EE 249 Discussion: Synthesis of Embedded Software using Free-Choice Petri Nets

By :Marco Sgroi, Luciano Lavagno, Alberto Sangiovanni-Vincentelli

Shanna-Shaye Forbes

Software synthesis from a concurrent functional specification is a problem in the design of embedded systems.

What's wrong?

You don't have pure data flow.

If pure data flow you can apply a fully static scheduling technique.(complete behavior is predicted at compile time)

What is the alternative?

If some data dependent structures (if-then-else,while-do) are present apply quasi static scheduling. The system can't be predicted at compile time b/c some decisions are made at run-time.

The paper proposed a quazi-static scheduling algorithm that generates a schedule in which run-time decisions are made only for data- depended control structures.

The algorithm takes as input a Petri Nets (PN) model of the system and produces as output a software implementation consisting of a set of software tasks that are invoked at run-time by the Real-Time Operating System (RTOS).

Why Petri Nets?

Petri Nets can express concurrency, non-deterministic choice, synchronization and causality and because most properties, including schedulability, are decidable for PNs.

In particular, we use a sub-class of PNs called *Free-Choice* (FCPNs), because they exhibit clear distinction between the notions of concurrency and choice.

3) A *free-choice net* (FC) is an ordinary Petri net such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition, i.e.,

for all $p \in P$, $|p\bullet| \leq 1$ or $\bullet(p\bullet) = \{p\}$; equivalently,

for all $p_1, p_2 \in P$, $p_1\bullet \cap p_2\bullet \neq \varnothing \Rightarrow |p_1\bullet| = |p_2\bullet|$

$= 1$.

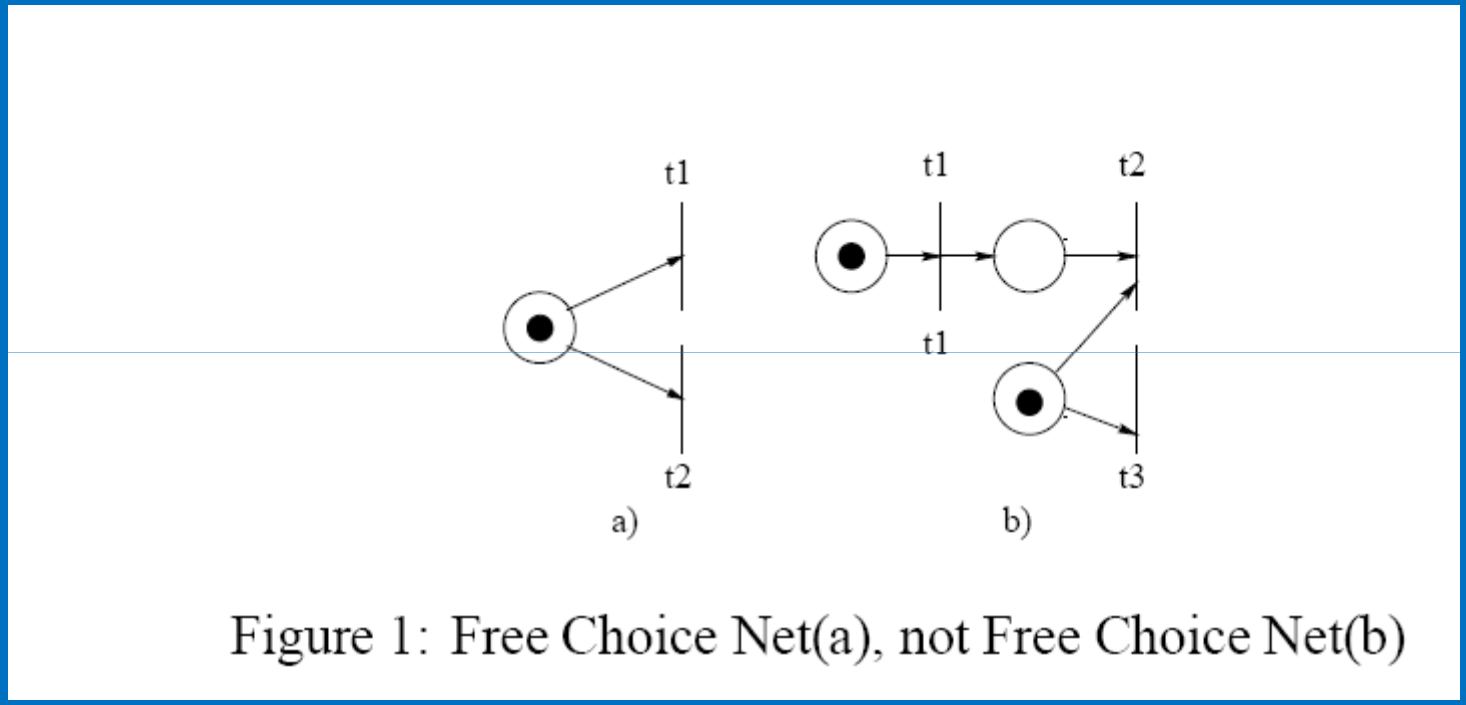
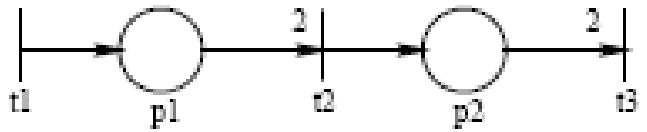


Figure 1: Free Choice Net(a), not Free Choice Net(b)

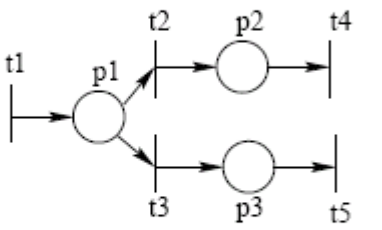
- Can one transition be enabled without the other?

