CE653 – Asynchronous Circuit Design

Instructor: C. Sotiriou

http://inf-server.inf.uth.gr/courses/CE653/

Contents

PTnet Definitions

- Definition [**PTnet**]:
  - A Place Transition net (or PTnet) is a triple $N = (S, T, F)$, where
    - S is the set of *places*,
    - T is the set of *transitions* ($S \cap T = \varnothing$)
    - $F \subseteq (S \times T) \cup (T \times S)$ is the *flow relation*
- Elements of $S \cup T$ are called nodes
- *pre-set* $\bullet x$ of x in $(S \cup T)$ is given by
  - $\bullet x = \{y \text{ in } S \cup T \mid (y, x) \text{ in } F\}$
- *post-set* $x\bullet$ of x in $(S \cup T)$ is given by
  - $x\bullet = \{y \text{ in } S \cup T \mid (x, y) \text{ in } F\}$
- A *Marking M* is a function $M: S \rightarrow \mathbb{N}$
- A *Marked PTnet* <*N, M0*> is a PTnet with an initial marking

PTnet Definitions

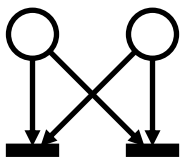
- A transition t in T is *enabled* at marking M iff:
  - For all p in $\bullet t$: $M(p) > 0$
- When t *fires* at M a new marking M' is produced:
  - $M'(p) = M(p) - \mathbb{F}(p, t) + \mathbb{F}(t, p)$ ($\mathbb{F}$ is characteristic function of F)
  - M[t>M' denotes that M' is reachable from M by the occurrence of transition t
- The set of markings reachable from the initial marking M0 by the occurrence of a sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ is denoted by *R(N, M0)*

PTnet Definitions – Net Subclasses

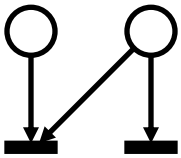
- Definition [**S-graph – State Machine**]:
  - A net N = (S, T, F) is called an *S-graph or State Machine,* iff each transition has exactly one input place and one output place
  - for all t in T: |•t| = 1 = |t•|
- Definition [**T-graph – Signal Transition Graph**]:
  - A net N = (S, T, F) is called an *T-graph or STG,* iff each place has exactly one input transition and one output transition
  - for all s in S: |•s| = 1 = |s•|

PTnet Definitions – Net Subclasses

- Definition [**Free-Choice**]:
  - N is called *free-choice* iff all p in S such that |p•|>1, •(p•) = p

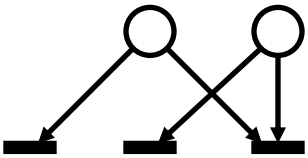


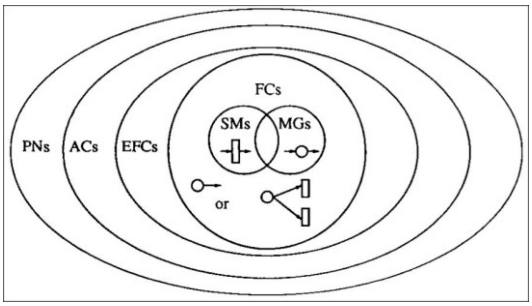
- Definition [**Asymmetric-Choice**]:
  - N is called *asymmetric-choice* (or simple net) iff for every two places p1, p2, p1•∩ p2• ≠ ∅ ➔ p1• ≤ p2• or p1• ≥ p2•



PTnet Definitions – Net Subclasses

- Definition [**General PTnet**]:
  - N is called *general PTnet* iff for every two places p1, p2, p1•∩ p2• ≠ ∅ ➔ p1• ≦ p2• or p1• ≧ p2•





