Previous methods only use limited knowledge of delays.
Very robust systems, but extremely conservative.
Large functional units do not have zero delay.
Gates and wires do not have an infinite delay.
Timing analysis can identify additional unreachable states.
These unreachable states are additional don’t cares.
*Timed circuits* use this information to optimize the design.
A Simple Example

- Shopkeeper actively calls winery and patron.
- Calls the patron immediately after calling the winery without waiting for the wine to arrive.
- The shopkeeper does the following:
  - Calls the winery,
  - Calls the patron,
  - Peers out the window until he sees both the wine delivery boy and the patron,
  - Lets them in, and
  - Completes the sale.
Timing Relationships

Call_WInery [uniform(2,3)]

Call_Patron [uniform(2,3)]

Wine_Arrives [uniform(2,3)]

Patron_Arrives [uniform(5,inf)]

Wine_Is_Purchased [uniform(2,3)]
There is a timer $t_i$ associated with each arc in the graph. A *timed state* is an *untimed state* and value of all active *timers*.

\[
\left( \{p_6\}, t_6 = 0 \right)
\]

A timer is allowed to advance by any amount less than its upper bound resulting in a new timed state.

\[
\left( \{p_6\}, t_6 = 1.1 \right)
\]
\[
\left( \{p_6\}, t_6 = 2.22 \right)
\]
\[
\left( \{p_6\}, t_6 = 2.71828182846 \right)
\]
When a timer reaches its lower bound, it becomes *satisfied*.

When a timer reaches its upper bound, it becomes *expired*.

An event enabled by a single rule must happen sometime after its timer becomes satisfied and before it becomes expired.

When an event is enabled by multiple rules, it must happen after all of its rules are satisfied, but before all of its rules are expired.

Extend the notion of allowed sequences to timed states paired with the time of the state transition.

State transition can be either time advancement or a change in the untimed state.
Example: Timing Sequence

Call_Winery [uniform(2,3)]

Call_Patron [uniform(2,3)]

Wine_Arrives [uniform(2,3)]

Patron_Arrives [uniform(5,inf)]

Wine_Is_Purchased [uniform(2,3)]
Example: Timing Sequence

\[
\left( \left( \left( \left( \left( \{ p_6 \}, t_6 = 0 \right), 0 \right),
\right) \right), 0 \right),
\right)
\]
Example: Timing Sequence

\[
(\{p_6\}, t_6 = 0, 0),
(\{p_6\}, t_6 = 2.22, 2.22),
\]
Example: Timing Sequence

\[
\begin{align*}
\text{Call_WInery} & \quad \text{uniform}(2,3) \\
p1 & \quad \text{Wine_Arrives} & \quad \text{Call_Patron} & \quad \text{uniform}(2,3) \\
p2 & \quad \text{Wine_Is_Purchased} & \quad \text{Call_Patron} & \quad \text{uniform}(2,3) \\
p3 & \quad \text{p1, p2} & \quad \text{p6} & \quad \text{Wine_Arrives} & \quad \text{uniform}(2,3) \\
p4 & \quad \text{Patron_Arrives} & \quad \text{uniform}(5,\infty) \\
p5 & \quad \text{Wine_Is_Purchased} & \quad \text{uniform}(2,3) \\
p6 & & & & \\
\end{align*}
\]

\[
\begin{align*}
\left( \{ p_6 \}, t_6 = 0, 0 \right), \\
\left( \{ p_6 \}, t_6 = 2.22, 2.22 \right), \\
\left( \{ p_1, p_2 \}, t_1 = t_2 = 0, 2.22 \right),
\end{align*}
\]
Example: Timing Sequence

\[
(\{p_6\}, t_6 = 0, 0), \\
(\{p_6\}, t_6 = 2.22, 2.22), \\
(\{p_1, p_2\}, t_1 = t_2 = 0, 2.22), \\
(\{p_1, p_2\}, t_1 = t_2 = 2.1, 4.32),
\]
Example: Timing Sequence

\[
\begin{align*}
( & \{p_6\}, t_6 = 0, 0), \\
( & \{p_6\}, t_6 = 2.22, 2.22), \\
( & \{p_1, p_2\}, t_1 = t_2 = 0, 2.22), \\
( & \{p_1, p_2\}, t_1 = t_2 = 2.1, 4.32), \\
( & \{p_1, p_3\}, t_1 = 2.1, t_3 = 0, 4.32),
\end{align*}
\]
Example: Timing Sequence

\[
(\{p_6\}, t_6 = 0, 0), \\
(\{p_6\}, t_6 = 2.22, 2.22), \\
(\{p_1, p_2\}, t_1 = t_2 = 0, 2.22), \\
(\{p_1, p_2\}, t_1 = t_2 = 2.1, 4.32), \\
(\{p_1, p_3\}, t_1 = 2.1, t_3 = 0, 4.32), \\
\ldots
\]
Since time can take on any real value, there is an uncountably infinite number of timed states and timed allowed sequences.

Must either group timed states into finite number of equivalence classes or restrict the values of the timers.

