Automated Synthesis of Protocols from Flows

Murali Talupur
SCL, Intel
Jun 10 2013
The Flows2HLM Method

• A novel way of generating executable and formal models
  – Easy to use
  – Leverages information already present in MAS
  – Enables very quick modeling

• Addresses the long standing problems
  – Viable way of generating formal models
  – Enabling early validation
• Validation can begin in earnest only when both RTL and BFM are ready

• Differing interpretations possible at validation and design ends
A better way

• Create executable, high level model which is formally analyzable (HLM)
  – Validation can begin early
• Leverage this to create validation IP
• Designers benefit from having correct, unambiguous reference model

**But no viable way to create high level formal models**
Enter Flows

• We can synthesize formal executable models from Flows
• Flows are partial orders on system events
  – Such as sending and receiving of messages
  – Provide natural descriptions for message passing systems
• Readily available in MAS documents
  – Architects think in these terms
    • Flows in fact can be thought of as specifying the protocol
  – Succinct and easy to understand
    • Flows refer to high level events mainly
    • Even a large protocol has about 40 odd Flows which is orders of magnitude smaller than the protocol itself

Two Flows from German’s protocol
Flows with state annotations

• Annotate each event with:
  – A **guard** enabling that event
  – And **updates** to states variables
• Both are Murphi snippets
• Flow language has 3 additional fields:
  – What messages to **recv**
  – What messages to **send**
  – Which **agent** is executing

Event **e2**:
- **agent**: Dir
- **recv**: (ReqS, i)
- **guard**: CurrCmd = Empty
- **update**: CurrPtr := i
- **send**: (GntS, i)
Each event converted to a Murphi rule
- Special functions for sending and receiving messages
- Track where an agent is in a particular flow
- Guards and updates copied into the Murphi rule

```plaintext
ruleset i: cbox do
    rule "e2"
        CurrCmd = Empty & Recv(ReqS, i) & Check_txn_buffer(i)
    =>
    Begin
        CurrPtr := i;
        Send(GntS, i, tracking_info);
        Update_txn_buffer(i);
    End
```

Corresponding Murphi rule
Applications

- Two large cache coherence protocols
  - Both from real designs
  - 5-6K lines of Murphi from ~40 Flows
  - Modeling done in just couple of weeks
    - Not counting tool development time
- One low level link layer protocol and one bus lock protocol
  - Both from real designs as well
  - 1-2K lines of Murphi from 3-6 Flows
Advantages

• Book keeping is all free
  – Lots of boring code avoided and focuses on those aspects which are most important to architects
    • *Less chance of making mistakes*

• Amount of annotation we have to add is tiny compared to the final protocol size
  – 1:20 ratio typically
  – Consistent with our earlier project on synthesizing invariants from Flows
    • Which was the inspiration for this project in the first place
Advantages

• We don’t break the natural organization of a protocol
  – Describe what happens within a single logical transaction
  – *No context switching between Flows*
    • We just have to describe what an agent does in a given scenario
    • In table based and direct coding, we have to consider what an agent can do in different scenarios at once

• Intuitive and easy to understand
  – Easy to modify and maintain
  – We can give enriched flows back to the architect
    • Or tables or Murphi model as they require

*Flow based modeling allows us to go after large designs and a wide variety of designs*
With Flows based Modeling

Low effort!  

Leverage

Clean and unambiguous

Getting executable formal models written in Murphi or other high level languages is hard

Our project shows a way around this by working directly with Flows

– Flows are already available in the MAS documents
– Additional state annotations are easy to supply