PTnet Definitions - Subnets

- Definition [**S-Components and T-Components**]:
  - N' = (S', T', F') is a *subnet* of N = (S, T, F) iff S' ≤ S, T' ≤ T and F' = F ∩ ((S' x T') U (T' x S'))
  - N' is an ***S-component*** (***T-component***) of N iff N' is a strongly connected S-graph (T-graph) and| T' = •S'∩S'• (S' = •T'∩T'•).
  - N' is generated by a set X' ≤ S ∩ T iff:

    S' = (X' ∩ S) U •(X' ∩ T) U (X' ∩ T)•

    T' = (X' ∩ T) U •(X' ∩ S) U (X' ∩ S)•

PTnet Definitions – Behavioural Properties

- Definition [**Bounded Net**]:
  - A *marked net (N, M0)* is bounded iff:
  - for all p in S, there exists k in $\mathbb{N}$, s.t. for all markings M reachable from M0: M(p) ≤ k
- Definition [**Structurally Bounded Net**]:
  - A net N is *structurally bounded* iff it is bounded for any initial marking M0.
- Definition [**Liveness**]:
  - A transition t in T is *live* in (N, M0) iff:
    - For all M in R(N, M0) there exists M' in R(N, M): M' enables t.
  - The marked net (N, M0) is *live* iff all t in T are live.
  - N is *structurally live* iff there exists initial marking M0 such that (N, M0) is live

PTnet Definitions – Graph Properties

- Definition [**Path**]:
  - A *path* of a net N=(S, T, F) is an alternating sequence $\pi = (x_0 f_0 x_1 \ldots f_{r-1} x_r)$ of elements X = S ∪ T such that: for all I, $0 \le i \le r - 1$: $f_i = (x_i, x_{i+1})$ in F
  - A path is *elementary* iff all x_i are distinct except x_0 and x_r
  - A *circuit* is a path such that $x_0 = x_r$
  - A circuit is *elementary* iff it is elementary as a path
- Definition[**Alternating Circuit**]
  - Let N = (S, T, F) be a net. A circuit Γ of N (not necessarily elementary) is an *alternating circuit* iff for all arcs in Γ of the form (p, t) the equality •t = {p} holds

PTnet Definitions - Subnets

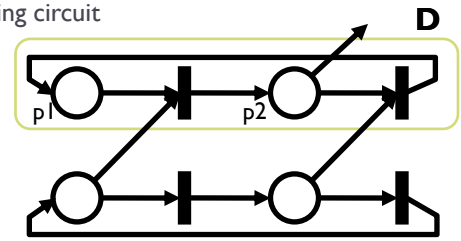
- Definition [**Well-formed Net**]:
  - A Net N is *well-formed* if **there exists a marking** M0 of N such that (N, M0) is a live and bounded system
- Thus, the net is not necessarily live at the current marking…
- Theorem [**S-Components and Well-Formed Nets**]:
  - Well formed Nets are covered by S-Components

PTnet Definitions – Siphons and Traps

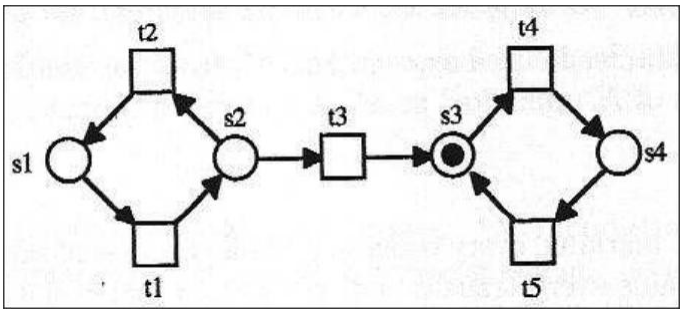
- Definition [**Siphons (Deadlocks) and Traps**]:
  - Let N = (S, T, F) be a net.
  - D ≤ S is a *siphon* (*deadlock*) iff D ≠ ∅ and •D ≤ D•
  - Θ ≤ S is a *trap* iff Θ ≠ ∅ and Θ• ≤ •Θ
  - A siphon (deadlock) or trap is *minimal* iff there exists no deadlock or trap D' such that D' ≤ D
  - A siphon (deadlock) or trap is *strongly-connected* iff the subnet generated by D U •D is strongly connected