A FCPN is quasi-statically schedulable if for every possible resolution of the control at the choice places, there exists a **cyclic finite sequence** that returns the tokens of the net to their initial places.



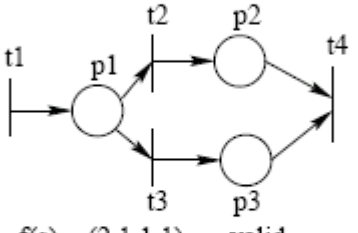
$$f(\sigma) = (4, 2, 1)^T$$

Figure 2: Cyclic schedule



f(s) = a(1,1,0,1,0) + b(1,0,1,0,1)
a , b = 0 , 1 , 2 ...

a)



f(s) = (2,1,1,1) valid
f(s) = (2,2,0,1) unbounded
f(s) = (2,0,2,1) unbounded

b)

Figure 3: Schedulable (a) and not schedulable (b) FCPNs

In a) $\Sigma = \{(t_1 t_2 t_4), (t_1 t_3 t_5)\}$ is a valid schedule because for every solution of the conflict between transitions t2 and t3, it is possible to complete a cycle that returns the net to the initial marking by firing t4 after t2, or t5 after t3.

b) is not schedulable because there exists no finite complete cycle if the conflict is always solved choosing t2 (t3). In fact if the token values in p1 are such that t2 (t3) always fired, unbounded accumulation of tokens occurs at place p2(p3).

Why is this schedulable if 3b wasn't schedulable?

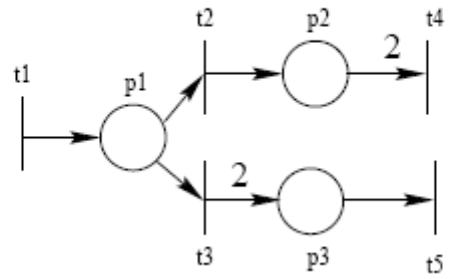


Figure 4: Schedulable Petri Net with weighted arcs

The net shown in figure 4 is schedulable and $\Sigma = \{(t_1t_2t_1t_2t_4)(t_1t_3t_5t_5)\}$ is a valid schedule.

If $t_1t_2t_1t_3t_5t_5$ fire in this order, one token remains in place p2 and the net does not return to the initial marking The net is considered schedulable because repeated executions of this sequence do not result in unbounded accumulation of tokens (as soon as there are two tokens in place p2 transition t4 Is fired and the tokens are consumed).

How do you find a valid schedule?

Step 1. Decompose the net into Conflict Free (CF) components.

Step 2. Check if every CF component is statically schedulable

Step 3. Derive a valid schedule, if there exists one

Bird’s eye view of the algorithm

1. Check the schedulability of the petri net.
    - If not schedulable, tell the designer no execution exists that can be implemented forever with bounded memory
    - Else if the net is schedulable:
        - Compute a quasi-static schedule (decompose net into statically schedulable components)
        - Derive a software implementation by traversing the schedule and replacing transitions with the corresponding code.

How do u generate code from the schedule?

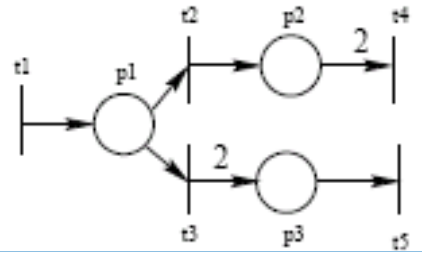


Figure 4: Schedulable Petri Net with weighted arcs

Schedule (Σ)
while ($t_i \neq EOS$) { Task(Σ,i); $i = i + 1$; }

Task (Σ,i)
while ($t_i \neq EOT$) {
  if (t_i is already visited) { insert **goto** label t_i }
  else{
    if (t_i is a conflicting transition) { insert **if..then..else** }
    if ($f(t_i) < f(t_{i-1})$) { insert counting var and **if** test }
    if ($f(t_i) > f(t_{i-1})$) { insert counting var and **while** test }
    if ($f(t_i) = f(t_{i-1})$) { insert t_i } } }

$\Sigma = \{(t_1 t_2 t_1 t_2 t_4)(t_1 t_3 t_5 t_5)\}$.
while (true) {
  t_1;
  if (p_1) {
    t_2; count(p_2)++;
    if (count(p_2) == 2) {
      t_4; count(p_2)-=2; }
    } else {
      t_3; count(p_3)+=2;
      while (count(p_3) $\geq$ 1) {
        t_5; count(p_3)- -; } } }

Questions?