CIS 525 Software Development of Parallel and Distributed Systems

TEMPORAL LOGIC

In Petri Nets we can prove system properties, such as deadlock, liveness, after first building a PN model of a system at certain level of abstraction.

For concurrent programs we want to be able to assert properties of processes and relate them to various states during the processes execution (without modeling state changes). Temporal logic provides this capability (Lamport 1980, Manna and Pnueli 1981, Kniiger 1987).

Temporal logic specifications are able to relate properties to state sequences associated with a concurrent program's execution. This gives us a means to asset both safety and liveness properties (two types of correctness properties)

Safety properties – program never does something bad (freedom from deadlock, mutual exclusion)

Liveness properties – assert that program will eventually do something good (example: message sent is eventually received, termination)

Fairness – helps to determine when competing events will occur (weak fairness and strong fairness)

Weak fairness – events that always are ready to be executed will eventually be executed

Strong fairness – events that are infinitely often become ready for execution will eventually be executed.

LOGIC-BASED SPECIFICATION FOR SYSTEMS

Temporal operators = take into account the time-dependent nature of truth values.

$\Box$ Always operator ($\Box P$ always true in any state)
$\Diamond$ Eventually operator ($\Diamond P \equiv \neg (\Box \neg P)$ = P is not always false)
O Next time operator
U Until operator

Examples: (of temporal logic statements)

1> $\Box(P \rightarrow \Diamond Q)$ it is always true that if P is true in the current state, then Q will eventually become true (Q is true either now or in some future state)

(This may correspond to the liveness property that "if a message is sent, it will eventually be received")

2> $\Box(A\rightarrow \bigcirc B)$whenever A is true, B will become true immediately after that

3> $\Box(A\rightarrow\neg B$ $\bigcirc B)$ whenever A is true then B will be false but it will become true immediately thereafter

4> P U Q either P is true in the current state or P will be at least until first time that Q becomes true.

USING TEMPORAL LOGIC AS A SPECIFICATION MODEL:

1> Identify a set of system properties (objective).

2> Formalize the properties using the temporal logic notation. Properties themselves become the model.(serious weakness comparing with PNs)

3> Temporal logic is the most effective in specifying those system properties identified by a system developer or analyst.

WHERE WAS TEMPORAL LOGIC APPLIED

a> Program synthesis as a way to specify formally those properties that drive a particular synthesis procedure.

b> Program analysis – to specify formally those properties to be checked by an analysis procedure.

MODEL CHECKING

Automated Program Analysis Technique that uses temporal logic → checks validity of temporal logic formulas for finite-state concurrent programs (Clatke, Emerson, Sistla 1986).

a> <u>Use a BDD (Binary Decision Diagram)</u> to represent state space symbolically; adopted successfully to handle the analysis of digital circuits with very large state spaces (Burch et al. 1980).

b> <u>Karam and Buhr (1990, 91)</u> → developed a temporal logic-based specification language to identify the behaviors of concurrent programs expressed in the ADA; these behavior specifications are then used in various analysis algorithms (example: deadlock analysis→ automatic theorem prover technique is used)

c> <u>Temporal logic formulas</u> do not give us info about a single only, the complexity of the analysis depends on the type of state sequences being considered:

- <u>linear-time temporal logic</u> – time has linear structure; each state has one successor state
- <u>branching-time temporal logic</u> – time has linear structure; a state may have more than one successor state.

Example: Binary semaphore; Karp; 1984

Goal: To specify the requirements of a semaphore operations P and V. V operation sets to 1 the value of the semaphore variable. P operation sets to 0 the value of the semaphore variable.

$p \in P$: a process p has called a P operation and is waiting for the operation to be completed

$p \in V$: a process p has called a V operation and is waiting for the operation to be completed.

s: value of the semaphore variable (1 or 0)

The following properties must be true if we are to have a correct semaphore implementations:

1> Property 1: A P (safety property) operation must not complete if the semaphore has value 0 (semaphore is "unavailable") and the value never changes. Otherwise, a user of the semaphore might violate mutual exclusion on a resource guarded by the semaphore.

