Títol: Verificació de sistemes concurrents basada en *Constraint-Programming*

Volum: I / I

Alumne: Josep Maria Royuela Alcázar

Director/Ponent: Josep Carmona Vargas

Departament: Llenguatges i Sistemes Informàtics

Data: 12 de Juliol del 2010

DADES DEL PROJECTE

Títol del Projecte: Verificació de sistemes concurrents basada en *Constraint-Programming*

Nom de l'estudiant: Josep Maria Royuela Alcázar

Titulació: Enginyeria en Informàtica

Crèdits: 37,5

Director/Ponent: Josep Carmona Vargas

Departament: Llenguatges i Sistemes Informàtics

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Francisco Javier Larrosa Bondia

Vocal: Montserrat Maureso Sánchez

Secretari: Josep Carmona Vargas

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D'INFORMÀTICA DE BARCELONA

PROJECTE FINAL DE CARRERA

Verificació de sistemes concurrents basada en *Constraint-Programming*

ENGINYERIA INFORMÀTICA

Alumne: ROYUELA ALCÁZAR, Josep Maria
Director/Ponent: CARMONA VARGAS, Josep

Agraïments

a la meva família – Mama, Papa i Sariur- pel seu constant i infinit ajut i recolzament. I, amb especial èmfasi, a l'Ani i a la Nane que, des d'on siguin, segur que se senten orgullosos d'aquesta segona titulació.
No puc oblidar-me tampoc de la Montse qui, en aquesta última etapa, m'ha empès i comprès com qui més.
I, com no, agraïments també pel meu tutor Josep, per la seva inestimable direcció.

"*En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.*[…]Vale."

El ingenioso hidalgo don Quijote de la Mancha

Miguel de Cervantes Saavedra

Índex

1 Prefaci

La concurrència és la coincidència, concurs simultani de diverses circumstàncies[1]. És a dir, és la coincidència en l'espai o en el temps de dues o més persones o coses. Al cas de la informàtica, és l'existència simultània de dos o més processos en execució. El paral·lelisme és un cas particular de la concurrència en què, l'execució de les instruccions és simultània. Exemples de paral·lelisme els trobem a arquitectures paral·leles, processament paral·lel, algorismes paral·lels i programació paral·lela.

Una manera de veure la concurrència és, com un conjunt d'activitats que es desenvolupen de forma simultània a les quals, s'anomena processos. Així, es pot trobar la concurrència en el *hardware*, com a eina per a solucionar problemes com l'execució paral·lela d'instruccions, el funcionament paral·lel de perifèrics, els processadors múltiples i els sistemes distribuïts. Per tant, un sistema concurrent és un sistema informàtic en el qual, la concurrència hi juga un paper molt important. Per exemple, els sistemes operatius, els sistemes gestors de bases de dades (DBMS), els sistemes en temps real i els sistemes distribuïts.

Tenim que, la concurrència està present a la naturalesa i als sistemes informàtics. I, aquí, és on apareix la necessitat de modelar sistemes concurrents i, la motivació per desenvolupar una eina que permeti l'anàlisi i la verificació de propietats sobre els mateixos.

[1] Definició del *Diccionario de la Real Academia Española de la Lengua.*

2 Introducció

2.1 Context

Els sistemes de transicions (TS) són una formalització específica dels sistemes de canvi d'estats en els quals, un sistema és representat per un conjunt de variables i un estat consisteix en una assignació de valors a aquests estats. En aquest projecte ens centrem en els sistemes concurrents, cada vegada més presents al món informàtic. Avui dia trobem sistemes concurrents tant pel que fa referència al *hardware*, com per exemple amb els sistemes multiprocessador, com en el *software* el qual, és desenvolupat per tal d'aprofitar aquestes noves arquitectures.

L'ús de models ajuda a l'anàlisi dels sistemes. Aquests ens permeten la modelització de sistemes reals de manera que podem experimentar situacions límit així com verificar propietats que compleix el sistema. Les xarxes de Petri (PN) han estat proposades per a una àmplia varietat d'aplicacions. Això és degut a la generalitat i permissivitat inherent a les PNs. Aquestes es poden aplicar informalment a qualsevol àrea o sistema que es pugui descriure gràficament com a un diagrama de fluxos i que necessiti algun mitjà de representació d'activitats paral·leles o concurrents.

Dues àrees d'aplicació meritoses de les PNs són l'avaluació del rendiment i els protocols de comunicació. S'aplica però a moltes altres àrees, com la modelització i anàlisi de sistemes de *software* distribuït, sistemes de bases de dades distribuïdes, programes concurrents i paral·lels, sistemes de control industrial flexible, sistemes discrets, *Multiprocessor Memory Systems* (MMS), entre d'altres. Tanmateix, s'ha d'arribar a un compromís entre la generalitat del modelatge i la capacitat d'anàlisi. És a dir, com més general és el model menys amè és el seu anàlisi. A continuació es mostra una PN corresponent a un sistema multiprocessador, i el seu graf d'abastabilitat.

Exemple 2.1: Sistema multiprocessador

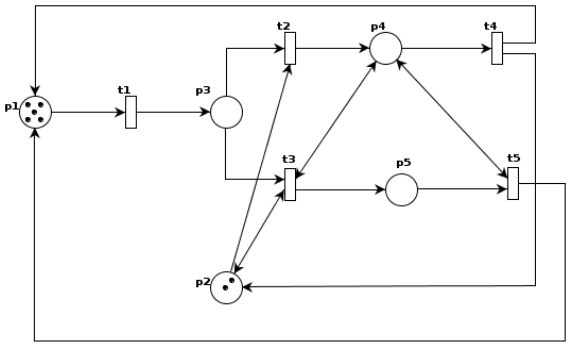


Figura 2.1: Una PN modelant un sistema multiprocessador on, els *tokens* a p_1 representen processadors actius, p_2 els busos disponibles, p_3, p_4 i p_5 els processadors esperant tenir accés, encuats a memòries comunes, respectivament.

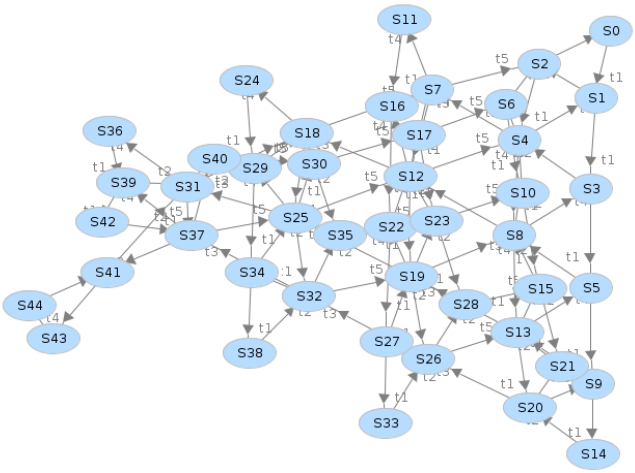


Figura 2.2: Graf d'abastabilitat del sistema multiprocessador de la figura 2.1.

Aquest és el motiu (simplicitat i generalitat) pel qual, aquest projecte es centra en la modelització de sistemes concurrents mitjançant PNs per tal d'analitzar i verificar propietats dels mateixos.

La lògica temporal lineal en el temps (LTL) es va introduir per tal d'especificar les propietats de les execucions d'un sistema. , aquest projecte utilitza la LTL per tal de verificar les propietats temporals dels sistemes. L'abast del projecte no se centra només en la verificació de propietats estàtiques (operadors *booleans* estàndards), sinó que pretén verificar propietats dinàmiques (operadors temporals) del mateix. Aquest és el motiu pel qual, el projecte inclou metodologies per tal de traduir aquestes propietats LTL en estructures de dades tals que puguin ser validades.

El *model-checking* és una tècnica per a verificar sistemes concurrents d'estats finits tals com, el disseny de circuits seqüencials i els protocols de comunicació. Té un gran nombre d'avantatges respecte de les aproximacions tradicionals basades en simulació, test i, raonament inductiu. En particular, el *model-checking* és automàtic i, usualment, molt ràpid. A més, si el disseny conté un error, el *model-checking* produeix un contraexemple que pot ser usat com a eina per arribar a l'origen de l'error. Aquest mètode, ha estat usat satisfactòriament a la pràctica en la verificació de dissenys industrials reals i, les companyies estan començant a comercialitzar *model-checkers*.

A més, la programació amb restriccions (CP) es relaciona molt amb la programació lògica i amb la investigació operativa. De fet, qualsevol programa lògic pot ser traduït a un programa amb restriccions i viceversa. Moltes vegades, els programes lògics són traduïts a programes amb restriccions degut a què, la solució és més eficient que la seva contrapartida. La diferència entre ambdós radica principalment en els seus estils i enfocaments al modelatge del món. Per a certs problemes és més natural (i, per tant, més simple) implementar-los com a programes lògics mentre que, en d'altres, és més natural implementar-los com a programes amb restriccions.

És per això que, aquest projecte inclou la CP com a eina de resolució de la verificació de propietats LTL en sistemes concurrents modelats mitjançant PNs. I, en particular, empra una eina de CP actual (*Gecode*) la qual, proporciona moltes possibilitats respecte de les seves predecessores com són la possibilitat d'implementar noves restriccions, l'eficiència i, la visualització gràfica dels arbres de cerca de les solucions del nostre sistema.

2.2 Objectius del projecte

L'objectiu d'aquest projecte és la verificació de propietats temporals LTL en sistemes concurrents modelats amb xarxes de Petri mitjançant *constraint programming*.

Per tal d'assolir aquest objectiu, en primer lloc s'ha de realitzar un estudi previ de les xarxes de Petri, les fórmules LTL, el *model-checking* i, la *constraint programming*.

Així, l'objectiu principal del projecte es pot dividir de la següent manera:

(1) Xarxes de Petri.

- Estudi de les xarxes de Petri: anàlisi, propietats i aplicacions.
- PNML[2]:
    i. Anàlisi i comprensió de les estructures definides.
    ii. Realització d'un *parser*.
- Creació d'una estructura de dades que contingui tota la informació necessària per a la verificació de les propietats LTL a les PNs.

(2) *Model-checking* de fórmules LTL usant *constraint programming*.

- Estudi de les fórmules LTL:
    i. Fórmules que defineixen, representació i propietats.
    ii. Realització d'un *parser* mitjançant una gramàtica en ANTLR.
    iii. Anàlisi i comprensió de l'estructura d'autòmats.
    iv. Aplicació d'un algorisme de construcció d'autòmats a partir de fórmules LTL.
- *Model-checking* de sistemes concurrents mitjançant *constraint programming*.
    i. Realització de la composició de l'autòmat construït a partir de la fórmula LTL i la xarxa de Petri.
    ii. Implementació de diversos tests de semidecisió sobre la satisfacció de propietats LTL sobre sistemes concurrents, mitjançant una llibreria de *constraint programming*.

Per tant, mitjançant la integració de diverses llibreries externes i la implementació dels diferents conceptes i algorismes de resolució necessaris, s'ha d'implementar una eina capaç de verificar propietats LTL sobre sistemes concurrents, modelats amb xarxes de Petri, mitjançant *constraint programming*.

[2] PNML: *Petri Net Markup Language*.

2.3 Estructura de la memòria

L'estructura de la memòria, a continuació, és la següent:

Capítol 3 - Teoria bàsica: on es presenten els conceptes i definicions necessaris per a la comprensió del projecte. S'introdueix als sistemes de transicions, a la teoria de xarxes de Petri, a les fórmules LTL, al *model-checking* i a la *constraint programming*.

Capítol 4 - Algorismes: on es descriuen i expliquen amb detall els algorismes implementats amb un exemple d'aplicació pas a pas.

Capítol 5 - Enginyeria del software: on es presenta l'especificació formal de l'aplicació, alguns aspectes sobre el seu disseny i, la seva implementació així com les tecnologies emprades.

Capítol 6 - Experimentació: on es presenten diversos exemples d'aplicació pas a pas de l'aplicació implementada, així com els resultats obtinguts i comparatives entre aquests.

Capítol 7 - Valoració econòmica: on es presenta la planificació de la realització del projecte, així com la valoració econòmica del mateix.

Capítol 8 - Conclusions: on es presenten les conclusions a les quals s'ha arribat en finalitzar el projecte, així com una descripció del treball futur a realitzar.

Capítol 9 - Bibliografia: on es presenta un ampli repertori de referències emprades, d'utilitat per tal d'ampliar o aprofundir en alguns dels temes tractats al projecte.

Annex A - Teoria de *Petri Nets*: on es presenta una ampliació de la teoria de xarxes de Petri presentada al projecte.

3 Teoria bàsica

3.1 Sistema de transicions

Un TS és un model que descriu el comportament dels sistemes. És un dígraf on els nodes representen *estats* i, les arestes modelen *transicions*. En aquest sentit, un estat és el valor actual del sistema i una transició representa un canvi d'estat.

Definició 3.1: (*Transition system*) Un sistema de transicions etiquetat és una 4-tupla T = (Act, Q, Δ, q_0), on:

(1) Act és un alfabet d'accions;

(2) Q és un conjunt d'estats;

(3) Δ és un conjunt de transicions (etiquetades) tal que $\Delta \subseteq Q$ x Act x Q;

(4) $q_0 \in Q$ és l'estat inicial. □

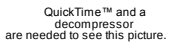


Figura 3.1: TS d'una màquina de venda de begudes.

Un recorregut complet (*full run*) d'un TS etiquetat és una seqüència infinita $q_0\, a_0\, q_1\, a_1\, q_2 \ldots$ tal que $(q_i,a_i,q_{i+1}) \in \Delta$ per a tot $i \geq 0$.

Quan els TS etiquetats són usats com a semàntica de processos algebraics, només les etiquetes de les transicions contenen informació rellevant; els estats entremitjos són irrellevants. En aquest cas, es parla de semàntica *action-based* (*basada en l'acció*). Tenim que, una seqüència infinita $a_0\, a_1\, a_2 \ldots$ d'accions de T és un recorregut sobre les accions si existeix un recorregut complet $q_0\, \underline{a_0}\, q_1\, \underline{a_1}\, q_2 \ldots$. El llenguatge sobre les accions L_a(T) de T és el conjunt de tots els recorreguts sobre les accions.

Quan els TS etiquetats són usats com a semàntica de llenguatges amb variables, la informació dels valors actuals de les variables està codificat als estats; les etiquetes de les transicions són usualment irrellevants. En aquest cas, es parla de semàntica *state-based* (*basada en l'estat*). Tenim que, una seqüència infinita $q_0\, q_1\, q_2 \ldots$ d'estats de T és un recorregut sobre els estats si existeix un recorregut complet $q_0\, \underline{a_0}\, q_1\, \underline{a_1}\, q_2 \ldots$. El llenguatge sobre els estats L_s(T) de T és el conjunt de tots els recorreguts sobre els estats.

Per a semàntiques basades en l'estat, és convenient usar TS (no etiquetats). Un TS (no etiquetat) és una 3-tupla T = (Q, Δ, q_0) on, $\Delta \subseteq Q$ x Q. Es pot considerar com un cas particular de TS etiquetat en què, totes les transicions tenen la mateixa etiqueta.

3.2 Xarxes de Petri

Les PNs són una eina de modelatge gràfic i matemàtica aplicable a molts sistemes. Són una prometedora eina per a descriure i analitzar sistemes de processament d'informació que es caracteritzen per ser concurrents, asíncrons, distribuïts, paral·lels, no deterministes i/o estocàstics.

Com a eina de modelatge gràfic, les PNs poden ser usades com a una ajuda a la comunicació visual similar als diagrames de fluxos, als diagrames de blocs i, a les xarxes. A més, els *tokens* són usats, en aquestes xarxes, per tal de simular les activitats dinàmiques i concurrents dels sistemes.

Com a eina matemàtica, és possible d'establir equacions d'estat, equacions algebraiques i, d'altres models matemàtics que governen el comportament dels sistemes.

La teoria aquí explicada està extreta de la referència [2]. Una ampliació de la mateixa es pot trobar a l'annex A Teoria de Petri Nets.

3.2.1 Definició

Una PN és un tipus particular de graf dirigit, juntament amb un estat inicial anomenat *marcatge inicial*, M_0. El subjacent graf N d'una PN és un graf dirigit, ponderat i bipartit el qual, consta de dos tipus de nodes, anomenats *places* i *transicions*, on els arcs són sempre entre un *place* i una transició o bé, entre una transició i un *place*.

En quant a la seva representació gràfica, els *places* es representen mitjançant cercles i les transicions mitjançant barres o caixes. Els arcs estan etiquetats amb els seus pesos (enters positius) on, un *k*-arc es pot interpretar com a un conjunt de *k* arcs paral·lels.

Un *marcatge* assigna a cada *place* un enter no negatiu. Si un marcatge assigna a un *place p* un enter no negatiu *k* llavors, *p* està marcat amb *k tokens*. Gràficament, es posen *k* punts negres (*tokens*) en el *place p*. Un marcatge es denota per *M*, un *m*-vector on, *m* és el nombre total de *places*. El *p*-èssim component de *M*, denotat per *M*(*p*), és el nombre de *tokens* al *place p*.

Definició 3.2: (*Petri net*) Una xarxa de Petri és una 5-tupla PN = (P, T, F, W, M_0), on:

(1) $P = \{p_1, p_2, \ldots, p_m\}$ és un conjunt finit de *places*;

(2) $T = \{t_1, t_2, \ldots, t_n\}$ és un conjunt finit de transicions;

(3) $F \subseteq (P \times T) \cup (T \times P)$ és un conjunt d'arcs (flux);

(4) $W: F \rightarrow \{1, 2, 3, \ldots\}$ és una funció de pes;

(5) $M_0: P \rightarrow \{0, 1, 2, 3, \ldots\}$ és el marcatge inicial;

(6) $P \cap T = \varnothing$ i $P \cup T \neq \varnothing$.

Una PN tal que $N = (P, T, F, W)$ sense cap marcatge inicial, es denota per N.

Una PN amb un marcatge inicial donat, es denota per (N, M_0).

□

El comportament de molts sistemes es pot descriure en termes d'estats del sistema i els seus canvis. Amb l'objectiu de simular el comportament dinàmic d'un sistema, un estat o un marcatge en una PN és canviat d'acord a la següent regla de transició (*disparament*).

Definició 3.3: (*Transition (firing) rule*)

(1) Una transició *t* es diu que està *disponible* si cada *place* d'entrada (*input place*) de *t* està marcat amb almenys *w*(*p*,*t*) *tokens* on, *w*(*p*,*t*) és el pes de l'arc de *p* a *t*.

(2) Una transició disponible pot o no disparar (depenent de si l'esdeveniment actual pren lloc o no).

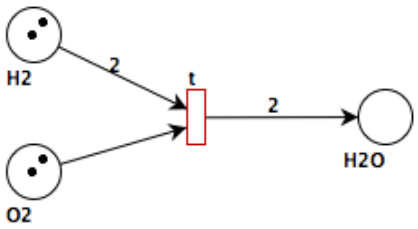
(3) Una transició disponible *t* que es dispara, esborra *w*(*p*,*t*) *tokens* de cada *place* d'entrada *p* de *t* i, afegeix *w*(*p*,*t*) *tokens* a cada *place* de sortida (*output place*) *p* de *t* on, *w*(*p*,*t*) és el pes de l'arc de *t* a *p*.

□

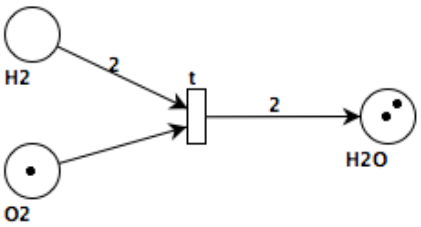
Una transició sense cap *place* d'entrada s'anomena *transició font* (*source transition*) i, una sense cap *place* de sortida s'anomena *transició pica* (*sink transition*). Notar que una transició font és incondicionalment disponible i que, el disparament d'una transició pica consumeix *tokens* però no en produeix cap.

Un parell format per un *place p* i una transició *t* s'anomena *auto-bucle* (*self-loop*) si *p* és alhora *place* d'entrada i de sortida de *t*. Una PN s'anomena *pura* (*pure*) si no conté cap auto-bucle. Una PN s'anomena *ordinària* (*ordinary*) si el pes de tots els seus arcs és 1.

Exemple 3.1: ($2H_2 + O_2 \rightarrow 2H_2O$)



(a)



(b)

Figura 3.2: Exemple de Regla de disparament:
(a) El marcatge abans del disparament habilita la transició *t*.
(b) El marcatge posterior al disparament de *t* on, *t* ja no està disponible.

3.2.2 Algunes propietats

Dos tipus de propietats es poden estudiar amb un model PN: aquelles que de penen del marcatge inicial i, aquelles que en són independents. Les propietats dependents del marcatge s'anomenen propietats d'entorn mentre que, les propietats que no, s'anomenen propietats estructurals. A continuació, es mostren diverses propietats bàsiques de les PNs i l'anàlisi dels seus problemes.

3.2.2.1 Abastabilitat (*Reachability*)

L'*abastabilitat* és una base fonamental per a l'estudi de les propietats dinàmiques d'un sistema. El disparament d'una transició disponible canviarà la distribució de *tokens* de la xarxa (*marcatge*). Un marcatge M_n és *abastable* des d'un marcatge M_0 si existeix una seqüència de disparaments que transforma M_0 en M_n. Un *disparament* o *seqüència ocurrent* es denota per $\sigma = M_0\ t_1\ M_1\ t_2\ M_2\ \dots\ t_n\ M_n$ o, simplement, $\sigma = t_1\ t_2\ \dots\ t_n$. En aquest cas, M_n és abastable des de M_0 per σ i, es denota per $M_0\ [\sigma > M_n]$.

El conjunt de tots els possibles marcatges assolibles des de M_0 a una xarxa (N, M_0) es denota per $R(N, M_0)$ o, simplement, $R(M_0)$. El conjunt de totes les possibles seqüències de disparaments des de M_0 a una xarxa (N, M_0) es denota per $L(N, M_0)$ o, simplement, $L(M_0)$.

Per tant, el *problema de l'abastabilitat* per a PNs és el problema de trobar si $M_n \in (N, M_0)$, per a un marcatge donat M_n a una xarxa (N, M_0). El problema de l'abastabilitat és decidible tot i que, com a mínim, exponencial a l'espai i en el temps, en el cas general. El *problema de la igualtat* és indecidible. No existeix cap algorisme per tal de determinar si $L(N, M_0) = L(N', M'_0)$ per a dues PNs N i N'.

3.2.3 Matriu d'incidència i equació d'estat

El comportament dinàmic de molts sistemes estudiats a l'enginyeria pot ser descrit mitjançant equacions diferencials o equacions algebraiques.

Matriu d'incidència

Per a una PN N amb n transicions i m *places*, la *matriu d'incidència* $A = [a_{ij}]$ és una matriu n x m d'enters i la seva entrada típica ve donada per:

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} \tag{1}$$

on, $a_{ij}^{+} = w(i, j)$ és el pes de l'arc de la transició i al seu *place* de sortida j i, $a_{ij}^{-} = w(j, i)$ és el pes de l'arc a la transició i del seu *place* d'entrada j.

a_{ij}^{-}, a_{ij}^{+}, a_{ij} representen, respectivament, el nombre de *tokens* eliminats, afegits i canviats al *place* j quan es dispara la transició i. Una transició i està disponible per a un marcatge M si i, només si:

$$a_{ij}^{-} \le M(j), \quad j = 1, 2 \cdots m. \tag{2}$$

Equació d'estat

Un marcatge M_k s'escriu com un vector m x 1. La j-èssima entrada de M_k denota el nombre de *tokens* al *place* j immediatament després del k-èssim disparament en alguna seqüència de disparament. El k-èssim disparament o *vector de control* u_k és un vector n x 1 d'entrada n-1 0's i un 1 a la i-èssima posició indicant que, la transició i es dispara al k-èssim disparament. Podem escriure, per tant, la següent equació d'estat per a una PN:

$$M_k = M_0 + A^T u_k, \quad k = 1, 2, \cdots. \tag{3}$$

□

Exemple 3.2:

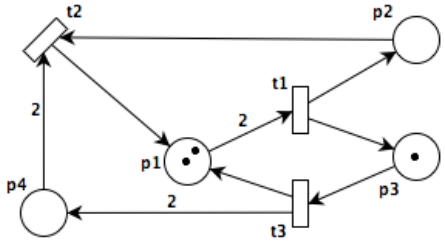


Figura 3.3: PN Exemple 3.2.

L'equació d'estat (3) on, la transició t_3 es dispara per assolir el marcatge $M_1 = (3\ 0\ 0\ 2)^T$ des de $M_0 = (2\ 0\ 1\ 0)^T$ és:

$$\begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

□

3.3 LTL (*Linear-time Temporal Logic*)

Linear-time temporal logic (LTL) és un formalisme per a especificar propietats temporals en una lògica determinada. La semàntica de LTL usualment defineix si una execució σ d'un sistema donat satisfà una fórmula. LTL està construïda a partir d'operadors *booleans* estàndards els quals, expressen propietats estàtiques i, per operadors temporals els quals, expressen propietats dinàmiques en el temps.

Definició 3.5: (LTL) Sigui Σ un alfabet finit i, sigui Π un conjunt de proposicions sobre Σ, és a dir, un conjunt de mapejos amb Σ com a domini i el conjunt $\{cert, fals\}$ com a rang. El conjunt de fórmules de *lògica temporal lineal en el temps* (LTL) sobre el conjunt Π es defineix per inducció com segueix:

- si $\phi \in \Pi$ llavors, ϕ és una fórmula;
- si ϕ i ψ són fórmules llavors, també ho són $\phi \wedge \psi$, $\neg\phi$, $X\phi$ i, ϕ U ψ.

Es fa ús de les següents abreviatures:

- $\phi \vee \psi = \neg (\neg\phi \wedge \neg\psi)$;
- $\phi \, \mathrm{V} \, \psi = \neg (\neg\phi \, \mathrm{U} \, \neg\psi)$, (dual de U);
- $\Diamond \, \phi =$ **cert** U ϕ, (eventualment);
- $\Box \, \phi = \neg \Diamond \neg\phi$, (sempre).

Una interpretació d'una fórmula LTL és una paraula infinita $\xi \in \Sigma^{\omega}$. Per tal de definir formalment la relació de satisfacció $\models$ de LTL, sigui $\xi(0)$ el primer element de ξ i, sigui $\xi^{(i)}(x) = \xi(x + i)$ el sufix de ξ que comença a la i-èssima posició. Tenim:

- $\xi \models \pi$, per a $\pi \in \Pi$ si $\pi(\xi(0)) = cert$;
- $\xi \models \neg\phi$, si *no* $\xi \models \phi$;
- $\xi \models \phi \wedge \psi$, si $\xi \models \phi$ i $\xi \models \psi$;
- $\xi \models X\phi$, si $\xi^{(1)} \models \phi$;
- $\xi \models \phi \, \mathrm{U} \, \psi$, si $\exists i \in \mathrm{N}$: $\xi^{(i)} \models \psi$ i $\forall j \leq i$: $\xi^{(j)} \models \phi$.

□

El llenguatge $L(\phi)$ d'una fórmula ϕ sobre Π és el conjunt de totes les paraules de Σ^{ω} que satisfan ϕ.

3.4 *Model checking*

Al camp de la lògica a la ciència de computadors, el *model-checking* fa referència al següent problema: donat un model d'un sistema, testejar automàticament si aquest model compleix una especificació donada. Típicament, els sistemes als quals es fa referència són sistemes *hardware* i *software* i, l'especificació conté requeriments de seguretat tals com l'absència de *punts morts* (*deadlocks*) i estats crítics similars que poden causar la fallida del sistema.

Per tal de resoldre aquest problema algorísmicament ambdós, el model del sistema i l'especificació, han de ser formulats en un cert llenguatge matemàtic. En aquest sentit, és formulat com una tasca en lògica, és a dir, controlar si l'estructura donada satisfà una fórmula lògica donada. El concepte és general i s'aplica en tot tipus d'estructures lògiques i convenients. Un problema simple de *model-checking* és el de verificar si una fórmula donada en la lògica proposicional és satisfeta per una estructura donada.

El problema de *model-checking* per a una PN segura i una fórmula LTL consisteix en decidir, donada una PN segura i una fórmula LTL, si la PN satisfà la propietat especificada a la fórmula.

3.5 *Constraint Programming*

La *constraint programming* (programació amb restriccions) és una aproximació alternativa a la programació convencional ja que, depèn d'una combinació de tècniques entre el raonament i la computació. El concepte principal és la restricció. Informalment, una restricció sobre una seqüència de variables és una relació entre els seus dominis. Es pot veure com la combinació de valors sobre els dominis de la variable que són admesos. I, per tant, un problema de satisfacció de restriccions consisteix en un conjunt finit de restriccions, cadascuna sobre una subseqüència de variables.

CP és un paradigma de la programació a la informàtica on, les relacions entre les variables són expressades en termes de restriccions (equacions). Actualment, és usada como una tecnologia de *software* per a la descripció i resolució de problemes combinatoris particularment difícils, en especial, a les àrees de planificació i programació de tasques (calendarització).

Es tracta d'un paradigma de programació basat en l'especificació d'un conjunt de restriccions les quals, han de ser satisfetes per qualsevol solució del problema plantejat, en lloc d'especificar les passes per tal d'obtenir aquesta solució.

L'enfoc de la CP es basa principalment en buscar un estat al qual, una gran quantitat de restriccions siguin satisfetes simultàniament. Un problema es defineix típicament com a un estat de la realitat en el qual, existeix un nombre de variables amb valor desconegut. Un programa basat en restriccions busca aquests valors per a totes les variables.

Degut a què l'objectiu del projecte és la verificació de propietats LTL en sistemes concurrents modelats amb xarxes de Petri mitjançant *constraint programming*, l'elecció d'una eina adequada per a tal objectiu és bàsica. En aquest sentit, s'ha considerat la decisió d'escollir una llibreria actual com *Gecode* la qual, proporciona una capacitat d'implementació elevada i, permet la visualització dels arbres de cerca de les solucions dels nostre sistema, a més de ser eficient.

4 Algorismes

4.1 Visió general

L'objectiu del projecte és la implementació d'una aplicació de verificació de propietats LTL a sistemes concurrents modelats amb xarxes de Petri, especificades en PNML, mitjançant *constraint programming*, amb la realització d'un algorisme de semidecisió.

En aquest sentit, les tasques a realitzar per l'aplicació són les següents:

- Carregar una xarxa de Petri especificada en PNML.