PTnet Definitions – Siphons and Traps

- Theorem[**Minimal Siphon for General PTnets**]
  - Let N = (S,T, F) be a net, D ≤ S a siphon of N and N_D the subnet of N generated by D U •D.
  - D is minimal iff there exists a *circuit* Γ in N_D (not necessarily elementary) that passes through all places of D such that for every transition t in Γ either:
  - |•t ∩ D| = 1 (*if net is FC*) or
  - |•t ∩ D| ≥ 2 and the places of (•t ∩ D) belong to an alternating circuit



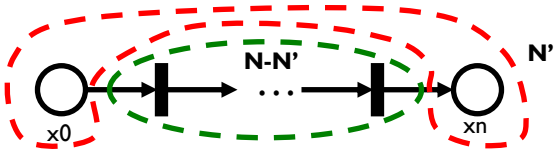
PTnet Definitions – Siphons and Traps



- The set {s1, s2} is a siphon; the set {s3, s4} is a trap

PTnet Definitions – Graph Properties

- Definition [**Handle**]:
  - Let N' be a partial subnet of N.
  - An elementary path $\pi = (x_0 f_0 x_1 \ldots f_{n-1} x_n)$ is a *handle* of N' iff $\pi \cap N' = \{x_0, x_n\}$



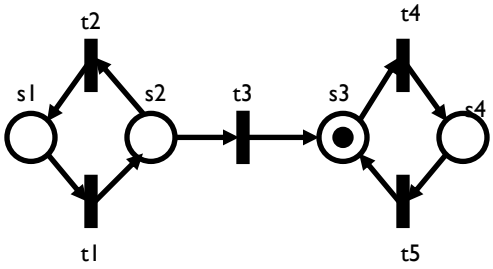
PTnet Examples

PTnet Examples - 1

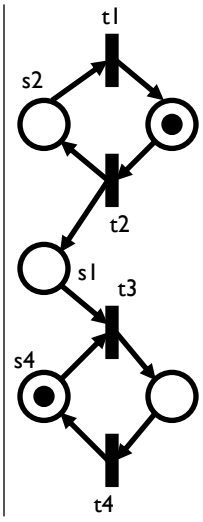
1.

B = 2

2.



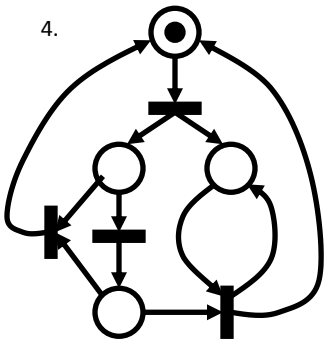
B, L'



B', L

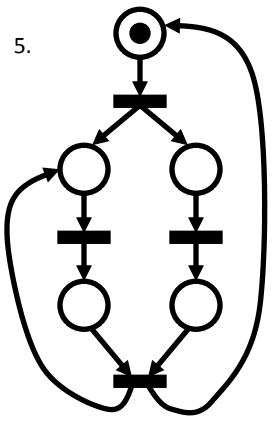
PTnet Examples - 2

4.



B', L'

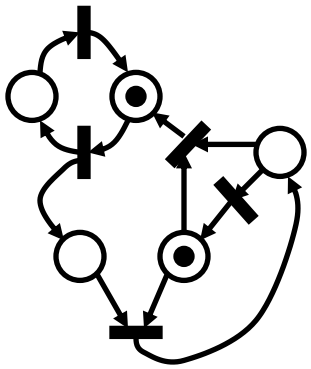
5.



B', L

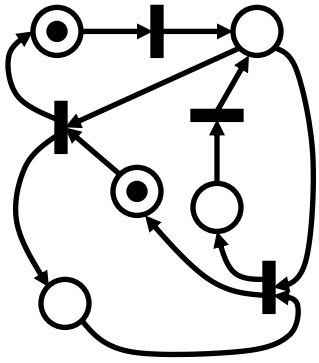
PTnet Examples - 3

6.