$p \in P$ □ $\Box(s=0) \rightarrow \Box\,(p \in P)$

2> Property 2: only a V operation can set the semaphore value to 1:
$\Box(\text{for all } p,\ p \notin V)$ □$(s=0) \rightarrow \Box(s=0)$

3> Property 3 (liveness property): A V operation must eventually be completed and in doing so set the semaphore value to 1
$p \in V \rightarrow \Diamond(p \notin V$ □ $(s=1)$

4> Property 4 (fairness property): A process that has requested P operation will not be indefinitely blocked if the semaphore value becomes 1 infinitely often. This is a fairness property that is typically satisfied by implementing a semaphore with FIFo queue.

$p \in P$ □$\Diamond(s=1) \rightarrow \Diamond(p \notin P)$

REQUIREMENTS SPECIFICATIONS FOR DISTRIBUTED SOFTWARE

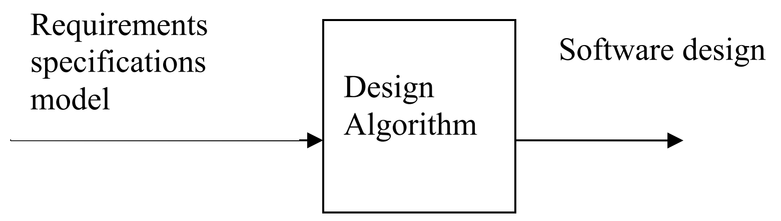
- Properties characterizing the intended behavior of the software system.
- Elements of distributed systems that have to be supported:
    - asynchronous execution (source of improved performance)
    - non deterministic actions (choice)
    - communication (with delay)

Categories of models:

a> Process algebras, like CCS (Lin, 1991, process algebra tools)
CCS = communicating concurrent systems

b> Petri-nets – explicit means to specify parallelism among asynchronous activities; the non deterministic selection of events, and synchronization.

c> Temporal logic – logic-based model; it asserts a set of properties for a system (without explicitly modeling a system)

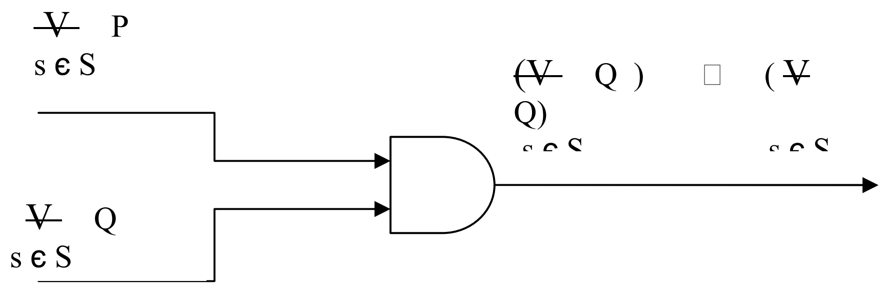


Task partitioning ← 1> identification of processes (cooperative program units)

Task allocation ← 2> allocation of processes to processors

Figure 1

PROBLEM 3.5 PAGE 59, SHOL SHATZ'S BOOK:



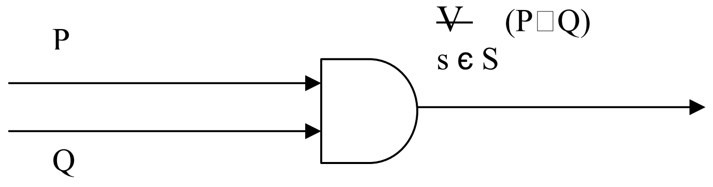


Figure 2

L= eventually P will be true □
Eventually Q will be true
Well, they may not be true at the same time

P= eventually P □ Q will be true (this must be at the same time)

PROBLEM 3.4 PAGE 59, SHOL SHATZ'S BOOK:

a> True
b> Not true
c> True

PROBLEM 3.1 PAGE 57-58; SHOL SHATZ'S BOOK

t1, t2, t3 are concurrent transitions that can execute in any order.

