For the case that the timed domain is the natural numbers and all the discrete variables range over finite domains, we can use conventional finite-state model checking to verify properties of CTS's. Some minimal modification to the model are necessary:

- We must ensure that \( T \) remains bounded. We can associate an idle transition with all terminating locations, and periodically reset the clocks so that \( T \) decreases modulo some constant.

- We must ensure that \( I \) remains bounded. Either omit \( I \) altogether, or let it be unbounded.

For some properties of CTS's, we can use conventional finite-state model checking to verify them. However, for the case that the time domain is the natural numbers and all the discrete variables range over finite domains, we must ensure that \( T \) remains bounded. We can associate an idle transition with all terminating locations, and periodically reset the clocks so that \( T \) decreases modulo some constant. This modification is necessary to ensure that \( I \) remains bounded. Either omit \( I \) altogether, or let it be unbounded.

### MODULE \texttt{MI}

This module represents process \( P_1 \) of program \texttt{ANY-Y}.

```plaintext
MODULE MI

VAR
  \( T = 0 \ldots 20 \);
  \( T_1 = 0 \ldots 6 \);
  \( T_2 = 0 \ldots 6 \);
  \( y = 0 \ldots 6 \);
  x : boolean;

DEFINE prog = \( T \geq T_1 \).

MODULE L(x,y,t,Low,High)

DEFINE prog := t < High;

VAR
  loc : 0..2;
  \( t_1 = 0 \ldots 7 \);
  \( T_1 = 0 \ldots 7 \);
  \( Y = 0 \ldots 2 \);

ASSIGN

// Initial states
init(loc) := 0;
init(y) := 0;
init(t) := 0;
next(loc) := case
  t < Low: loc;
  loc = 0 & !x: 1;
  loc = 0 & x: 2;
  loc = 1: 0;
  1: loc; esac;
next(y) := case
  loc = 1 & next(loc) = 0: (y + 1) mod 7;
  1: y; esac;
next(t) := case
  t < Low: 0;
  1: t; esac;
```

### MODULE \texttt{Tick}

This module represents process \( T \) of program \texttt{ANY-Y}.

```plaintext
MODULE Tick

VAR
  \( T = 0 \ldots 7 \);
  \( T_1 = 0 \ldots 7 \);

DEFINE prog = \( T \geq T_1 \).
```

### MODULE \texttt{Tick(MI)}

This module represents process \( T \) of program \texttt{ANY-Y}.

```plaintext
MODULE Tick(MI)

VAR
  \( T = 0 \ldots 7 \);
  \( T_1 = 0 \ldots 7 \);

DEFINE prog = \( T \geq T_1 \).
```

### MODULE \texttt{R(x,y,t,Low,High)}

This module represents process \( P_2 \) of program \texttt{ANY-Y}.

```plaintext
MODULE R(x,y,t,Low,High)

DEFINE prog := t < High;

VAR
  loc : 0..1;

ASSIGN

// Initial states
init(loc) := 0;
init(x) := 0;
init(t) := 0;
next(loc) := case
  t < Low: loc;
  loc = 0: 1;
  1: loc; esac;
next(x) := case
  loc = 0 & next(loc) = 1: 1;
  1: x; esac;
next(t) := case
  t >= Low: 0;
  1: t; esac;
```

### MODULE \texttt{PROC (x,y,Low,High)}

This module represents process \( P \) of program \texttt{ANY-Y}.

```plaintext
MODULE PROC(x,y,Low,High)

DEFINE prog := t < High;

VAR
  \( T = 0 \ldots 20 \);
  \( T_1 = 0 \ldots 7 \);

ASSIGN

// Initial states
init(x) := 0;
init(y) := 0;
init(t) := 0;
next(x) := case
  t < Low: 0;
  1: x; esac;
next(t) := case
  t < Low: 0;
  1: t; esac;
```

### MODULE \texttt{Any-Y}

This module represents process \( P \) of program \texttt{ANY-Y}.

```plaintext
MODULE Any-Y

ASSIGN

// Initial states
init(x) := 0;
init(y) := 0;
init(t) := 0;
```

This module represents process \( P \) of program \texttt{ANY-Y}.

```plaintext
MODULE Any-Y

ASSIGN

// Initial states
init(x) := 0;
init(y) := 0;
init(t) := 0;
```
This module represents the tick transition. Parameter prog is true if all processes allow time to progress. To simplify matters, the tick transition always increments all clocks by 1.

\[
\text{MODULE Tick}(\text{Prog}, t_1, t_2, T)
\]

\[
\begin{align*}
\text{ASSIGN init}(T) & := 0; \\
\text{next}(t_1) & := \text{case} \begin{cases} 
\text{Prog}: t_1 + 1; \\
1: t_1; 
\end{cases} \\
\text{next}(t_2) & := \text{case} \begin{cases} 
\text{Prog}: t_2 + 1; \\
1: t_2; 
\end{cases} \\
\text{next}(T) & := \text{case} \begin{cases} 
\text{Prog}: (T + 1) \mod 21; \\
1: T; 
\end{cases}
\end{align*}
\]

\[
\text{JUSTICET}=0, T \neq 0
\]

The algorithm is correct for any time bounds \([L; U]\) such that \(2L > U\). In our case, we run it with \(T = 9\) and \(L = 6\). This guarantees that the correct timing constraints imposed on the system.