B', L'

7.



B, L

Minimal Siphons and S-Components – Esparsa/Kemper Algorithm

Get Handle Algorithm

Algorithm Get_Handle(S, S', F, t, p)

```
// S ∩ S' = ø, p in S, t in S' and t is in •p //
i = 1; Stack = empty;
forall x in S' num(x) = 0;
forall x in S num(x) = -1;

push(Stack, p);
if (handle-DFS(t) == 0)
{
  return NULL;
  printf("No handle exists\n");
}
else
{
  return Stack; // Stack contains Handle //
}
```

Algorithm handle-DFS(v)

```
num(v) = i; i = i + 1;

push(Stack, v);

forall (w in •v)
  if (num(w) == -1) // start node of handle //
  {
     push(Stack, w);
     return 1;
  }
forall (w in •v)
  if (num(w) == 0) // new non-start node //
     if (dfs(w) == 1) return 1;

pop(Stack, v);
return 0;
```

- N is strongly-connected FC-Net with S, S' ≤ P U T
- S ∩ S' = Ø, p in S, t in S', t is in •p, Outputs handle H = (x_0, x_1, …, x_{n-2}, t, p) or 0
- Num(v) values:
  - num(v) == -1 ➔ v in S, start point of handle, num(v) ==0 ➔ v in S', not visited before
  - num(v) > 0 ➔ v has been visited before and either (a) no handle was found or (b) has been reached again. In both cases v cannot belong to the handle

Get Minimal Siphon (Deadlock) Algorithm

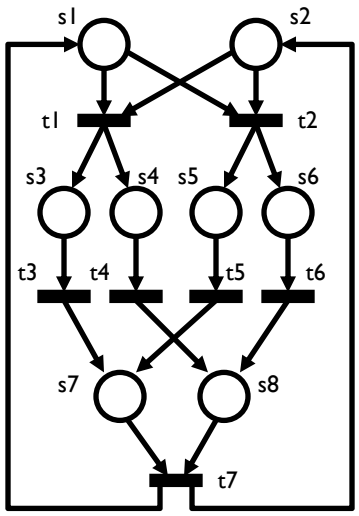
Algorithm Get_Minimal_Deadlock(P,T, F, p, D, Td)

```
Pc = {p}; Tc = 0; // current sets of Places and Transitions

while (there exists p' in Pc, there exists t in •p' and t not in Tc)
{
    H = Get_Handle((Pc U Tc), (P U T)-(Pc U Tc), F, p', t);
    Pc = Pc U (H ∩ P); // discard multiple instances of places
    Tc = Tc U (H ∩ T); // and transitions
}
D = Pc; Td = Tc;

return D, Tc; // return Deadlock places and transitions
```

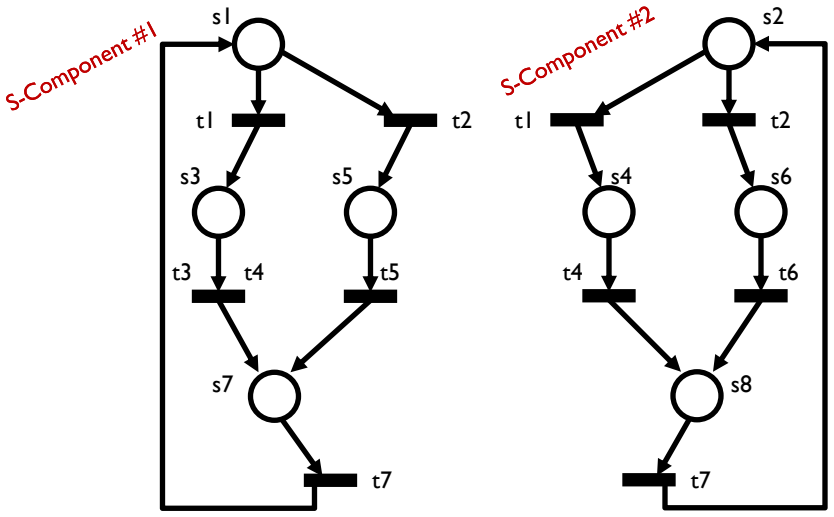
- N = (P,T, F) strongly-connected FC-Net with p in P
- D is minimal deadlock D ≤ P, containing P
- To be an S-Component, minimal deadlock D must satisfy:
  - It is strongly connected
  - For all t in D: |•t∩D| = |t•∩D| = 1