Several possible methods for *timed state space exploration*:

- Region method
- Discrete-time method
- Zone method
- POSET method
A region is described by the integer component of each timer and the relationship between the fractional components.

- \( f(t_1) = f(t_2) = 0 \): region is a point.
- \( f(t_1) = 0 \) and \( f(t_2) > 0 \): region is a vertical line segment.
- \( f(t_1) > 0 \) and \( f(t_2) = 0 \): region is a horizontal line segment.
- \( f(t_1) = f(t_2) > 0 \): region is a diagonal line segment.
- \( f(t_1) > f(t_2) > 0 \): region is a lower triangle.
- \( f(t_2) > f(t_1) > 0 \): region is an upper triangle.
Possible Timed States Using Regions

$\langle 0, 5 \rangle$

171 distinct timed states
Timed Sequence Using Regions

Call_Patron
[uniform(2,3)]

Wine_Is_Purchased
[uniform(2,3)]

Call_WInery
[uniform(2,3)]

Wine_Arrives
[uniform(5,inf)]

Patron_Arrives
[uniform(5,inf)]

Wine_Arrives
[uniform(2,3)]

Call_Patron
[uniform(2,3)]

Wine_Is_Purchased
[uniform(2,3)]

Call Winery

t1=t2=0
f(t1)=f(t2)=0
Timed Sequence Using Regions

\[ t_1 = t_2 = 0 \]
\[ f(t_1) = f(t_2) = 0 \]

\[ t_1 = t_2 = 0 \]
\[ f(t_1) = f(t_2) > 0 \]
Timed Sequence Using Regions

Call_WInery [uniform(2,3)]

Wine_Arrives [uniform(2,3)]

Call_Patron [uniform(2,3)]

Patron_Arrives [uniform(5,inf)]

Wine_Is_Purchased [uniform(2,3)]

Call_Patron [uniform(2,3)]

Wine_Arrives [uniform(2,3)]

Wine_Is_Purchased [uniform(2,3)]

\[ t_1 = t_2 = 0 \]
\[ f(t_1) = f(t_2) > 0 \]

\[ t_1 = t_2 = 1 \]
\[ f(t_1) = f(t_2) = 0 \]
Timed Sequence Using Regions

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)
Timed Sequence Using Regions

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)
Timed Sequence Using Regions

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)

16 / 116
Timed Sequence Using Regions

- Call_Patron
  \[\text{uniform}(2,3)\]
  \[p_3\]
- Wine_Is_Purchased
  \[\text{uniform}(2,3)\]
- Call_WInery
  \[\text{uniform}(2,3)\]
- Patron_Arrives
  \[\text{uniform}(5,\text{inf})\]
- Wine_Arrives
  \[\text{uniform}(2,3)\]
- Call_Patron
  \[\text{uniform}(2,3)\]
  \[p_6\]
- \[p_1\]
- \[p_2\]
- \[p_3\]
- \[p_4\]
- \[p_5\]
- \[p_6\]

\[t_1=t_2=2\]
\[f(t_1)=f(t_2)>0\]
\[t_3=0, t_1=2\]
\[f(t_1) > f(t_3)=0\]
Timed Sequence Using Regions

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)

Page 18 / 116
Timed Sequence Using Regions

```
Call_Patron [uniform(2,3)]
p3
Wine_Is_Purchased [uniform(2,3)]
Call_WInery [uniform(2,3)]
p1
p2
Wine_Arrives [uniform(2,3)]
Patron_Arrives [uniform(5,inf)]
p5
Call_Patron
p6
Wine_Arrives [uniform(2,3)]
p4
p5
Wine_Is_Purchased [uniform(2,3)]
```

- \( t_3 = 0, t_1 = 2 \)
  - \( f(t_1) > f(t_3) > 0 \)
- \( t_3 = 0, t_1 = 3 \)
  - \( f(t_3) > f(t_1) = 0 \)

Chris J. Myers (Lecture 7: Timed Circuits)
Timed Sequence Using Regions

Call_Patron [uniform(2,3)]

Wine_Is_Purchased [uniform(2,3)]

Call_WInery [uniform(2,3)]

Patron_Arrives [uniform(5,inf)]

Wine_Arrives [uniform(2,3)]

Wine Arrives

t3=0, t1=3
f(t3) > f(t1)=0

Wine Arrives

t3=0, t4=0
f(t3) > f(t4)=0

f(t3) > f(t1)=0

f(t3) > f(t4)=0
Timed State Space Using Regions

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)
Timed State Space Explosion

- Requires 26 timed states to represent all the timing relationships for only 4 untimed states.
- Worst-case complexity is:

\[ |S| \frac{n!}{\ln 2} \left( \frac{k}{\ln 2} \right)^n 4^{1/k} \]

where \( S \) is number of untimed states, \( n \) is the number of rules enabled concurrently, and \( k \) is maximum timing constraint.
For timed labeled Petri-nets, all timing requirements are of the form $\leq$ or $\geq$, since timing bounds are inclusive.

In this case, fractional components are not necessary.

Only need to track *discrete-time* states.

