Boundedness of Petri Nets

- Def: A Petri Net is **bounded** wrt an initial state if the set of states reachable from the initial state is finite.

**Bounded Unbounded**

In general, boundedness is a “good thing”.
- It is essential if the system is to be realized with finite memory.
- It allows the system to be analyzed as a finite-state machine.
- However, a type of unboundedness can be useful for mathematical analysis, in the form of **firing counters**.

Firing Counters

should be unbounded as a necessary condition to be free of deadlock.

Boundedness of Individual Places

- Def: A set of places (circular nodes) in a Petri net is **simultaneously unbounded** wrt. an initial state if ($\forall n \in \mathbb{N}$), there is a reachable state in which each place has $\geq n$ tokens.

Theorem (Karp & Miller, 1966)

- There is an algorithm for deciding whether any given place in a Petri net is unbounded.

Reachability-Tree Algorithm

- Construct a tree with the initial state as root.
- Construct successive nodes for each firable transition, as if constructing a state diagram.
- Whenever a node is added that has a predecessor which is pointwise $\leq$ this node, set to $\sim$ any place that is $<$ in the predecessor. If the result is a repeat, that branch ends.
- This process will terminate. Sets of places with $\sim$ are simultaneously unbounded.
Reachability-Tree Algorithm

Unboundedness of a Counter for a Transition is a Necessary, but not Sufficient, Condition for the Transition to have an Infinite Firing Sequence

Generalizing Petri Nets

- Adding inhibitory arcs:
  - For finite-state systems: ok, can simulate without inhibitory arcs anyway.
  - For unbounded systems: can destroy essential decidability properties (now can simulate a Turing machine)
- “Colored” tokens (see Kurt Jensen, 3 vols., Springer, 1997)
- Program variables
- Enabling predicates on transitions

Inhibition

Simulating Inhibitory Arcs

This transformation only works if the place X in question is bounded by 1.
Simulating Inhibitory Arcs

Adding *Time* to Petri Nets

- **Variation 1**: Transitions have a delay time; firing takes a non-zero time from enabling. Time may be bounded from above or below.

- **Variation 2**: Places have a delay time: A token must dwell on a place a certain amount of time (determined by the place) before becoming usable in firing.

- **Variation 3**: Like 2, but tokens have a delay time.

Variation 1 is Prevalent


Barthomieu & Diaz

- A Time Petri Net is like a Petri Net with a time interval on each transition: 
  \[ \langle t_1, t_2 \rangle \text{ or } \langle t_1, \infty \rangle \]

  From the time the transition is enabled, it cannot fire before \( t_1 \) and must fire by \( t_2 \) (unless disabled by firing another transition).

Example: Representing Time-out

- Showed that boundedness is decidable for Time Petri Nets with rational time bounds.
Example: Sending messages between two sites.

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>A message sent by a sender could get lost or be garbled.</td>
</tr>
<tr>
<td>Thus the receiver must ack each message.</td>
</tr>
<tr>
<td>If the ack is not received in a specified time, the transmission is regarded as having timed out and the sender must resend.</td>
</tr>
<tr>
<td>The ack could also get lost or be garbled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem, continued</th>
</tr>
</thead>
<tbody>
<tr>
<td>The sender might decide to resend although the original message has just been delayed, not lost.</td>
</tr>
<tr>
<td>How can the receiver tell whether an incoming message is new or just a retransmission of an earlier message?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each message is uniquely timestamped by a sequence number.</td>
</tr>
<tr>
<td>The receiver only accepts and acknowledges the next number in sequence, not the replay of an earlier number.</td>
</tr>
<tr>
<td>The acknowledgment indicates the number of the message being acknowledged.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem &amp; Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>The set of timestamps is not bounded.</td>
</tr>
<tr>
<td>Alternating bit protocol (ABP):</td>
</tr>
<tr>
<td>Use only 0 and 1 as numbers.</td>
</tr>
<tr>
<td>Sender sends 1 only when 0 has been successfully acknowledged.</td>
</tr>
<tr>
<td>Sender sends 0 only when 1 has been successfully acknowledged.</td>
</tr>
</tbody>
</table>

| Drawback: Only one message can be in transit at a time. |
More general possibilities (not ABP)

Example:
Representing ABP in TPN: Basic

We will develop only the 0 half of the protocol.
The 1 half fits symmetrically.

ABP:
Add possible loss

ABP (continued)

ABP (continued)
ABP (continued)

Summary

- A message can be sent at an arbitrary time.
- Once sent, the message can be received or lost, within a bounded time.
- If the message is received, an ack is sent.
- The ack can be received or lost.
- If an ack is not received in a bounded time, the message is resent.
- If a resent message is received, it is ack’ed.

Homework, Part 2

Construct a Time Petri Net model for the home alarm system described earlier.

Temporal Constraint Networks

Temporal Constraint Networks: Basic Idea

- Nodes are “timepoints”; they represent points in time.
- Timepoints are not necessarily bound to a specific time; they may be “floating”.
- A timepoint becomes “grounded” when it is associated with a specific time.
- Arcs represent temporal constraints between timepoints.
- Each arc is labeled with a minimum and maximum time between the timepoints it connects.