<u>Method 1: to make order of execution strictly defined for instance in order</u>
t1 → t2 → t3

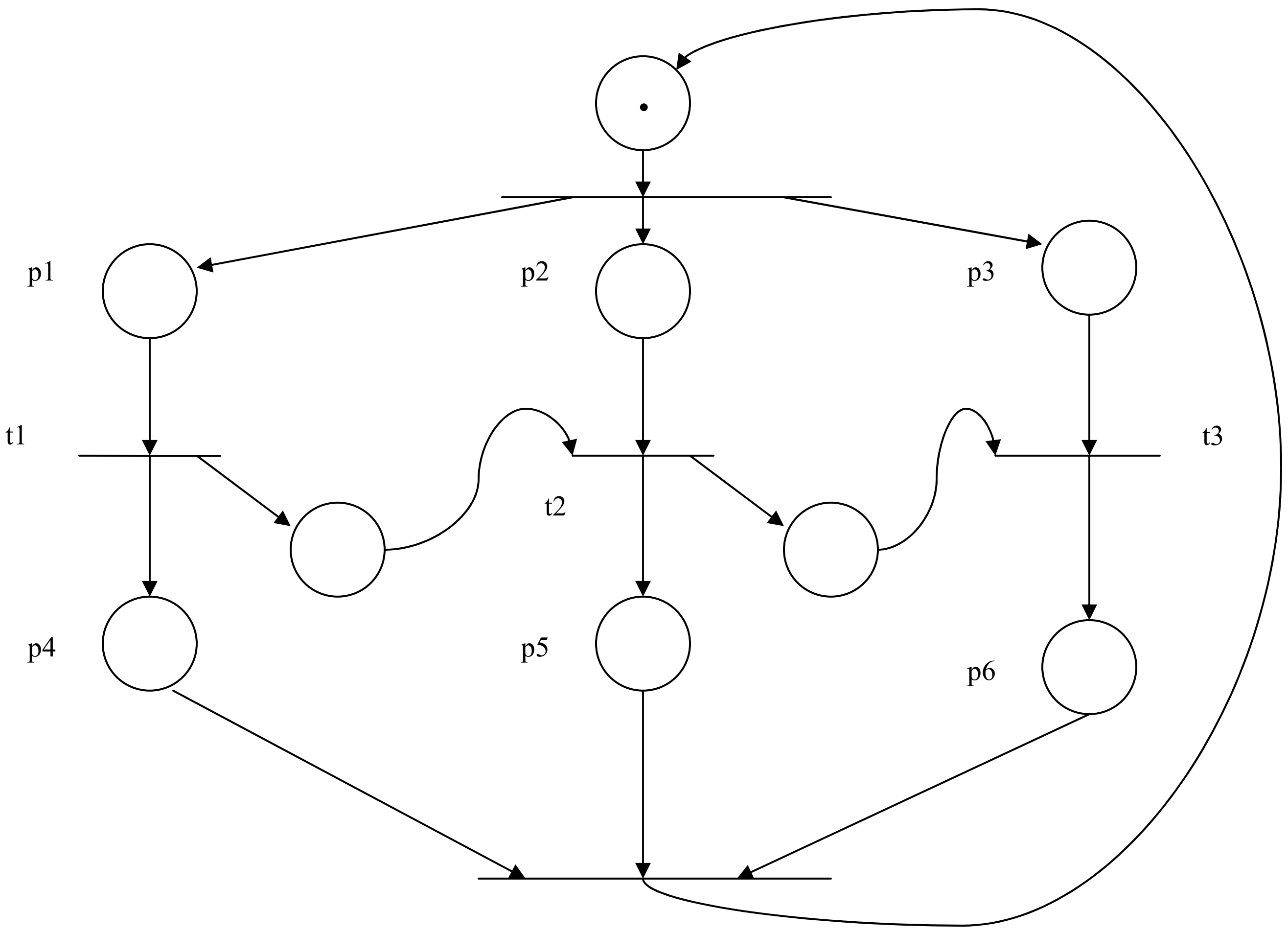


Figure 3

Method 2: with inhibitor arcs

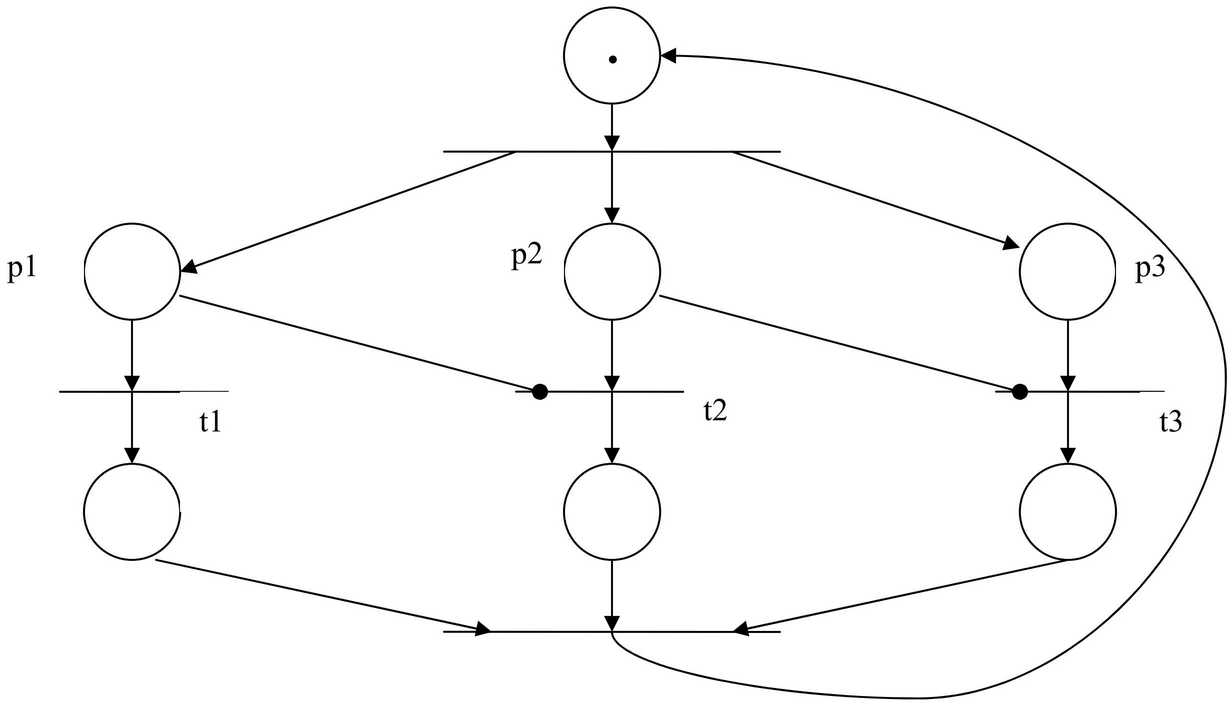
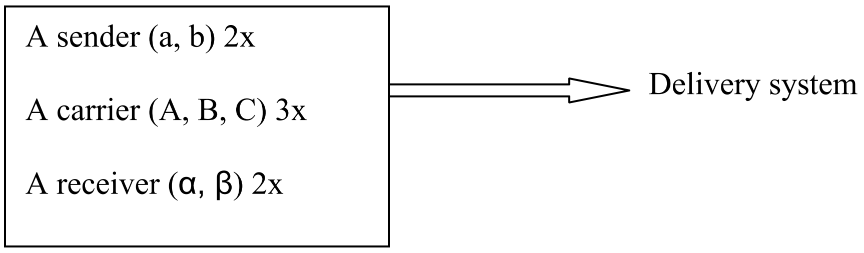


Figure 4

PROBLEM 3.2 PAGE 58 SHOL SHATZ'S BOOK:



<u>Specification requirements:</u>

A sender can hire any number of carriers to send a parcel to any of the receivers:

1> (a, A, α)
2> (a, A, β)
3> (a, B, α)
4> (a, B, β)
5> (a, C, α)
6> (a, C, β)
7> (b, A, α)
8> (b, A, β)
9> (b, B, α)
10> (b, B, β)
11> (b, C, α)
12> (b, C, β)