Worst-case complexity is now:

$$|S|(k + 1)^n$$

Reduction by a factor of more than $n!$. 
36 distinct timed states
Timed State Space Using Discrete-Time

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)
Unfortunately, the discrete-time technique is still exponential in the number of concurrent timers and size of the timing bounds.

Changing each timing bound of $[2, 3]$ to $[19, 31]$ and $[5, \infty]$ to $[53, \infty]$, number of timed states goes from 69 to more than 3000.

Changing each timing bound of $[2, 3]$ to $[191, 311]$ and $[5, \infty]$ to $[531, \infty]$, number of timed states goes to over 300,000!
Another approach is to use convex polygons, called *zones*, to represent equivalence classes of timed states.

One zone is representing 171 regions or 36 discrete-time states.
Convex polygons can be represented using linear inequalities.

Introduce a dummy timer \( t_0 \) which always takes the value 0.

For each pair of timers, introduce inequality of the form:

\[
t_j - t_i \leq m_{ij}
\]

Example:

\[
\begin{align*}
t_0 - t_0 & \leq 0 \\
t_1 - t_0 & \leq 5 \\
t_2 - t_0 & \leq 5 \\
t_0 - t_1 & \leq 0 \\
t_1 - t_1 & \leq 0 \\
t_2 - t_1 & \leq 5 \\
t_1 - t_2 & \leq 5 \\
t_2 - t_2 & \leq 0
\end{align*}
\]
Set of inequalities can be collected into a data structure called a *difference bound matrix* (DBM).

The difference bound matrix for this example is shown below:

\[
\begin{array}{ccc}
  & t_0 & t_1 & t_2 \\
 t_0 & 0 & 5 & 5 \\
 t_1 & 0 & 0 & 5 \\
 t_2 & 0 & 5 & 0 \\
\end{array}
\]
Recanonicalization

- Many DBMs represent the same zone.
- Need unique DBM representation to determine when a zone has been seen before during the depth first search.
- Each zone has a canonical DBM representation when all entries are minimal.
Recanonicalization Example

\[
\begin{array}{c|ccc}
 t_0 & t_1 & t_2 \\
\hline
 t_0 & 0 & 5 & 5 \\
 t_1 & 0 & 0 & 7 \\
 t_2 & 0 & 5 & 0 \\
\end{array}
\]

\[
\begin{align*}
t_{2} - t_{1} & \leq 7 \\
t_{2} - t_{0} & \leq 5 \\
t_{0} - t_{1} & \leq 0
\end{align*}
\]

- Add last two equations to get:

\[
t_{2} - t_{1} \leq 5
\]

\[
\begin{array}{c|ccc}
 t_0 & t_1 & t_2 \\
\hline
 t_0 & 0 & 5 & 5 \\
 t_1 & 0 & 0 & 5 \\
 t_2 & 0 & 5 & 0 \\
\end{array}
\]
Recanonicalization equivalent to all pairs shortest path problem.

Create a labeled digraph where:
- There is a vertex for each timer $t_i$,
- An arc from $t_i$ to $t_j$ for each linear inequality of the form $t_j - t_i \leq m_{ij}$ when $i \neq j$.
- Each arc is labeled by $m_{ij}$. 
**Floyd’s Algorithm**

**Recanonicalization** \((M)\)

for \(k = 1\) to \(n\)
  
  for \(i = 1\) to \(n\)
    
    for \(j = 1\) to \(n\)
      
      if \((m_{ij} > m_{ik} + m_{kj})\) then
        
        \[m_{ij} = m_{ik} + m_{kj};\]
Floyd’s Algorithm Example

\[
\begin{array}{c|ccc}
   & t_0 & t_1 & t_2 \\
\hline
   t_0 & 0 & 5 & 5 \\
t_1 & 0 & 0 & 7 \\
t_2 & 0 & 5 & 0 \\
\end{array}
\]
After a rule fires:

- Restrict to reflect minimum firing time.
- Recononicalize
- Project out information on rule that fired.
- Extend matrix with new rows and columns for new rules.
- Advance time
- Recononicalize
Example: firing of a rule $r_k = \langle e_k, f_k, l_k, u_k \rangle$ where

- $e_k$ is the enabling transition,
- $f_k$ is the enabled transition,
- $l_k$ is the lower bound of the corresponding timer $t_k$, and
- $u_k$ is the upper bound on the timer.

Constrain DBM to indicate rule has reached its lower bound.

$t_0 - t_k \leq -l_k$, so set $m_{k0}$ to $-l_k$.

DBM may no longer be maximally tight.