Temporal Constraint Networks

- TCN’s are a graphical model used to reason about temporal systems.
- They have also been used as a form of control specification for real-time systems.
Temporal Constraints between Timepoints

\( a \quad \text{u to v} \quad b \)

means that between the time at which \( a \) is grounded and the time at which \( b \) is grounded, there is at least \( u \) units of time and at most \( v \) units of time:

\[ \text{t}(a) + u \leq \text{t}(b) \leq \text{t}(a) + v \]

\( u \) or \( v \) can be negative: \( u \) units before is the same as -\( u \) units after.

\( a \quad \text{-v to -u} \quad b \)

Other Ways of Writing

\( a \quad \text{u to v} \quad b \)

\[ \begin{align*}
\text{t}(a) + u & \leq \text{t}(b) \leq \text{t}(a) + v \\
\text{u} & \leq \text{t}(b) - \text{t}(a) \leq \text{v} \\
\text{-}u & \geq \text{t}(a) - \text{t}(b) \geq \text{-}v \\
\text{-}v & \leq \text{t}(a) - \text{t}(b) \leq \text{-}u \\
\end{align*} \]

Temporal Constraint Example

\( \text{launch} \quad 5 \text{ to } 20 \quad \text{orbit} \)

means that orbit can occur no sooner than 5 time units, nor no later than 20 time units, after launch.

Absolute Time

\( \text{reference} \quad 5 \text{ to } 10 \text{ seconds} \quad \text{event} \)

supposing that reference is a fixed reference time, such as Jan. 1, 2000, this says that event can occur not before 5 seconds and not later than 10 seconds into the year.

\( \text{reference} \quad 10 \text{ to } 10 \text{ seconds} \quad \text{event} \)

says event must occur exactly 10 seconds into the year.

Reasoning with TCN's

\( a \quad \text{u to v} \quad b \quad \text{w to x} \quad c \)

\( \sqcup \text{ implies} \)

\( a \quad \text{(u+w) to (v+x)} \quad c \)

Reasoning with TCN's

\( a \quad \text{u to v} \quad c \)

\( b \quad \text{w to x} \quad \text{c} \)

\( \sqcup \text{ implies} \)

\( a \quad \text{??} \quad b \)
Reasoning with TCN's

Check Extreme Cases

Consistency of TCN's

Example of Consistent vs. Inconsistent

Consistency Checking Algorithm

Def: A temporal constraint network is **consistent** if it is possible to assign times to each of the timepoints such that all constraints are simultaneously satisfied.

Convert the TCN to a labeled directed graph with just one distance on each arc, using the following transformation:
Consistency Checking Algorithm

- Test the resulting graph to see if any negative-sum cycles are present (e.g., using Floyd’s algorithm).
- The original TCN is consistent iff there is no negative-sum cycle in the transformed graph.

Example

\[
\begin{array}{c}
\text{a} & \text{b} & \text{c} \\
5 \text{ to } 10 & 6 \text{ to } 9 & 5 \text{ to } 10 \\
\end{array}
\]

\[
\begin{array}{c}
\text{a} & \text{b} & \text{c} \\
6 \text{ to } 9 & 6 \text{ to } 9 & -6 \\
\end{array}
\]

\[
\begin{array}{c}
\text{a} & \text{b} & \text{c} \\
10 & 9 & -6 \\
\end{array}
\]

Consistent

Constraint Propagation

- Suppose a node in a consistent TCN is grounded (is assigned a fixed time).
- Then for each other node in the TCN, a window for possible groundings can be determined from the temporal constraints.

Example

\[
\begin{array}{c}
\text{a} & \text{b} & \text{c} \\
5 \text{ to } 10 & 6 \text{ to } 9 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{a} & \text{b} & \text{c} \\
\text{ground} & 0 & [6, 9] (= [0+6, 0+9]) \\
\end{array}
\]

Implied window
**Constraint Propagation Example**

- **implied window**
- 

  \([-4, 4]\) (= \([6-10, 9-5]\))

- **Note:** The window idea assumes "controllable" constraints: We get to choose the actual distance between timepoints consistent with the constraint.

---

**Conjecture**

- If a TCN is consistent, then grounding any node within its window, by adding a constraint between a reference timepoint and the node, results in a consistent net.
- The effect of grounding is to narrow other windows.

---

**Sequential Grounding Example**

- **Step 1**
- **Step 2**, say 3

- So a possible grounding sequence is: b(0), a(3), c(9).

---

**Possible use of TCN’s for RT Control**

- Windows are maintained with earliest and latest groundable times for all nodes.
- Some reference node fires at "time 0".
- While( unfired nodes exist )
  
  
  choose ungrounded node with earliest window;
  ground that node;
  propagate constraints, updating windows;
  }

---

**Difficulties in using TCN’s for RT Control**

- The constraints have to be controllable in the sense mentioned earlier.
- The check for consistency can be computationally expensive.