S-Covering - Example

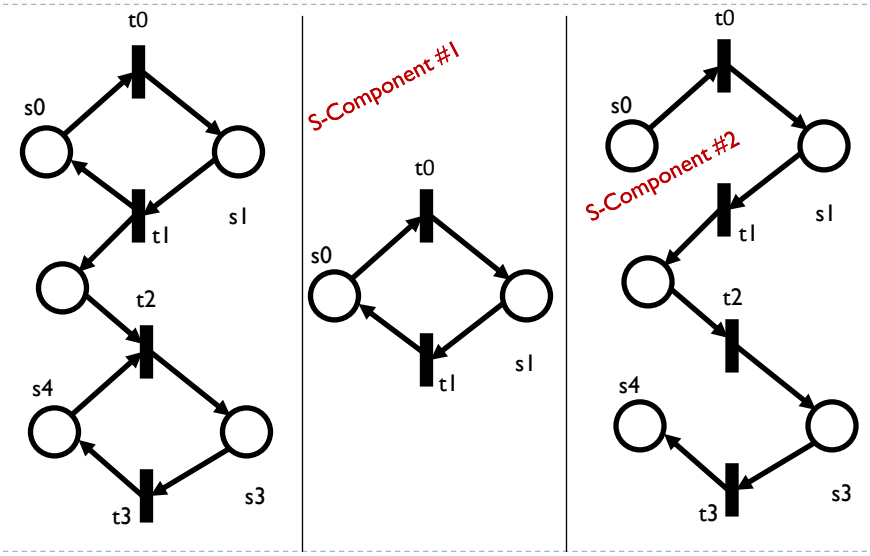
- Original Net:



S-Covering – Example – S-Cover



S-Cover for non Well-formed Net



Minimal Siphons – Cordone et. al (PIPE2) Algorithms

Rules for Siphon Extraction/Minimisation

- Definition **[PTnet Reduction Function RED]**
  - Let G = (P,T, F) be a PTnet and ~P ≤ P.
  - The reduction function RED is defined as follows:
  - ~G = *red*(G, ~P), where the reduced net ~G = (~P, ~T, ~F) is defined by:
    - ~T = {t in T | (•t U t•) ∩ ~P ≠∅}
    - ~F(p, t) = F(p, t), ~F(t, p) = F(t, p), forall p in ~P, t in ~T
- Rule 1 [*place subset reduction*]
  - Let G = (P,T, F) be PTnet and ~P ≤ P
  - Set of siphons of G contained in ~P is the same as siphons of reduced net ~G = *red*(G, ~P)

Rules for Siphon Extraction/Minimisation

- Places with ***empty subset*** and places with ***all net transitions*** as post-set are special cases!
- Rule 2 [*empty subset places*]
  - Let G = (P,T, F) be PTnet and _P ≤ P such that •_P = ∅
  - Then, minimal siphons of G are minimal siphons of ~G = *red*(G, P - _P), **plus the individual places in _P**
- Rule 3 [*all net transitions in post-set places*]
  - Let G = (P,T, F) be a PTnet such that P• = T
  - Then P is a siphon