Recanonicalize DBM using Floyd’s algorithm.
Remove the row and column corresponding to $t_k$.
If rule firing causes an transition, new rules may be enabled.
For newly enabled rules, introduce a new timer $t_l$ with a row and column in the DBM.
Initialize $m_{l0}$ and $m_{0l}$ to 0.
Initialize each $m_{lj}$ to $m_{0j}$.
Initialize each $m_{il}$ to $m_{i0}$.
Advance Time

- Set all timers to their upper bound (i.e., $m_{0j} = u_j$).
- Recanonicalize the DBM using Floyd’s algorithm.
update_zone \((M, r_j, \text{event\_fired, } R_{en}, R_{new})\)

if \(m_{j0} > -l_j\) then \(m_{j0} = -l_j\)
recanonicalize\((M)\)
project\((M, r_j)\)

if (event\_fired) then

foreach \(r_i \in R_{new}\)

\(m_{i0} = m_{0i} = 0\)

foreach \(r_k \in R_{new}\)

\(m_{ik} = m_{ki} = 0\)

foreach \(r_k \in (R_{en} - R_{new})\)

\(m_{ik} = m_{0k}\)
\(m_{ki} = m_{k0}\)

foreach \(r_i \in R_{en}\)

\(m_{0i} = u_i\)
recanonicalize\((M)\)
normalize\((M, R_{en})\)
\textbf{Normalize}

\begin{verbatim}
\textbf{normalize} (M, R_{en})
  \textbf{foreach} \( r_i \in R_{en} \)
    \textbf{if} \( (m_{i0} < -\text{premax}(r_i)) \) \textbf{then}
      \textbf{foreach} \( r_j \in R_{en} \)
        \( m_{ij} = m_{ij} - (m_{i0} + \text{premax}(r_i)) \)
        \( m_{ji} = m_{ji} + (m_{i0} + \text{premax}(r_i)) \)
      \textbf{foreach} \( r_i \in R_{en} \)
        \textbf{if} \( (m_{0i} > \text{premax}(r_i)) \) \textbf{then}
          \( m_{0i} = \max_j (\min(m_{0j}, \text{premax}(r_j)) - m_{ij}) \)
  \textbf{recanonicalize}(M)
\end{verbatim}
Initial Zone

Call_Patron [uniform(2, 3)]

Wine_Arrives [uniform(2, 3)]

Patron_Arrives [uniform(5, inf)]

Wine_Is_Purchased [uniform(2, 3)]
Initial Zone

```
Initial Zone

Initial

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th>t6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdvTime/Recanon/Norm.</td>
<td>0 3</td>
<td>0 0</td>
</tr>
</tbody>
</table>
```

```
Initial

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th>t6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>
```

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Winery Called

Asynchronous Circuit Design

Restrict/Recanon

<table>
<thead>
<tr>
<th>t0</th>
<th>t6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Winery Called

Call_Patron [uniform(2,3)]

Wine_Is_Purchased [uniform(2,3)]

Call_WInery [uniform(2,3)]

Wine_Arrives [uniform(2,3)]

Patron_Arrives [uniform(5,inf)]

Wine_Is_Purchased [uniform(2,3)]

Restrict/Recanon

Project

\[
\begin{array}{c|cc}
 & t_0 & t_6 \\
\hline
 t_0 & 0 & 3 \\
 t_6 & -2 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c}
 & t_0 \\
\hline
 t_0 & 0 \\
\end{array}
\]
Zone After Winery Called

Asynchronous Circuit Design

Project

\[
\begin{array}{c|c}
\text{t} & \text{t}_0 \\
\hline
0 & 0
\end{array}
\]

Extend

\[
\begin{array}{c|ccc}
\text{t} & \text{t}_0 & \text{t}_1 & \text{t}_2 \\
\hline
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
2 & 0 & 0 & 0
\end{array}
\]

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Winery Called

AdvTime/Recanon/Norm.

Extend

<table>
<thead>
<tr>
<th></th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₂</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Wine Arrives

```
Wine Arrives [uniform(2,3)]
```

```
Wine Is Purchased [uniform(2,3)]
```

```
Patron Arrives [uniform(5, inf)]
```

```
Wine Arrives [uniform(2,3)]
```

```
Restrict
```

```
t_0 t_1 t_2
0 3 3
-2 0 0
0 0 0
```

Chris J. Myers (Lecture 7: Timed Circuits)  Asynchronous Circuit Design
Zone After Wine Arrives

\[
\begin{align*}
\text{Restrict} & \\
\begin{array}{c|ccc}
t & t_0 & t_1 & t_2 \\
0 & 0 & 3 & 3 \\
1 & -2 & 0 & 0 \\
2 & 0 & 0 & 0 \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\text{Recanon} & \\
\begin{array}{c|ccc}
t & t_0 & t_1 & t_2 \\
0 & 0 & 3 & 3 \\
1 & -2 & 0 & 0 \\
2 & -2 & 0 & 0 \\
\end{array}
\end{align*}
\]
Zone After Wine Arrives

Recanon

\[
\begin{array}{c|ccc}
  t_0 & t_1 & t_2 \\
  \hline
  t_0 & 0 & 3 & 3 \\
  t_1 & -2 & 0 & 0 \\
  t_2 & -2 & 0 & 0 \\
\end{array}
\]

Project

\[
\begin{array}{c|c}
  t_0 & t_2 \\
  \hline
  t_0 & 0 & 3 \\
  t_2 & -2 & 0 \\
\end{array}
\]
Zone After Wine Arrives

```
<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patron Arrives</td>
<td>t0-4</td>
</tr>
<tr>
<td>Wine Arrives</td>
<td>t2</td>
</tr>
<tr>
<td>Wine Is Purchased</td>
<td>t0-2</td>
</tr>
</tbody>
</table>
```