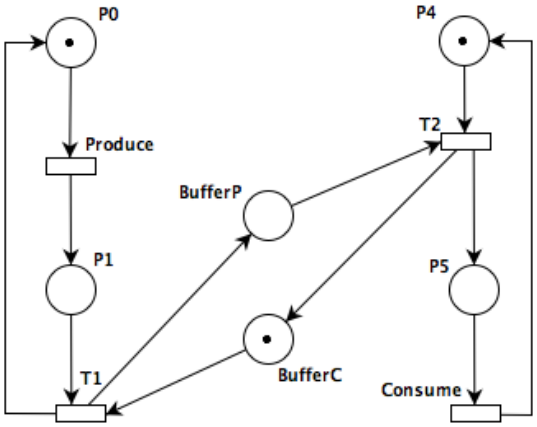


Figura 4.1: Càrrega d'una PN modelant un sistema productor/consumidor *pull*.

- Construir un autòmat (l'autòmat de la fórmula negada) a partir de la fórmula LTL introduïda el qual, accepta exactament totes les seqüències infinites que violen la fórmula ϕ.

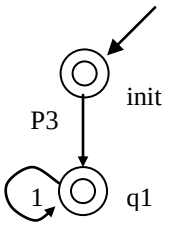


Figura 4.2: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg p3 \wedge \neg p2$.

- Construir un Büchi autòmat corresponent al producte de la xarxa de Petri i de l'autòmat generat el qual, accepta totes les infinites computacions de la xarxa de Petri que són acceptades per la propietat LTL, és a dir, totes les infinites computacions que violen ϕ. Aquest apartat es farà mitjançant dues metodologies.

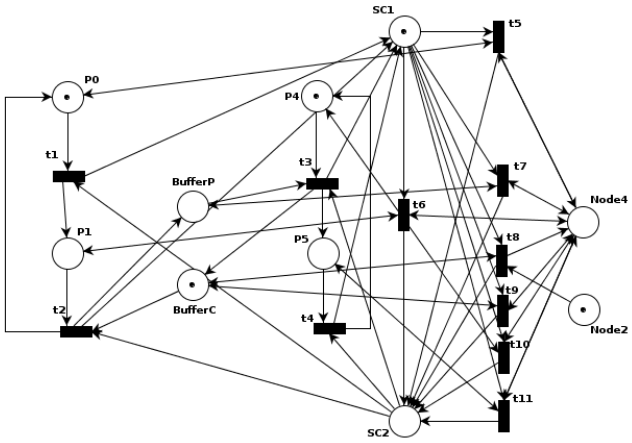


Figura 4.3: Büchi autòmat producte de la PN carregada i l'autòmat de la fórmula negada generat.

- *Xequejar* si l'autòmat producte és buit, és a dir, si accepta seqüències no infinites. D'aquesta manera, s'obté que la xarxa de Petri satisfà la propietat LTL ϕ si i, només si, l'autòmat producte és buit. La tècnica a emprar, mitjançant *model-checking* és el *test de semidecisió* pel qual, un procediment pot respondre 'sí', cas en què la propietat a *xequejar* es compleix o bé, pot respondre 'no sé' el qual, s'implementarà via *constraint programming*.

A continuació es presenten els dos algorismes principals de l'aplicació a implementar per a la realització d'aquest procediment.

4.2 Construcció d'un autòmat a partir d'una LTL

L'algorisme aquí presentat així com la teoria associada al mateix, han estat extrets, principalment, de la referència [4]. Al document al qual es fa referència, es pot trobar una explicació encara més exhaustiva del tema.

A continuació es presenta un algorisme basat en una taula (*tableau-based*), per a l'obtenció d'un autòmat a partir d'una propietat LTL. L'algorisme està dirigit a ser usat en *model-checking* d'una manera *on-the-fly*, és a dir, l'autòmat pot ser construït simultàniament amb, i guiat per, la generació del model. En particular, és possible detectar que la propietat no es compleix només amb la construcció de part del model i de l'autòmat. L'algorisme també pot ser usat per *xequejar* la validesa d'una afirmació lògica temporal. Tot i que el problema general és *PSPACE*-complet, experiments realitzats mostren un bon comportament per a fórmules temporals típicament usades a la verificació.

4.2.1 Introducció

Aquest algorisme se centra en una classe de mètodes basats en l'espai d'estats anomenat *model-checking*. La idea del *model-checking* és veure la verificació com un *xequeig* de si el graf que representa l'espai d'estats del protocol satisfà (és model de) la propietat a ser *xequejada*. Específicament, se centra en el *model-checking* de fórmules lògiques temporals lineals en el temps (LTL). En aquest sentit, es pot *xequejar* que totes les seqüències d'execucions infinites que es poden extreure del graf de l'espai d'estats satisfan (són models de) la fórmula lògica temporal o, equivalentment que, cap d'aquestes seqüències falsifica la fórmula.

El problema de *model-checking* així com el problema de validació de LTL és *PSPACE*-complet. A la pràctica, les aplicacions de mètodes de *model-checking* tenen dues limitacions complexes principals:

- la mida de l'autòmat, tant per al protocol com per a la propietat ja que, el temps d'execució és proporcional al producte del nombre de nodes de l'autòmat;
- i, la mida de part de l'autòmat producte el qual, s'ha de mantenir a memòria per tal de *xequejar* el seu buit (*emptyness*) ja que, la memòria disponible és una limitació per a la mida dels problemes que es poden tractar.

L'autòmat corresponent a la propietat pot tenir fins a $2^{O(n)}$ nodes on, n és el nombre de subfórmules de la pròpia fórmula. Així, la mida de l'autòmat producte el qual, determina la complexitat del mètode, és proporcional a $N \cdot 2^{O(n)}$ on, N és el nombre d'estats (abastables).

L'algorisme aquí presentat té com a objectiu mantenir l'autòmat generat tant petit com sigui possible i, amb una implementació senzilla. Així, es procedeix *on-the-fly* en el sentit de què, l'autòmat només és generat a mida que es necessita durant el procés de verificació. Tècnicament, l'algorisme tradueix una LTL proposicional en un *Büchi autòmat general* usant cerca en profunditat (DFS).

4.2.2 Quadre de construcció

L'objectiu és la construcció d'un autòmat (TS) el qual, generi totes les infinites seqüències que satisfan una LTL φ. L'autòmat que es construeix és un *Büchi autòmat general*, és a dir, un Büchi autòmat amb múltiples conjunts d'estats acceptadors, en contraposició al *Büchi autòmat simple* el qual, només té un conjunt d'estats acceptadors.

Definició 4.1: (*Generalized Büchi Automaton*) Un Büchi autòmat general és una 4-tupla GBA = (Q, I, $\rightarrow$, F), on:

(1) Q és un conjunt finit d'estats;

(2) $I \subseteq Q$ és el conjunt d'estats inicials;

(3) $\rightarrow \subset Q$ x Q és la relació de transició;

(4) $F \subseteq 2^{2^Q}$ és el conjunt d'estats acceptadors $F = \{F_1, F_2, \ldots F_n\}$. Notar que F pot ser buit.

□

Una *execució* de GBA és una seqüència infinita $\sigma = q_0\, q_1\, q_2 \ldots$ tal que $q_0 \in I$ i, per a cada $i \geq 0$, $q_i \rightarrow q_{i+1}$. Una *execució acceptadora* σ és una execució tal que, per a cada conjunt acceptador $F_i \in F$, existeix com a mínim un estat $q \in F_i$ el qual, apareix amb freqüència infinita a σ.

L'autòmat aquí definit no té entrada i, per tant, no defineix cap seqüència. Per tant, es necessita afegir etiquetes a l'autòmat. Aquestes són afegides als estats. Un *Büchi autòmat general etiquetat* (LGBA) és un 3-tupla (GBA, *D*, *L*) on, GBA és un Büchi autòmat general, *D* és algun domini finit i, L: $Q \rightarrow 2^D$ és una *funció d'etiquetatge* dels estats de A als subconjunts del domini *D* (un estat té un conjunt d'etiquetes de *D*). Un LGBA *accepta* una paraula $\xi = x_0\, x_1\, x_2\, \ldots$ de D^w si i, només si, existeix una execució acceptadora $\sigma = q_0\, q_1\, q_2\, \ldots$ de A tal que, per a cada $i \geq 0$, $x_i \in L(q_i)$. Llavors, es diu que l'execució σ accepta ξ.

El procediment de construcció de l'autòmat genera un graf el qual, defineix els estats i transicions de l'autòmat. Els nodes del graf estan etiquetats per conjunts de fórmules i s'obtenen per descomposició de fórmules d'acord amb la seva estructura *booleana* i, per expansió dels operadors temporals amb l'objectiu de separar allò que és immediatament cert d'allò que ho ha de ser al següent estat. La identitat fonamental usada per això és $\mu\, \mathrm{U}\, \psi \equiv \psi \vee (\mu \wedge \mathrm{X}(\mu\, \mathrm{U}\, \psi))$.

4.2.2.1 Estructura de dades

L'estructura de dades usada per a representar els nodes del graf conté informació suficient per tal que l'algorisme de construcció del graf sigui capaç d'operar de manera DFS.

Un *node* del graf conté els següents caps:

- *Name*: *string* (identificador) que representa el nom del node.
- *Incoming*: les arestes d'entrada representades pels noms dels nodes amb aresta de sortida que porta al node actual.
- *New*: un conjunt de propietats temporals (fórmules) les quals, s'han de complir a l'estat actual i encara no han estat processades.
- *Old*: les propietats que s'han de complir al node i encara no han estat processades. Eventualment, *New* serà buit, deixant totes les seves obligacions a *Old*.
- *Next*: propietats temporals que s'han de complir a tots els estats que són immediatament successors dels estats que satisfan les propietats de *Old*.
- *Father*: durant la construcció, els nodes es divideixen. Aquest camp conté el nom del node del qual l'actual ha estat dividit. Aquest camp és usat només per raons de correctesa de l'algorisme i, no és important per a la construcció.

Es manté una llista de nodes *Nodes_Set* la construcció de la qual és completa, amb tots els nodes amb els camps anteriors. Els camps anteriors es denoten com segueix: el camp *New* del node *q* com *New*(*q*), etc..

4.2.2.2 Algorisme

Per tal de facilitar la construcció de l'autòmat, es considera que la fórmula ha estat primerament transformada de manera que, la fórmula està en *forma negada normal*, és a dir, les negacions han estat empeses cap als literals. Es considera també que els operadors F ($\equiv \Diamond$) i G ($\equiv \Box$) no apareixen a la fórmula.

Les línies que apareixen en la següent descripció de l'algorisme fan referència a als números que apareixen a l'algorisme 4.1 següent. L'algorisme de traducció de la fórmula φ comença amb un node (línies 38-39). Aquest node té una aresta d'entrada (*dummy*), etiquetada com a init per tal de marcar el fet que és el node inicial. Així, una vegada acabada la construcció de l'autòmat, un node serà inicial si i, només si, conté aquesta etiqueta a la seva llista d'arestes d'entrada.

Amb el node actual *N*, l'algorisme *xequeja* si n'hi han obligacions encara no processades a *New* (línia 4). Si no, el node actual ja està completament processat i llest per ser afegit a *Nodes_Set*. Si ja hi ha algun node a *Nodes_Set* amb les mateixes obligacions als camps *Old* i *Next* (línia 5), la còpia que ja existeix només necessita ser actualitzada en les seves arestes d'entrada; el conjunt d'arestes d'entrada de la nova còpia són afegides a la resta de còpies de *Nodes_Set* (línia 6).

Si no hi ha un node tal a *Nodes_Set* llavors, el node actual és afegit a la llista i, un nou node actual és creat com a successor seu com segueix (línies 8-10):

- Inicialment, hi ha una aresta des de *N* fins al nou node actual.
- El conjunt *New* conté inicialment el camp *Next* de *N*.
- Els conjunts *Old* i *Next* del nou node actual estan inicialment buits.

Quan es processa el node actual, una fórmula η de *New* és esborrada de la llista. En el cas en què η sigui una proposició o una negació d'una proposició (un literal) llavors, si $\neg\eta$ està a *Old*, el node actual és descartat ja que conté una contradicció (línies 16-17). Altrament, η és afegida a *Old* (si no hi està ja).

Quan η no és un literal, el node actual es pot dividir en dos (línies 21-26) o no (línies29-31, 33-35) i, noves fórmules són afegides als camps *New* i *Next* (línies 22-23, 25-26, 30-31, 34-35). Les accions exactes depenen de la forma de η i són com segueix:

- $\eta = \mu \wedge \psi$: llavors, μ i ψ són afegides a *New* ja que, la certesa d'ambdues fórmules és necessària per tal de complir η.
- $\eta = \mu \vee \psi$: llavors, el node es divideix, afegint μ al *New* d'una còpia i, ψ al de l'altra. Aquests nodes es corresponen amb les dues vies per les quals η es pot complir.
- $\eta = \mu \,\mathrm{U}\, \psi$: novament, el node es divideix: per a la primera còpia, μ és afegida al *New* i $\mu \,\mathrm{U}\, \psi$ al *Next*. Per a l'altra còpia, ψ és afegida al *New*. Aquesta divisió és explicada tot observant que $\mu \mathrm{U} \psi$ és equivalent a $\psi \vee (\mu \wedge \mathrm{X}(\mu \,\mathrm{U}\, \psi))$.
- $\eta = \mu \,\mathrm{V}\, \psi$: llavors, el node es divideix: ψ és afegida al *New* d'ambdues còpies, μ és afegida al *New* d'una còpia i, $\mu \mathrm{V} \psi$ és afegida al *Next* de l'altra. Aquesta divisió és explicada tot observant que $\mu \mathrm{V} \psi$ és equivalent a $\psi \vee (\mu \wedge \mathrm{X}(\mu \,\mathrm{V}\, \psi))$.
- $\eta = \mathrm{X}\mu$: llavors, μ és afegida al *Next* ja que, la certesa d'aquesta fórmula és necessària al node posterior per tal de complir η. En quant al node actual, no requereix de cap condició, no hi ha cap propietat específica a complir.

Les còpies són processades en ordre DFS, és a dir, quan l'expansió del node actual i dels seus successors ha finalitzat, la expansió de la segona còpia i dels seus successors comença.

La funció **new_name()** genera un nou *string* per a cada crida successiva. La funció **Neg** es defineix com: **Neg**(P_n) = $\neg P_n$, **Neg**($\neg P_n$) = P_n i, de manera similar per a les constants *booleanes* T i F. Les funcions **New1**(η), **New2**(η) i **Next1**(η) es defineix segons la taula següent:

η	New1(η)	Next1(η)	New2(η)
μ U ψ	$\{\mu\}$	$\{\mu$ U $\psi\}$	$\{\psi\}$
μ V ψ	$\{\psi\}$	$\{\mu$ V $\psi\}$	$\{\mu, \psi\}$
$\mu \vee \psi$	$\{\mu\}$	$\varnothing$	$\{\psi\}$

Taula 4.1: Taula de funcions de l'algorisme 4.1.

Algorisme 4.1: Construcció *on-the-fly* d'un autòmat a partir d'una LTL

```
1   record graph_node = [Name: string, Incoming: set of string,
2     New: set of formula, Old: set of formula, Next: set of formula];

3   function expand (Node, Nodes_Set)
4     si New(Node) = ∅ llavors
5       si ∃ND ∈ Nodes_Set amb Old(ND) = Old(Node) i Next(ND) = Next(Node)
6       llavors Incoming(ND) = Incoming(ND) ∪ Incoming(Node);
7         return (Nodes_Set);
8       sinó return (expand([Name⇐Father⇐new_name(),
9         Incoming⇐{Name(Node)}, New⇐{Next(Node)},
10        Old⇐∅, Next⇐∅], {Node} ∪ Nodes_Set));
11    sinó
12      sigui η ∈ New;
13      New(Node) := New(Node)\{η};
14      cas de η:
15      η = P_n, o ¬P_n o η = T o η = F ⇒
16        si η = F o Neg(η) ∈ Old(Node) // el node actual conté una contradicció
17        llavors return (Nodes_Set); // es descarta el node actual
18        sinó Old(Node) := Old(Node) ∪ {η};
19          return (expand(Node, Nodes_Set));
20      η = μ U ψ, o μ V ψ, o μ ∨ ψ ⇒
21        Node1 := [Name⇐new_name(), Father⇐Name(Node),
22          Incoming⇐Incoming(Node),New⇐New(Node)∪({New1(η)}\Old(Node)),
23          Old⇐Old(Node) ∪ {η}, Next⇐Next(Node) ∪ {Next1(η)}];
24        Node2 := [Name⇐new_name(), Father⇐Name(Node),
25          Incoming⇐Incoming(Node),New⇐New(Node)∪({New2(η)}\Old(Node)),
26          Old⇐Old(Node) ∪ {η}, Next⇐Next(Node)];
27        return (expand(Node2, expand(Node1, Nodes_Set)));
28      η = μ ∧ ψ ⇒
29        return (expand([Name⇐Name(Node), Father⇐Father(Node),
30          Incoming⇐Incoming(Node), New⇐New(Node) ∪ ({μ,ψ}\Old(Node)),
31          Old⇐Old(Node) ∪ {η}, Next⇐Next(Node)], Nodes_Set));
```

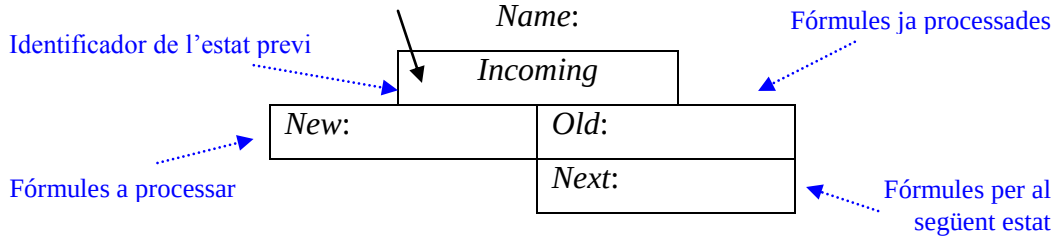
```
32        η = Xμ ⇒
33          return (expand([Name⇐Name(Node), Father⇐Father(Node),
34            Incoming⇐Incoming(Node), New⇐New(Node),
35            Old⇐Old(Node) ∪ {η}, Next⇐Next(Node) ∪ {μ}], Nodes_Set));
36  end expand;

37  function create_graph (φ)
38    return (expand([Name⇐Father⇐new_name(), Incoming⇐{init},
39      New⇐{φ}, Old⇐∅, Next⇐∅], Nodes_Set))
40  end create_graph;
```

□

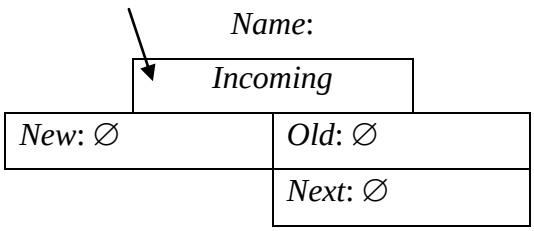
L'algorisme presentat genera un LGBA el qual, pot ser usat per a *model-checking* o per a *xequejar* la validesa d'una propietat temporal. En qualsevol cas, no cal completar la construcció de l'autòmat per tal de fer *model-checking*. La construcció dels nodes es pot fer per demanda a mida que s'interseca amb l'autòmat protocol. Llavors, quan els successors d'un node a l'autòmat de la propietat són construïts, no cal immediatament continuar amb la construcció dels seus successors. En canvi, és possible que la violació d'una propietat *xequejada* pugui ser descoberta abans de generar l'autòmat de la propietat sencera.

4.2.2.3 Visualització gràfica

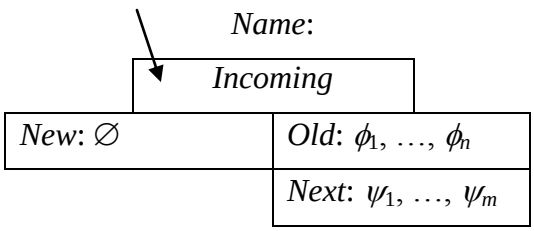
Estat de l'autòmat



Nodes inicials

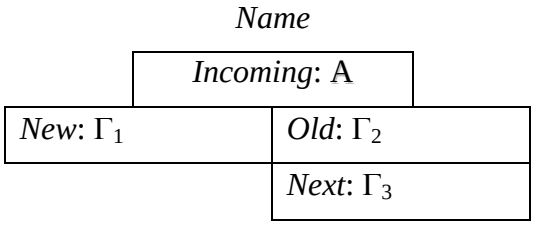


Nodes finals = estats de l'autòmat



Pas base

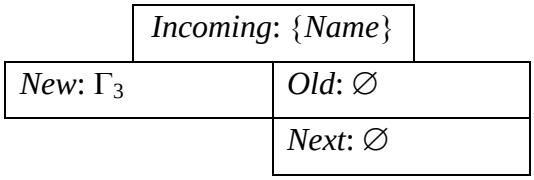
Configuració actual:



Si $\Gamma_1 = \varnothing$ llavors, totes les fórmules ja han estat processades.

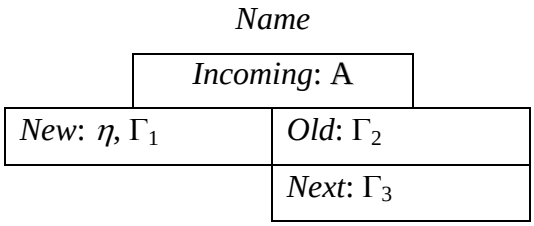
Existeix un node *Name'* amb idèntics *Old*, *Next*?

- Llavors, descartar *Name* i afegir *Name.Incoming* a *Name'.Incoming*.
- Altrament:
  - *Name* és un nou estat.
  - Crear un nou nom i node:



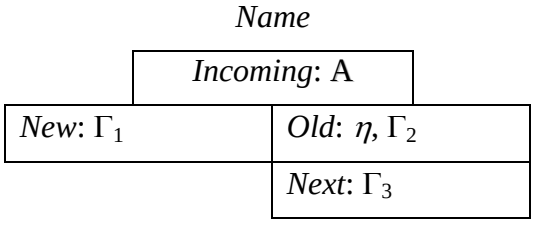
Cas: Literal

Configuració actual:



És $\neg\eta \in \Gamma_2$?

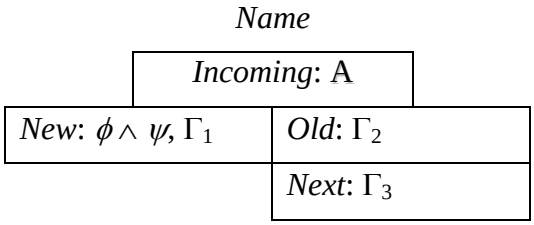
- Sí: es descarta el node.
- No: següent configuració:



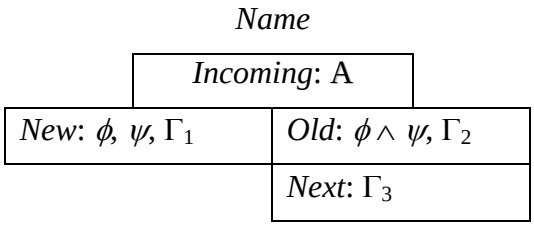
El cas per a $\neg\eta$ a *New* és similar.

Cas: Conjunció

Configuració actual:

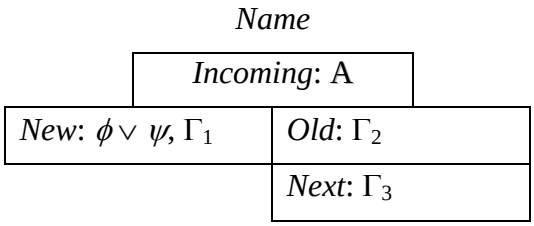


Següent configuració:

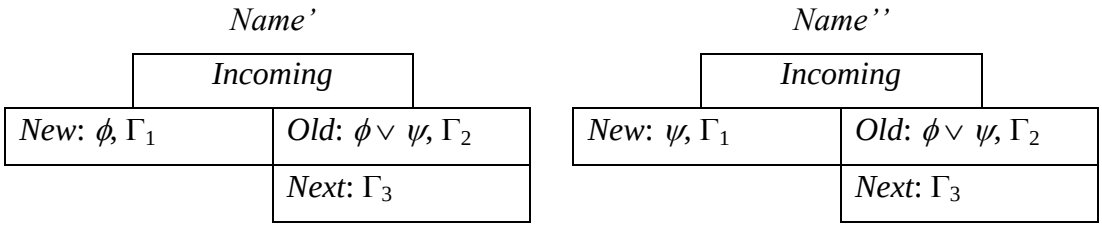


Cas: Disjunció

Configuració actual:

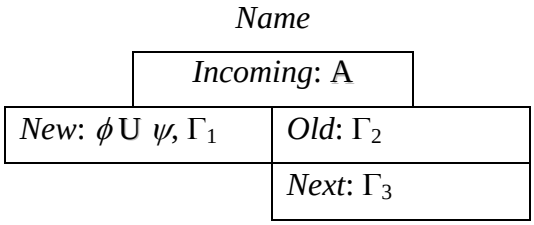


Configuració dividida en dues:



Cas: Until

Configuració actual:



Configuració dividida en dues:

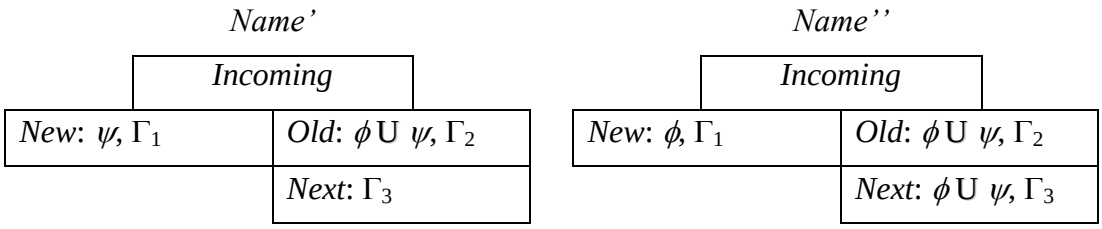
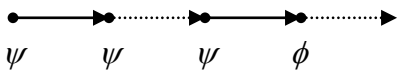
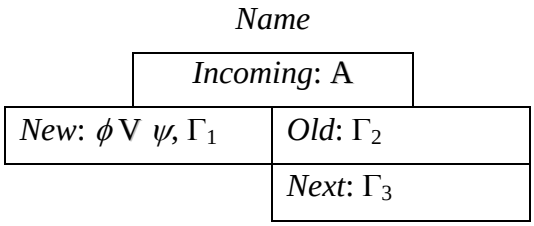


Diagrama temporal de la fórmula LTL ϕ U ψ:

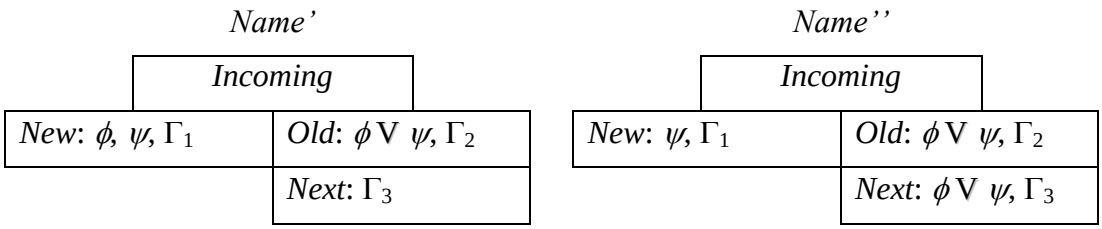


Cas: Release

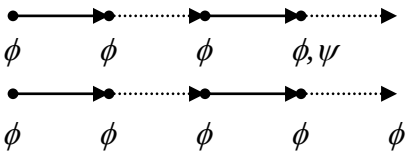
Configuració actual:



Configuració dividida en dues:

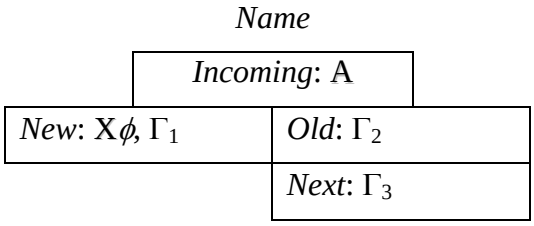


Diagrames temporal de la fórmula LTL ϕ R ψ:



Cas: Next

Configuració actual:



Següent configuració:

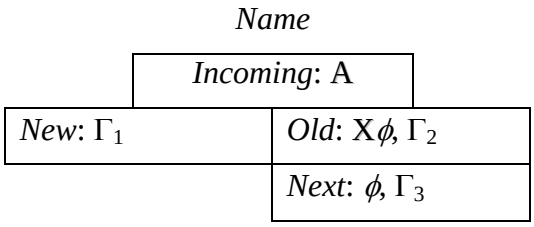
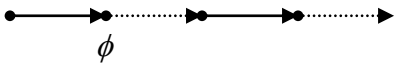


Diagrama temporal de la fórmula LTL Xϕ:



4.3 *Model-checking* de propietats LTL a una *Petri Net* usant *Constraint Programming*

L'algorisme aquí presentat així com la teoria associada al mateix, han estat extrets, principalment, de la referència [7]. Al document al qual es fa referència i, especialment, a la referència [7.1], es pot trobar una explicació encara més exhaustiva del tema.

El problema de *model-checking* per a PNs segures i, lògiques temporals lineals en el temps (LTL) consisteix en decidir, donada una PN segura i una fórmula LTL si, la PN satisfà la propietat codificada per la fórmula. En aquest sentit, s'introdueix un test de semidecisió per a aquest problema. Per *test de semidecisió* s'entén aquell que, pot respondre 'sí', cas en què la PN satisfà la propietat o bé, pot respondre 'no sé'. El test està basat en una variant de l'aproximació teòrica d'autòmats (*automata-theoretic approach*) al *model-checking* i, en la noció de *T-invariant*.

4.3.1 Introducció

LTL és un formalisme per a l'especificació de propietats de sistemes concurrents. El problema de decidir si un sistema concurrent satisfà una fórmula LTL s'anomena *problema de model-checking* (de LTL). L'aproximació teòrica d'autòmats assumeix que existeix un mapeig semàntic el qual, associa a un sistema concurrent *sys* un sistema de transicions finit (etiquetat) A_{sys}. Així, calen implementar les següents tres tasques:

- Construir un Büchi autòmat $A_{\neg\phi}$ per a la negació de la fórmula ϕ a *xequejar*. $A_{\neg\phi}$ accepta exactament totes les seqüències infinites que violen la fórmula ϕ.
- Construir un Büchi autòmat A_p, anomenat el producte de A_{sys} i $A_{\neg\phi}$. A_p accepta totes les infinites computacions de A_{sys} que són acceptades per $A_{\neg\phi}$, és a dir, totes les infinites computacions de A_{sys} que violen ϕ.
- *Xequejar* si l'autòmat producte A_p és buit, és a dir, si accepta seqüències no infinites. A_{sys} satisfà ϕ si i, només si, A_p és buit.

