Synthesize the ABP (Alternating Bit Protocol)
A challenge problem in decentralized controller synthesis

Stavros Tripakis
UC Berkeley

NSF ExCAPE PI meeting – June 2012
ABP: **reliable transmission** over an **unreliable channel**

Channels are lossy but FIFO.
Challenge problem: *synthesize the ABP automatically!*

Why:
- Warm up
- Gauge state of art
- Education
- Instance of generic problem

Channels are lossy but FIFO.
Can be formalized as a decentralized controller synthesis problem.
Can be formalized as a decentralized controller synthesis problem
Input: the **Sending and Receiving Clients**

(Actual input may be in your favorite formalism/language)

Sending client:

Receiving client:
In Promela / Spin

proctype SendClient()
{
    do
    :: atomic{ msg!0; pending = 1 } ->
        atomic{ done?0; pending = 0 } 
    od
}

proctype ReceiveClient()
{
    do
    :: atomic{ deliver?0; pending = 2 } 
    od
}
Input: the **Forward and Backward Channels**

(Actual input may be in your favorite formalism/language)

**Forward channel:**

**Backward channel:**
In Promela / Spin

proctype Channel(chan input, output; byte id)
{
    bool x;
    do
        :: input?x ->
            if
            :: atomic{ lost[id] = 0 -> output!x } 
            :: atomic{ lost[id] = 1 -> skip }  // msg lost
        fi
    od
}
Input: the **Specification**

(Actual input may be in your favorite formalism/language)

\[
\text{fair\_channels} \Rightarrow \text{request\_response}
\]

\[
\text{fair\_channels} = \text{do not lose messages forever}
\]

\[
\text{request\_response} = \Box (\text{msg} \Rightarrow \Diamond (\text{deliver} \land \Diamond \text{done}))
\]
#define fair_channels
( (!(!( <> [] lost[0])) ) && (!(!(! <> [] lost[1]))) )

#define req_resp
([]((pending==1) -> (<>((pending==2)&&(<>((pending==0)))))))

ltl abp { fair_channels -> req_resp }
Output: the ABP
(Alternating Bit Protocol)

ABP sender:
(left controller)

ABP receiver:
(right controller)
proctype ABP_Sender(chan request, response, output, input)
{
    bool alt_bit = 0;  /* alternating bit */
    bool ack;         /* expected acknowledgment */
    do
        :: request?0;
        output!alt_bit;
        do
            :: input?ack;
                if :: ack == alt_bit /* correct acknowledgment received */
                    -> response!0
                    -> alt_bit = !alt_bit
                    -> break
                :: ack != alt_bit /* incorrect acknowledgment received, consume it and do nothing */
                    fi
            :: timeout -> output!alt_bit /* retransmit */
        od
    od
}

proctype ABP_Receiver(chan delivery, input, output)
{
    bool alt_bit = 0;  /* alternating bit */
    bool seq = 0;      /* received sequence number */
    do
        :: input?seq;
            if :: seq == alt_bit /* correct seq. number received */
                -> output!alt_bit /* send acknowledgment */
                -> delivery!0     /* deliver to receiving client */
                -> alt_bit = !alt_bit
            :: seq != alt_bit /* incorrect seq. number received */
                -> output!seq     /* resend ack */
            fi
    od
}
Challenges (?

- Decentralized control (and observation) problems are undecidable in general [Pnueli-Rosner’90, Lamouchi-Thistle’00, Tripakis’01]
  - Note: all inputs are finite-state
- Getting the specification right => getting the right controllers
- Representation of controllers
- Understanding the result
- …
- Applications to ExCAPE challenge domains:
  - Multicore protocols
  - Robotics
  - Concurrent programs
  - Network protocols
References

• Pnueli, Rosner, 1990, “Distributed Reactive Systems are Hard to Synthesize” http://dl.acm.org/citation.cfm?id=1398817
• Lamouchi, Thistle, 2000, “Effective control synthesis for DES under partial observations” http://dl.acm.org/citation.cfm?id=1654806