The next example shows an interesting case where mutual exclusion is guaranteed due to the correct timing constraints imposed on the system.

**Fischer’s Mutual Exclusion Program**

```
FILE Fischer.smv

MODULE MI

process Tick(prog, T, low, high)!
process Idle()
process proc(1, x, t_1, Low, High)
process proc(2, x, t_2, Low, High)
process proc(1, x, t_1, low, high)
process proc(2, x, t_2, low, high)

VAR

prog = 0
high = 0
low = 0

MODULE main

DEFINE Low := 3;
High := 5;
Prog := P[1].prog & P[2].prog;
VAR

x: 0..2;
t_1: 0..6;
t_2: 0..6;
T: 0..7;
P[1]: process proc(1, x, t_1, Low, High);
P[2]: process proc(2, x, t_2, Low, High);
Idle: process MI;
tick: process Tick(Prog, t_1, t_2, T);

MODULE MI

process Tick(prog, T, low, high)!

\[
\begin{align*}
0 = x : 6
\end{align*}
\]

```

The two properties we model check for time bounds \([3; 5]\) are

- Property 2: states that variable \(y\) never rises above the value of 1. This is an example of a property which does not mention time explicitly but is valid only due to the timing constraints.

The next examples show an interesting case where mutual exclusion is guaranteed due to the correct timing constraints imposed on the system.
Lecture 8: Model Checking Discrete CTS

A. Pnueli

Module

MODULE proc(id, x, t, Low, High)

DEFINE prog := t < High;

VAR loc : 1..9;

ASSIGN

init(loc) := 1;
init(x) := 0;
init(t) := 0;

next(loc) :=
    case
    t < Low : loc;
    loc = 1 : {1, 2};
    loc in {2, 5, 6, 9} : (loc mod 9) + 1;
    loc = 3 & x != id : 4;
    loc = 3 : 9;
    loc = 4 & x = 0 : 5;
    loc = 7 & x = id : 8;
    loc in {7, 8} : 3;
    1 : loc;
    esac;

next(x) :=
    case
    loc = 5 & next(loc) = 6 : id;
    loc = 9 & next(loc) != loc : 0;
    1 : x;
    esac;

next(t) :=
    case
    >= Low : 0;
    1 : t;
    esac;

Tick

MODULE Tick(Prog, t_1, t_2, T)

ASSIGN

init(T) := 0;

next(t_1) :=
    case
    Prog : t_1 + 1;
    1 : t_1;
    esac;

next(t_2) :=
    case
    Prog : t_2 + 1;
    1 : t_2;
    esac;

next(T) :=
    case
    Prog : (T + 1) mod 8;
    1 : T;
    esac;

Justice

T = 0, T != 0

The properties we wish to check for program Fischer are:

(! (P[1]: loc = 8 & P[2]: loc = 8))

Mutual Exclusion

(P[1]: loc = 2 ! (P[1]: loc = 8))

Accessibility

The properties follow:

Running the program with L = 3 and U = 5, both properties are invalid.

Running the program with L = 2 and U = 5, both properties are valid.

Counter Example to Mutual Exclusion

Counter Example to Mutual Exclusion
Lecture 8: Model Checking Discrete CTS

A. Pnueli

Counter Example to Accessibility:

From $h = 5$ to $h = 3$

```
loop forever do
    1: noncritical
    2: skip
    3: while $x \neq 1$ do
        4: await $x = 0$
        5: $x := 1$
        6: skip
        7: if $x = 1$ then
            8: critical
            9: $x := 0$
```

```
loop forever do
    1: noncritical
    2: skip
    3: while $x \neq 2$ do
        4: await $x = 0$
        5: $x := 2$
        6: skip
        7: if $x = 2$ then
            8: critical
            9: $x := 0$
```

St. 29 = $x:2, t_1:0, t_2:2, T:7, loc_1:3, loc_2:3$

St. 30 = $x:2, t_1:0, t_2:0, T:7, loc_1:3, loc_2:9, tick:2$

St. 31 = $x:2, t_1:2, t_2:2, T:0, loc_1:3, loc_2:9$

St. 32 = $x:0, t_1:2, t_2:0, T:0, loc_1:3, loc_2:1, tick:2$

St. 33 = $x:0, t_1:4, t_2:2, T:2, loc_1:3, loc_2:1$

St. 34 = $x:0, t_1:4, t_2:0, T:2, loc_1:3, loc_2:2, tick:1$

St. 35 = $x:0, t_1:5, t_2:1, T:3, loc_1:3, loc_2:2$

St. 36 = $x:0, t_1:0, t_2:5, T:5, loc_1:5, loc_2:5$

St. 37 = $x:0, t_1:5, t_2:5, T:5, loc_1:5, loc_2:5$

St. 38 = $x:0, t_1:5, t_2:5, T:5, loc_1:5, loc_2:5$

```
loop forever do
    1: noncritical
    2: skip
    3: while $x \neq 1$ do
        4: await $x = 0$
        5: $x := 1$
        6: skip
        7: if $x = 1$ then
            8: critical
            9: $x := 0$
```

```
loop forever do
    1: noncritical
    2: skip
    3: while $x \neq 2$ do
        4: await $x = 0$
        5: $x := 2$
        6: skip
        7: if $x = 2$ then
            8: critical
            9: $x := 0$
```

Timed and Hybrid Systems, NYU, Spring 2007