El problema principal d'aquesta aproximació és el fenomen de l'explosió d'estats: la mida del TS A_{sys} pot créixer exponencialment respecte a la mida de *sys*. Per tal d'evitar l'explosió d'estats, s'introdueix una tècnica la qual, pot ser emprada quan el sistema es modela com una PN segura. La tècnica és el *test de semidecisió* pel qual, un procediment pot respondre 'sí', cas en què la propietat a *xequejar* es compleix o bé, pot respondre 'no sé'.

Per a sistemes modelats com a PNs, el sistema de transicions A_{sys} és, precisament, el graf d'abastabilitat. Una aplicació senzilla de l'aproximació teòrica d'autòmats seria (1) construcció del graf d'abastabilitat i, (2) construcció de l'autòmat producte. La principal contribució que es fa és que, el pas (2) pot ser implementat abans que el pas (1). Més específicament, es descriuen diferents vies de construcció de la Büchi xarxa producte N_p a partir d'una PN N_{sys} i un Büchi autòmat $A_{\neg\phi}$. Amb aquesta construcció, es redueix el problema de *model-checking* a un cert problema de *xarxa buida* (*net emptyness*), molt similar al problema del Büchi autòmat buit. Així, el test es basa en la noció de *T-invariant*. A més, el test pot ser implementat en el marc de la *constraint programming*.

4.3.2 Conceptes bàsics

4.3.2.1 LTL en sistemes de transicions

Es vol usar LTL per tal de descriure propietats tant de semàntiques basades en l'acció com basades en l'estat d'un TS etiquetat T = (Act, Q, Δ, q_0).

Al cas de les semàntiques basades en l'acció, es pren $\Sigma = Act$. Π és, per tant, un conjunt de proposicions sobre el conjunt d'accions i, el llenguatge $L(\phi)$ d'una fórmula ϕ és el conjunt de recorreguts sobre accions. Es diu que T satisfà ϕ si $L_a(\mathrm{T}) \subseteq L(\phi)$, és a dir, si cada recorregut sobre accions de T satisfà ϕ.

Al cas de les semàntiques basades en l'estat, es pren $\Sigma = Q$ i, per tant, Π és un conjunt de proposicions sobre el conjunt d'estats. Anàlogament, es diu que T satisfà ϕ si $L_s(\mathrm{T}) \subseteq L(\phi)$.

4.3.2.2 Büchi autòmat

Sigui ϕ una fórmula LTL sobre un conjunt de proposicions Π.

Definició 4.2: (*Labelled Büchi Automaton*) Un Büchi autòmat etiquetat sobre Π és una 5-tupla A = (2^{Π}, Q, Δ, q_0, F), on:

(1) Q és un conjunt finit d'estats;

(2) $\Delta \subseteq Q$ x 2^{Π} x Q és la relació de transició;

(3) $q_0 \in Q$ és l'estat inicial;

(4) $F \subseteq Q$ és el conjunt d'estats acceptadors.

□

Una *execució acceptadora* de A és una seqüència infinita $\sigma = q_0\ \Pi_0\ q_1\ \Pi_1\ q_2\ \ldots$ tal que $(q_i,\ \Pi_i,\ q_{i+1}) \in \Delta$ per a cada $i \geq 0$ i, algun estat de F apareix amb freqüència infinita a σ. A *accepta* una paraula infinita $a_0\ a_1\ a_2\ \ldots \in \Sigma^{\omega}$ si existeix una execució acceptadora $q_0\ \Pi_0\ q_1\ \Pi_1\ q_2\ \ldots$ tal que a_i satisfà cada predicat de Π_i, per a cada $i \geq 0$.

Es defineix el llenguatge L(A) del Büchi autòmat etiquetat A com el conjunt de paraules infinites acceptades per A. Així, si ϕ és una fórmula LTL llavors, existeix un Büchi autòmat A tal que $L(\phi) = L$(A). I, es denota com A_ϕ un Büchi autòmat tal que, satisfà $L(\phi) = L(A_\phi)$.

També s'usen els Büchi autòmats no etiquetats els quals, són 4-tuples A = (Q, Δ, q_0, F) on, $\Delta \subseteq Q$ x Q. Es poden veure com a un cas particular de Büchi autòmats etiquetats en què, totes les transicions estan etiquetades pel conjunt buit de proposicions.

El problema del no buit (*nonemptyness*) per a Büchi autòmats no etiquetats consisteix en decidir si L(A) és no buit. El problema és NLOGSPACE-complet.

4.3.2.3 Autòmat producte

Sigui T_{sys} un TS etiquetat finit i, sigui ϕ una fórmula sobre les accions o els estats de T_{sys}. El procediment basat en autòmats per tal de *xequejar* si T_{sys} satisfà ϕ consisteix en les següents passes:

- Construir un Büchi autòmat etiquetat $A_{\neg\phi}$ el qual, accepta $L(\neg\phi)$.
- Construir un Büchi autòmat no etiquetat A_p, anomenat el producte de T_{sys} i $A_{\neg\phi}$ el qual, és buit si i, només si, $L(T_{sys}) \cap L(\neg\phi) = \varnothing$.
- *Xequejar* si $L(A_p)$ és no buit.

Clarament, $L(A_p)$ és buit si i, només si, $L(T_{sys}) \cap L(\neg\phi) = \varnothing$ si i, només si, $L(T_{sys}) \subseteq L(\phi)$ si i, només si, T_{sys} satisfà ϕ.

Semàntiques basades en l'estat

Sigui T_{sys} = (Q_{sys}, Δ_{sys}, q_{0sys}) un sistema de transicions no etiquetat i, sigui $A_{\neg\phi}$ = (2^{Π}, $Q_{\neg\phi}$, $\Delta_{\neg\phi}$, $q_{0\neg\phi}$, $F_{\neg\phi}$) el Büchi autòmat etiquetat corresponent a la negació de ϕ on, Π és el conjunt de proposicions sobre Q_{sys}. L'autòmat producte de T_{sys} i $A_{\neg\phi}$ és el Büchi autòmat no etiquetat A_p = (Q, Δ, q_0, F) donat per:

- $Q = Q_{sys}$ x $Q_{\neg\phi}$.
- Δ és el conjunt més petit tal que, si $(q_1, q_2) \in \Delta_{sys}$, $(r_1, \{\pi_1, \ldots, \pi_n\}, r_2) \in \Delta_{\neg\phi}$ i, q_1 satisfà π_i per a cada $1 \leq i \leq n$ llavors, $((q_1, r_1), (q_2, r_2)) \in \Delta$.
- $q_0 = (q_{0\,sys}, q_{0\neg\phi})$.
- $F = Q_{sys}$ x $F_{\neg\phi}$.

Les proposicions π_i són avaluades sobre l'estat q_1.

D'aquesta definició s'obté que, A_p és buit si i, només si, el conjunt $L_s(T_{sys}) \cap L(A_{\neg\phi})$ és també buit.

4.3.2.4 Notació multiconjunt

Un *multiconjunt* sobre un conjunt X és un mapeig μ: X a N. Les operacions unió, intersecció, suma i, diferència sobre multiconjunts es defineixen de manera equivalent a com es fa sobre conjunts. El conjunt de multiconjunts sobre X es denota per $\mathcal{M}(X)$.

Definició 4.3: (*Labelled Petri Net*) Una xarxa de Petri etiquetada és una 4-tupla N = (Act, P, T, M_0), on:

(1) Act és un conjunt d'accions;

(2) P és un conjunt finit de *places*;

(3) $T \subseteq \mathcal{M}(P)$ x Act x $\mathcal{M}(P)$ és un conjunt de transicions;

(4) $M_0 \in \mathcal{M}(P)$ és un marcatge.

□

Per a una transició t = (P, Q), anomenem P (respectivament Q) al *preconjunt* (respectivament *postconjunt*) i s'escriu $\cdot t$ (respectivament $t\cdot$). Els multiconjunts de *places* s'anomenen *marcatges* i, M_0 s'anomena *marcatge inicial* de N.

$M \xrightarrow{t} M'$ denota que la transició t ocorre al marcatge M assolint M'.Una seqüència finita o infinita $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} M_2 \cdots$ s'anomena *seqüència ocurrent*. $M \xrightarrow{a} M'$ per $a \in \Sigma$ denota que existeix una transició t = (P_1, a, P_2) tal que $M \xrightarrow{t} M'$.

Una *execució completa* d'una PN és una seqüència infinita $M_0\, a_0\, M_1\, a_1\, M_2\, a_2 \ldots$ tal que $M_i \xrightarrow{a_i} M'_i$ per a tot $i \geq 0$. També es denota una execució completa com $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} M_2 \cdots$. Notar que per a cada execució completa existeix una seqüència ocurrent associada.

Una seqüència infinita $a_0\ a_1\ a_2\ \ldots$ d'accions és una *action run* si existeix una execució completa $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} M_2 \cdots$. El llenguatge sobre les accions L_a(N) de N és el conjunt de tots els recorreguts sobre les accions. Una seqüència infinita $M_0\ M_1\ M_2\ \ldots$ de marcatges és un recorregut sobre els estats si existeix una execució completa $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} M_2 \cdots$. El llenguatge sobre els estats L_s(N) de N és el conjunt de tots els recorreguts sobre els estats.

Les PNs no etiquetades s'obtenen de PNs etiquetades, ometent les etiquetes de les transicions. Per tant, una PN no etiquetada és una 3-tupla (P, T, M_0) on, $T \subseteq M(P)$ x $M(P)$.

4.3.2.5 LTL en *Petri Nets* segures

A continuació definim quan una PN segura satisfà un fórmula LTL.

Al cas de semàntiques basades en l'estat, Π és un conjunt de proposicions sobre el conjunt d'estats. Com què, els estats d'una PN són els seus marcatges abastables, per a les PNs es pren Π com a un conjunt arbitrari de proposicions sobre el conjunt de marcatges. Per altra banda, es restringeixen a les proposicions π_p on, p és un *place* de la xarxa, amb la següent interpretació: un marcatge M satisfà π_p si i, només si, marca el *place* p. Una xarxa N satisfà una fórmula ϕ si L_s(N) $\subseteq L(\phi)$.

4.3.2.6 *Büchi Nets*

El producte d'un Büchi autòmat i una PN segura és una Büchi xarxa, la xarxa equivalent al Büchi autòmat producte no etiquetat definit a l'apartat 4.3.2.3 per a TSs.

Una *Büchi xarxa* és una tupla N = (P, T, M_0, F) on, (P, T, M_0) és una PN no etiquetada i F és un subconjunt de P. Una *execució acceptadora* de N és una *state run* $M_0\ M_1\ M_2\ \ldots$ tal que, algun *place* de F apareix a infinits marcatges M_i. N és no buida si té una execució acceptadora.

El problema del no buit (*nonemptyness*) per a una Büchi xarxa N = (P, T, M_0, F) és el problema de decidir si N és no buit. El problema és PSPACE-complet.

4.3.3 Test del buit en Büchi xarxes usant *T-invariants*

4.3.3.1 *Product Nets* sobre semàntiques basades en els estats

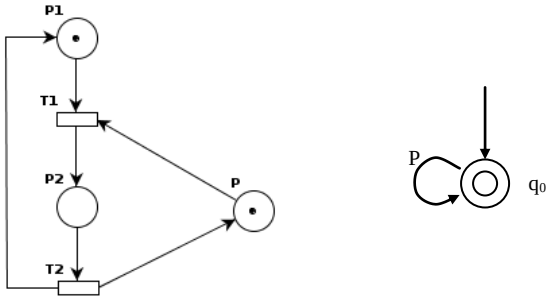
Sigui una PN segura no etiquetada N_{sys} = (P_{sys}, T_{sys}, M_{0sys}) fixada. Considerem que el conjunt Π de proposicions sobre els marcatges de N_{sys}, usats per a la construcció de les fórmules LTL, conté només predicats π_p els quals es compleixen si i, només si, el *place* p és marcat. Es pot, per tant, identificar clarament la proposició π_p i el *place* p. Amb aquesta identificació, el Büchi autòmat $A_{\neg\phi}$ de la negació de la fórmula ϕ té la forma $A_{\neg\phi}$ = ($2^{P_{sys}}$, $Q_{\neg\phi}$, $\Delta_{\neg\phi}$, $q_{0\neg\phi}$, $F_{\neg\phi}$).

L'objectiu és construir una Büchi xarxa producte, satisfent la següent propietat: la xarxa producte es pot moure des del marcatge (M_1, q_1) al marcatge (M_2, q_2) si i, només si:

(1) N_{sys} es pot moure des de M_1 a M_2;

(2) existeix $(q_1, R, q_2) \in \Delta_{\neg\phi}$;

(3) i, M_1 marca cada place de R.



(a) (b)

Figura 4.4: (a) PN N_{sys} i, (b) Büchi autòmat $\Delta_{\neg\phi}$.

Es mostren dues diferents construccions el resultat de les quals, es mostrarà seguint l'exemple de la figura 4.4.

La idea és la següent: si (P_1, P_2) és una transició de la PN i, (q_1, R, q_2) és una transició del Büchi autòmat llavors, afegim la següent transició al producte:

$$(P_1 + (R - P_1) + \{q_1\}, P_2 + (R - P_1) + \{q_2\}).$$

Amb aquesta solució se satisfan les tres restriccions anteriors. L'autòmat producte queda definit a continuació.

Definició 4.4: (*Product Büchi Net*) Una *Büchi xarxa producte* $N_p = (P, T, M_0, F)$ de N_{sys} i $A_{\neg\phi}$ ve donada per:

- $P = P_{sys} \cup Q_{\neg\phi}$;
- T és el conjunt més petit tal que satisfà: si $(P_1, P_2) \in T_{sys}$ i $(q_1, R, q_2) \in \Delta_{\neg\phi}$ llavors, $(P_1 + (R - P_1) + \{q_1\}, P_2 + (R - P_1) + \{q_2\}) \in T$;
- $M_0 = M_{0\,sys} + \{q_{0\neg\phi}\}$;
- $F = F_{\neg\phi}$. □

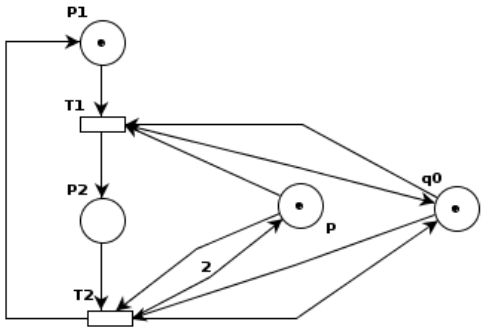


Figura 4.5: La xarxa producte N_p, dels N_{sys} i $\Delta_{\neg\phi}$ mostrats a la figura 4.4, segons la definició 4.4.

En la segona construcció, l'autòmat i la PN alternen els seus moviments: l'autòmat testeja si el marcatge M_1 marca cada place de R. Si aquest és el cas llavors, es mou de q_1 a q_2 i, transfereix els controls a la xarxa la qual, fa el seu moviment i, retorna el control a l'autòmat. L'alternança es pot implementar mitjançant dos *scheduling places* SC_1 i, SC_2. Un *token* a SC_1 (SC_2) significa que l'autòmat (xarxa) ha de moure en el següent pas.

Definició 4.5: (*Product Büchi Net*) Una *Büchi xarxa producte* $N_p = (P, T, M_0, F)$ de N_{sys} i $A_{\neg\phi}$ ve donada per:

- $P = P_{sys} \cup Q_{\neg\phi} \cup \{SC_1, SC_2\}$;
- T és el conjunt més petit tal que satisfà: si $(P_1, P_2) \in T_{sys}$ llavors, $(P_1 + \{SC_2\}, P_2 + \{SC_1\}) \in T$ i, si $(q_1, R, q_2) \in \Delta_{\neg\phi}$ llavors, $(\{q_1, SC_1\}+R, \{q_2, SC_2\}+R) \in T$;
- $M_0 = M_{0\,sys} + \{q_{0\neg\phi}, SC_1\}$;
- $F = F_{\neg\phi}$.

□

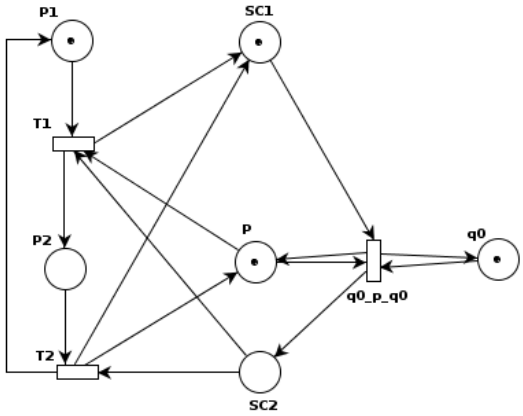


Figura 4.6: La xarxa producte N_p, dels N_{sys} i $\Delta_{\neg\phi}$ mostrats a la figura 4.4, segons la definició 4.5.

Aquesta segona construcció, contràriament a la primera, resta més petita: la seva mida és essencialment la suma de les mides de N_{sys} i $\Delta_{\neg\phi}$.

4.3.3.2 Test

A continuació, es desenvolupa un test de semidecisió per problema del buit (*emptyness*) a les Büchi xarxes el qual, evita la construcció del graf d'abastabilitat.

El fet que una variable x tingui valor v es modela posant un *token* al *place* x_v. Així, afirmacions com "la variable x pren el valor 1 infinitament" es formalitzen emprant semàntiques basades en l'estat.

El test es basa en la noció de *T-invariant*.

Definició 4.6: (*T-invariant*)

- Un *T-invariant* és un vector tal que, conté una seqüència de disparaments que dóna com a marcatge resultant el marcatge inicial.
- Un *T-invariant* d'una PN amb matriu d'incidència A és, un vector y d'elements racionals tal que $Ay = 0$.
- Un *T-invariant* d'una xarxa és un mapeig J el qual, assigna a cada transició t un número racional $J(t)$ i, satisfà la següent propietat per a cada *place p*:

$$\sum_{t \in \bullet p} J(t) = \sum_{t \in p \bullet} J(t).$$ □

Els *T-invariants* tenen la següent propietat fonamental. Siguin M i M' dos marcatges de la xarxa N i, sigui σ la seqüència de transicions tal que $M \xrightarrow{\sigma} M'$. Tenim que $M = M'$ si i, només si, el mapeig el qual associa a cada transició t el nombre de vegades que apareix a σ és un *T-invariant* de N.

Un *T-invariant* J d'una Büchi xarxa N és *realitzable* si, existeix un marcatge abastable M i, una seqüència de transicions σ no buida tal que $M \xrightarrow{\sigma} M$ i cada transició t ocorre exactament $J(t)$ vegades a σ. La seqüència $M \xrightarrow{\sigma} M$ s'anomena *realització* de J. Els *T-invariants* realitzables són sempre semipositius, és a dir, els seus components han de ser no negatius i, com a mínim un d'ells ha de ser diferent de 0. Un *T-invariant* J és *final* si $J(t) > 0$ per alguna transició t en el postconjunt d'un *place* final de N.

Tenim que, una Büchi xarxa és no buida si i, només si, té un *T-invariant* final realitzable.

Com a conseqüència immediata d'aquesta proposició, si una Büchi xarxa no té *T-invariants* semipositius finals, realitzables o no llavors, és buida. Aquesta condició suficient del buit porta a un test de semidecisió simple ja que, l'absència de *T-invariants* semipositius es pot *xequejar* solucionant un sistema d'(in)equacions lineals de la forma:

$$\begin{aligned} & N \cdot X = 0 \\ & X \geq 0 \\ & \textstyle\sum_{t \in F \bullet} X(t) > 1 \end{aligned} \qquad (1)$$

on, N és la matriu d'incidència de la Büchi xarxa i, F és el conjunt de *places* finals.

5 Enginyeria del *software*

L'enginyeria del *software* és la disciplina o àrea de la informàtica que ofereix mètodes i tècniques per al desenvolupament i manteniment *software* de qualitat (Zelkovitz, 1978).

Les etapes del desenvolupament de *software* són les següents:

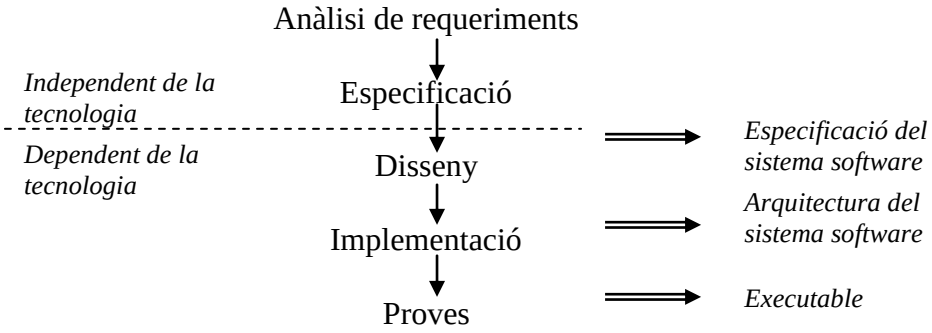


Figura 5.1: Etapes del desenvolupament *software*.

El model de desenvolupament inicialment emprat per a la realització de *software* ha estat el *model en cascada* o clàssic (model tradicional):

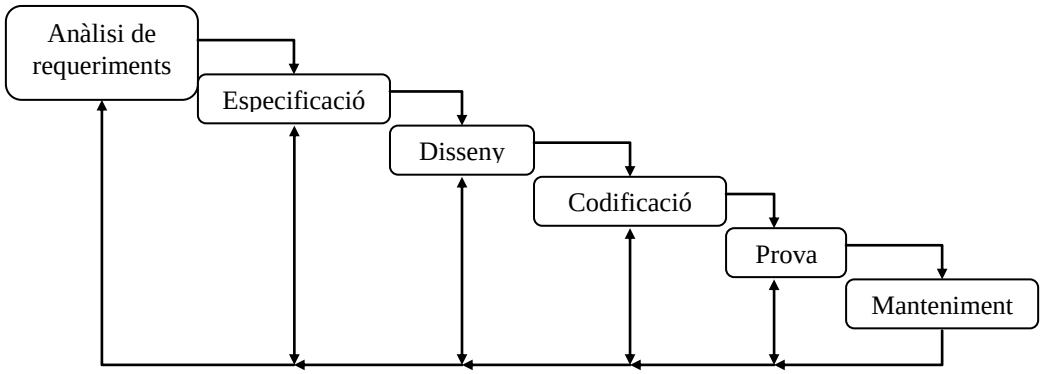


Figura 5.2: Model de cascada.

Com a conseqüència de l'evolució de les necessitats al llarg del desenvolupament del *software*, el model emprat ha acabat sent un *model de prototipatge*:

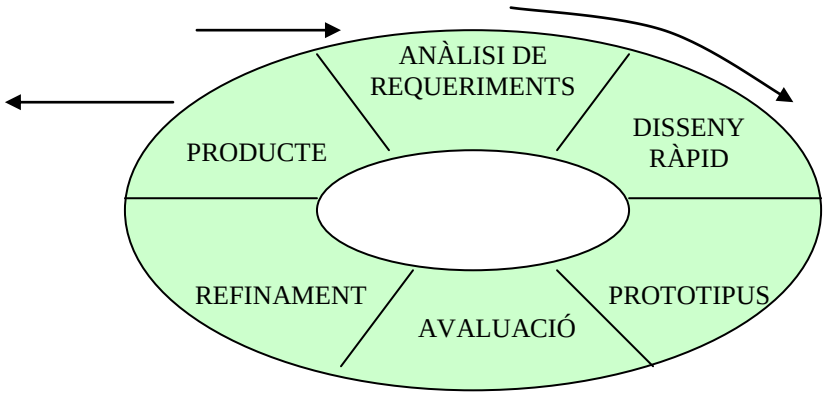


Figura 5.3: Model de prototipatge.

5.1 Especificació

L'especificació de requeriments descriu el comportament esperat del *software* una vegada desenvolupat. És important la identificació de les necessitats del negoci, així com la interacció amb els usuaris funcionals per a la recol·lecció, classificació, identificació, priorització i especificació dels requeriments del *software*.

Per tant, en aquest apartat es defineixen quins són els requeriments que haurà de satisfer l'aplicació desenvolupada.

5.1.1 Anàlisi de requeriments

L'anàlisi de requeriments engloba aquelles tasques que determinen les necessitats i/o condicions que l'usuari funcional requereix que el sistema compleixi amb la finalitat d'assolir els seus objectius. És a dir, es tracta d'una especificació a alt nivell de la descripció i abast del projecte de *software*, així com les funcionalitats que ha de proporcionar el mateix.

A continuació s'especifiquen els següents tipus de requeriments:

- *Requeriments funcionals*: descriuen què fa el sistema, quines són les seves entrades i sortides i les relacions entre ambdues i, les dades i processos.
- *Requeriments no funcionals*: descriuen les qualitats generals que ha de complir el sistema per tal de poder realitzar, amb certes garanties de qualitat, les seves funcions.

5.1.1.1 Requeriments funcionals

A continuació es llisten les activitats que ha de realitzar el sistema:

- *Entrades del sistema*:
    - *Fitxer XSD*: és un fitxer *XML Schema* el qual, especifica l'esquema en format PNML (*Petri Net Markup Language*), que s'accepta per al reconeixement dels fitxers XML que continguin les xarxes de Petri a carregar al sistema.
    - *Fitxer PNML*: és un fitxer que segueix l'esquema especificat pel fitxer anterior el qual, conté la xarxa de Petri a carregar al sistema.
    - *Fitxer PC*: és un fitxer que conté les *P-components* de la xarxa de Petri carregada amb el fitxer anterior.
    - *Fitxer LTL*: és un fitxer que conté la fórmula LTL a verificar si compleix la xarxa de Petri carregada.
- *Sortides del sistema*:
    - *Fitxer PNML*: és un fitxer que segueix l'esquema especificat pel fitxer d'entrada *XSD* el qual, conté la Büchi xarxa producte construïda mitjançant la xarxa de Petri carregada i l'autòmat $A_{\neg\phi}$, construït a partir de la fórmula LTL carregada, segons la definició seguida per a tal construcció.
    - *Matriu d'incidència*: s'ha de visualitzar la matriu d'incidència corresponent a la Büchi xarxa producte anterior.

    - *Resolució de la verificació de la fórmula LTL*: s'ha de visualitzar la resolució de l'aplicació del test per a la Büchi xarxa producte anterior.

- *PetriNet*: mòdul de gestió de les xarxes de Petri el qual, ha de complir:
    - Ha de poder validar el fitxer d'entrada *PNML* a partir de la definició especificada al fitxer d'entrada *XSD*.
    - Ha de poder crear una estructura de dades a partir d'aquest fitxer d'entrada *PNML* la qual, contingui les subestructures que formen una xarxa de Petri (*places*, transicions i arcs).
    - Per a cadascuna d'aquestes subestructures, ha de permetre la consulta de la seva informació.
- *LTLAutomaton*: mòdul de gestió de les fórmules LTL el qual, ha de complir:
    - Ha de poder validar el fitxer d'entrada *LTL* a partir de la gramàtica especificada per a la definició de fórmules LTL.
    - Ha de poder transformar la fórmula LTL carregada amb el fitxer d'entrada *LTL* a forma normal negada.
    - Ha de poder crear un autòmat corresponent a la fórmula LTL carregada, una vegada ja transformada a forma normal negada, tot creant les estructura de dades necessàries per a la gestió d'autòmats.
- *BüchiNet*: mòdul de gestió de les Büchi xarxes el qual, ha de complir:
    - Ha de crear la Büchi xarxa producte, segons la definició especificada, de la xarxa de Petri i l'autòmat creats als mòduls anteriors.
    - Ha de generar el fitxer de sortida *PNML* corresponent a la Büchi xarxa producte, tot seguint l'especificació continguda a l'esquema del fitxer d'entrada *XSD*.
- *Gecode*: mòdul de gestió de la *constraint programming* el qual, ha de complir:
    - Ha de poder realitzar el test de semidecisió pel qual, s'indicarà si la xarxa de Petri carregada compleix o no la fórmula LTL carregada, usant *constraint programming*.
    - Ha de visualitzar la matriu d'incidència de la Büchi xarxa producte.
    - Ha de visualitzar la resolució del test de semidecisió.

5.1.1.2 Requeriments no funcionals

Els requeriments no funcionals considerats per tal de poder complir el major nombre de factors que faran que el nostre sistema de *software* sigui de qualitat són els següents: usabilitat, mantenibilitat, portabilitat, modularitat, fiabilitat i eficiència.

5.1.2 Anàlisi funcional

5.1.2.1 Model de casos d'ús

El model de casos d'ús del sistema ens proporciona les següents informacions:

- les funcionalitats que el nostre sistema *software* proporciona a l'exterior;
- i, les entitats externes (actors) que interactuen amb el nostre sistema.

Actors

L'aplicació *software* a desenvolupar és mono usuari, és a dir, el sistema només interactua amb un usuari. Anomenem aquest actor *User*.

Existeixen però, a part del *Sistema*, tres actors més els quals, fan referència als programes que interaccionen amb la nostre aplicació els quals, són: *Xerces*, *ANTLR* i *Gecode*.

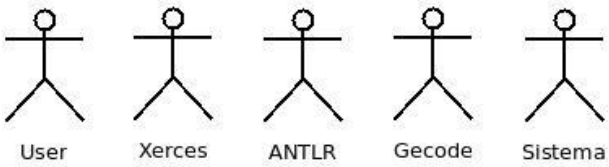


Figura 5.4: Actors del sistema.

Diagrames de casos d'ús

Els diagrames de casos d'ús permeten veure les funcionalitats que el nostre sistema *software* ofereix a l'exterior, és a dir, què permet fer la nostra aplicació. També mostren quins són els actors que inicien i interactuen amb aquestes funcionalitats.

Per tal de facilitar-ne la comprensió, s'ha separat el diagrama de casos d'ús en subdiagrames segons els diferents mòduls funcionals.

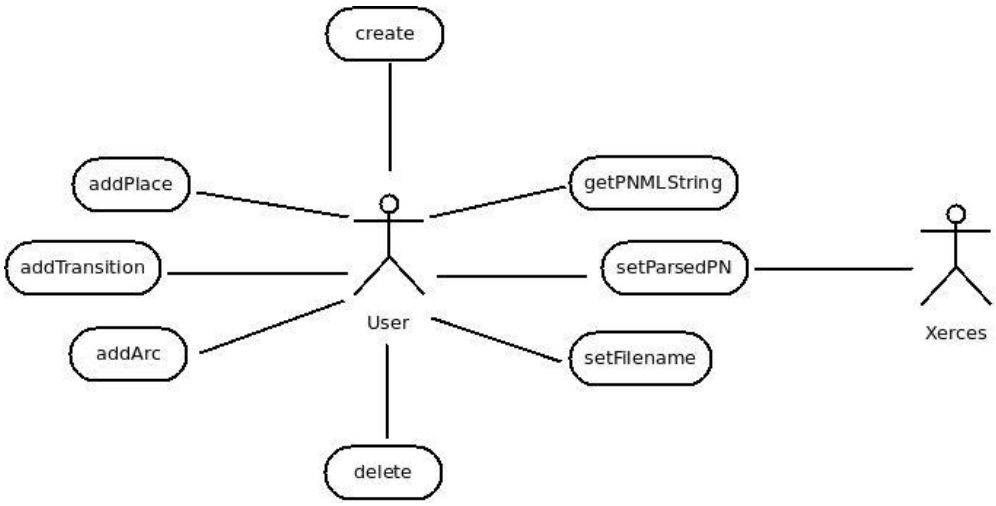


Figura 5.5: Subdiagrama dels casos d'ús del mòdul de *PetriNet*.

