Monitoring Real-Time Properties in Copilot

Lauren Pick

August 20, 2015

Copilot Streams and Iterations

- Copilot manipulates and receives data in the form of streams
  - The nth value of a stream s is denoted as $s(n)$
- Copilot programs have a cyclic behavior, where each iteration consists of
  1. sampling externs
  2. updating internal variables
  3. firing external triggers

  - So for a stream s, the nth iteration provides the value $s(n)$, where the first iteration is the 0th iteration.
  - For a Copilot program that has been compiled into C, an iteration occurs each time the `step()` function is called.

Monitoring Properties Involving Real-Time with Copilot

- Given a knowledge of a constant sampling rate for externs of a monitor, real-time properties can be monitored, since each iteration would correspond to a known, constant amount of time elapsing.
  - e.g. if the sampling period is known to be 1 second:

```
bools = extern "bools" Nothing
-- 1 second delay:
bools' :: Stream Bool
bools' = [False] ++ bools
```

- Monitoring properties involves checking if the current and past values in streams match the values that would be expected if the property held
  - As a result, it may be necessary to delay streams or express the property in terms of past values if the original property involves future values

A Property Involving Real-Time

A property that is to be monitored assuming that the global sampling period is 0.1 second:

A sampled temperature value should not exceed the threshold temperature *thresh* for 0.3 second or longer.

- If *over* is the stream of booleans indicating whether or not the temperature *temp* exceeds *thresh* ($temp > thresh$), then this means that for all n, it is not the case that

$$\begin{array}{lllllllll} over(n+3) & = & \texttt{true} & \wedge & over(n+2) & = & \texttt{true} & \wedge \\ over(n+1) & = & \texttt{true} & \wedge & over(n) & = & \texttt{true} & \end{array}$$

- or, if expressed in terms of past values, then for all $n > 2$, it is not the case that

$$\begin{array}{lllllllll} over(n) & = & \texttt{true} & \wedge & over(n-1) & = & \texttt{true} & \wedge \\ over(n-2) & = & \texttt{true} & \wedge & over(n-3) & = & \texttt{true} & \end{array}$$

Monitoring Properties Involving Real-Time with Copilot

```
propTempExceedThreshold :: Spec
propTempExceedThreshold =
  trigger "over_threshold" (overThresh) []

  where
  thresh = 500

  over :: Stream Bool
  over = [False, False, False, False] ++ (temp > thresh) -- delay

  temp :: Stream Float
  temp = extern "temp" Nothing

  overThresh :: Stream Bool
  overThresh = drop 3 over &&
               drop 2 over &&
               drop 1 over &&
               over
```

Monitoring Properties Involving Real-Time with Copilot

```
propTempExceedThreshold :: Spec
propTempExceedThreshold =
  trigger "over_threshold" (overThresh) []

  where
  thresh = 500

  over :: Stream Bool
  over = temp > thresh

  temp :: Stream Float
  temp = extern "temp" Nothing

  overThresh :: Stream Bool
  overThresh = over &&
               ([False] ++ over) &&
               ([False, False] ++ over) &&
               ([False, False, False] ++ over)
```

The LTL and PTLTL libraries

- The LTL library implements bounded Linear Temporal Logic
  - E.g., the property that $\Diamond^n p$ can be expressed as `eventually n b`, where `b` is a boolean stream indicating whether p currently holds
  - When using the library's functions, it is necessary that the streams have sufficient history to "drop" from
- The PTLTL library implements past time Linear Temporal Logic
  - Other than `previous`, all functions provided in the PTLTL library range over the entirety of a streams' history
    - E.g., the property $\overleftarrow{\Diamond} p$ can be expressed as `eventuallyPrev b`
- Using these libraries, properties such as the one mentioned previously become easier to express within a monitor
  - Using bounded LTL and the assumed 0.1 second sampling period, the property can be written as

$$\neg \Box^3 (\texttt{temp} > \texttt{thresh})$$

Monitoring Real-Time Properties with Bounded LTL

```
propTempExceedThreshold :: Spec
propTempExceedThreshold =
  trigger "over_threshold" (overThresh) []

  where
  thresh = 500

  over :: Stream Bool
  over = [False, False, False, False] ++ (temp > thresh) -- delay

  temp :: Stream Float
  temp = extern "temp" Nothing

  overThresh :: Stream Bool
  overThresh = always 3 over
```