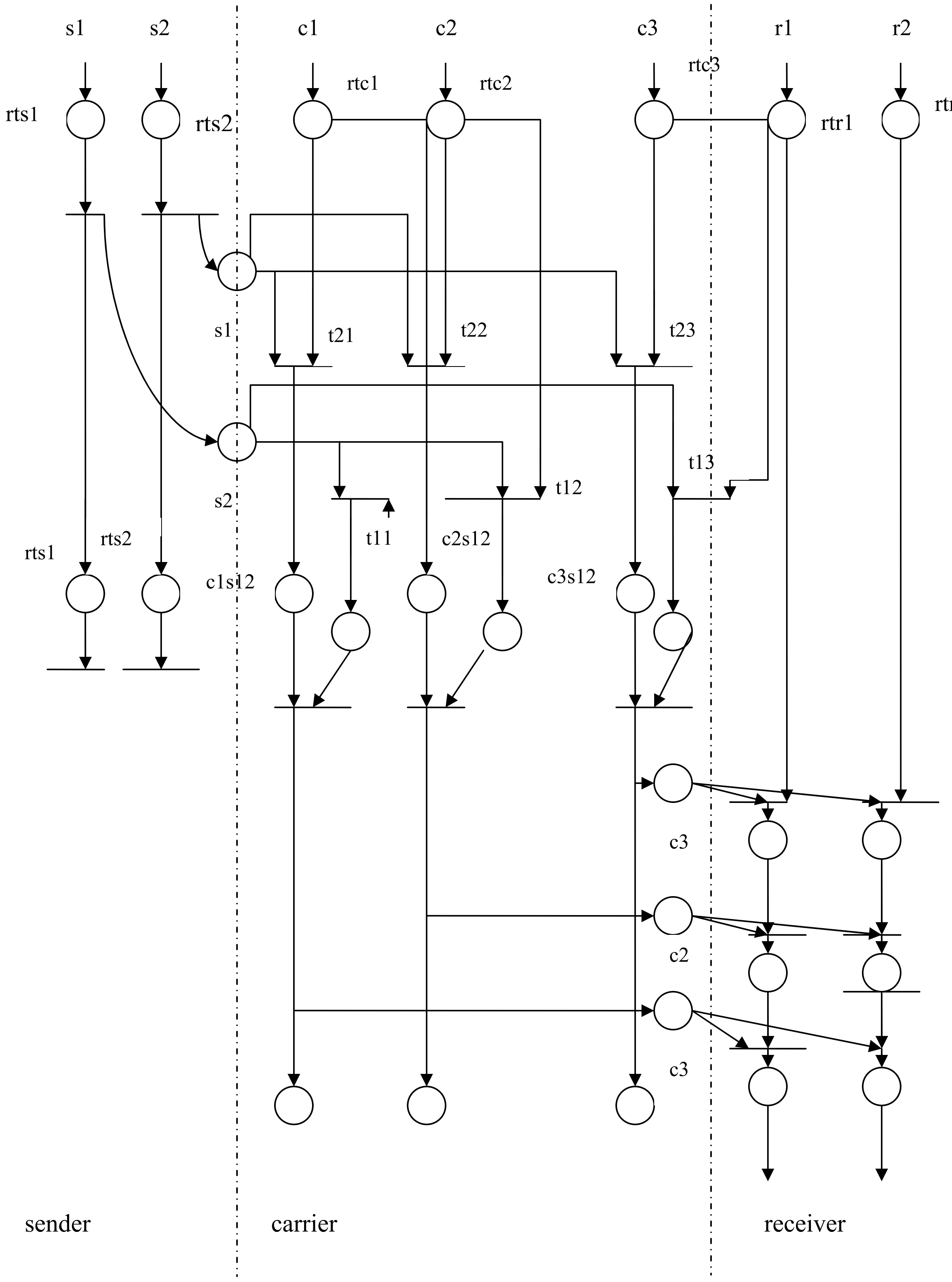

s1
s2
c1
c2
c3
r1
r2
rtc3
rtc1
rtc2
rtr2
rts1
rts2
rtr1
s1
t21
t22
t23
t13
s2
t12
t11
c2s12
rts1
rts2
c1s12
c3s12
c3
c2
c3
sender
carrier
receiver

Figure 5