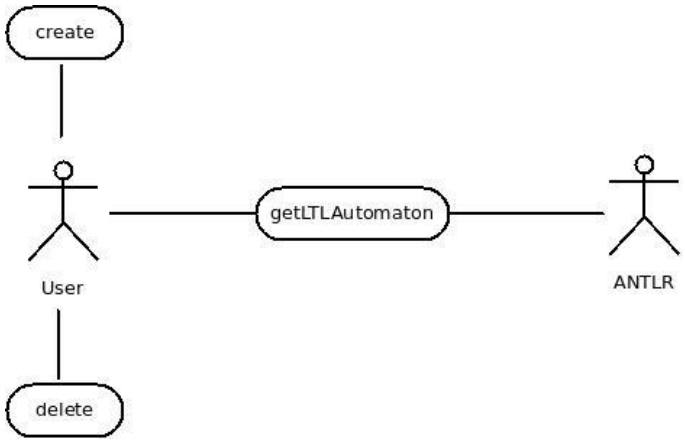


Figura 5.6: Subdiagrama dels casos d'ús del mòdul de *LTLAutomaton*.

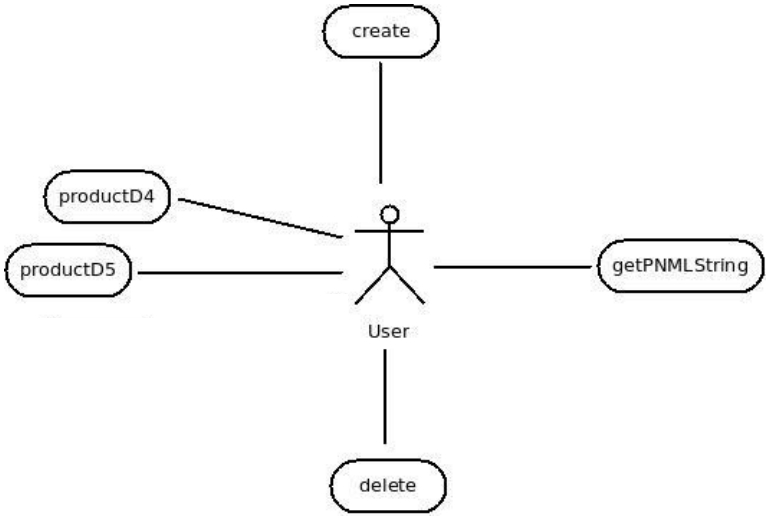


Figura 5.7: Subdiagrama dels casos d'ús del mòdul de *BuchiNet*.

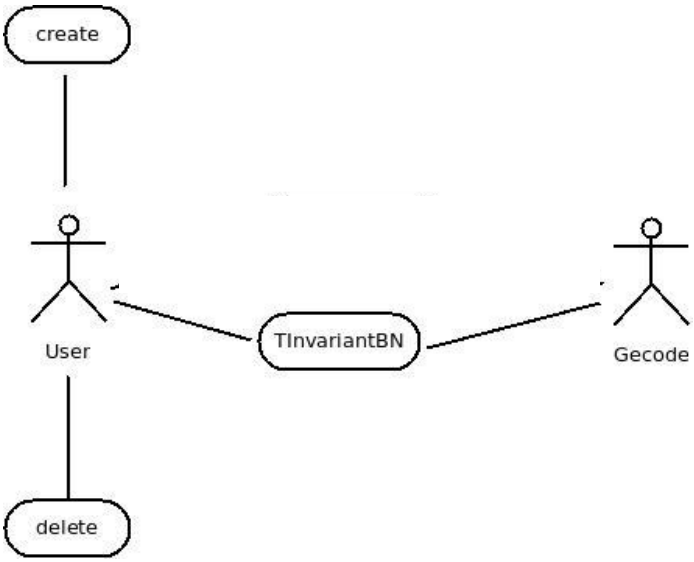


Figura 5.8: Subdiagrama dels casos d'ús del mòdul de *Gecode*.

Especificació dels casos d'ús

A continuació, es mostra l'especificació formal dels casos d'ús presentats a l'apartat anterior.

- **Casos d'ús del mòdul de PetriNet**

Cas d'ús: create

Context: El *User* vol crear una nova PN.

Actors: *User*, *Sistema*.

Precondicions: ∅

Postcondicions:

- El *Sistema* ha creat l'objecte *PetriNet*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* el nom del fitxer que conté/contindrà la PN.
2. El *Sistema* crea un objecte *PetriNet* amb fitxer de referència l'indicat.

Cas d'ús: addPlace

Context: El *User* vol afegir un *place* a una PN existent.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *PetriNet*.

Postcondicions:

- El *Sistema* ha afegit el *place* a l'objecte *PetriNet*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* els paràmetres del *place* a afegir a la PN.
2. El *Sistema* crea un objecte *Place* amb els paràmetres indicats.
3. El *Sistema* afegeix a l'objecte *PetriNet* l'objecte *Place* creat.

Cas d'ús: addTransition

Context: El *User* vol afegir una transició a una PN existent.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *PetriNet*.

Postcondicions:

- El *Sistema* ha afegit la transició a l'objecte *PetriNet*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* els paràmetres de la transició a afegir a la PN.
2. El *Sistema* crea un objecte *Transition* amb els paràmetres indicats.
3. El *Sistema* afegeix a l'objecte *PetriNet* l'objecte *Transition* creat.

Cas d'ús: addArc

Context: El *User* vol afegir un arc a una PN existent.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *PetriNet*.

Postcondicions:

- El *Sistema* ha afegit l'arc a l'objecte *PetriNet*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* els paràmetres de l'arc a afegir a la PN.
2. El *Sistema* crea un objecte *Arc* amb els paràmetres indicats.
3. El *Sistema* afegeix a l'objecte *PetriNet* l'objecte *Arc* creat.

Cas d'ús: getPNMLString

Context: El *User* vol generar un *string* el qual, contingui la PN en el format PNML especificat al fitxer XSD.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *PetriNet*.

Postcondicions:

- El *Sistema* retorna l'objecte *PetriNet* en un *string* en el format PNML especificat al fitxer XSD.

Escenari d'èxit principal:

1. El *User* sol·licita al *Sistema* la creació del *string* amb la PN en el format PNML especificat al fitxer XSD.
2. El *Sistema* retorna el *string* que descriu l'objecte *PetriNet* segons el fitxer XSD.

Cas d'ús: setParsedPN

Context: El *User* vol carregar una *Petri Net* a partir d'un fitxer que segueix la descripció especificada al fitxer XSD.

Actors: *User*, *Sistema*, *Xerces*.

Precondicions:

- Existeix l'objecte *PetriNet*.
- Existeix el fitxer amb la PN descrita segons l'especificació del fitxer XSD.

Postcondicions:

- El *Sistema* ha carregat l'objecte *PetriNet*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* l'objecte que conté la PN.
2. El *Sistema* es comunica amb *Xerces* per tal d'interpretar les dades contingudes a l'objecte indicat.
3. A mida que s'interpreten els objectes (*Place*, *Transition* i *Arc*) aquests, s'afegeixen a l'objecte *PetriNet*.

Escenaris alternatius:

- Si el fitxer indicat no segueix l'especificació descrita al fitxer XSD, el *Sistema* retorna un error i no modifica el seu estat (l'objecte *PetriNet*).

Cas d'ús: setFilename

Context: El *User* vol canviar el fitxer que referencia a l'objecte *Petri Net*.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *PetriNet*.

Postcondicions:

- El *Sistema* ha canviat la referència a l'objecte *PetriNet*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* el paràmetre del fitxer que referencia a la PN.
2. El *Sistema* canvia el fitxer que referencia a l'objecte *PetriNet*.

Cas d'ús: delete

Context: El *User* vol eliminar un objecte *PetriNet*.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *PetriNet*.

Postcondicions:

- El *Sistema* ha eliminat l'objecte *PetriNet*.

Escenari d'èxit principal:

1. El *User/Sistema* elimina l'objecte *PetriNet* existent.

- **Casos d'ús del mòdul de LTLAutomaton**

Cas d'ús: create

Context: El *User* vol crear un objecte *LTLAutomaton* el qual, gestionarà la construcció de l'autòmat corresponent a la negació de la fórmula LTL indicada al fitxer LTL, en forma negada normal.

Actors: *User*, *Sistema*.

Precondicions: ∅

Postcondicions:

- El *Sistema* ha creat l'objecte *LTLAutomaton*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* el nom del fitxer que la fórmula LTL que es vol verificar.
2. El *Sistema* crea un objecte *LTLAutomaton* amb fitxer de referència l'indicat.

Cas d'ús: getLTLAutomaton

Context: El *User* vol generar els nodes de l'autòmat corresponent a la negació de la fórmula LTL indicada al fitxer LTL, en forma normal negada.

Actors: *User*, *Sistema*, *ANTLR*.

Precondicions:

- Existeix l'objecte *LTLAutomaton*.
- Existeix el fitxer amb la fórmula LTL descrita segons l'especificació de la gramàtica ANTLR especificada.

Postcondicions:

- El *Sistema* retorna la llista de nodes corresponents a l'autòmat que representa la negació de la fórmula LTL indicada, en forma normal negada.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* l'objecte que conté la fórmula LTL.
2. El *Sistema* es comunica amb *ANTLR* per tal d'interpretar les dades contingudes a l'objecte indicat i generar l'arbre gramatical de la fórmula LTL indicada.
3. El *Sistema* nega la fórmula LTL llegida i empeny les negacions, deixant la fórmula LTL en forma negada normal, tot guardant-la en un fitxer de sortida.
4. El *Sistema* tradueix la fórmula LTL negada, en forma negada normal, en un autòmat, segons l'algorisme 4.1.
5. El *Sistema* retorna els nodes corresponents a aquest autòmat generat.

Escenaris alternatius:

- Si el fitxer indicat no segueix l'especificació descrita a la gramàtica ANTLR especificada, el *Sistema* retorna un error i finalitza l'execució de l'aplicació.

Cas d'ús: delete

Context: El *User* vol eliminar un objecte *LTLAutomaton*.

Actors: *User, Sistema*.

Precondicions:

- Existeix l'objecte *LTLAutomaton*.

Postcondicions:

- El *Sistema* ha eliminat l'objecte *LTLAutomaton*.

Escenari d'èxit principal:

1. El *User/Sistema* elimina l'objecte *LTLAutomaton* existent.

- **Casos d'ús del mòdul de BuchiNet**

Cas d'ús: create

Context: El *User* vol crear un objecte *BuchiNet* per tal de comprovar si una propietat LTL es compleix per a la PN carregada.

Actors: *User, Sistema*.

Precondicions:

- Existeix l'objecte *PetriNet*.
- Existeix l'objecte *Automaton*.

Postcondicions:

- El *Sistema* ha creat l'objecte *BuchiNet*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* el nom del fitxer a on guardar la xarxa de Büchi resultant en el format especificat al fitxer XSD, l'objecte *Automaton* que representa la fórmula LTL negada i l'objecte *PetriNet*.
2. El *Sistema* crea un objecte *BuchiNet* amb fitxer de referència l'indicat.

Cas d'ús: productD4

Context: El *User* vol construir una *Büchi Net* segons la definició 4.4, corresponent al producte de la PN i de l'autòmat corresponent a la negació de la fórmula LTL indicada al fitxer LTL, en forma normal negada.

Actors: *User, Sistema*.

Precondicions:

- Existeix l'objecte *BuchiNet*.

Postcondicions:

- El *Sistema* ha construït la Büchi xarxa producte segons la definició 4.4.

Escenari d'èxit principal:

1. El *Sistema* crea la Büchi xarxa producte segons l'algorisme de la definició 4.4.

Cas d'ús: productD5

Context: El *User* vol construir una xarxa de Büchi segons la definició 4.5, corresponent al producte de la PN i de l'autòmat corresponent a la negació de la fórmula LTL indicada al fitxer LTL, en forma normal negada.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *BuchiNet*.

Postcondicions:

- El *Sistema* ha construït la Büchi xarxa producte segons la definició 4.5.

Escenari d'èxit principal:

1. El *Sistema* crea la Büchi xarxa producte segons l'algorisme de la definició 4.5.

Cas d'ús: getPNMLString

Context: El *User* vol generar un *string* el qual, contingui la xarxa de Büchi en el format PNML especificat al fitxer XSD.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *BuchiNet*.

Postcondicions:

- El *Sistema* retorna l'objecte *BuchiNet* en un *string* en el format PNML especificat al fitxer XSD.

Escenari d'èxit principal:

1. El *User* sol·licita al *Sistema* la creació del *string* amb la xarxa de Büchi en el format PNML especificat al fitxer XSD.
2. El *Sistema* retorna el *string* que descriu l'objecte *BuchiNet* segons el fitxer XSD.

Cas d'ús: delete

Context: El *User* vol eliminar un objecte *BuchiNet*.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *BuchiNet*.

Postcondicions:

- El *Sistema* ha eliminat l'objecte *BuchiNet*.

Escenari d'èxit principal:

1. El *User/Sistema* elimina l'objecte *BuchiNet* existent.

- **Casos d'ús del mòdul de Gecode**

Cas d'ús: create

Context: El *User* vol crear un objecte *Gecode* el qual, gestionarà la *constraint programming* per a la verificació de la propietat LTL a la PN.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *BuchiNet*.

Postcondicions:

- El *Sistema* ha creat l'objecte *Gecode*.

Escenari d'èxit principal:

1. El *User* indica al *Sistema* l'objecte *BuchiNet* a verificar, juntament amb el vector de *places* finals i el vector de *P-components*.
2. El *Sistema* crea un objecte *Gecode*.

Cas d'ús: TInvariantBN

Context: El *User* vol resoldre la verificació de la propietat LTL a la PN.

Actors: *User*, *Sistema*, *Gecode*.

Precondicions:

- Existeix l'objecte *Gecode*.

Postcondicions:

- El *Sistema* ha resolt mitjançant *constraint programming* la verificació de la propietat LTL a la PN.
- El *Sistema* visualitza l'arbre de resolució mitjançant BAB.

Escenari d'èxit principal:

1. El *Sistema* es comunica amb *Gecode* per tal de resoldre mitjançant *constraint programming*, el test presentat a l'apartat 4.3.3.
2. *Gecode* resol el sistema mitjançant BAB, tot visualitzant l'arbre de resolució del sistema el qual, permet veure les solucions parcials, així com desenvolupar branques no explorades de l'arbre.

Cas d'ús: delete

Context: El *User* vol eliminar un objecte *Gecode*.

Actors: *User*, *Sistema*.

Precondicions:

- Existeix l'objecte *Gecode*.

Postcondicions:

- El *Sistema* ha eliminat l'objecte *Gecode*.

Escenari d'èxit principal:

1. El *User/Sistema* elimina l'objecte *Gecode* existent.

5.2 Disseny

L'arquitectura consisteix en afegir valor als processos de negoci tenint en compte les solucions tecnològiques. L'arquitectura de sistemes en general, és una activitat de planificació, tant a nivell d'infraestructura de xarxa i *hardware*, com de *software*. Consisteix en el disseny dels components de l'aplicació (entitats de negoci), mitjançant l'ús de patrons d'arquitectura. El disseny arquitectònic permet visualitzar la interacció entre les entitats del negoci i, a més, pot ser validat, per exemple, mitjançant diagrames de seqüència. El disseny arquitectònic descriu en general com es construirà l'aplicació *software*.

En aquest sentit, la part de disseny de *software* consisteix en establir una estratègia de solució per a les entitats de negoci, a partir dels requeriments especificats a l'apartat anterior (5.1). En aquesta solució, s'ha de definir el sistema amb el detall suficient com per a poder-lo desenvolupar.

5.2.1 Arquitectura del sistema

L'arquitectura emprada és l'*arquitectura per capes* per tal de separar la lògica del sistema de la lògica del disseny. En particular, s'ha emprat una arquitectura de tres capes, amb capa de presentació, capa de domini i capa de dades.

- **Capa de Presentació**

  S'encarrega de la comunicació amb l'usuari. És la part de l'aplicació que conté la interfície gràfica a més de la lògica que permet la comunicació entre l'usuari i el sistema. Per tant, la seva principal funció és la recepció de peticions per part de l'usuari per tal de reportar-li els resultats obtinguts del programa d'una manera intel·ligible.

- **Capa de Domini**

  S'encarrega de l'execució de les accions sol·licitades, tot canviant l'estat del domini. Aquesta capa té com a funcionalitat principal la coordinació de l'aplicació. S'encarrega de la presa de decisions i de la realització de càlculs a partir de les peticions de la capa de presentació així com, així com del processament d'aquesta informació i la seva comunicació amb les altres dues capes contigües. A més, s'encarrega de la comunicació amb la resta de llibreries externes a l'aplicació, tot realitzant un control d'errors de manera que es puguin detectar a les seves entrades i ser degudament tractats per tal de poder informar l'usuari.

- **Capa de Dades**

  S'encarrega de gestionar la persistència de les dades. Aquesta capa és la que gestiona les dades emprades per l'aplicació, tant d'entrada com de sortida, així com el lloc on es troben.

Aquesta divisió per capes, juntament amb l'aplicació de patrons de disseny adequats els quals, limiten la comunicació entre capes, mantindran un alt nivell d'abstracció i una baixa cohesió entre els diferents elements del sistema. D'aquesta manera, es garanteixen la mantenibilitat, canviabilitat, reusabilitat i, portabilitat del sistema.

L'aplicació desenvolupada no requereix de Sistema Gestor de Base de Dades (SGBD) ja què, les dades permanents les quals, corresponen a xarxes de Petri, són guardades en fitxers emmagatzemats a memòria. És la capa de dades l'encarregada de gestionar la persistència de les dades.

Figura 5.9: Arquitectura de capes del sistema.

5.2.2 Diagrames de classe

Els diagrames de classe presentats es corresponen amb els casos d'ús especificats a l'apartat anterior (5.1), tot seguint l'arquitectura de tres capes emprada, presentada a l'apartat anterior. En aquest cas, s'especifiquen tots els casos d'ús per classes, amb un nivell de detall exhaustiu, així com les cardinalitats i relacions entre les diferents classes i capes i, les restriccions textuals als casos que procedeix.

Primerament, mostrem el diagrama de classes general de la nostra aplicació, sense atributs ni accions per tal de fer-lo més entenedor:

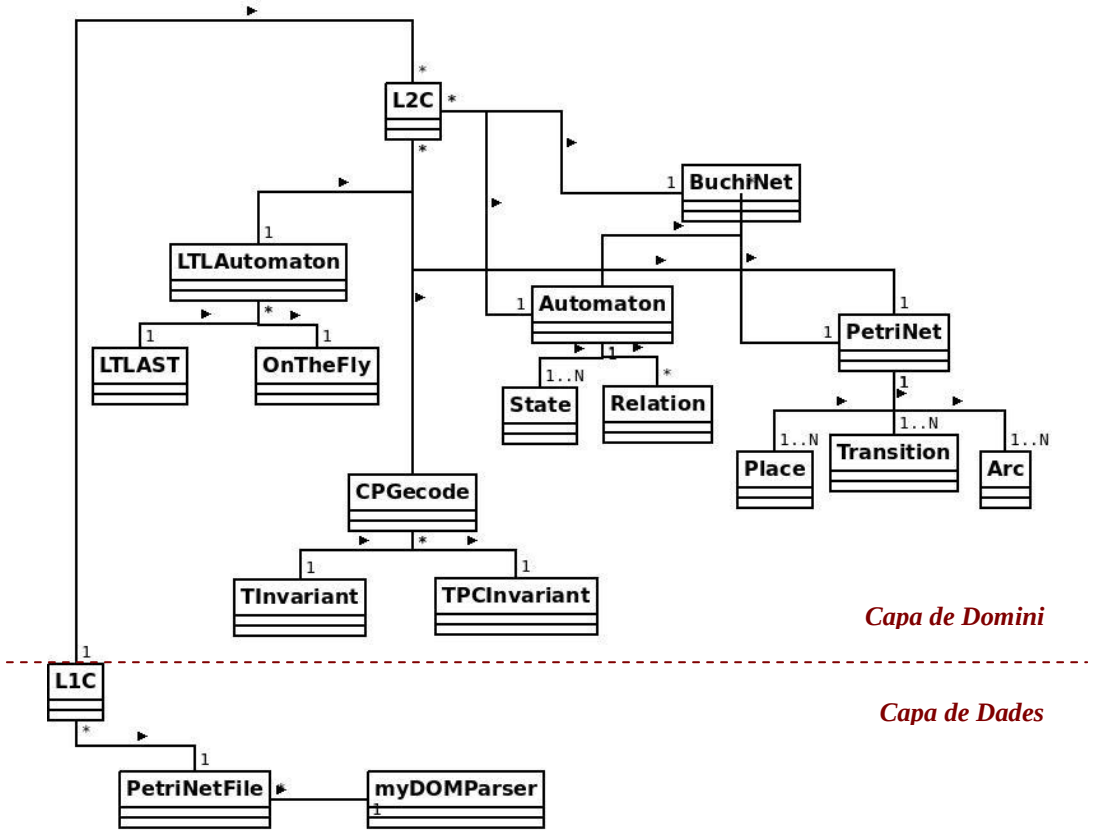


Figura 5.10: Diagrama de classes general esquemàtic.[3]

[3] No s'inclouen les classes corresponents a les llibreries externes.

A continuació es mostra, el mateix diagrama de classes general amb totes les classes del nostre sistema separades per capes i, amb tots els atributs i accions:

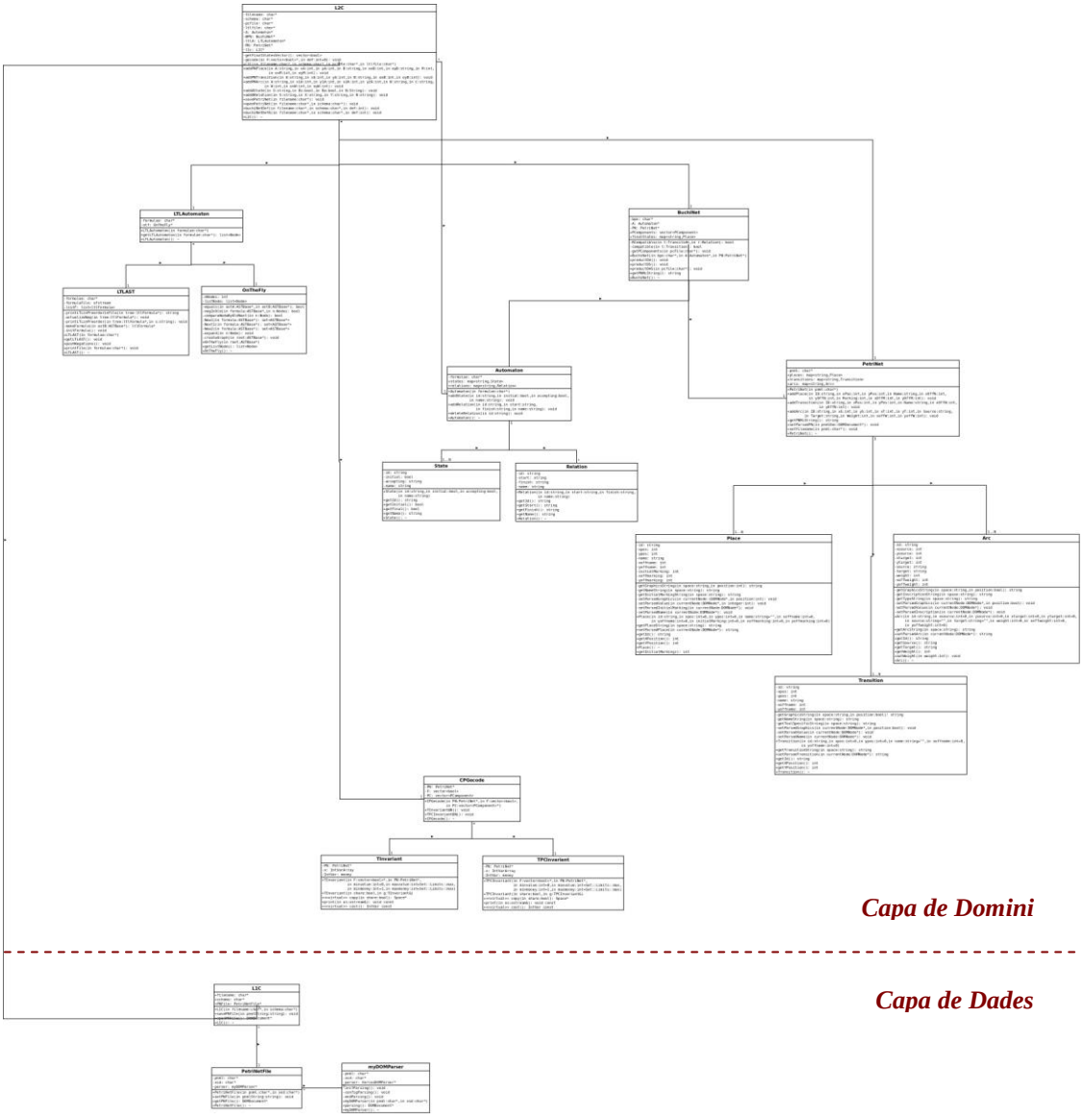


Figura 5.11: Diagrama de classes general.[4]

Com es pot veure, es tracta d'un disseny orientat a objectes a on, cada classe d'objectes té les seves pròpies operacions de manipulació d'informació. Els objectes interactuen per tal de satisfer les operacions del sistema.

[4] No s'inclouen les classes corresponents a les llibreries externes.

5.2.3 Patrons de disseny

En un sistema orientat a objectes a on aquests, intercanvien informació, responen a crides d'altres objectes i, canvien el seu estat, cal dissenyar el sistema de manera que els objectes tinguin cadascun d'ells, les seves responsabilitats clarament assignades.

Per tant, tal i com hem indicat abans, l'ús d'una arquitectura per capes, juntament amb l'aplicació de patrons de disseny, mantindrà un alt nivell d'abstracció i una baixa cohesió entre els diferents elements del sistema.

Un patró de disseny es defineix com un esquema de com solucionar un problema de disseny el qual, pot ser emprat en múltiples situacions o bé, pot ajudar a garantir i/o mantenir els nivells d'abstracció i cohesió desitjats.

En aquest sentit, es mostren els diferents patrons de disseny emprats a la nostra aplicació *software*.

5.2.3.1 Patró de disseny Controlador Façana

El patró de disseny Controlador permet l'assignació de la responsabilitat de rebre un esdeveniment del sistema a un únic objecte. Així, es té que els clients del sistema desconeixen l'estructura interna del sistema. A més, és l'objecte Controlador qui delega sobre un o més objectes del sistema el tractament de l'esdeveniment. I, de la mateixa manera que passa amb els clients, passa amb els objectes que tracten l'esdeveniment, que desconeixen l'existència d'un controlador ni el seu tipus.

A la nostra aplicació, s'ha dissenyat un sistema amb patró controlador de tipus façana, és a dir, amb objectes que representen tot el sistema. L'aplicació d'aquest patró es pot veure a cadascuna de les capes del diagrama de classes de la figura 5.11.

A continuació mostrem el detall de la capa de dades de l'aplicació, on es pot veure l'objecte *L1C*, controlador façana de la capa de dades:

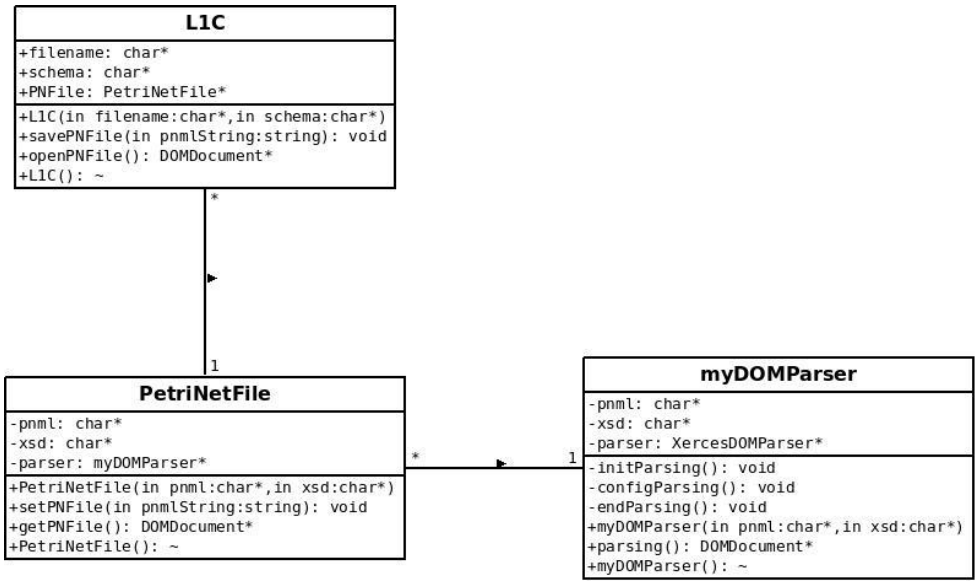


Figura 5.12: Controlador façana per a la Capa de Dades.

5.2.3.2 Patrons de disseny Iterador i Diccionari

El patró de disseny Iterador resol la necessitat de fer recorreguts seqüencials sobre un agregat. D'aquesta manera, s'allibera l'agregat de la responsabilitat del recorregut i s'assigna aquesta responsabilitat a l'objecte Iterador. El patró Iterador proporciona operacions per tal de recórrer els objectes de l'agregat. En quant a les responsabilitats, la classe que conté l'agregat és la responsable de la creació i destrucció de l'objecte Iterador.

El patró de disseny Diccionari resol la necessitat d'accedir a un agregat mitjançant una clau. D'aquesta manera, es poden inserir i esborrar objectes a l'agregat i, es poden recórrer tots els objectes de l'agregat (mitjançant el patró Iterador). En quant a les responsabilitats, la classe que conté l'agregat és la responsable de mantenir-lo actualitzat.

A la nostra aplicació, el sistema dissenyat no presenta cap exemple d'ús del patró Iterador no del patró Diccionari ja què, aquests han estat emprats de manera directe durant la implementació mitjançant les eines que proporcionava el llenguatge de programació utilitzat C++. Per tant, tot i haver estat emprats, no s'han reflectit al diagrama de classes.