Drawbacks of Non-Explicit Real-Time

In the monitors for the `propTempExceedThreshold` specification so far, any connection to real-time values was implicit; the monitors were stated to assume a constant sampling period of 0.1 second.

- If the sampling period is changed, then it is then necessary to alter the monitors each time this occurs
- If samples do not occur with exactly 0.1 second in between (e.g. if one sample is, for some reason, 0.1 second late), then the properties are no longer monitored correctly

Metric Temporal Logic

Metric Temporal Logic is an extension to Linear Temporal Logic where operators are augmented with *intervals* over which the properties should hold.

- An implementation of bounded MTL in Copilot that uses iterations as time units rather than real-time units would resemble the bounded LTL library but with each function accepting a lower bound in addition to the upper bound
- If the lower bound were set to zero, then the result would be the same as the LTL library could provide; e.g., given an integer value `n` and streams `s0` and `s1`, the following would result in the same streams:
  - `until n s0 s1`
  - `until 0 n s0 s1`

Making Time More Explicit

- The most straightforward way to incorporate real-time values into monitors is by sampling the clock, i.e., introducing a `clk` stream
- By adding `clk` and `dist` (the minimum sampling period) parameters to MTL functions, intervals can refer to real-time values instead of iterations
- So MTL properties such as $\mathbf{R}_{[l,u]}(p_1, p_2)$ can be expressed as `release l u clk dist b1 b2`

A Property Involving Real-Time

In MTL, the property that the temperature should not exceed the threshold temperature for 0.3 second (300 ms) or longer can be written as:

$$\neg \, \Box_{[0,300]}(\texttt{temps} > \texttt{threshold})$$

For the purpose of monitoring, this property should instead be expressed in terms of a past-time operator:

$$\neg \, \overleftarrow{\Box}_{[0,300]}(\texttt{temps} > \texttt{threshold})$$

Monitoring Real-Time Properties with MTL

```
propTempExceedThreshold :: Spec
propTempExceedThreshold =
  trigger "over_threshold" (overThresh) []

  where
  thresh = 500

  temp :: Stream Float
  temp = extern "temp" Nothing

  clk :: Stream Word64
  clk = extern "clk_ms" Nothing

  overThresh :: Stream Bool
  overThresh = alwaysBeen 0 300 clk 100 (temp > thresh)
```

Monitoring Real-Time Properties with MTL

- No longer the requirement that the global sampling period is exactly 0.1 second
  - The monitor instead assumes that the sampling period for all the streams used within the specification is *at least* 0.1 second
  - Increasing the global sampling period to e.g. 0.15 second requires no change to the monitor
- May set off "false alarms" until 300 ms pass
  - Can prevent this at the expense of complicating the monitor slightly, e.g. changing the `trigger` to

```
trigger "over_threshold" (overThresh && canTrigger) []
```

where

- `canTrigger = (clk - hold(clk)) >= 300`
- ```
  hold s = s'
    where s' = mux ([True] ++ false) s ([0] ++ s')
  ```
  ```

Summary of Logic Libraries in Copilot

- LTL
  - an implementation of bounded Linear Temporal Logic
  - the parameter `n` provides the number of *iterations* over which the property should hold
- PTLTL
  - an implementation of past-time Linear Temporal Logic
  - properties should hold over the current and all previous sampled values
- MTL
  - an implementation of bounded Metric Temporal Logic with
    - bounds l, u that must be nonnegative integers
    - a time domain consisting of all time values sampled
  - the parameters `l` and `u` provide the *real-time values* for which the property should hold