Project

```
<table>
<thead>
<tr>
<th>Time</th>
<th>t0</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Extend

```
<table>
<thead>
<tr>
<th>Time</th>
<th>t0</th>
<th>t4</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Zone After Wine Arrives

Call_Winery [uniform(2,3)]

p2

Call_Patron [uniform(2,3)]

Wine_Arrives [uniform(2,3)]

p1

p3

Patron_Arrives [uniform(5,inf)]

p6

Wine_Is_Purchased [uniform(2,3)]

p5

p4

Extend

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th>t4</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>t4</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>t2</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>

AdvTime

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th>t4</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>t4</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>t2</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Wine Arrives and Patron is Called

- Wine Arrives and Patron is Called
  - Wine Is Purchased
  - Call Winery
  - Patron Arrives
  - Wine Arrives

Restrict/Reconon

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_4$</th>
<th>$t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$t_2$</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Wine Arrives and Patron is Called

Call_Patron

Wine_Is_Purchased

Call_WInery

Patron_Arrives

Wine_Arrives

Restrict/Reconon

Project

\[
\begin{array}{c|ccc}
    & t_0 & t_4 & t_2 \\
    t_0 & 0 & 1 & 3 \\
    t_4 & 0 & 0 & 3 \\
    t_2 & -2 & -2 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c}
    & t_0 & t_4 \\
    t_0 & 0 & 1 \\
    t_4 & 0 & 0 \\
\end{array}
\]
Zone After Wine Arrives and Patron is Called

Call_WInery [uniform(2,3)]

p2

p1

Call_Patron [uniform(2,3)]

Wine_Arrives [uniform(2,3)]

p3

Patron_Arrives [uniform(5,inf)]

p6

Wine_Is_Purchased [uniform(2,3)]

p5

p4

Project

<table>
<thead>
<tr>
<th>t0</th>
<th>t4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Extend

<table>
<thead>
<tr>
<th>t0</th>
<th>t4</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Wine Arrives and Patron is Called

Asynchronous Circuit Design

Extend

AdvTime
Zone After Wine Arrives and Patron is Called

AdvTime

Reconon/ Norm.

Asynchronous Circuit Design
Zone After Rule Expires

Asynchronous Circuit Design

Restrict

<table>
<thead>
<tr>
<th></th>
<th>t₀</th>
<th>t₄</th>
<th>t₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>t₄</td>
<td>-2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₃</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Reconon

<table>
<thead>
<tr>
<th></th>
<th>t₀</th>
<th>t₄</th>
<th>t₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>t₄</td>
<td>-2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₃</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Rule Expires

Call_Patron
\[\text{uniform}(2,3)\]

Wine_Is_Purchased
\[\text{uniform}(2,3)\]

Call_WInery
\[\text{uniform}(2,3)\]

Patron_Arrives
\[\text{uniform}(5,\infty)\]

Wine_Arrives
\[\text{uniform}(2,3)\]

Reconon

\[
t_0 \quad t_4 \quad t_3 \\
\begin{array}{c}
0 \\
-2 \\
-1
\end{array}
\]

Project

\[
t_0 \quad t_3 \\
\begin{array}{c}
0 \\
-1
\end{array}
\]
Zone After Rule Expires

![Diagram of asynchronous circuit design with events and time intervals]