5.2.3.3 Patrons de disseny Creador i Expert

L'assignació de responsabilitats a objectes consisteix a determinar/assignar quines són les obligacions/responsabilitats concretes dels objectes del diagrama de classes, per tal de donar resposta als esdeveniments externs.

El patró creador és un patró de disseny emprat quan l'assignació de responsabilitats fa referència a la creació d'instàncies de classes. D'aquesta manera, s'aconsegueix mantenir un baix nivell d'acoblament.

A continuació mostrem el detall del mòdul gestor d'autòmats on, els objectes *State* i *Relation* són creats per l'objecte *Automaton*. Per tant, quan una instància d'una altra classe vol crear, modificar o eliminar qualsevol d'aquests objectes, ho ha de fer a través de l'objecte *Automaton*:

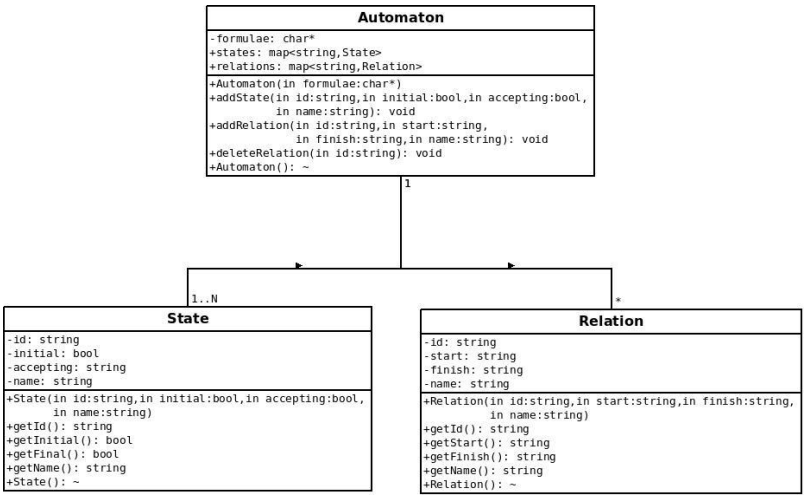


Figura 5.13: Patró Creador per a l'objecte *Automaton*.

El mateix patró s'aplica, per exemple, per a la creació, modificació i eliminació dels objectes de l'objecte *PetriNet* (*Place*, *Transition* i *Arc*).

El patró expert és un patró de disseny emprat per tal de prendre la decisió de quina classe és la que ha de tenir una responsabilitat concreta. D'aquesta manera, s'aconsegueix mantenir un baix nivell d'acoblament, ja que es manté l'encapsulament i, una alta cohesió, ja que es manté una conducta distribuïda entre les classes que tenen la informació. És a dir, el patró expert assigna una responsabilitat a qui té la informació necessària per a realitzar-la.

Aquest patró ha estat emprat al llarg de tota l'aplicació, reduint el sistema al mínim nombre de classes, tot mantenint un nivell de comunicació entre aquestes de manera que es mantingués el mínim acoblament possible.

5.3 Implementació

L'objectiu principal de la implementació és la reducció del disseny a codi.

5.3.1 Tecnologies emprades

Assenyalar que s'ha implementat l'aplicació seguint el criteri d'utilitzar només *software* lliure. Per tant, totes les eines i tecnologies mostrades a continuació són conseqüència directe d'aquesta premissa d'implementació.

5.3.1.1 Llenguatges de programació

A l'hora de triar un llenguatge de programació, calia pensar en primer lloc, en la programació orientada a objectes i, en segon lloc, en la intenció d'ús d'algunes llibreries específiques. Això és el que ha determinat que el nostre llenguatge de programació emprat per a la implementació de l'aplicació hagi estat C++.

En quant a la gestió de la persistència de les dades, s'ha pensat en un llenguatge estàndard d'intercanvi de dades suficientment estès i, amb experiència ja en l'àmbit de programació en què es mou la nostra aplicació com és PNML.

- **C++**

C++ [15] és un llenguatge procedural (orientat a algorismes) i orientat a objectes. Per tant, C++ és un llenguatge que aporta solució a tots els requeriments del sistema desenvolupat: pot ser compilat en múltiples sistemes operatius sense necessitat de modificar el codi (portabilitat), consta d'una llibreria àmplia estàndard (llibreria emprada com explicat a l'apartat anterior de patrons de disseny, com per exemple, amb l'ús de patrons de disseny com Iterador o Diccionari) i, permet l'ús de la memòria de manera explícita o implícita.

- **PNML**

PNML [8] és una proposta de format d'intercanvi de xarxes de Petri el qual, suporta tots els tipus de xarxes de Petri, basat en el format XML [14.1].

En aquest sentit, s'ha creat un fitxer *Schema* [14.2] el qual, defineix un subconjunt de les xarxes de Petri especificades amb XML. Aquesta especificació només obvia certs camps d'especificació gràfica i de modularitat i multiespecificació de xarxes de Petri.

El fitxer *Schema* que defineix una xarxa de Petri vàlida per al nostre sistema és el següent:

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema       attributeFormDefault="unqualified"
                  elementFormDefault="qualified"
                  version="1.0"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="pnml" type="pfcPNML"/>
    <xsd:complexType name="pfcPNML">
        <xsd:all>
            <xsd:element name="net" type="pfcNet"/>
        </xsd:all>
    </xsd:complexType>

    <xsd:complexType name="pfcNet">
        <xsd:sequence>
                <xsd:element name="name" type="pfcName" minOccurs="0" maxOccurs="1"/>
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="place" type="pfcPlace"/>
                        <xsd:element name="transition" type="pfcTransition"/>
                        <xsd:element name="arc" type="pfcArc"/>
                </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="type" type="xsd:string" use="required"/>
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="pfcPlace">
        <xsd:sequence>
                <xsd:element      name="graphics" type="pfcPositionGraphics"
                                  minOccurs="0" maxOccurs="1"/>
                <xsd:element name="name" type="pfcName" minOccurs="0" maxOccurs="1"/>
                <xsd:element      name="initialMarking" type="pfcInitialMarking"
                                  minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="pfcInitialMarking">
        <xsd:sequence>
                <xsd:element name="value" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
                <xsd:element      name="graphics" type="pfcOffsetGraphics"
                                  minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="pfcTransition">
        <xsd:sequence>
                <xsd:element      name="graphics" type="pfcPositionGraphics"
                                  minOccurs="0" maxOccurs="1"/>
                <xsd:element name="name" type="pfcName" minOccurs="0" maxOccurs="1"/>
                <xsd:element      name="toolpecific" type="pfcToolSpecific"
                                  minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
    </xsd:complexType>
```

```
    <xsd:complexType name="pfcToolSpecific">
        <xsd:all>
            <xsd:element name="hidden"/>
        </xsd:all>
        <xsd:attribute name="tool" type="xsd:string" use="required"/>
        <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="pfcArc">
        <xsd:sequence>
            <xsd:element    name="graphics" type="pfcPositionGraphics"
                            minOccurs="0" maxOccurs="1"/>
            <xsd:element    name="inscription" type="pfcName"
                            minOccurs="0" maxOccurs="1"/>
            <xsd:element    name="type" type="pfcType"
                            minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
        <xsd:attribute name="source" type="xsd:IDREF" use="required"/>
        <xsd:attribute name="target" type="xsd:IDREF" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="pfcType">
        <xsd:attribute name="value" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="pfcName">
        <xsd:sequence>
            <xsd:element name="value" type="xsd:string"/>
            <xsd:element    name="graphics" type="pfcOffsetGraphics"
                            minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="pfcPositionGraphics">
        <xsd:sequence>
            <xsd:element    name="position" type="pfcPosition"
                            minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="pfcPosition">
        <xsd:attribute name="x" type="xsd:integer" use="required"/>
        <xsd:attribute name="y" type="xsd:integer" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="pfcOffsetGraphics">
        <xsd:sequence>
            <xsd:element    name="offset" type="pfcPosition"
                            minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

Figura 5.14: Fitxer *Schema* d'especificació de xarxes de Petri (fitxer XSD de l'especificació (5.1))[5].

[5] Aquest fitxer es pot trobar al CD del projecte: PFC/PFC_Codi/jp/jpnml.xsd.

5.3.1.2 Llibreries

Per a les classes del nostre sistema, s'han triat tres llibreries: *Xerces-C++*, *PCCTS 1.33* i, *Gecode*.

- **Xerces-C++**

Xerces-C++ és un *parsejador* (analitzador sintàctic) escrit en C++, per tal de validar documents en format XML. Aquesta llibreria permet llegir i escriure documents en format XML. Proveeix d'una llibreria per a *parsejar*, generar, manipular i, validar, documents XML usant les APIs DOM, SAX i SAX2. Aquest *parsejador* proporciona un alt rendiment, modularitat i, escalabilitat.

El sistema utilitza aquesta llibreria per analitzar, manipular i validar el fitxer d'entrada PNML indicat a les especificacions (5.1) el qual, conté una xarxa de Petri. En aquest sentit, s'ha emprat la metodologia DOM[6] per tal d'accedir a la informació continguda al fitxer d'entrada la qual, construeix un arbre a memòria el qual és accessible. El fitxer que especifica el format de les xarxes de Petri a validar per aquesta llibreria és el fitxer XSD (figura 5.14).

- **PCCTS 1.33**

PCCTS 1.33 és la versió prèvia de *ANTLR*[7] el qual, fou escrit en C. És un conjunt de *software* lliure que permet la construcció de reconeixedors i traductors de llenguatges (compiladors) el qual, està comprès de les següents eines:

1. ANTLR: un generador de *parsers* (analitzadors sintàctics).
2. DLG: un analitzador lèxic (*scanner*).
3. SORCERER: un generador d'arbres de *parseig* els quals, permeten l'especificació de l'estructura de dades en arbre mitjançant una gramàtica.

PCCTS 1.33 treballa amb llenguatges LL(k), amb $k \geq 1$ mitjançant un mètode d'anàlisi sintàctica descendent, és a dir, analitza les entrades d'esquerre a dreta (*left-to-left*).

Aquesta llibreria s'utilitza en aquesta aplicació per a l'anàlisi del fitxer d'entrada LTL indicat a les especificacions (5.1) el qual, representa la propietat LTL a ser verificada sobre una xarxa de Petri. Per a poder ser emprada ha calgut generar els fitxers en llenguatge C++ corresponents a la validació gramaticals de les propietats LTL.

De les tres eines de què disposa la llibreria, hem usat ANTLR i DLG:

1. Creació d'un fitxer que conté la descripció lèxica i sintàctica de la gramàtica. En aquest fitxer s'indica com es farà l'anàlisi, quins seran els *tokens* de la gramàtica i quina serà la gramàtica.

[6] DOM: *Document Object Model*.
[7] ANTLR: *Another Tool for Language Recognition*.

A continuació es mostren els *tokens* i la gramàtica considerada:

tokens

```
#token AND              "AND"
#token OR               "OR"
#token U                "U"
#token R                "R"
#token X                "X"
#token NOT              "NOT"

#token OPENPAR          "\("
#token CLOSEPAR         "\)"

#token COMA             ","

#token IDENTIFIER       "[a-zA-Z][a-zA-Z0-9]*"
#token NUMBER           "[0-9]+"

#token WHITESPACE       "[\ \t]+"           << skip(); >>
#token NEWLINE          "\n"                << skip(); newline(); >>

#token END              "@"
```

gramàtica

```
class LTLExpression {
        ltl
        : formulae END!
        ;

        formulae
        : formulae1 ((U^|R^) formulae1)*
        ;

        formulae1
        : formulae2 ((AND^|OR^) formulae2)*
        ;

        formulae2
        : ((X^|NOT^) formulae2 | formula)
        ;

        formula:
        OPENPAR! formulae CLOSEPAR!
        | parameters
        ;

        function
        : ENABLE
        | TOKEN
        ;

        parameters
        : IDENTIFIER
        | NUMBER
        |
        ;
}
```

Figura 5.15: Fitxer d'especificació de la gramàtica[8].

[8] Aquest fitxer es pot trobar al CD del projecte: PFC/PFC_Codi/Level2/LTLAutomaton/ANTLR/test.g.

2. Mitjançant la comanda *antlr* es generen els fitxers encarregats de la validar sintàcticament la gramàtica, així com els fitxers que permeten generar el codi per a reconèixer el lèxic. Aquests fitxers s'inclouen com a part de la nostra aplicació.
3. Mitjançant la comanda *dlg* es generen els fitxers de reconeixement lèxic de la gramàtica. Aquests fitxers també s'inclouen com a part de la nostra aplicació.

- **Gecode**

Gecode és una eina moderna (la versió 3.1.0 emprada a la aplicació és de l'any 2009) de *constraint programming*. És un entorn de desenvolupament de sistemes i aplicacions basats en restriccions el qual, és:

- **obert**: permet la programació de nous propagadors (com la implementació de restriccions) entre d'altres.
- **lliure**: *software* de llicència MIT de distribució lliure.
- **portable**: està implementat en C++ i es pot emprar en qualsevol sistema operatiu (inclou instruccions de compilació, *linkatge* i execució).
- **accessible**: inclou una àmplia documentació que contempla la programació de diferents tasques.
- **eficient**: ofereix una bona resposta en temps d'execució, ús de memòria i, escalabilitat.
- **paral·lel**: ofereix un bon sistema que s'adapta al *hardware* actual *multicore*.
- **viu**: té una àmplia comunitat d'usuaris.

A la nostra aplicació, utilitzem aquesta llibreria per a la resolució dels algorismes (capítol 4) de verificació de propietats LTL en xarxes de Petri. Ofereix a més, una interfície gràfica on presenta els arbres de cerca d'aquests algorismes, segons la metodologia emprada: *depth-search first* (DFS) o *branch-and-bound* (BAB).

5.3.1.3 Tecnologies addicionals

- **Pipe2**

Pipe2 (*Platform Independent Petrinet Esditor*2) és *software* lliure i independent de la plataforma. Es tracta d'en editor i analitzador de xarxes de Petri.

Aquesta eina ha estat emprada per a dibuixar les xarxes de Petri d'aquest document. De la mateixa manera, les xarxes de Petri exemple, emprades per a l'execució de l'aplicació ja que, *Pipe2* permet guardar les xarxes de Petri en format XML. Addicionalment i, també durant l'execució de les proves, s'ha utilitzat per a la verificació dels resultats obtinguts ja que, també incorpora un analitzador que, entre d'altres, calcula matrius d'incidència i *T-invariants*.

- **Dia**

Dia és un editor de diagrames per sistemes UNIX i Windows. En aquest sentit, s'ha utilitzat per a l'edició dels diagrames de classes i dels diagrames de casos d'ús d'aquest document.

5.3.2 Persistència de les dades

En aquesta aplicació, gairebé tot el que fa referència a la persistència de les dades ha estat explicat a l'apartat anterior (5.3.1) estant lligat amb les llibreries *Xerces-C++*, *PNML* i, *ANTLR*. Només queda pendent la persistència de les dades corresponents al fitxer d'entrada PC, el fitxer que conté les *P-components* de la xarxa de Petri.

5.3.2.1 Format de les *P-components*

El format de codificació de les *P-components* al fitxer d'entrada PC, és el següent:

n p_1 P_{10} P_{11} … $P_{1(p1-1)}$ t_1 T_{10} T_{11} … $T_{1(t1-1)}$ … … p_n P_{n0} P_{n1} … $P_{n(pn-1)}$ t_n T_{n0} T_{n1} … $T_{n(tn-1)}$	n: nombre de *P-components* p_i: nombre de *places* de la *P-component i*-èssima P_{ik}: identificador del k-èssim place de la *P-component* i-èssima t_j: nombre de transicions de la *P-component j*-èssima P_{jk}: identificador de la *k*-èssima transició de la *P-component j*-èssima

Figura 5.16: Format del fitxer d'entrada PC de les *P-components*.

6 Experimentació

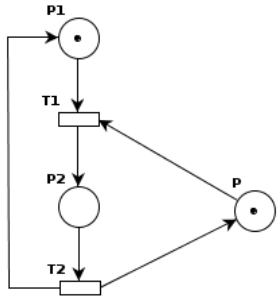
En aquest capítol es mostren els resultats obtinguts en implementar els algorismes anteriorment presentats (al capítol 4 Algorismes). Aquests resultats es presenten tant en mode analític com en mode gràfic.

En quant la visualització gràfica es mostren, tant els autòmats corresponents a la negació de les fórmules LTL a verificar ($A_{\neg\phi}$) així com les Büchi xarxes producte obtingudes en aplicar aquests algorismes (definició 4.4 i definició 4.5), a més de la visualització de la resolució del sistema d'inequacions (*Constraint Programming*) mitjançant la llibreria *Gecode* esmentada al capítol anterior (5.3 Implementació).

Pel que fa als resultats analítics es mostren, tant la matriu d'incidència calculada per a les Büchi xarxes producte generades, com el resultat (*T-invariant*) que s'obté en resoldre el sistema d'inequacions (*constraint programming*), mitjançant novament la llibreria *Gecode*.

6.1 Exemple Base

Aquest primer exemple fa referència a la PN presentada al capítol 4 Algorismes (apartat 4.3.3).

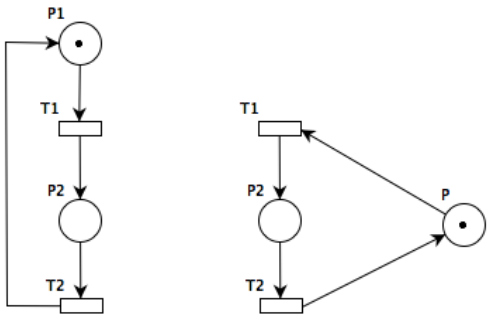


Matriu d'incidència:

$$\begin{array}{c} \\ P \\ P1 \\ P2 \end{array} \begin{array}{c} \begin{array}{cc} T1 & T2 \end{array} \\ \begin{bmatrix} -1 & 1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \end{array}$$

Figura 6.1: PN $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$, amb P_{sys} = (P, P1, P2), T_{sys} = (T1, T2) i M_{0sys} = (P, P1).

Les *P-components* d'aquesta PN són les següents:



(a) (b)

Figura 6.2: *P-components* de la PN de la figura 6.1:
(a) $N_1 = (P_1, T_1)$, amb P_1 = (P1, P2), T_1 = (T1, T2).
(b) $N_2 = (P_2, T_2)$, amb P_2 = (P, P2), T_2 = (T1, T2).

6.1.1 Fórmula LTL: *Next*

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL $\phi = X$ P2 i a la seva negada. D'aquesta manera, es podran observar de manera paral·lela els resultats obtinguts per a totes dues.

6.1.1.1 $\phi = X$ P2

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg(X\,P2) = X\neg P2.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

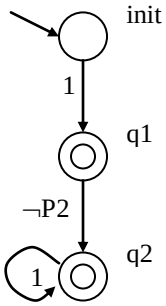
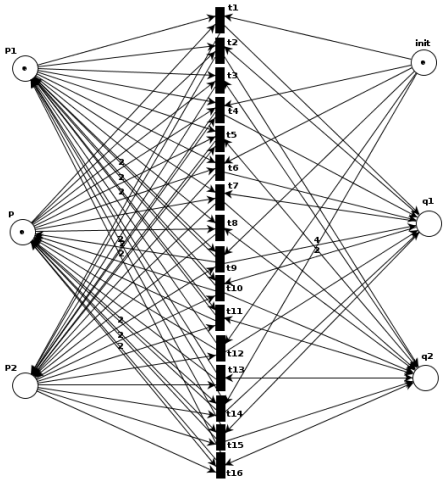


Figura 6.3: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = X$ P2.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4:



Matriu d'incidència:

	t1	t10	t11	t12	t13	t14	t15	t16	t2	t3	t4	t5	t6	t7	t8	t9
init	-1	0	0	-1	0	-1	0	0	0	0	-1	0	-1	0	0	-1
q1	1	-2	0	1	0	1	-1	0	-1	0	1	0	1	-1	0	4
q2	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0	0
P	-1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	2
P1	-1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	1
P2	1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	-1

Figura 6.4: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$ segons la definició 4.4.

Mostrem ara els resultats obtinguts mitjançant *Constraint Programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.4:

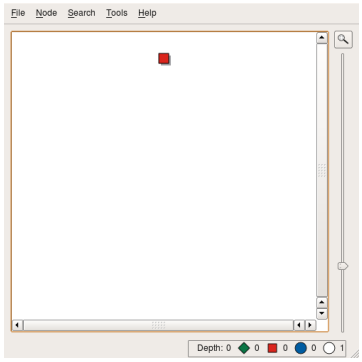
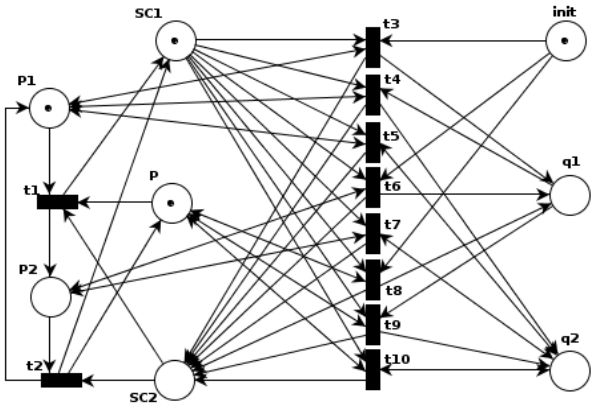


Figura 6.5: Resultat de la Büchi xarxa producte de la figura 6.4.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.3, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = X$ P2, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5:



Matriu d'incidència:

	t1	t10	t2	t3	t4	t5	t6	t7	t8	t9
init	0	0	0	-1	0	0	-1	0	-1	0
q1	0	0	0	1	-1	0	1	0	1	-1
q2	0	0	0	0	1	0	0	0	0	1
P	-1	0	1	0	0	0	0	0	0	0
P1	-1	0	1	0	0	0	0	0	0	0
P2	1	0	-1	0	0	0	0	0	0	0
SC1	1	-1	1	-1	-1	-1	-1	-1	-1	-1
SC2	-1	1	-1	1	1	1	1	1	1	1

Figura 6.6: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.5.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.6:

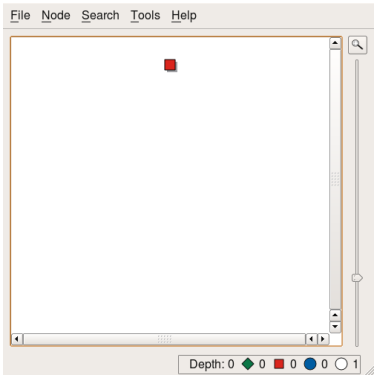


Figura 6.7: Resultat de la Büchi xarxa producte de la figura 6.6.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.3, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = X$ P2, segons la definició 4.5.

Així, podem concloure que per a totes dues definicions (definició 4.4 i definició 4.5) en quant a la construcció de la Büchi xarxa producte, el resultat de l'aplicació del test de l'apartat 4.3.3 és el mateix, que es compleix la fórmula LTL $\phi = X$ P2 analitzada.

6.1.1.2 $\phi = \neg(X$ P2)

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg(X\ P2) = X\ P2.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

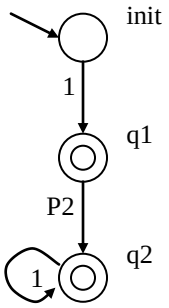
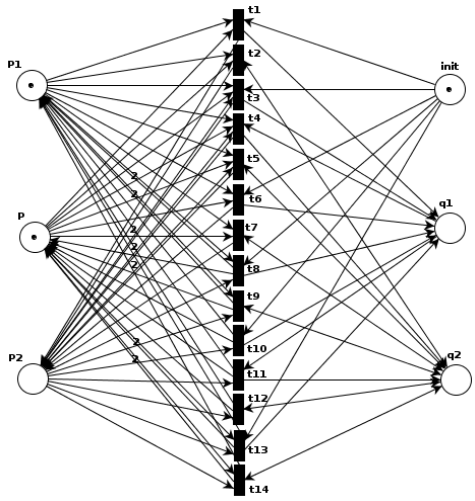


Figura 6.8: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg(X$ P2).

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4:



Matriu d'incidència:

	t1	t10	t11	t12	t13	t14	t2	t3	t4	t5	t6	t7	t8	t9
init	-1	-1	0	0	-1	0	0	-1	0	0	-1	0	-1	0
q1	1	1	-1	0	1	0	0	1	-1	0	1	0	1	0
q2	0	0	1	0	0	0	0	0	1	0	0	0	0	0
P	-1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
P1	-1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1
P2	1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1

Figura 6.9: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.4.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.9:

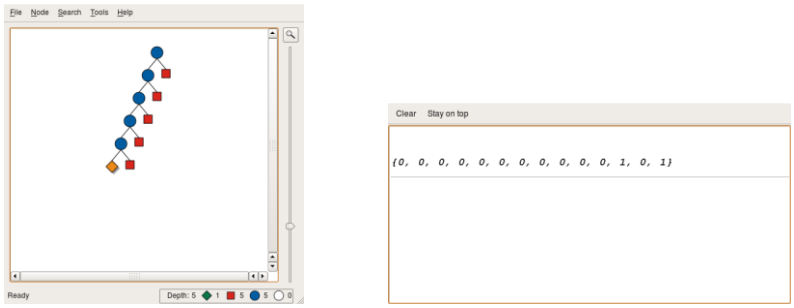
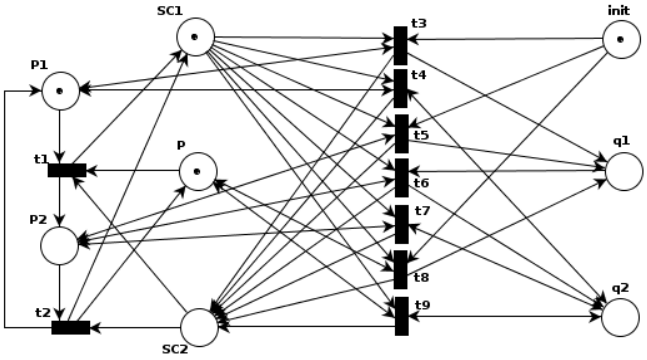


Figura 6.10: Resultat de la Büchi xarxa producte de la figura 6.9.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.8, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix o no la fórmula $\phi = \neg(X\ \mathrm{P2})$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5:



Matriu d'incidència:

	t1	t2	t3	t4	t5	t6	t7	t8	t9
init	0	0	-1	0	-1	0	0	-1	0
q1	0	0	1	0	1	-1	0	1	0
q2	0	0	0	0	0	1	0	0	0
P	-1	1	0	0	0	0	0	0	0
P1	-1	1	0	0	0	0	0	0	0
P2	1	-1	0	0	0	0	0	0	0
SC1	1	1	-1	-1	-1	-1	-1	-1	-1
SC2	-1	-1	1	1	1	1	1	1	1

Figura 6.11: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.5.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.11:

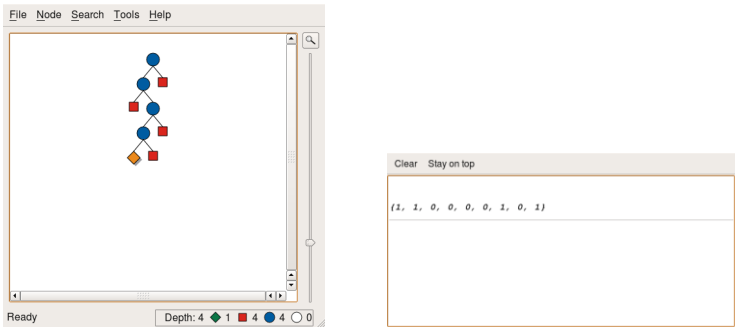


Figura 6.12: Resultat de la Büchi xarxa producte de la figura 6.11.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.8, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix la fórmula $\phi = \neg(X\ \mathrm{P2})$, segons la definició 4.5.

Així, podem concloure que per a totes dues definicions (definició 4.4 i definició 4.5) en quant a la construcció de la Büchi xarxa producte, el resultat de l'aplicació del test de l'apartat 4.3.3 és el mateix, que no se sap si es compleix la fórmula LTL $\phi = \neg(X\ \mathrm{P2})$ analitzada.

6.1.1.3 Resum de l'aplicació de l'operand *Next*

Podem concloure que, aplicant a totes dues fórmules LTL, ϕ = *X* P2 i la seva negada, el test de l'apartat 4.3.3 dóna al primer cas (ϕ) que, es compleix la fórmula i, per al segon cas ($\neg\phi$) que, no se sap. Observem doncs, clarament, la bondat del test de semidecisió aplicat.

6.1.2 Fórmula LTL: *OR* ($\vee$)

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL ϕ = P $\vee$ P2 i a la seva negada. D'aquesta manera, es podran observar de manera paral·lela els resultats obtinguts per a totes dues.