Rules for Siphon Extraction/Minimisation

- Places in ***post-set of transitions with empty pre-set cannot belong to a siphon*** and can be eliminated
- If all transitions are in post-set of some place, T = P• then P is a siphon (see Rule 3)
- Rule 4 [*trap places elimination*]
  - Let G = (P,T, F) be a PTnet and let _T ≤ T be such that
  - •_T = Ø. Then, if _P = _T•, G has same siphons as
  - ~G = *red*(G, P - _P)
- Rule 5 [*redundant places*]
  - Let G = (P,T, F) be a PTnet and S ≤ P a siphon of G.
  - If there exists _p in S: for all t in _p• either
    (t• ∩ S) > {_p} or
    (•t ∩ S) = Ø, then S – {_p} is also a siphon of G

Rules for Siphon Extraction/Minimisation

- Rule 6 [*Siphon Minimality*]
  - Let G = (P,T, F) be a PTnet and S ≤ P a siphon of G.
  - S is minimal for G, iff all reduced nets:
  - ~Gp = *red*(G, S-{p}) for all p in S do not contain siphons
- Rule 7 [*Siphon Decomposition into Smaller Siphons*]
  - Let G = (P,T, F) be a PTnet and ~P = {p1, p2, …, pn} ≤ P
  - The set of Siphons of G NOT containing ~P is the Union

$$\bigcup_{i=}^{n} S_i$$

where Si is the set of siphons of the reduced net:
~Gi = *red*(G, P – {pi}), *i.e.* containing the n places except i.

Find a Siphon - Algorithm

Algorithm Find_Siphon(G, ~P, P)

```
while (1)
{
    if ((exists p in P ∩ ~P) && (exists t in •p such that t not in P•))
    {
        S = Ø;                                  // Rule 4 //
        return;
    }
    if ((exists p in P - ~P) && (exists t in •p such that t not in P•))
        G = red(G, P - {p});                    // modifies local P and G//
    else
    {
        S = P;
        return;
    }
}
```

- ~P is a set of places (one or more) which should be contained in the siphon
- Elimination based on **Rule 4** (trap places)

Find a Minimal Siphon - Algorithn

Algorithm Find_Min_Siphon(G, ~S, ~P, P)

```
while (exists p in (P - ~P) ∩ ~S such that (•t ∩ ~S) < {p} or
                                           (t• ∩ ~S) = Ø)   // Rule 5 //
    ~S = ~S - {p};
while (1) {
    if (~S < P) G = red(G, ~S); Pnew = P - ~P;
    if (Pnew == Ø)
    {
        S = S~;
        return;                             // found minimal siphon //
    }
    forall (p in Pnew)                      // find smaller siphons of Pnew //
    {
        Gp = red(G, P - {p}); Pnew = Pnew - p;
        Sp = Find_Siphon(Gp, ~P, P);
        if (Sp != Ø)
        {
            ~S = Sp;
            break;                      // found smaller siphon - to outer loop //
        }
    }
}
```

- Computes one minimal siphon S ≤ ~S and containing ~P

Find all Minimal Siphons - Algorithn

Algorithm Find_All_Min_Siphons(G, ~P, P)

```
SS = Ø; // Minimal Siphon Set //
while ((~P == Ø) && (exists p in P such that •p = Ø)) {
    S = {p}; SS = SS U {S}; G = red(G, P - {p});          // Rule 2 //
}
~S = Find_Siphon(G, ~P, P);
if (~S == Ø) return;
S = Find_Min_Siphon(G, ~S, ~P, P);
SS = SS U S;
Pnew = S - ~P; Pold = Ø;
if (Pnew == Ø) return;
forall (p in Pnew) {
    Gp = red(G, P - {p});
    SSp = Find_All_Min_Siphons(Gp, ~P U Pold, P);          // Rule 7 //
    SS = SS U SSp;
    Pnew = Pnew - {p}; Pold = Pold U {p};
}
```

- Find_All_Min_Siphons(G, Ø) returns all minimal siphons of G
- Worst-case complexity is $O(2^n)$
  - Due to Recursive call for Rule 7