**Project**

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

**AdvTime/Reconon**

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Rule Expires

Asynchronous Circuit Design
Zone After Patron Arrives

- **Call_Patron [uniform(2,3)]**: p3
- **Wine_Is_Purchased [uniform(2,3)]**: Call_WInery
- **Wine_Arrives [uniform(2,3)]**: p4
- **Patron_Arrives [uniform(5,inf)]**: p5
- **Restrict/Recanon**

```
<table>
<thead>
<tr>
<th>t0</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>-5</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Zone After Patron Arrives

Restrict/Recanon

Project

\[
\begin{array}{c|c|c}
\text{t}_0 & \text{t}_3 \\
\hline
0 & 5 \\
\hline
-5 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{t}_0 & \text{t}_0 \\
\hline
0 & \\
\end{array}
\]
Zone After Patron Arrives

Asynchronous Circuit Design

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Patron Arrives

Call_Patron [uniform(2,3)]
p3
Wine_Is_Purchased [uniform(2,3)]
Call_WInery [uniform(2,3)]
p6
p2 p1
Patron_Arrives [uniform(5,inf)]
p5
Wine_Arrives [uniform(2,3)]
p4

Extend

AdvTime/ Recanon/ Norm.
Zone After Wine is Purchased

\[
\begin{align*}
\text{Call_WInery} & \quad \text{[uniform(2,3)]} \\
\text{Call_Patron} & \quad \text{[uniform(2,3)]} \\
\text{Wine_Arrives} & \quad \text{[uniform(2,3)]} \\
\text{Patron_Arrives} & \quad \text{[uniform(5,inf)]} \\
\text{Wine_Is_Purchased} & \quad \text{[uniform(2,3)]} \\
\end{align*}
\]

Restrict/Recanon

\[
\begin{array}{c|cc}
\text{t} & \text{t}_0 & \text{t}_5 \\
\hline
\text{t}_0 & 0 & 3 \\
\text{t}_5 & -2 & 0 \\
\end{array}
\]
Zone After Wine is Purchased

**Asynchronous Circuit Design**

---

**Project**

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$t_5$</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Restrict/Recanon**

---

**Call_Patron**

---

**Wine_Is_Purchased**

---

**Call_WInery**

---

**Wine_Arrives**

---

**Patron_Arrives**

---

**Wine_Is_Purchased**
Zone After Wine is Purchased

Project

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
</tr>
</tbody>
</table>

Extend

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_6$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Wine is Purchased

\begin{align*}
\text{Call}_\text{Patron} & \quad \text{[uniform(2,3)]} \\
\text{Wine}_\text{Is}_\text{Purchased} & \quad \text{[uniform(2,3)]} \\
\text{Call}_\text{WInery} & \quad \text{[uniform(2,3)]} \\
\text{Patron}_\text{Arrives} & \quad \text{[uniform(5, inf)]} \\
\text{Wine}_\text{Arrives} & \quad \text{[uniform(2, 3)]} \\
\end{align*}

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AdvTime/Recanon/Norm.</strong></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Extend</strong></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Patron is Called

Asynchronous Circuit Design

Initial

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Patron is Called

Initial

Restrict

\[
\begin{array}{c|ccc}
  & t_0 & t_1 & t_2 \\
\hline
  t_0 & 0 & 3 & 3 \\
  t_1 & 0 & 0 & 0 \\
  t_2 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|ccc}
  & t_0 & t_1 & t_2 \\
\hline
  t_0 & 0 & 3 & 3 \\
  t_1 & 0 & 0 & 0 \\
  t_2 & -2 & 0 & 0 \\
\end{array}
\]
Zone After Patron is Called

Restrict

Recanon

\[
\begin{array}{c|ccc}
  \text{t} & t_0 & t_1 & t_2 \\
  t_0 & 0 & 3 & 3 \\
  t_1 & 0 & 0 & 0 \\
  t_2 & -2 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|ccc}
  \text{t} & t_0 & t_1 & t_2 \\
  t_0 & 0 & 3 & 3 \\
  t_1 & -2 & 0 & 0 \\
  t_2 & -2 & 0 & 0 \\
\end{array}
\]
Zone After Patron is Called

Recanon

Project

Chris J. Myers (Lecture 7: Timed Circuits)  Asynchronous Circuit Design
Zone After Patron is Called

Project

\[
\begin{array}{c|c|c}
  t_0 & t_1 \\
  0 & 3 \\
\end{array}
\]

Extend

\[
\begin{array}{c|c|c|c}
  t_0 & t_1 & t_3 \\
  0 & 3 & 0 \\
  -2 & 0 & -2 \\
  0 & 3 & 0 \\
\end{array}
\]
Zone After Patron is Called

Call_Winery \[\text{uniform}(2,3)]

Wine_Arrives \[\text{uniform}(2,3)]

Call_Patron \[\text{uniform}(2,3)]

Patron_Arrives \[\text{uniform}(5, \infty)]

Wine_Is_Purchased \[\text{uniform}(2,3)]

Extend

\[
\begin{array}{ccc}
  t_0 & t_1 & t_3 \\
  0 & 3 & 0 \\
  -2 & 0 & -2 \\
  0 & 3 & 0 \\
\end{array}
\]

AdvTime

\[
\begin{array}{ccc}
  t_0 & t_1 & t_3 \\
  0 & 3 & \infty \\
  -2 & 0 & -2 \\
  0 & 3 & 0 \\
\end{array}
\]
Zone After Patron is Called

AdvTime

\[
\begin{array}{c|ccc}
  & t_0 & t_1 & t_3 \\
\hline
t_0 & 0 & 3 & \infty \\
t_1 & -2 & 0 & -2 \\
t_3 & 0 & 3 & 0 \\
\end{array}
\]

Recanon/ Norm.

\[
\begin{array}{c|ccc}
  & t_0 & t_1 & t_3 \\
\hline
t_0 & 0 & 3 & 1 \\
t_1 & -2 & 0 & -2 \\
t_3 & 0 & 3 & 0 \\
\end{array}
\]

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Patron is Called and Wine Arrives

Asynchronous Circuit Design

Restrict/Recanon

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$t_1$</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Patron is Called and Wine Arrives

Call_Patron

Wine_Is_Purchased

Call_WInery

Patron_Arrives

Wine_Arrives

Restrict/Recanon

Project
Zone After Patron is Called and Wine Arrives

Project

\[