6.1.2.1 ϕ = P $\vee$ P2

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg(P \vee P2) = \neg P \wedge \neg P2.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

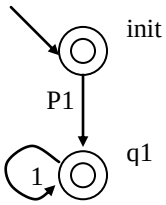
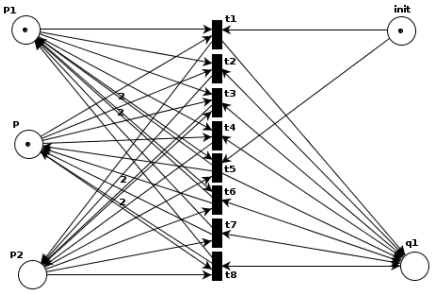


Figura 6.13: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL ϕ = P $\vee$ P2.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4:



Matriu d'incidència:

	t1	t2	t3	t4	t5	t6	t7	t8
init	-1	0	0	0	-1	0	0	0
q1	1	0	0	0	1	0	0	0
P	-1	-1	-1	-1	1	1	1	1
P1	-1	-1	-1	-1	1	1	1	1
P2	1	1	1	1	-1	-1	-1	-1

Figura 6.14: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.4.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.14:

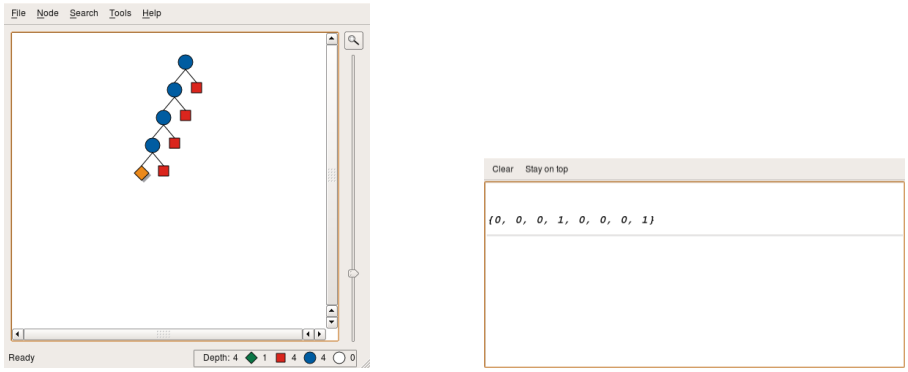
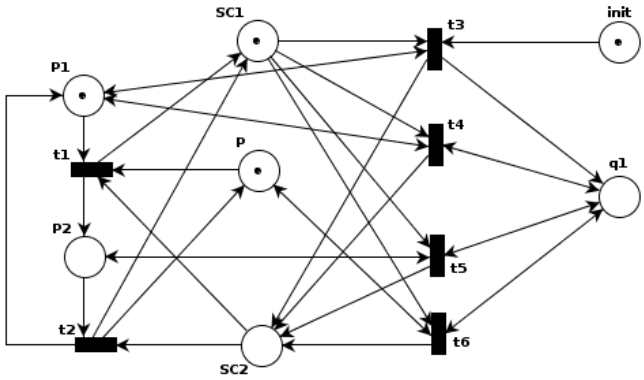


Figura 6.15: Resultat de la Büchi xarxa producte de la figura 6.14.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.13, té *T-invariants* i, per tant, no se sap la PN N_{sys} compleix la fórmula $\phi = \mathrm{P} \vee \mathrm{P2}$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5:



Matriu d'incidència:

	t1	t2	t3	t4	t5	t6
init	0	0	-1	0	0	0
q1	0	0	1	0	0	0
P	-1	1	0	0	0	0
P1	-1	1	0	0	0	0
P2	1	-1	0	0	0	0
SC1	1	1	-1	-1	-1	-1
SC2	-1	-1	1	1	1	1

Figura 6.16: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.5.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.16:

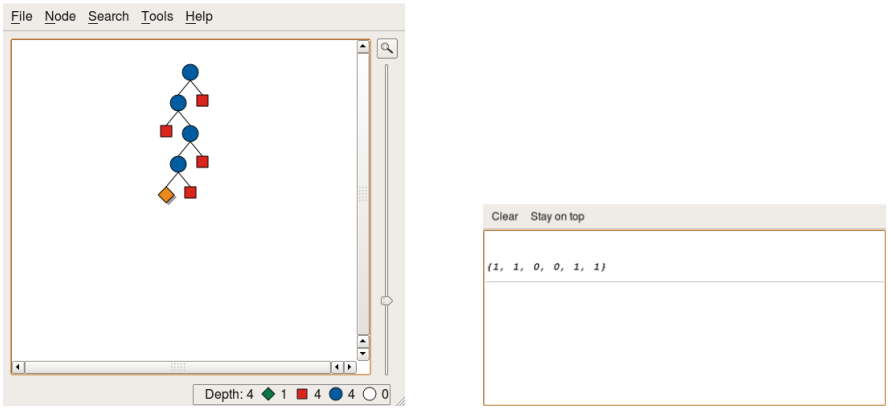


Figura 6.17: Resultat de la Büchi xarxa producte de la figura 6.16.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.13, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix o no la fórmula $\phi = P \vee P2$, segons la definició 4.5.

Així, podem concloure que per a totes dues definicions (definició 4.4 i definició 4.5) en quant a la construcció de la Büchi xarxa producte, el resultat de l'aplicació del test de l'apartat 4.3.3 és el mateix, que no se sap si es compleix la fórmula LTL $\phi = P \vee P2$ analitzada.

6.1.2.2 $\phi = \neg(P \vee P2)$

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg(P \vee P2) = P \vee P2.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

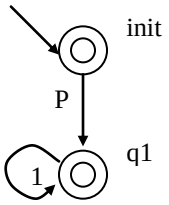
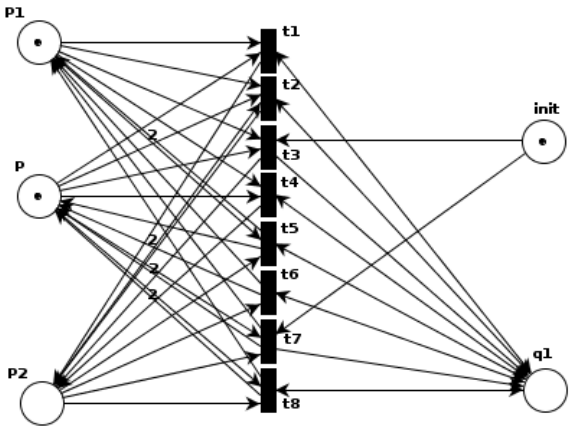


Figura 6.18: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg(P \vee P2)$.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4:



Matriu d'incidència:

	t1	t2	t3	t4	t5	t6	t7	t8
init	0	0	-1	0	0	0	-1	0
q1	0	0	1	0	0	0	1	0
P	-1	-1	-1	-1	1	1	1	1
P1	-1	-1	-1	-1	1	1	1	1
P2	1	1	1	1	-1	-1	-1	-1

Figura 6.19: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.4.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.19:

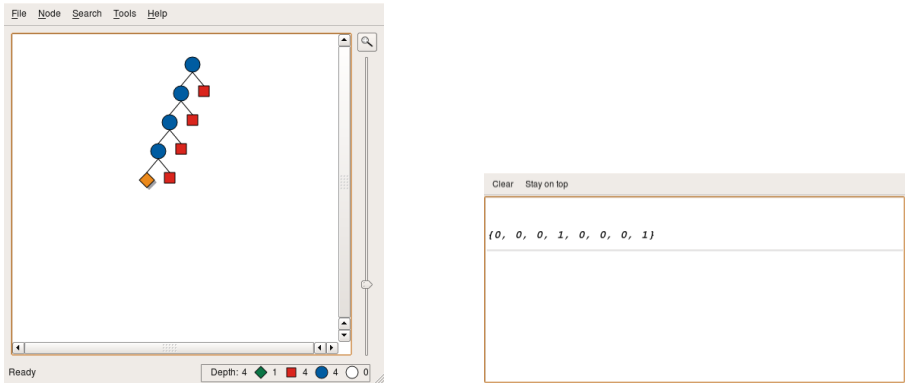
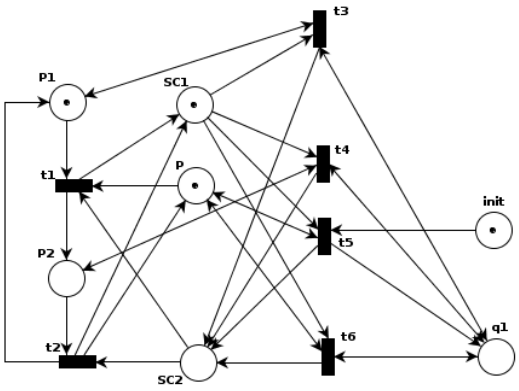


Figura 6.20: Resultat de la Büchi xarxa producte de la figura 6.19.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.18, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix o no la fórmula $\phi = \neg(P \vee P2)$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5:



Matriu d'incidència:

	t1	t2	t3	t4	t5	t6
init	0	0	0	0	-1	0
q1	0	0	0	0	1	0
P	-1	1	0	0	0	0
P1	-1	1	0	0	0	0
P2	1	-1	0	0	0	0
SC1	1	1	-1	-1	-1	-1
SC2	-1	-1	1	1	1	1

Figura 6.21: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.5.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.21:

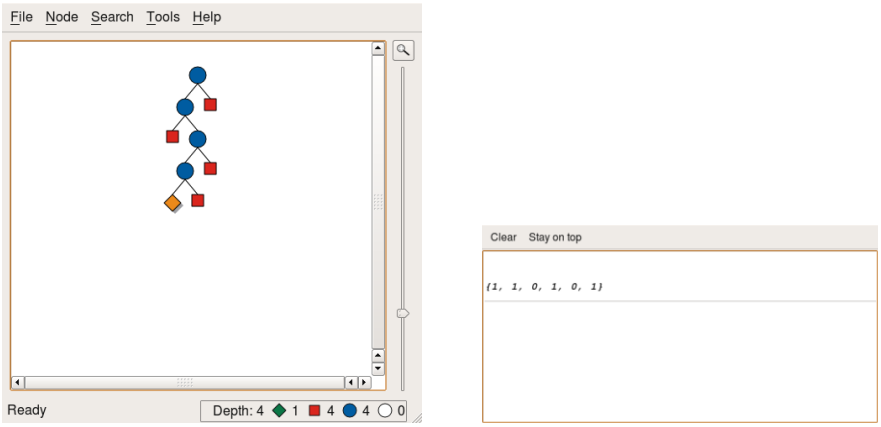


Figura 6.22: Resultat de la Büchi xarxa producte de la figura 6.21.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.18, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix o no la fórmula $\phi = \neg(P \vee P2)$, segons la definició 4.5.

Així, podem concloure que per a totes dues definicions (definició 4.4 i definició 4.5) en quant a la construcció de la Büchi xarxa producte, el resultat de l'aplicació del test de l'apartat 4.3.3 és el mateix, que no se sap si es compleix la fórmula LTL $\phi = \neg(P \vee P2)$ analitzada.

6.1.2.3 Resum de l'aplicació de l'operand *OR* (∨)

Podem concloure que, aplicant a totes dues fórmules LTL, ϕ = P ∨ P2 i la seva negada, el test de l'apartat 4.3.3 dóna per ambdós casos que, no se sap si es compleixen les fórmules, ni per ϕ, ni per a la seva negada $\neg\phi$. En aquest cas, el test de semidecisió aplicat no sap decidir.

6.1.3 Fórmula LTL: *Until*

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL ϕ = P *U* P2 i a la seva negada. D'aquesta manera, es podran observar de manera paral·lela els resultats obtinguts per a totes dues.

6.1.3.1 ϕ = P *U* P2

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg(P\ U\ P2) = \neg P\ V\ \neg P2,$$ amb *V* (*Release*) dual de *U* (*Until*).

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

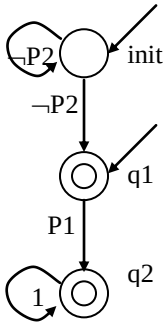


Figura 6.23: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL ϕ = P *U* P2.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4:

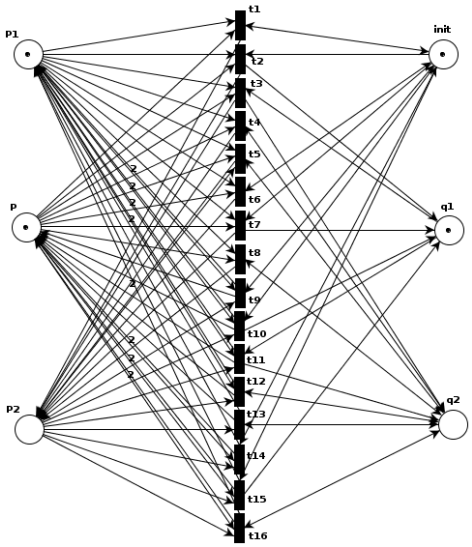


Figura 6.24 (a): Büchi xarxa producte de N_{sys} = (P_{sys}, T_{sys}, M_{0sys}) i $A_{\neg\phi}$, segons la definició 4.4.

Matriu d'incidència:

	t1	t10	t11	t12	t13	t14	t15	t16	t2	t3	t4	t5	t6	t7	t8	t9
init	0	-1	0	0	0	0	-1	0	-1	0	0	0	0	-1	0	0
q1	0	1	-1	0	0	0	1	0	1	-1	0	0	0	1	0	0
q2	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
P	-1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	1
P1	-1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	1
P2	1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1	-1

Figura 6.24 (b): Matriu d'incidència de la Büchi xarxa producte de la figura 6.24 (a).

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.24:

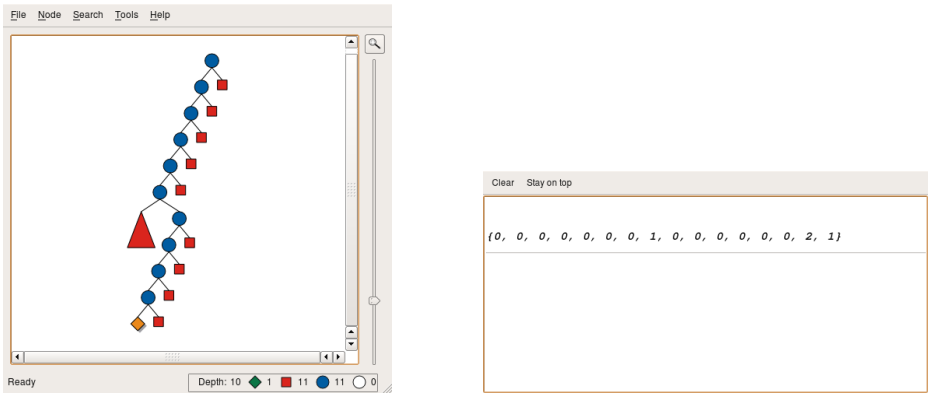


Figura 6.25: Resultat de la Büchi xarxa producte de la figura 6.24.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.23, té *T-invariants* i, per tant, no se sap la PN N_{sys} compleix o no la fórmula ϕ = P *U* P2, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5:

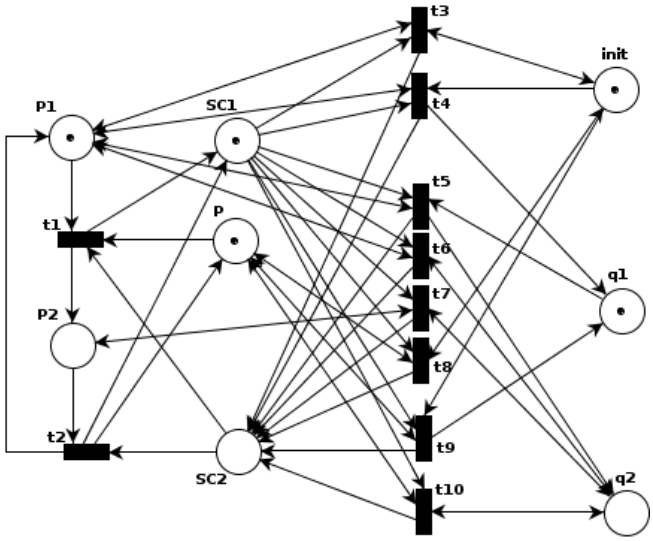


Figura 6.26 (a): Büchi xarxa producte de N_{sys} = (P_{sys}, T_{sys}, M_{0sys}) i $A_{\neg\phi}$, segons la definició 4.5.

Matriu d'incidència:

	t1	t10	t2	t3	t4	t5	t6	t7	t8	t9
init	0	0	0	0	-1	0	0	0	0	-1
q1	0	0	0	0	1	-1	0	0	0	1
q2	0	0	0	0	0	1	0	0	0	0
P	-1	0	1	0	0	0	0	0	0	0
P1	-1	0	1	0	0	0	0	0	0	0
P2	1	0	-1	0	0	0	0	0	0	0
SC1	1	-1	1	-1	-1	-1	-1	-1	-1	-1
SC2	-1	1	-1	1	1	1	1	1	1	1

Figura 6.26 (b): Matriu d'incidència de la Büchi xarxa producte de la figura 6.26 (a).

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.26:

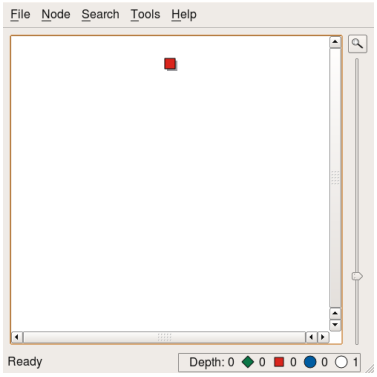


Figura 6.27: Resultat de la Büchi xarxa producte de la figura 6.26.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.23, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula ϕ = P *U* P2, segons la definició 4.5.

Així, veiem que per a la definició 4.4, l'aplicació del test de l'apartat 4.3.3 no sap si es compleix la fórmula però, quan s'aplica la definició 4.5, l'aplicació del test de l'apartat 4.3.3 resol que sí que es compleix la fórmula LTL ϕ = P *U* P2 analitzada.

6.1.3.2 $\phi = \neg($ P *U* P2)

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg(P\ U\ P2) = P\ U\ P2.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

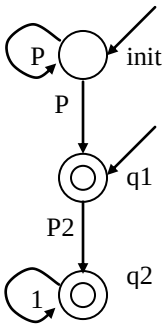
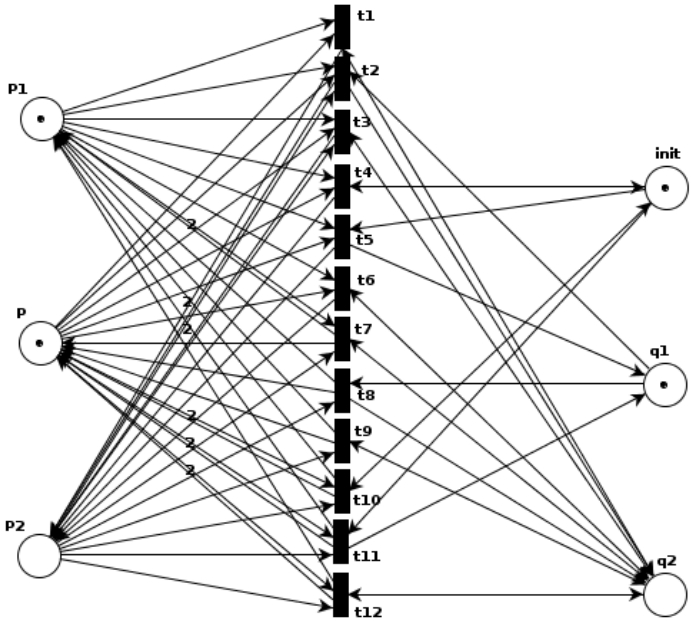


Figura 6.28: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg(\mathrm{P}\ U\ \mathrm{P2})$.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4:



Matriu d'incidència:

	t1	t10	t11	t12	t2	t3	t4	t5	t6	t7	t8	t9
init	0	0	-1	0	0	0	0	-1	0	0	0	0
q1	0	0	1	0	-1	0	0	1	0	0	-1	0
q2	0	0	0	0	1	0	0	0	0	0	1	0
P	-1	1	1	1	-1	-1	-1	-1	-1	1	1	1
P1	-1	1	1	1	-1	-1	-1	-1	-1	1	1	1
P2	1	-1	-1	-1	1	1	1	1	1	-1	-1	-1

Figura 6.29: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.4.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.29:

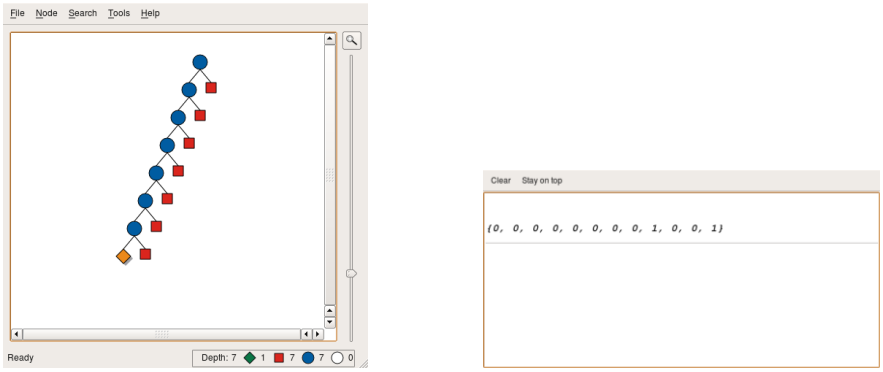
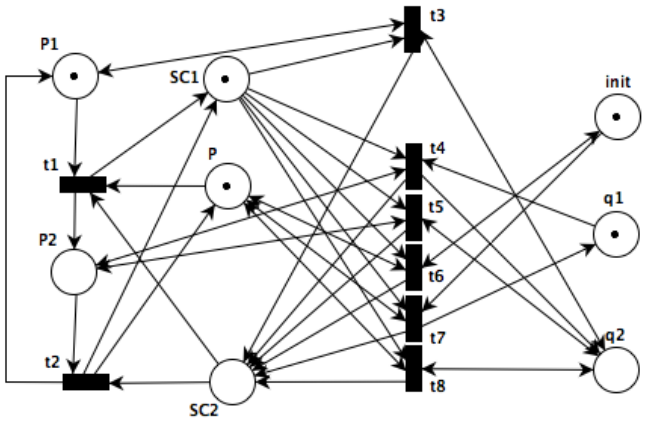


Figura 6.30: Resultat de la Büchi xarxa producte de la figura 6.29.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.28, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix o no la fórmula $\phi = \neg(\mathrm{P}\ U\ \mathrm{P2})$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5:



Matriu d'incidència:

	t1	t2	t3	t4	t5	t6	t7	t8
init	0	0	0	0	0	0	-1	0
q1	0	0	0	-1	0	0	1	0
q2	0	0	0	1	0	0	0	0
P	-1	1	0	0	0	0	0	0
P1	-1	1	0	0	0	0	0	0
P2	1	-1	0	0	0	0	0	0
SC1	1	1	-1	-1	-1	-1	-1	-1
SC2	-1	-1	1	1	1	1	1	1

Figura 6.31: Büchi xarxa producte de $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ i $A_{\neg\phi}$, segons la definició 4.5.

Mostrem ara els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3 a la Büchi xarxa producte de la figura 6.31:

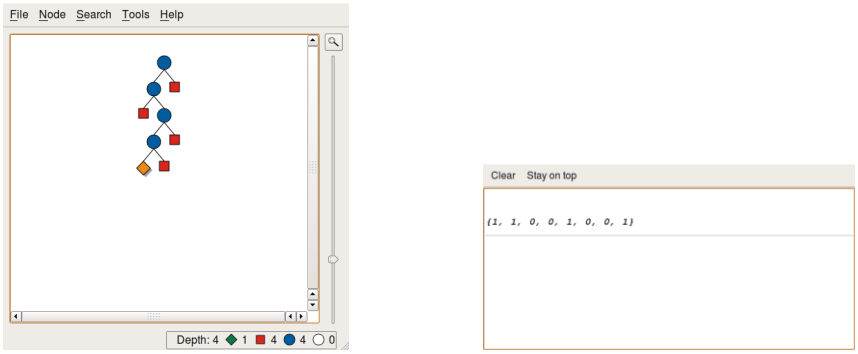


Figura 6.32: Resultat de la Büchi xarxa producte de la figura 6.31.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.1 i l'autòmat $A_{\neg\phi}$ presentat a la figura 6.28, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix la fórmula $\phi = \neg(\text{P } U \text{ P2})$, segons la definició 4.5.

Així, podem concloure que per a totes dues definicions (definició 4.4 i definició 4.5) en quant a la construcció de la Büchi xarxa producte, el resultat de l'aplicació del test de l'apartat 4.3.3 és el mateix, que no se sap si es compleix la fórmula LTL $\phi = \neg(\text{P } U \text{ P2})$ analitzada.

6.1.3.3 Resum de l'aplicació de l'operand *Until*

Podem concloure que, aplicant a totes dues fórmules LTL, $\phi = \text{P } U \text{ P2}$ i la seva negada, el test de l'apartat 4.3.3 dóna al primer cas (ϕ) que, es compleix la fórmula (per a la definició 4.4) i, per al segon cas ($\neg\phi$) que, no se sap. Observem doncs, clarament, la bondat del test de semidecisió aplicat.

6.1.4 Quadre resum

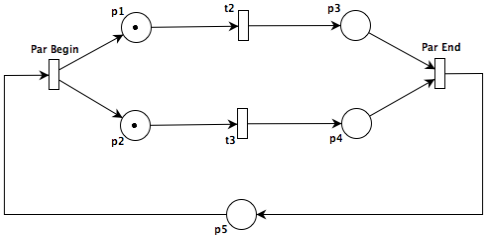
A continuació mostrem un resum dels diferents jocs de prova realitzats sobre la PN de la figura 6.1:

	Definició 4.4	Definició 4.5	Test de semidecisió (4.3.3)
$\phi = \text{X P2}$	■	■	Es compleix la propietat.
$\phi = \neg(\text{X P2})$	◆	◆	No se sap si es compleix la propietat.
$\phi = \text{P} \vee \text{P2}$	◆	◆	No se sap si es compleix la propietat.
$\phi = \neg(\text{P} \vee \text{P2})$	◆	◆	No se sap si es compleix la propietat.
$\phi = \text{P U P2}$	◆	■	Es compleix la propietat (definició 4.6).
$\phi = \neg(\text{P U P2})$	◆	◆	No se sap si es compleix la propietat.

Figura 6.33: Taula resum dels jocs de proves de l'apartat 6.1.

6.2 Exemple Real: Activitats Paral·leles

Aquest segon exemple fa referència a una PN que representa un sistema determinista d'activitats paral·leles:



Matriu d'incidència:

	Par Begin	t2	t3	Par End
p1	1	-1	0	0
p2	1	0	-1	0
p3	0	1	0	-1
p4	0	0	1	-1
p5	-1	0	0	1

Figura 6.34: PN N_{sys} = (P_{sys}, T_{sys}, M_{0sys}) amb,
P_{sys} = (p1, p2, p3, p4, p5), T_{sys} = (Par Begin, t2, t3, Par End) i M_{0sys} = (p1, p2).

6.2.1 ϕ = (p1 $\vee$ p2) U (X p5)

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL ϕ = (p1 $\vee$ p2) U (X p5).

En aquest cas, la fórmula LTL representa la propietat de què, totes les execucions sempre passen per un dels dos camins paral·lels. Com veurem a continuació, es compleix la propietat, és a dir, sempre es passa per un dels dos camins paral·lels. Es tracta d'una propietat simple que pretén mostrar l'abast de l'eina per a casos més complexes.

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg((p1 \vee p2)\, U\, (X\ p5)) = (\neg p1 \wedge \neg p2)\, R\, (X \neg p5).$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

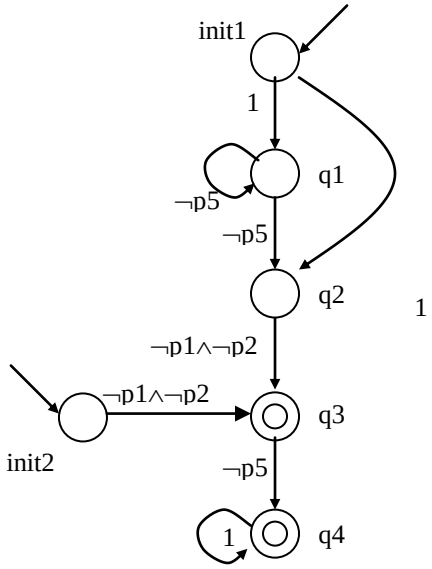


Figura 6.35: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL ϕ = (p1 $\vee$ p2) U (X p5).

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar-li el test de l'apartat 4.3.3:

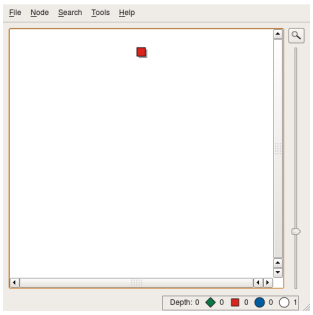


Figura 6.36: Resultat de la Büchi xarxa producte.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.34 i l'autòmat $A_{\neg\phi}$, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = (p1 \vee p2)\ U\ (X\ p5)$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3:

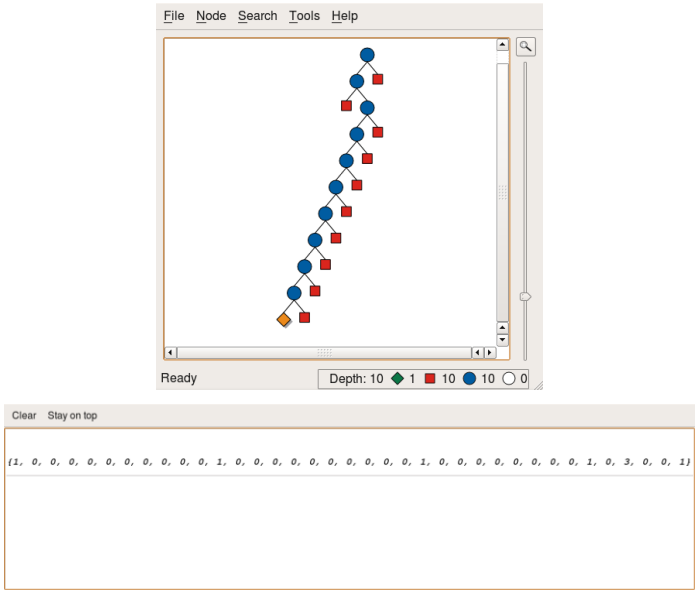


Figura 6.37: Resultat de la Büchi xarxa producte.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.34 i l'autòmat $A_{\neg\phi}$, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix la fórmula $\phi = (p1 \vee p2)\ U\ (X\ p5)$, segons la definició 4.5.

Així, veiem que per a la definició 4.5, l'aplicació del test de l'apartat 4.3.3 no sap si es compleix la fórmula però, quan s'aplica la definició 4.4, l'aplicació del test de l'apartat 4.3.3 resol que sí que es compleix la fórmula LTL $\phi = (p1 \vee p2)\ U\ (X\ p5)$ analitzada.

6.2.2 $\phi = \neg((p1 \wedge p5)\ U\ p2)$

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL $\phi = \neg((p1 \wedge p5)\ U\ p2)$.

En aquest cas, la fórmula LTL representa la propietat de què, totes les execucions no sempre passen pel mateix camí, el de p2 i p4 en aquest cas. Com veurem a continuació, es compleix la propietat, és a dir, no sempre es passa pel mateix camí.

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg((p1 \wedge p5)\ U\ p2) = (p1 \wedge p5)\ U\ p2.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

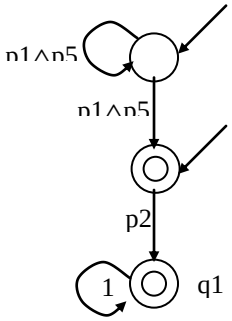


Figura 6.38: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg((p1 \wedge p5)\ U\ p2)$.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar-li el test de l'apartat 4.3.3:

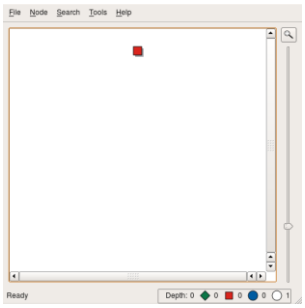


Figura 6.39: Resultat de la Büchi xarxa producte.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.34 i l'autòmat $A_{\neg\phi}$, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = \neg((p1 \wedge p5)\ U\ p2)$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3:

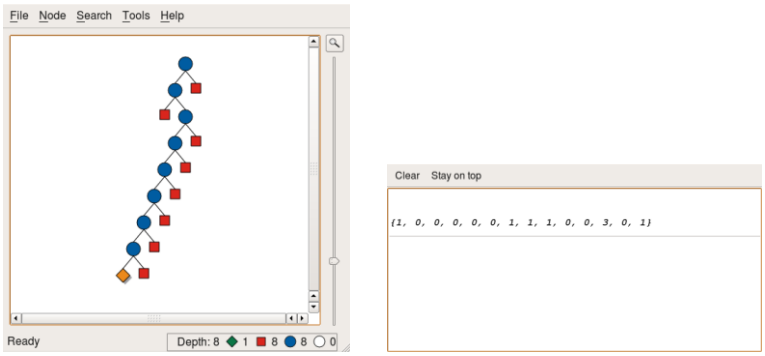


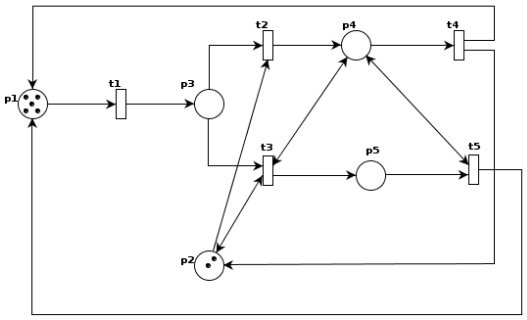
Figura 6.40: Resultat de la Büchi xarxa producte.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.34 i l'autòmat $A_{\neg\phi}$, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix la fórmula $\phi = \neg((p1 \wedge p5)\ U\ p2)$, segons la definició 4.5.

Així, veiem que per a la definició 4.5, l'aplicació del test de l'apartat 4.3.3 no sap si es compleix la fórmula però, quan s'aplica la definició 4.4, l'aplicació del test de l'apartat 4.3.3 resol que sí que es compleix la fórmula LTL $\phi = \neg((p1 \wedge p5)\ U\ p2)$ analitzada.

6.3 Exemple Real: Sistema Multiprocessador

Aquest tercer exemple fa referència a una PN que representa un sistema multiprocessador:



Matriu d'incidència:

	t1	t3	t2	t4	t5
p1	-1	0	0	1	1
p3	1	-1	-1	0	0
p4	0	0	1	-1	0
p5	0	1	0	0	-1
p2	0	0	-1	1	0

Figura 6.41: Una PN modelant un sistema multiprocessador on, els *tokens* a p_1 representen processadors actius, p_2 els busos disponibles, p_3, p_4 i p_5 els processadors esperant tenir accés, encuats a memòries comunes, respectivament.

6.3.1 $\phi = \neg(p4\ U\ p5)$

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL $\phi = \neg(\ p4\ U\ p5)$.

En aquest cas, la fórmula LTL representa la propietat de què, no n'hi han prorocessadors accedint a memòria fins a que s'encuen.

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg(p4\ U\ p5) = p4\ U\ p5.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

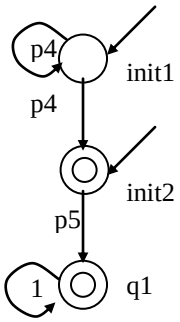


Figura 6.42: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg(\ p4\ U\ p5)$.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar-li el test de l'apartat 4.3.3:

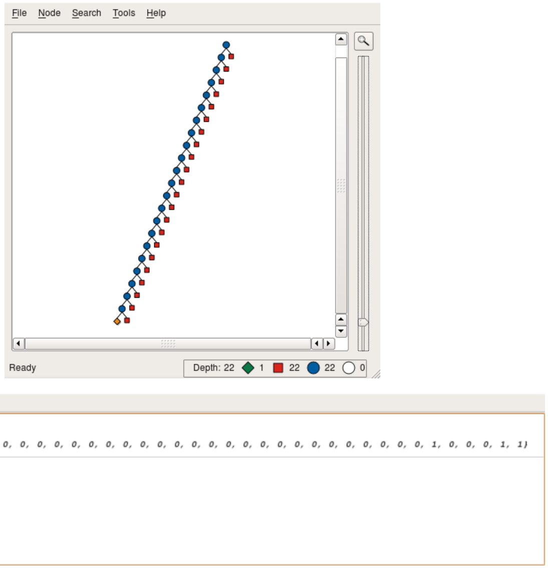


Figura 6.43: Resultat de la Büchi xarxa producte.

Podem observar que troba una solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.41 i l'autòmat $A_{\neg\phi}$, té *T-invariants* i, per tant, la PN N_{sys} no se sap si compleix la fórmula $\phi = \neg($ p4 U p5$)$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3:

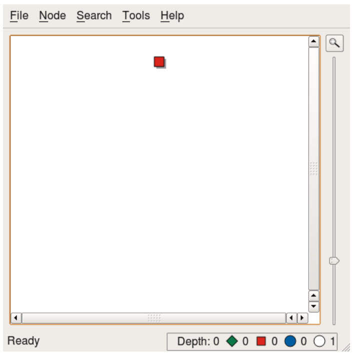


Figura 6.44: Resultat de la Büchi xarxa producte.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.41 i l'autòmat $A_{\neg\phi}$, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = \neg($ p4 U p5$)$, segons la definició 4.5.

Així, veiem que per a la definició 4.4, l'aplicació del test de l'apartat 4.3.3 no sap si es compleix la fórmula però, quan s'aplica la definició 4.5, l'aplicació del test de l'apartat 4.3.3 resol que sí que es compleix la fórmula LTL $\phi = \neg($ p4 U p5$)$ analitzada.

6.3.2 $\phi = \neg(p2 \wedge p5)$

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL $\phi = \neg(p2 \wedge p3)$.

En aquest cas, la fórmula LTL representa la propietat de què, no n'hi han busos disponibles o no n'hi han processadors esperant a ser atesos.

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg(p2 \wedge p5) = p2 \wedge p5.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

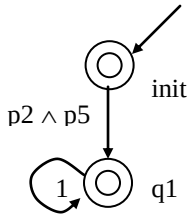


Figura 6.45: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg(p2 \wedge p5)$.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar-li el test de l'apartat 4.3.3:

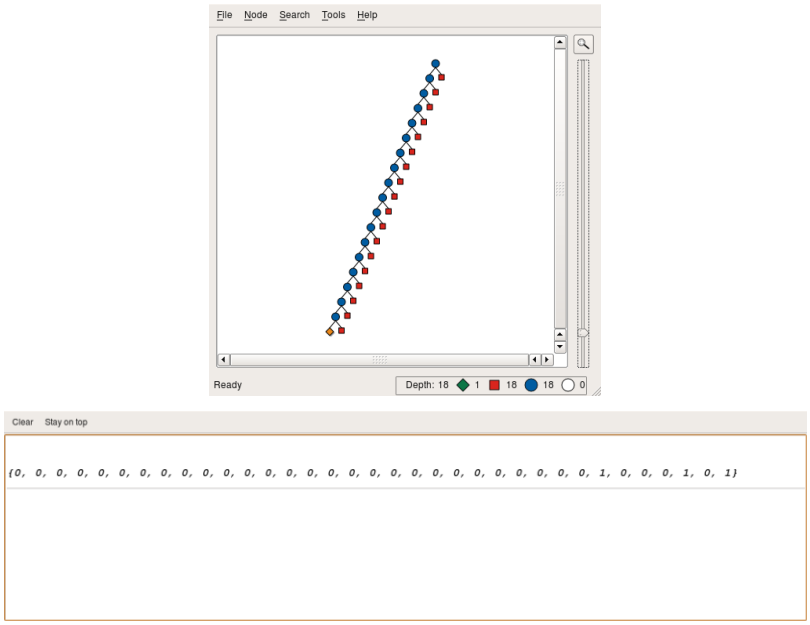


Figura 6.46: Resultat de la Büchi xarxa producte.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.41 i l'autòmat $A_{\neg\phi}$, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix la fórmula $\phi = \neg(p2 \wedge p5)$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3:

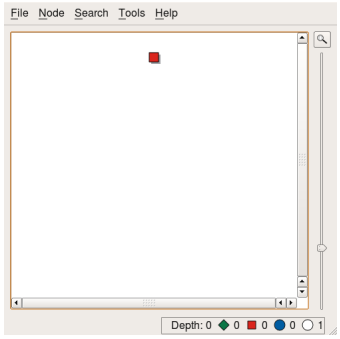


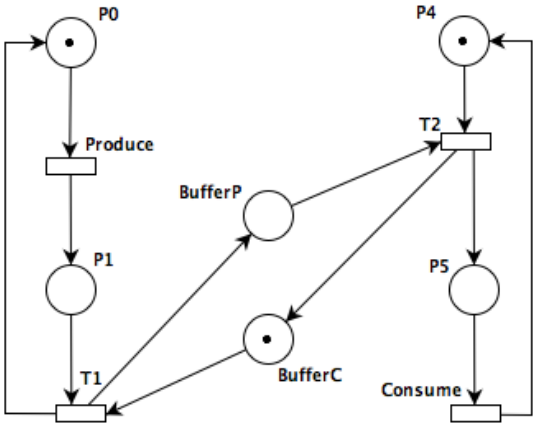
Figura 6.47: Resultat de la Büchi xarxa producte.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.41 i l'autòmat $A_{\neg\phi}$, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = \neg(p2 \wedge p5)$, segons la definició 4.5.

Així, veiem que per a la definició 4.4, l'aplicació del test de l'apartat 4.3.3 no sap si es compleix la fórmula però, quan s'aplica la definició 4.5, l'aplicació del test de l'apartat 4.3.3 resol que sí que es compleix la fórmula LTL $\phi = \neg(p2 \wedge p5)$ analitzada.

6.4 Exemple Real: Sistema Productor/Consumidor

Aquest quart exemple fa referència a una PN que representa un sistema productor/consumidor. En particular, es tracta d'un sistema *pull*, és a dir, el sistema productor produeix només quan el sistema consumidor ho requereix:



Matriu d'incidència:

	Produce	T1	T2	Consume
P0	-1	1	0	0
P1	1	-1	0	0
BufferP	0	1	-1	0
BufferC	0	-1	1	0
P4	0	0	-1	1
P5	0	0	1	-1

Figura 6.48: Una PN modelant un sistema productor/consumidor *pull*.

6.4.1 $\phi = \neg((P0 \wedge P4)\ U\ P1)$

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL $\phi = \neg((P0 \wedge P4)\ U\ P1)$.

En aquest cas, la fórmula LTL representa la propietat de què, no sempre estan productor i consumidor preparats per iniciar el seu procés i, és el productor qui comença.

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg((P0 \wedge P4)\ U\ P1) = (P0 \wedge P4)\ U\ P1.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

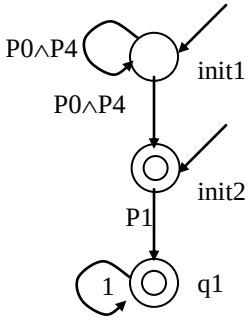


Figura 6.49: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg((P0 \wedge P4)\ U\ P1)$.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar-li el test de l'apartat 4.3.3:

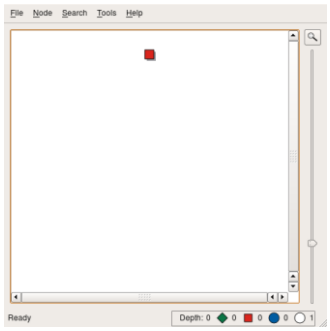


Figura 6.50: Resultat de la Büchi xarxa producte.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.48 i l'autòmat $A_{\neg\phi}$, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = \neg((P0 \wedge P4)\ U\ P1)$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3:

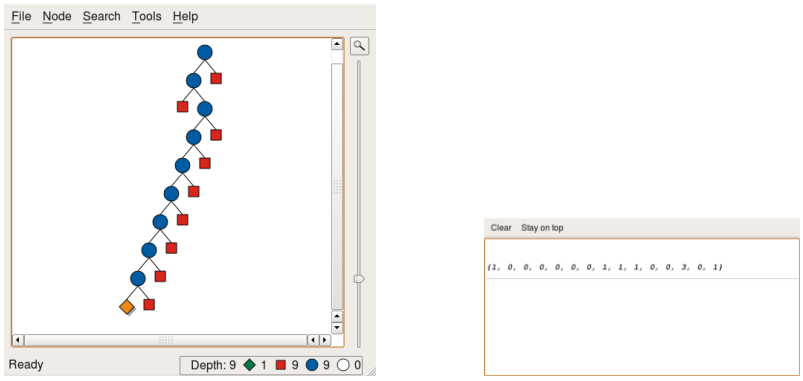


Figura 6.51: Resultat de la Büchi xarxa producte.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.48 i l'autòmat $A_{\neg\phi}$, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix la fórmula $\phi = \neg((P0 \wedge P4)\ U\ P1)$, segons la definició 4.5.

Així, veiem que per a la definició 4.5, l'aplicació del test de l'apartat 4.3.3 no sap si es compleix la fórmula però, quan s'aplica la definició 4.4, l'aplicació del test de l'apartat 4.3.3 resol que sí que es compleix la fórmula LTL $\phi = \neg((P0 \wedge P4)\ U\ P1)$ analitzada.

6.4.2 $\phi = \neg(\neg P3 \wedge \neg P2)$

A continuació s'aplicaran els algorismes presentats (al capítol 4 Algorismes) a la fórmula LTL $\phi = \neg(\neg P3 \wedge \neg P2)$.

En aquest cas, la fórmula LTL representa la propietat de què, no pot estar tots dos subsistemes sense producció, és a dir, o bé el sistema productor ha deixat producció al seu *buffer* o bé, el sistema consumidor està sol·licitant producció al seu *buffer*. Com veurem a continuació, la propietat que es compleix es la seva negada, és a dir, mai estan buits tots dos *buffers* o, el que és equivalent, el sistema és continu i sempre produeix quan es demana.

En primer lloc, es construeix l'autòmat de la negació de la fórmula LTL ($A_{\neg\phi}$), en forma negada normal:

$$\neg\phi = \neg\neg(\neg P3 \wedge \neg P2) = \neg P3 \wedge \neg P2.$$

Com a resultat d'aplicar l'algorisme 4.1, s'obté el següent autòmat ($A_{\neg\phi}$):

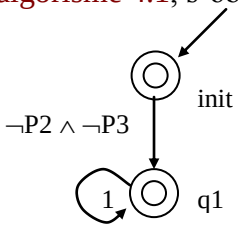


Figura 6.52: Autòmat $A_{\neg\phi}$ corresponent a la fórmula LTL $\phi = \neg(\neg P3 \wedge \neg P2)$.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.4 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar-li el test de l'apartat 4.3.3:

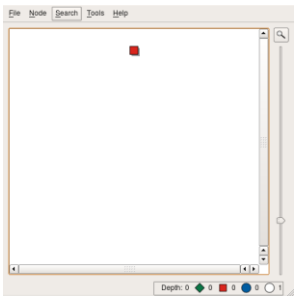


Figura 6.53: Resultat de la Büchi xarxa producte.

Podem observar que no es troba cap solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.48 i l'autòmat $A_{\neg\phi}$, no té *T-invariants* i, per tant, la PN N_{sys} compleix la fórmula $\phi = \neg(\neg P3 \wedge \neg P2)$, segons la definició 4.4.

Calculem ara la Büchi xarxa producte de N_{sys} i $A_{\neg\phi}$, segons la definició 4.5 i, mostrem els resultats obtinguts mitjançant *constraint programming* en aplicar el test de l'apartat 4.3.3:

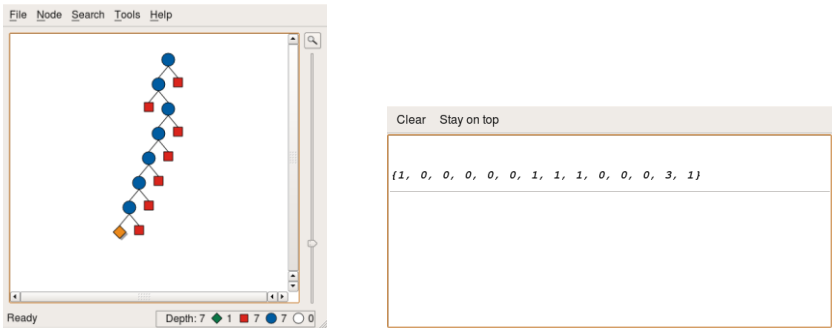


Figura 6.54: Resultat de la Büchi xarxa producte.

Podem observar que es troba solució per al sistema d'inequacions. Per tant, es pot dir que, la Büchi xarxa producte, corresponent al producte de la PN N_{sys} presentada a la figura 6.48 i l'autòmat $A_{\neg\phi}$, té *T-invariants* i, per tant, no se sap si la PN N_{sys} compleix la fórmula $\phi = \neg(\neg P3 \wedge \neg P2)$, segons la definició 4.5.

Així, veiem que per a la definició 4.5, l'aplicació del test de l'apartat 4.3.3 no sap si es compleix la fórmula però, quan s'aplica la definició 4.4, l'aplicació del test de l'apartat 4.3.3 resol que sí que es compleix la fórmula LTL $\phi = \neg(\neg p3 \wedge \neg p2)$ analitzada.

7 Valoració econòmica

7.1 Planificació temporal

El desenvolupament del projecte ha constat de diverses etapes. Tal i com s'ha indicat al capítol 5, com a conseqüència de l'evolució de les necessitats al llarg del desenvolupament del *software*, el model emprat ha acabat sent un model de prototipatge, en comptes del model clàssic o en cascada. Això, evidentment, també ha tingut les seves implicacions en les durades de les etapes del projecte.

Les etapes en les quals, s'ha dividit el projecte, són:

Etapa 1: Estudi i comprensió teòrica dels conceptes a utilitzar.

Etapa 2: Anàlisi de requeriments.

Etapa 3: Especificació i disseny.

Etapa 4: Estudi i aprenentatge de les tecnologies a emprar.

Etapa 5: Implementació.

Etapa 6: Proves.

Etapa 7: Documentació.

A continuació es mostra una taula amb les hores planificades per a cadascuna de les etapes i, les hores reals finalment dedicades a cadascuna d'elles:

ETAPA	HORES PLANIFICADES	HORES REALS
1	80	110
2	10	5
3	80	80
4	40	70
5	260	280
6	120	140
7	160	180
Total	**750**	**865**

Figura 7.1: Planificació temporal del projecte (valoració aproximada).

En quant a l'increment d'hores respecte a la previsió de duració d'un projecte es déu, principalment, al fet que la teoria i eines necessàries per al desenvolupament de l'aplicació han estat noves i, per tant, ha calgut comprendre i assimilar aquests nous conceptes alhora que s'anaven implementant.

7.2 Cost del projecte

Tal i com es va indicar a l'apartat 5.3.1, assenyalar que s'ha implementat l'aplicació seguint el criteri d'utilitzar només *software* lliure. Per tant, el seu cost és nul.

En quant al *hardware* emprat, es tracta d'un ordinador domèstic, valorat en aproximadament 1.000 €.

Finalment, en quant a les hores de treball, a continuació es llisten les etapes de desenvolupament del projecte, juntament amb les hores planificades, ja que són les hores compromeses i, per tant, acordades i, un rol i cost per hora d'aquest rol per a cadascuna de les etapes:

ETAPA	***ROL***	***€/HORA***	***HORES***	***COST***
1	Analista	36	80	2.880 €
2	Analista	36	10	360 €
3	Analista	36	80	2.880 €
4	Programador	30	40	1.200 €
5	Programador	30	260	7.800 €
6	Programador	30	120	3.600 €
7	Analista	36	160	5.760 €
Total			**750**	**24.480 €**

Figura 7.2: Cost de les hores de treball planificades.

Així, tenint en compte el cost de les hores de treball i el cost del *hardware*, el cost total del projecte és de **25.480 €**.

8 Conclusions

8.1 Objectius assolits

A la finalització d'aquest projecte i, observant els resultats obtinguts, podem concloure que, els objectius plantejats al principi del mateix s'han assolit en gairebé la seva totalitat.

Així, s'ha implementat una aplicació de verificació de propietats LTL a sistemes concurrents modelats amb xarxes de Petri, especificades en PNML, mitjançant *constraint programming*, amb la realització d'un algorisme de semidecisió. Aquest algorisme es presenta sobre dues metodologies de composició de les propietats LTL a verificar sobre la xarxa de Petri: una composició natural i, l'altre, amb la construcció d'una xarxa de Petri segura.

La bondat de l'aplicació implementada es presenta en un ampli ventall de jocs de proves on es mostren els resultats obtinguts, una comparativa entre aquests i, una visualització gràfica de la resolució del sistema.

8.2 Treball futur

Aquest projecte, tot i estar finalitzat, no és un projecte tancat. Encara resten obertes moltes possibilitats d'ampliació en les quals, treballar en un futur.

Es poden ampliar les propietats LTL acceptades per la gramàtica definida així com implementar algun algorisme de simplificació de les mateixes. Resten poques propietats a incloure, de fet, només resten operands ja que totes les propietats LTL poden ser verificades amb l'aplicació implementat.

8.3 Valoració global del projecte

La valoració global del projecte és més que positiva.

En primer lloc, cal destacar l'etapa de documentació. Els coneixements adquirits al llarg del desenvolupament del projecte són, pràcticament en la seva totalitat, addicionals als que s'havien adquirit durant la realització de la carrera. I, en quant a les habilitats adquirides, encara mereixen una major menció. Aquest projecte requereix d'una àmplia dedicació a l'estudi i comprensió de teories de l'àmbit informàtic complementàries als coneixements adquirits fins al moment. A més, cal una forta dedicació a la comprensió i capacitat d'anàlisi de *papers* de caràcter acadèmic, així com la necessitat d'abstracció suficient com per a desenvolupar eines informàtiques que implementin aquests nous conceptes.

En quant a l'etapa d'enginyeria del *software*, cal destacar el fet que s'ha de desenvolupar una aplicació complexa com la requerida en aquest projecte, en la seva totalitat. Cal una aplicació metòdica de les eines i metodologies estudiades al llarg de la carrera per tal de desenvolupar amb èxit l'aplicació desitjada. En aquest sentit, s'ha desenvolupat l'especificació formal del projecte, amb el corresponent anàlisi de requeriments, una etapa de disseny de l'aplicació i, finalment una etapa d'implementació. A més, totes elles desenvolupades de manera iterativa, tot permetent l'adequació de l'aplicació en desenvolupament als nous objectius que el projecte anava assolint. També cal destacar el fet que, la implementació ha requerit d'interaccionar amb moltes llibreries externes a la pròpia aplicació en sí, totes elles d'àmbits diferents, amb totes les dificultats que això comporta.

Per tot això i, pel fet de veure realitzat el projecte amb els seus objectius assolits, amb el caràcter tant teòric com pràctic que ha tingut, fa que la valoració del projecte sigui més que positiva.

8.4 Conclusions

L'eina desenvolupada aporta moltes possibilitats en l'estudi dels sistemes concurrents i, en general, de qualsevol sistema que es pugui modelitzar amb xarxes de Petri. A més, amb la inclusió d'algorismes que permeten la traducció de propietats en autòmats, en particular, la inclusió de propietats LTL, fa que l'aplicació desenvolupada tingui encara més unes majors possibilitats d'estudi, anàlisi i explotació.

També destacar l'ús de formats estàndard com són PNML o, el fet de definir gramàtiques (ANTLR) per tal d'incloure propietats LTL. Aquestes eines faciliten el fet que l'aplicació sigui portable i, que els resultats obtinguts puguin ser analitzats o emprats en d'altres estudis.

Finalment, cal destacar també, l'aplicació del *model-checking* mitjançant *constraint programming*, amb una llibreria actual (*Gecode*), amb els avantatges que això suposa. S'han pogut implementar restriccions noves, tot desenvolupant tests de semidecisió de l'àmbit del *model-checking*.

9 Bibliografia

Sistemes de transició

[1] Andrei Voronkov.
"*Logic in Computer Science*".
http://www.voronkov.com/lics.cgi

Xarxes de Petri

[2] Tadao Murata.
Petri Nets: Properties, Analysis and Applications.
Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, April 1989.
http://www.cs.sysu.edu.cn/selab/references/TCS/T.Murata(IEEE1989)%20Petri%20Nets%20-%20Properties,%20Analysis%20and%20Applications.pdf

[2.1] http://www.cs.unc.edu/~montek/teaching/spring-04/petrinets.ppt

[3] Jörg Desel and Javier Sparza.
Free Choice Petri Nets.
Cambridge University Press, 1995.

LTL

[4] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper.
Simple On-the-fly Automatic Verification of Linear Temporal Logic.
In Protocol Specification Testing and Verification. 1995.
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.2625

[4.1] Dimitra Giannakopoulou, and Flavio Lerda.
Efficient translation of LTL formulae into Büchi automata.
RIACS Technical Report 01.29. June 2001.
Research Institute for Advanced Computer Science.
http://www.riacs.edu/research/technical_reports/TR_pdf/TR_01.29.pdf

[4.2] Flavio Lerda.
LTL to Büchi Automata.
www.cs.cmu.edu/~emc/15-820A/reading/ltl_to_buechi_automata.ppt

[4.3] Mads Dam.
Translating LTL to Automata.
http://www.imit.kth.se/courses/2G1516/Docs04/Slides/LTL-to-automata.pdf

Constraint Programming

[5] Krzysztof R. Apt.
Principles of Constraint Programming.
Cambridge University Press, 2003.

[6] Stephan Melzer.
Verification of Parallel Systems Using Constraint Programming.
In Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97). Springer-Verlag, LNCS 1330. 1997.
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.8250

[7] Javier Esparza and Stephan Melzer.
Model Checking LTL using Constraint Programming.
In 18th International Conference on Application and Theory of Petri Nets. 1997.
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.1543

[7.1] Javier Esparza and Stephan Melzer.
Model Checking LTL using Constraint Programming.
Technical report. Technische Universität München, March 1997.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.5090&rep=rep1&type=ps

PNML *(Petri Net Markup Language)*

[8] http://www.pnml.org/

[8.1] Matthias Jüngel, Ekkart Kindler, and Michael Weber.
The Petri Net Markup Language.
S. Philippi (Ed.): 7. Workshop Algorithmen und Werkzeuge für Petrinetze.
Fachberichte Informatik 7/2000, Universität Koblenz-Landau.

[8.2] Ekkart Kindler and Michael Weber.
A Universal Module Concept for Petri Nets
- an implementation-oriented approach -.
Informatik-Berichte 150, Humboldt-Universität zu Berlin.

[8.3] Jonathan Billington, Soren Christensen, Kees van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber.
The Petri Net Markup Language: Concepts, Technology, and Tools.

[8.4] Matthias Jüngel, Ekkart Kindler, and Michael Weber.
Towards a Generic Interchange Format for Petri Nets - Position Paper -.
Bastide, R. et al (eds.): Meeting on XML/SGML based Interchange Formats for Petri Nets.
Aarhus, Denmark, 21st ICATPN.(pp.1-5) Jun.2000.

[8.5] Michael Weber and Ekkart Kindler.
The Petri Net Markup Language.

[9] http://www.informatik.hu-berlin.de/top/pnml/

Altres referències

[10] Jorge Muñoz Gama.
Process Mining: descobrint models formals a partir de logs d'un sistema.
Tutor: *Josep Carmona Vargas*.
PFC-FIB-LSI, Juny 2009.

[11] Sara Royuela Alcázar.
Una eina per verificar propietats en xarxes de Petri usant àlgebra lineal.
Tutor: *Josep Carmona Vargas*.
PFC-FIB-LSI, 1 de Juliol del 2010.

[12] Dolors Costal, Xavier Franch, M. Ribera Sancho & Ernest Teniente.
Enginyeria del software. Especificació.
Especificació de sistemes orientats a objectes amb la notació UML.
Edicions UPC, Febrer 2005.

[13] *Enginyeria del software* 2 i *Projecte d'enginyeria del Software i Bases de Dades*.
Apunts de l'assignatura.

[14] World Wide Web Consortium (W3C)

[14.1] Extensible Markup Language (XML)
http://www.w3.org/XML

[14.2] XML Schema
http://www.w3.org/XML/Schema

[15] http://www.cplusplus.com/

[16] A parser generator of ANTLR (PCCTS 1.33)
http://www.antlr2.org/pccts133.html

[17] Validating XML parser written in a portable subset of C++ (Xerces-C++)
http://xerces.apache.org/xerces-c/

[18] Platform Independent Petri net Editor 2 (PIPE2)
http://pipe2.sourceforge.net/

[19] A program to draw structured diagrams (DIA)
http://dia-installer.de/index_en.html

A Teoria de *Petri Nets*

La teoria aquí explicada està extreta de la referència [2], essent una ampliació de la teoria presentada a l'apartat 3.2. Al document al qual es fa referència, es pot trobar una teoria encara més exhaustiva del tema.

A.1 Definicions

A una PN que pot acomodar un nombre infinit de *tokens* a cada *place* (com a l'exemple anterior), se l'anomena *xarxa de capacitat infinita*. Per a modelar molts sistemes físics és natural considerar un límit superior per al nombre de *tokens* que cada *place* pot acumular. A una PN d'aquest tipus se l'anomena *xarxa de capacitat finita*.

A una xarxa de capacitat finita (N, M_0), cada *place* p té associat una *capacitat* $K(p)$, el nombre màxim de *tokens* que p pot acumular a cada moment. Per a xarxes de capacitat finita, existeix una condició addicional per a la disponibilitat d'una transició t, que el nombre de *tokens* a cada *place* de sortida no excedeixi la seva capacitat $K(p)$ després de disparar t.

Aquesta regla de la restricció de capacitat és anomenada *regla de transició estricta* mentre que, la regla sense aquesta restricció és anomenada *regla de transició* (*dèbil*). Donada una xarxa de capacitat finita (N, M_0), és possible aplicar tant la regla de transició estricta a la xarxa donada (N, M_0) o, equivalentment, la regla de transició (dèbil) a una xarxa transformada (N', M'_0), la xarxa obtinguda de (N, M_0) per la *transformació del place complementari* on, s'assumeix que N és pura.

Definició A.1: (*Complementary-place transformation*)

Pas 1: Afegir un *place* p' complementari per a cada *place* p on, el marcatge inicial de p' ve donat per $M'_0(p') = K(p) - M_0(p)$.

Pas 2: Entre cada transició t i alguns *places* p' complementaris, dibuixar nous arcs (t, p') o (p', t) on, $w(t, p') = w(t, p)$ i $w(p', t) = w(p, t)$ tal que, la suma de *tokens* a un *place* p i el seu *place* p' complementari siguin igual a la seva capacitat $K(p)$ per a cada *place* p, abans i després de disparar la transició t.

□

Sigui (N, M_0) una xarxa de capacitat finita pura a on, la regla de transició estricta és aplicada. Sigui (N', M'_0) la xarxa obtinguda de (N, M_0) a l'aplicar la transformació del *place* complementari a on, la regla de transició dèbil és aplicada. Llavors, les dues xarxes (N, M_0) i (N', M'_0) són equivalents en el sentit en què, ambdues tenen el mateix conjunt de possibles seqüències de disparaments.

A.2 Exemples

A continuació, es mostren diversos exemples de PNs que mostren els diversos conceptes bàsics explicats i modelen diversos tipus de sistemes.