\begin{array}{c|c|c}
\text{t_0} & \text{t_3} \\
0 & 1 \\
0 & 0 \\
\end{array}
\]

Extend

\[
\begin{array}{c|c|c|c}
\text{t_0} & \text{t_4} & \text{t_3} \\
0 & 0 & 1 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{array}
\]

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Patron is Called and Wine Arrives

- Call_Patron
- Call_WInery
- Patron_Arrives
- Wine_Arrives
- Wine_Is_Purchased

Extend

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th>t4</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>t4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AdvTime

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th>t4</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>0</td>
<td>3</td>
<td>∞</td>
</tr>
<tr>
<td>t4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Patron is Called and Wine Arrives

Call_Patron [uniform(2,3)] → p3
Wine_Is_Purchased [uniform(2,3)] → Call_WInery [uniform(2,3)] → p6
Patron_Arrives [uniform(5,inf)] → p5
Wine_Arrives [uniform(2,3)] → p4

AdvTime

<table>
<thead>
<tr>
<th>t0</th>
<th>t4</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>∞</td>
</tr>
</tbody>
</table>

Recanon/ Norm.

<table>
<thead>
<tr>
<th>t0</th>
<th>t4</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Zone After Rule Expires

Call_Patron

Wine_Is_Purchased

Call_WInery

Wine_Arrives

Patron_Arrives

Restrict

<table>
<thead>
<tr>
<th>t_0</th>
<th>t_4</th>
<th>t_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>t_4</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>t_3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Chris J. Myers (Lecture 7:Timed Circuits)
Zone After Rule Expires

Call_Patron

Wine_Is_Purchased

Call_WInery

Wine_Arrives

Patron_Arrives

Restrict

Recanon

\[
\begin{array}{c|ccc}
  t_0 & t_4 & t_3 \\
  \hline
t_0 & 0 & 3 & 4 \\
t_4 & -2 & 0 & 1 \\
t_3 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|ccc}
  t_0 & t_4 & t_3 \\
  \hline
t_0 & 0 & 3 & 4 \\
t_4 & -2 & 0 & 1 \\
t_3 & -2 & 0 & 0 \\
\end{array}
\]
Zone After Rule Expires

Recanon

Project

Chris J. Myers (Lecture 7: Timed Circuits)
Zone After Rule Expires

Project

AdvTime / Recanon

Chris J. Myers (Lecture 7: Timed Circuits)  Asynchronous Circuit Design
Zone After Rule Expires

AdvTime / Recanon

Normalize
Timed State Space using Zones

- Wine Is Purchased
- Call Winery
- Wine Arrives
- Call Patron
- Wine Arrives
- Call Patron
- rule expires
- Patron Arrives
- rule expires
- rule expires
Adverse Example

1 untimed state
2,825,761 discrete-time states
219,977,777 zones
Order versus Causality

Asynchronous Circuit Design
Using linear traces introduces fake orderings.

Need to separate concurrency from casuality.

Find zones on POSETs rather than linear traces.

Represent POSETs using graph/matrix.
Asynchronous Circuit Design

POSET Graph/Matrix/Zone

reset

\begin{array}{c}
\text{reset} \\
\text{[1,40]} \\
\text{[1,40]} \\
a \\
b
\end{array}

\begin{array}{c}
0 \\
40 \\
40
\end{array}

\begin{array}{c}
\text{[a,b]} & [b,a] \\
\end{array}

\begin{array}{c}
t_a \\
40 \\
0
\end{array}

\begin{array}{c}
0 \\
40 \\
40 \text{ t_b}
\end{array}

\begin{array}{cccc}
& r & a & b \\
\hline
r & 0 & 40 & 40 \\
a & -1 & 0 & 39 \\
b & -1 & 39 & 0
\end{array}

\begin{array}{c}
t_0 \\
t_a \\
t_b
\end{array}

\begin{array}{c}
0 \\
40 \\
40
\end{array}

\begin{array}{c}
0 \\
0 \\
39
\end{array}

\begin{array}{c}
0 \\
39 \\
0
\end{array}
1 untimed state
2,825,761 discrete-time states
219,977,777 zones
1 zone found using POSET timing
If event occurs, update POSET matrix and create zone:
- Set minimums to 0 (i.e., \( m_{i0} = 0 \)).
- Set maximums to the upper bound (i.e., \( m_{0j} = u_j \)).
- Copy relevant time separations from POSET matrix to zone (i.e., \( m_{ij} = p_{ij} \)).
- Recanonicalize.

Otherwise, project out timer corresponding to rule that fired.
Initial Zone using POSETs

Initial POSET

\[ r \]

reset

Initial zone /
Recanonicalize /
Normalize

\[
\begin{array}{c|c|c}
 t_0 & t_6 \\
 0 & 3 \\
 0 & 0 \\
\end{array}
\]
Zone after the Winery is Called using POSETs

Extend POSET

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>cw</td>
<td>cw</td>
<td>cw</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>

Project POSET

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cw</td>
<td>cw</td>
<td>cw</td>
<td>0</td>
</tr>
</tbody>
</table>

Initial zone /
Recanonicalize /
Normalize

<table>
<thead>
<tr>
<th>t_0</th>
<th>t_1</th>
<th>t_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

reset

[2, 3]

Call Winery
POSET after the Wine Arrives

Extend POSET

\[ \begin{array}{c|cc}
& cw & wa \\
\hline
cw & 0 & 3 \\
wa & -2 & 0 \\
\end{array} \]

reset

Call Winery

\[ [2,3] \]

Wine Arrives
Zone after the Wine Arrives using POSETs

Initial zone

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_4$</th>
<th>$t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>

Recanonicalize / Normalize

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_4$</th>
<th>$t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$t_2$</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>
**POSET after the Patron is Called**

<table>
<thead>
<tr>
<th></th>
<th>(cw)</th>
<th>(wa)</th>
<th>(cp)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Extend POSET</strong></td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>0</td>
<td>(\infty)</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(cw)</th>
<th>(wa)</th>
<th>(cp)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recanonicalize</strong></td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(wa)</th>
<th>(cp)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project POSET</strong></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- \(\text{reset}\)
- Call Winery
- Wine Arrives
- Call Patron
Zone after the Patron is Called using POSETs

Initial zone

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_4$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Recanonicalize / Normalize

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_4$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Zone after the Rule Expires using POSETs

**Project zone**

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Advance time / Recanonicalize**

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Normalize**

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
POSET after the Patron Arrives

Extend POSET

Recanonicalize

Project POSET

wa  cp  pa

wa | 0  1  ∞
cp | 1  0  ∞
pa | ∞ -5  0

reset

Call Winery

Wine Arrives

Call Patron

Patron Arrives

[2,3]

[2,3]

[2,3]

[5,inf]
Zone after the Patron Arrives using POSETs

Initial zone / Recanonicalize / Normalize

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$t_5$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
POSET after the Wine is Purchased

Extend POSET

\[
\begin{array}{c|cc}
\text{pa} & \text{wp} \\
\hline
0 & 3 \\
-2 & 0 \\
\end{array}
\]

Project POSET

\[
\begin{array}{c|c}
wp & \\
\hline
0 & \\
\end{array}
\]

Reset

Call Winery

\([2,3]\]

Call Patron

\([2,3]\)

Wine Arrives

\([5,\infty]\)

Patron Arrives

\([2,3]\)

Wine Is Purchased

\([2,3]\)
Zone after the Wine is Purchased using POSETs

Initial zone / $t_0$ $t_6$
Recanonicalize / $t_0$ 0 3
Normalize $t_6$ 0 0
POSET after the Patron is Called

Extend POSET

\[
\begin{array}{c|cc}
 cp & cw \\
 0 & -2 \\
 3 & 0 \\
\end{array}
\]

reset

\[ [2,3] \]

Call Winery

\[ [2,3] \]

Call Patron
Zone after the Patron is Called using POSEtS

Initial zone

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>(\infty)</td>
</tr>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Recanonicalize /
Normalize

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$t_1$</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
POSET after the Wine Arrives

<table>
<thead>
<tr>
<th></th>
<th>wa</th>
<th>cp</th>
<th>cw</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Extend POSET</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wa</td>
<td>0</td>
<td>∞</td>
<td>-2</td>
</tr>
<tr>
<td>cp</td>
<td>∞</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>cw</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>wa</th>
<th>cp</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Recanonicalize</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wa</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>cp</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cw</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>wa</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Project POSET</em></td>
<td></td>
</tr>
<tr>
<td>wa</td>
<td>0</td>
</tr>
<tr>
<td>cp</td>
<td>1</td>
</tr>
</tbody>
</table>

reset

\[ [2,3] \]

Call Winery

\[ [2,3] \] \[2,3\]

Wine Arrives Call Patron

Chris J. Myers (Lecture 7: Timed Circuits) Asynchronous Circuit Design 108 / 116
Zone after the Wine Arrives using POSETs

Zone

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_4$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Recanonicalize / Normalize

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_4$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Timed State Space using POSETs

- **Call Winery**: Wine Arrives → Call Patron
- **Wine Arrives**: Call Patron
- **Call Patron**: Wine Arrives
- **Wine Is Purchased**: Patron Arrives
- **Patron Arrives**: rule expires

---

Chris J. Myers (Lecture 7: Timed Circuits)
Wine Shop Example: Reduced State Graph

Asynchronous Circuit Design
Wine Shop Example: Speed-Independent Circuit

[Diagram of a speed-independent circuit with nodes labeled as follows:
- ack_wine
- req_patron
- ack_wine -> CSC0
- req_patron -> CSC0
- ack_patron -> CSC0
- req_wine
- ack_wine
- req_wine
- ack_wine -> CSC0
- req_wine
- ack_wine
]
Wine Shop Example: Timed Circuit

\[\text{ack\_wine} \quad \text{CSC0}\]
\[\text{req\_patron} \quad \text{C}\]
\[\text{CSC0} \quad \text{C}\]
\[\text{ack\_wine} \quad \text{req\_patron}\]
\[\text{ack\_patron} \quad \text{req\_patron}\]

\[\text{req\_wine} \quad \text{ack\_patron} \quad \text{req\_patron}\]
\[\text{ack\_wine}\]
Summary

- Regions
- Discrete-time states
- Zones
- Zones + POSETs
- Timed circuits