A.2.1 Màquina d'estats finita

Les màquines d'estat finites o, els seus diagrames d'estats, poden equivalentment ser representats com a subclasses de PNs.

Notar que cada transició d'aquesta xarxa té exactament un *arc d'entrada* (*incoming arc*) i exactament un *arc de sortida* (*outgoing arc*). La subclasse de les PNs amb aquesta propietat es coneix com *màquines d'estats*.

L'estructura d'un *place p* amb dos (o més) *transicions de sortida* (*outgoing transitions*) és referida com un *conflicte*, una *decisió* o una *selecció*, depenent de la seva aplicació (a l'exemple, el *place* "*0 c€*" i les transicions "*Deposit 5 c€*" i "*Deposit 10 c€*"). Les màquines d'estat permeten la representació de decisions però no d'activitats paral·leles.

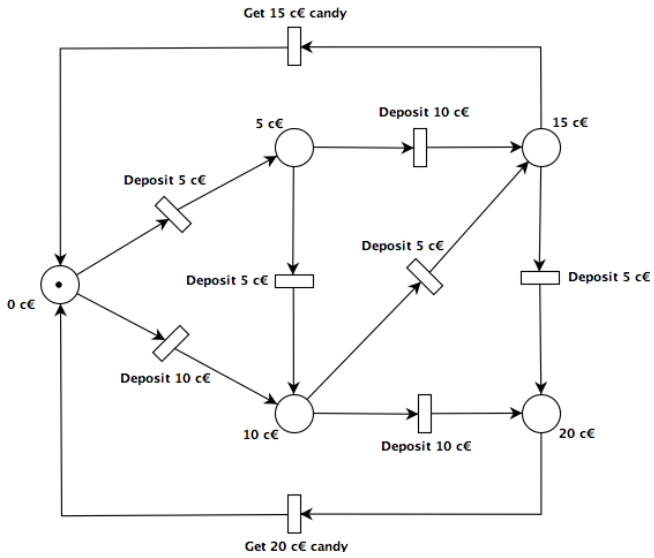


Figura A.1: Una PN (màquina d'estats) representant una màquina de venda de begudes, sense transicions de retorn de monedes.

A.2.2 Activitats paral·leles

Activitats paral·leles o concurrència es poden expressar fàcilment mitjançant PNs. En general, dues transicions s'anomenen *concurrents* si són casualment independents, és a dir, una transició es pot disparar abans o després que l'altre en paral·lel (a l'exemple, les transicions t_2 i t_3).

La *concurrència* es pot considerar com una relació binària (denotada per *co* al conjunt d'events $A = \{e_1, e_2, \ldots\}$) la qual és:

(1) *reflexiva* (e_i *co* e_j) i;

(2) *simètrica* (e_i *co* $e_j \Rightarrow e_j$ *co* e_i);

(3) però, *no transitiva* (e_i *co* e_j i e_j *co* $e_k \not\Rightarrow e_i$ *co* e_k).

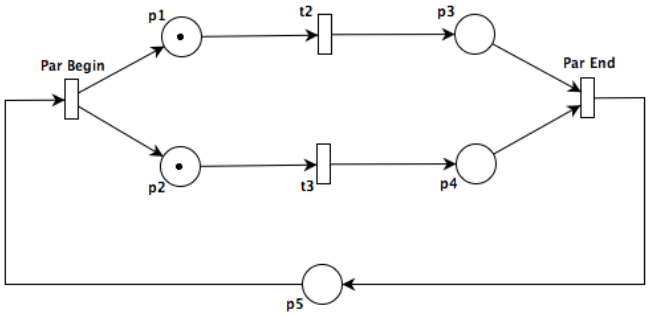


Figura A.2: Una PN (graf marcat) representant activitats paral·leles deterministes.

Notar que cada *place* d'aquesta xarxa té exactament un arc d'entrada i exactament un arc de sortida. La subclasse de les PNs amb aquesta propietat es coneix com *graf marcat* (*marked graphs*).

Dos events estan en *conflicte* si un dels dos pot ocórrer per no ambdós i, són *concurrents* si tots dos poden ocórrer en qualsevol ordre sense conflictes. Una situació en què tant conflicte com concurrència poden ocórrer s'anomena *confusió*.

A.2.3 Flux de dades de computacions

Les PNs poden ser usades no només per a representar el flux de control sinó també el flux de dades. La següent xarxa mostra un flux de dades.

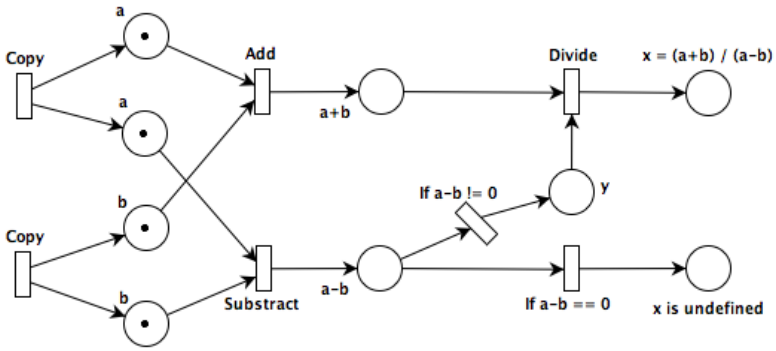


Figura A.3: Una PN mostrant el flux de dades de la computació de $x = (a+b) / (a-b)$.

Un *computador de flux de dades* és aquell en el qual, les instruccions estan disponibles per a executar-se amb l'arribada dels seus operands i poden ser executades concurrentment.

A.2.4 Protocols de comunicació

Els protocols de comunicació són una altra àrea on les PNs poden ser usades per a representar i especificar característiques específiques essencials d'un sistema. Les propietats de *vivacitat* i *seguretat* d'una PN són freqüentment usades com a criteri de correctesa en protocols de comunicació.

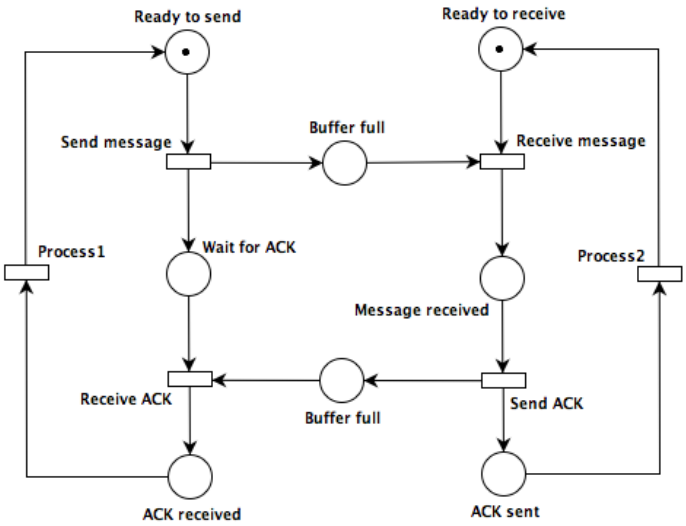


Figura A.4: Un model simplificat d'un protocol de comunicació.

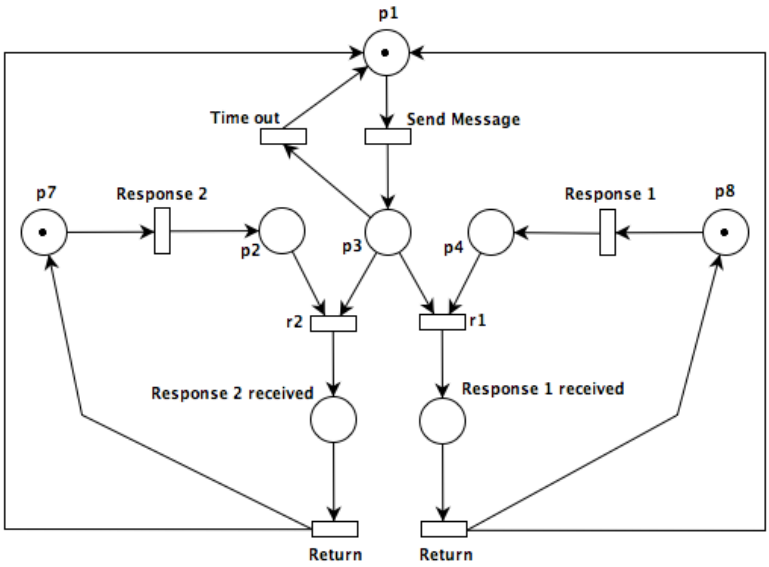


Figura A.5: Una PN representant un procés d'espera no determinista.

A.2.5 Control de sincronització

En un multiprocessador o sistema de processament distribuït, els recursos i la informació són compartits per diversos processadors. Aquesta compartició ha de ser controlada o sincronitzada per tal d'assegurar el correcte comportament de tot el sistema. Les PNs han estat emprades per a modelar una varietat de mecanismes de sincronització incloent, els problemes d'exclusió mútua, lectura/escriptura i productor/consumidor.

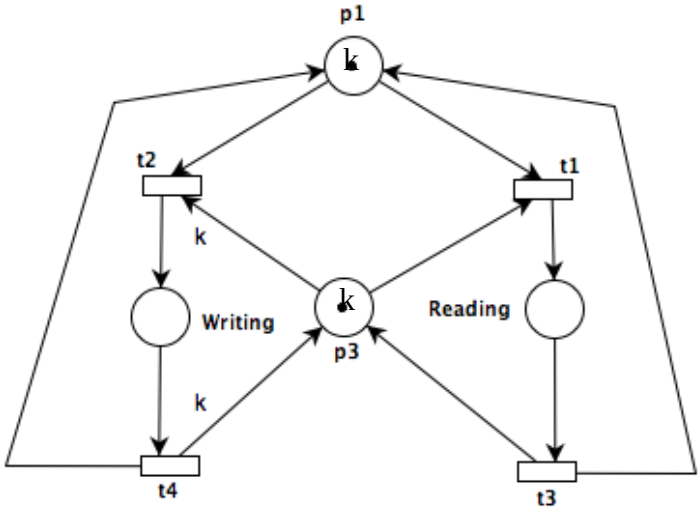


Figura A.6: Una PN representant un sistema de lectura/escriptura.

A.2.6 Sistema productor/consumidor amb prioritat

La següent xarxa representa un sistema productor/consumidor amb prioritat, és a dir, el "*Consumer A*" té prioritat sobre el "*Consumer B*" en el sentit en què *A* pot consumir mentre que el "*Buffer A*" tingui ítems (*tokens*) però, *B* pot consumir només si el "*Buffer A*" és buit i el "*Buffer B*" té ítems (*tokens*).

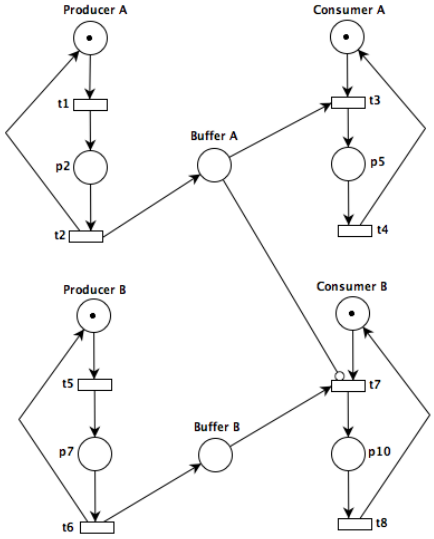


Figura A.7: Una PN estesa representant un sistema productor/consumidor amb prioritat.

Per a modelar aquest sistema s'ha d'introduir un nou tipus d'arc, anomenat *arc inhibidor* (*inhibitor arc*). Un arc inhibidor connecta un *place* a una transició i inhabilita la transició quan el *place* d'entrada té *tokens* i habilita la transició quan el *place* d'entrada no té *tokens* i la resta de *places* (normals) d'entrada tenen, com a mínim, el nombre de *tokens* corresponents al pes de l'arc. Cap *token* es mou a través de l'arc inhibidor quan la transició es dispara. La classe de les PNs amb arcs inhibidors s'anomena *PN estesa* (*extended Petri nets*). La introducció d'arcs inhibidors afegeix l'habilitat de testejar *zeros* (és a dir, l'absència de *tokens* a un *place*) i incrementa el poder de modelatge de les PNs al nivell de les *màquines de Turing*.

A.2.7 Llenguatges formals

Quan les transicions d'una PN estan etiquetades amb un conjunt de no necessàriament símbols diferents, una seqüència de disparaments de transicions genera un *string* de símbols. El conjunt de *strings* generat per les possibles seqüències de disparaments, defineix un llenguatge formal anomenat *llenguatge PN* (*Petri-net language*). Com què cada màquina d'estats finita es pot modelar amb una PN, cada llenguatge regular és un llenguatge PN. Els llenguatges PN són llenguatges sensibles al context.

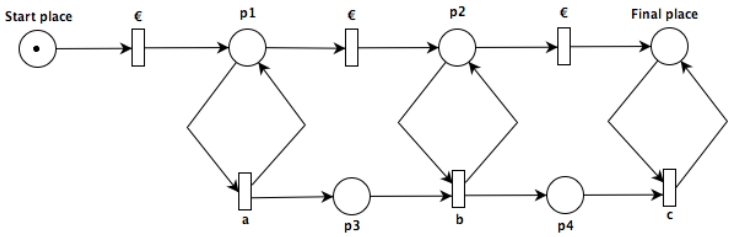


Figura A.8: Una PN etiquetada que genera el llenguatge sensible al context $L(M) = \{a^n\, b^n\, c^n \mid n \geq 0\}$.

A.2.8 Sistemes multiprocessador

A continuació mostrem una xarxa que modela un sistema multiprocessador amb 5 processadors, 3 memòries comunes i dos busos.

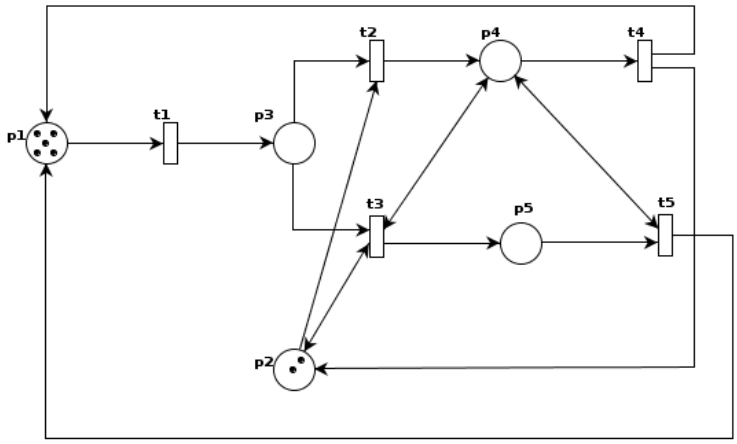


Figura A.9: Una PN modelant un sistema multiprocessador on,
els *tokens* a p_1 representen processadors actius, p_2 els busos disponibles,
p_3, p_4 i p_5 els processadors esperant tenir accés, encuats a memòries comunes, respectivament.

A.3 Altres propietats

A.3.1 Fita (Boundedness)

Una PN es diu *k-fitada* o, simplement, *fitada* si el nombre de *tokens* a cada *place* no excedeix un nombre finit *k* per a qualsevol marcatge abastable des de M_0, és a dir, $M(p) \leq k$ per a cada *place p* i per a cada marcatge $M \in R(M_0)$. Una PN (N, M_0), es diu que és *segura* (*safe*) si és 1-fitada.

A.3.2 Vivacitat (Liveness)

El concepte de *vivacitat* fa referència a la completa absència de punts morts als sistemes operatius. Una PN (N, M_0) es diu que és *viva* (o, equivalentment, M_0 és un *marcatge viu* per a *N*) si, sense importar quin marcatge s'ha assolit des de M_0, és possible disparar qualsevol transició de la xarxa progressant per alguna seqüència de disparament. Això significa que una PN viva garanteix operacions lliures de punts morts, sense importar quina seqüència de disparament és escollida.

La vivacitat és una propietat ideal per a molts sistemes. Per contra, és no pràctica i massa costosa per a ser verificada en molts sistemes com el sistema operatiu d'un gran computador. Per tant, es relaxa la condició de vivacitat i es defineixen els següents diferents nivells de vivacitat.

Una transició *t* en un PN (N, M_0) es diu que és:

(0) *L0-viva* o *morta* (*dead*) si *t* mai es pot disparar per a cap seqüència de disparament a $L(M_0)$.

(1) *L1-viva* (*potencialment disparable*) si *t* mai es pot disparar com a mínim en alguna seqüència de disparament a $L(M_0)$.

(2) *L2-viva* si, donat un enter positiu *k*, *t* es pot disparar com a mínim *k* vegades en alguna seqüència de disparament a $L(M_0)$.

(3) *L3-viva* si *t* apareix infinitament, freqüentment en alguna seqüència de disparament a $L(M_0)$.

(4) *L4-viva* o *viva* (*live*) si *t* és *L1-viva* per a tot marcatge *M* a $R(M_0)$.

Una PN (N, M_0) es diu que és *Lk-viva* si cada transició a la xarxa és *Lk-viva*, $k = 0, 1, 2, 3, 4$. Així, tenim que:

$$L4\text{-vivacitat} \Rightarrow L3\text{-vivacitat} \Rightarrow L2\text{-vivacitat} \Rightarrow L1\text{-vivacitat}.$$

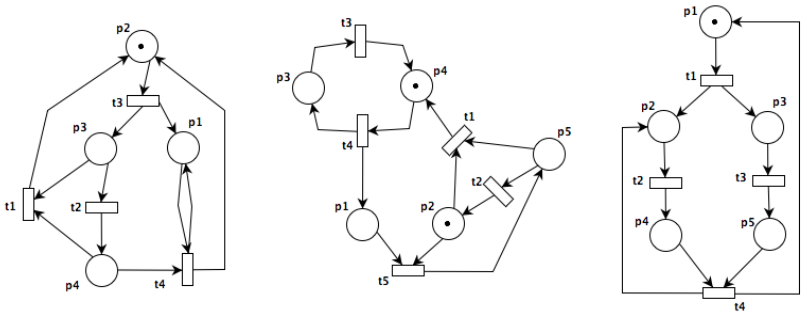
Una transició és *estrictament Lk-viva* si és *Lk-viva* però no $L(k+1)$*-viva*, $k = 0, 1, 2, 3$.

A.3.3 Reversibilitat i Estat Base (*Reversibility and Home State*)

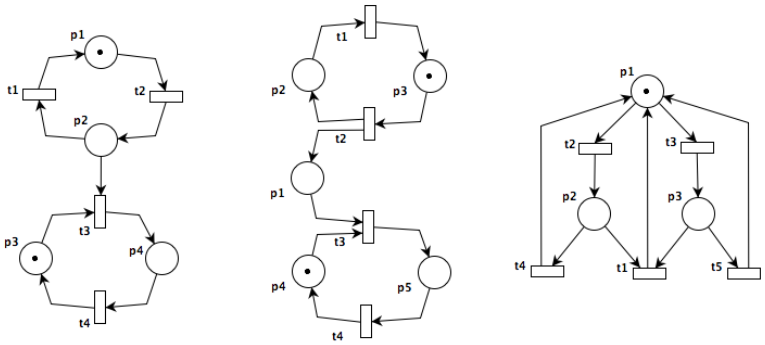
Una PN (N, M_0) es diu que és *reversible* si, per a cada marcatge *M* a $R(M_0)$, M_0 és abastable des de *M*. Per tant, a una xarxa reversible sempre es pot tornar a l'estat o marcatge inicial. Un marcatge *M'* es diu que és un *estat base* si, per a cada marcatge *M* a $R(M_0)$, *M'* és abastable des de *M*.

Notar que, les tres propietats anteriors (fita, vivacitat i, reversibilitat) són independents les unes de les altres.

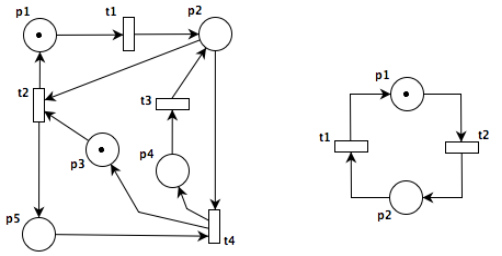
Exemple A.1: Exemples de PNs amb totes les possibles combinacions de les propietats de fita, vivacitat i reversibilitat. Es denoten com: B (fita), $\overline{B}$ (no fita), L (viva), $\overline{L}$ (no viva), R (reversible) i, $\overline{R}$ (no reversible).



$\overline{B}\ \overline{L}\ \overline{R}$ (a) $\overline{B}\ \overline{L}\ R$ (b) $\overline{B}\ L\ \overline{R}$ (c)



$B\ \overline{L}\ \overline{R}$ (d) $\overline{B}\ L\ R$ (e) $B\ \overline{L}\ R$ (f)



$B\ L\ \overline{R}$ (g) $B\ L\ R$ (h)

Figura A.10: PNs i les seves propietats de fita, vivacitat i, reversibilitat.

A.3.4 Cobertura (*Coverability*)

Un marcatge M en una PN (N, M_0) es diu que és cobrible si, existeix un marcatge M' a $R(M_0)$ tal que, $M'(p) \geq M(p)$ per a cada p de la xarxa. La *cobertura* està estretament relacionada amb la *L1-vivacitat* (potencial disparabilitat). Sigui M el marcatge mínim necessari per habilitar la transició t. Llavors, t és morta (no *L1-viva*) si i, només si, M és no cobrible. És a dir, t és *L1-viva* si i, només si, M és cobrible.

A.3.5 Persistència (*Persistence*)

Una PN (N, M_0) es diu que és *persistent* si, per a dues transicions disponibles qualssevol, el disparament d'una transició no inhabilita l'altre. Una transició a una xarxa persistent, una vegada estigui disponible, romandrà disponible fins a que sigui disparada. Notar que tots els grafs marcats són persistents però, no totes les xarxes persistents són grafs marcats.

A.3.6 Distància Síncrona (*Sincronic Distance*)

La distància síncrona és una mètrica estretament relacionada al grau de dependència mútua entre dos events a un sistema de condicions/events. Es defineix la *distància síncrona* entre dues transicions t_1 i t_2 a una PN (N, M_0) com:

$$d_{12} = \max_{\sigma} \left| \overline{\sigma}(t_1) - \overline{\sigma}(t_2) \right| \qquad (1)$$

on, σ és una seqüència de disparament començant a qualsevol marcatge M a $R(M_0)$ i, $\overline{\sigma}(t_i)$ és el nombre de vegades que la transició t_i, i = 1, 2, dispara a σ.

La distància síncrona representa una mètrica ben definida per a xarxes de condicions/events i grafs marcats.

A.3.7 Equitat (*Fairness*)

Aquí es presenten dos conceptes bàsics d'*equitat* d'entre els molts que han estat proposats: *equitat de fita* (*bounded-fairness*) i *equitat incondicional* (*unconditional (global) fairness*).

Dues transicions t_1 i t_2 es diu que estan amb una relació de *fita justa* o (*B-fair*) si, el nombre màxim de vegades que una es pot disparar mentre que l'altre no es dispara és fitat. Una PN (N, M_0) es diu que és una xarxa *B-fair* si, cada parell de transicions de la xarxa estan amb una relació *B-fair*.

Una seqüència de disparament σ es diu que és *incondicional (globalment) justa* si, és finita o cada transició de la xarxa apareix amb freqüència infinita a σ. Una PN (N, M_0) es diu que és *incondicionalment justa* si, cada seqüència de disparament σ de M a $R(M_0)$ és incondicionalment justa.

Tota xarxa *B-fair* és incondicionalment justa i, tota xarxa incondicionalment justa fitada és *B-fair*.

A.4 Mètodes d'anàlisi

Els mètodes d'anàlisi de les PNs es poden classificar segons els següents tres grups:

(1) el mètode de l'arbre de cobertura;

(2) l'aproximació per matriu/equació;

(3) i, les tècniques de reducció o descomposició.

A continuació es detalla el segon d'aquests.

A.4.1 Matriu d'incidència i equació d'estat

El comportament dinàmic de molts sistemes estudiats a l'enginyeria pot ser descrit mitjançant equacions diferencials o equacions algebraiques.

Matriu d'incidència

Per a una PN *N* amb *n* transicions i *m places*, la *matriu d'incidència A* = [a_{ij}] és una matriu *n* x *m* d'enters i la seva entrada típica ve donada per:

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} \tag{2}$$

on, a_{ij}^{+}= *w*(*i*, *j*) és el pes de l'arc de la transició *i* al seu *place* de sortida *j* i, a_{ij}^{-}= *w*(*j*, *i*) és el pes de l'arc a la transició *i* del seu *place* d'entrada *j*.

a_{ij}^{-}, a_{ij}^{+}, a_{ij} representen, respectivament, el nombre de *tokens* eliminats, afegits i canviats al *place j* quan es dispara la transició *i*. Una transició *i* està disponible per a un marcatge *M* si i, només si:

$$a_{ij}^{-} \le M(j), \quad j = 1, 2 \cdots m. \tag{3}$$

Equació d'estat

Un marcatge M_k s'escriu com un vector *m* x 1. La *j*-èssima entrada de M_k denota el nombre de *tokens* al *place j* immediatament després del *k*-èssim disparament en alguna seqüència de disparament. El *k*-èssim disparament o *vector de control* u_k és un vector *n* x 1 d'entrada *n*-1 0's i un 1 a la *i*-èssima posició indicant que, la transició *i* es dispara al *k*-èssim disparament. I, per tant, podem escriure la següent equació d'estat per a una PN:

$$M_k = M_0 + A^T u_k, \quad k = 1, 2, \cdots. \tag{4}$$

Condició necessària d'abastabilitat

Considerem que, un marcatge destí M_d és abastable des de M_0 per una seqüència de disparament { u_1, u_2, … u_d}. Així, escrivint l'equació d'estat (4) i sumant per a *i* = 1, 2, … d obtenim que:

$$M_d = M_0 + A^T \sum_{k=1}^{d} u_k \tag{5}$$

que pot ser reescrit com:

$$A^T x = \Delta M \tag{6}$$

on, $\Delta M = M_d - M_0$ i $x = \sum_{k=1}^{d} u_k$. Aquí, *x* és un vector *n* x 1 d'enters no negatius i, s'anomena *vector comptador de disparaments* (*firing count vector*). La *i*-èssima entrada de *x* denota el nombre de vegades que la transició *i* s'ha de disparar per a transformar M_0 en M_d. El conjunt d'equacions lineals algebraiques de (6) té solució *x* si i, només si, Δ*M* és ortogonal a cada solució *y* del seu sistema homogeni,

$$\Delta y = 0. \tag{7}$$

Sigui r el rang de A i, particionem A de la següent manera:

$$A = \begin{matrix} \underbrace{m-r} & \underbrace{r} \\ \end{matrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow n-r \end{matrix} \tag{8}$$

on, A_{12} és una matriu quadrada no singular d'ordre r. Un conjunt de (m-r) solucions y linealment independents a (7) es pot donar com les (m-r) files de la següent matriu B_f de (m-r) x m:

$$B_f = \left[I_\mu : -A_{11}^T \left(A_{12}^T \right)^{-1} \right] \tag{9}$$

on, I_μ és la matriu identitat d'ordre $\mu = m - r$. Notar que $AB_f^T = 0$. És a dir, el vector espai format pels vectors fila de A és ortogonal al vector espai format pels vectors fila de B_f. La matriu B_f correspon a la matriu circuit fonamental al cas del graf marcat. Ara, la condició de què ΔM és ortogonal a cada solució de $Ay = 0$ és equivalent a la següent condició:

$$B_f \Delta M = 0. \tag{10}$$

Així, si M_d és abastable des de M_0 llavors, el corresponent vector comptador de disparaments x ha d'existir i (10) s'ha d'acomplir. Per tant, tenim la següent condició necessària d'abastabilitat a una PN sense restriccions: si M_d és abastable des de M_0 a una PN (N, M_0) llavors, $B_f \Delta M = 0$ on, $\Delta M = M_d - M_0$ i B_f ve donat per (9).

La seva contraposició porta a la següent condició suficient de no abastabilitat: a una PN (N, M_0), un marcatge M_d no és abastable des de M_0 ($\neq M_d$) si, la seva diferència és una combinació lineal dels vectors fila de B_f, és a dir,

$$\Delta M = B_f z \tag{11}$$

on, z és un vector μ x 1 no nul.

Exemple 3.2:

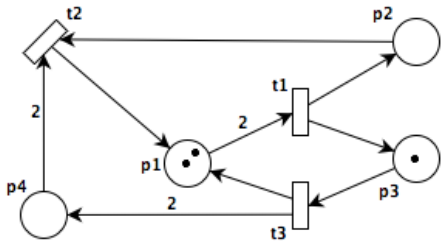


Figura 3.3: PN Exemple 3.2.

L'equació d'estat (4) on, la transició t_3 es dispara per assolir el marcatge $M_1 = (3\ 0\ 0\ 2)^T$ des de $M_0 = (2\ 0\ 1\ 0)^T$ és:

$$\begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

La matriu d'incidència A és de rang 2 i, es pot particionar en la forma de (8) on,

$$A_{11} = \begin{bmatrix} -2 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{i} \quad A_{12} = \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix}.$$

Llavors, la matriu B_f es pot trobar mitjançant (9):

$$B_f = \begin{bmatrix} 1 & 0 & 2 & 1/2 \\ 0 & 1 & -1 & -1/2 \end{bmatrix}$$

Es pot verificar que $B_f \Delta M = 0$ per a $\Delta M = M_1 - M_0 = (1\ 0\ -1\ 2)^T$. □

Una solució entera x de l'equació homogènia ($\Delta M = 0$ a (6))

$$A^T x = 0 \qquad (12)$$

s'anomena *T-invariant* i, una solució entera y de l'equació homogènia transposada $Ay = 0$ s'anomena *S-invariant*.

DEO GRATIAS

"*Aquí feneix lo llibre del valerós e estrenu cavaller Tirant lo Blanc, príncep e Cèsar de l'Imperi grec de Contestinoble, lo qual fon traduït d'anglès en llengua portuguesa, e aprés en vulgar llengua valenciana, per lo magnífic e virtuós cavaller Mossèn Joanot Martorell, lo qual, per mort sua, no en pogué acabar de traduir sinó les tres parts. La quarta part, que és la fi del llibre, és estada traduïda, a pregàries de la noble senyora Dona Isabel de Lloris, per lo magnífic cavaller Mossèn Martí Joan de Galba; e si defalt hi serà trobat, vol sia atribuït a la sua ignorància; al qual Nostre Senyor Jesucrist, per la sua immensa bondat, vulla donar, en premi de sos treballs, la glòria de paradís. E protesta que si en lo dit llibre haurà posades algunes coses que no sien catòliques, que no les vol haver dites, ans les remet a correcció de la santa catòlica Església.*"

Tirant lo Blanc

Joanot Martorell