Sample Lecture on System Modeling

- **Important simplifications** in building formal models of real-life systems
  - Discussed in steps throughout the lecture.
  - Summarized in second last slide.
- Discusses a **concrete example** (of a cache coherence protocol)
  - Modeled in the input language of a model checking tool SMV.

A Full Case Study using SMV

CS 4271

Abhik Roychoudhury

So far …

- **Basics of modeling**
  - Includes details of SMV syntax
- **Toy examples**
  - ABP, Traffic Light Controller
- **Motivational practical applications**
  - AMBA AHB protocol
- We need to model/verify a medium sized problem completely to get the feel

Case Studies

- Many well-publicized successes of Model Checking in the verif. of processors / cache coherence protocols
  - Encore Gigamax Cache coherence protocol
  - IEEE FutureBus+ standard
  - T9000 virtual channel processor
- Our case study is a slightly more software centric coherence protocol, one for distributed file systems

Note of Caution

- Following slides use CMU SMV syntax which is a bit different from Cadence SMV syntax.
  - This has been done to ensure uniformity with the reading material.
  - The syntax differences are however minimal e.g.
    - MODULE x(…) instead of MODULE x(…)
    - VAR {

Before starting …

- SMV employs symbolic model checking
  - More space efficient than the explicit state MC algorithm which proceeds by graph search.
  - Uses a data structure called Binary Decision Diagrams for compact internal representation of state space.
  - This will be covered in later lectures, but you do not need to understand symbolic MC for modeling and verifying using SMV.

Success of Model Checking

- Primarily in hardware verification
- Routinely used for processor verification (or modules of it) in Intel etc
- Employed by CAD giants and processor design companies
  - Cadence, Intel, Motorola …

Verification or Bug Hunting

- Model Checking verifies only the design, not the implementation
- So, MC is more of a bug detection mechanism
  - But "Automated" bug detection
- This line will be more acceptable to practitioners than presenting it as a formal verification technique.

Reference

- A Case Study in Model Checking Software Systems
  - Jeanette Wing and M. Vaziri-Farhana
  - FSE 1995, CMU Tech report CMU-CS-96-124
  - Another version in "Science of Computer Programming", v20, 1997, 273-299.

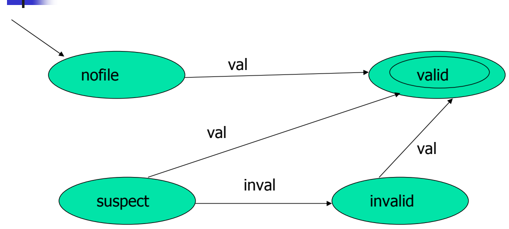
File System Cache coherence

- Several clients and servers
- Each file is authorized by a single server
- Clients cache files on obtaining them from the server
- Clients talk directly only to the server, not to other clients.

The Cache Coherence Problem

- A client can have a copy of a file, but
  - It may not be sure that it is the latest copy
  - It knows that it is an invalid copy
- The client will
  - Request for a validation from the server
  - Request a fresh copy

Client's belief about its copy



Client receives val or inval from server

Client – State Variables

- out – request to the server.
  - This can be a request to
    - Fetch a file copy
    - Validate a file copy
- belief
  - the current belief of the client about the status of its cached copy.

Simplification # 1

- Do not model contents of shared files.
  - The file contents does not affect the coherence mechanism.

Client – State Variables

- Possible values of the "belief" variable
  - Valid
    - the client knows that the cache copy is valid
  - Invalid
    - the client knows that the cached copy is invalid
  - Suspect
    - the client is not sure of the status of the cached copy
  - Nofile
    - client does not have a cached copy

Simplification # 2

- In reality
  - The client has a status for every shared file
- We model
  - Only one shared file, since the coherence issues of each file is independent
    - Hence modeling one server is enough.

Client SMV description

```
MODULE client(input)
VAR
        out: {0, fetch, validate};
        belief: {valid, invalid, suspect,nofile};
ASSIGN
         …
```

- Receives input from server
- out and belief are state variables

Client- Trans. Rel. in SMV

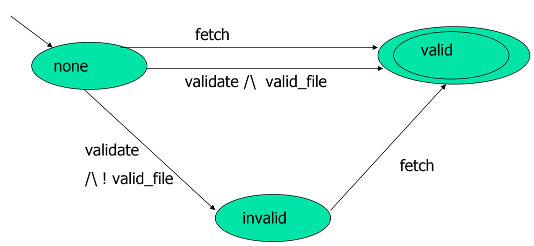
```
ASSIGN
        init(out) := 0;
        next(out) := case
                   (belief = nofile) : fetch;
                   (belief = invalid) : fetch;
                   (belief = suspect): validate;
                   1: 0;
         esac
…
```

Client – Trans. Rel. in SMV

```
init(belief) := nofile, suspect;
next(belief):=
case
   (belief=nofile)&(input=val): valid;
   (belief=suspect)&(input=val):valid;
   (belief=suspect)&(input=inval): invalid;
   (belief=invalid)&(input=val): valid;
     1  : belief
esac
```

- `belief` is updated based on current value, and input from server.

Server – State Diagram



Server receives fetch, validate from Client.

Server – State Variables

- belief : the belief of the server about the status of a cached copy in a particular client
  - Valid
  - Invalid
  - None : Server has no knowledge (the client might or might not have a cached copy)
- For each client i, the server should model a belief[i]
  - But we will simplify further and model only one client !!

Server – State Variables

- out : the output of the server to the client
  - val: indicates to the client that its cached copy is valid
  - inval : indicates to the client that its cached copy is invalid
  - 0 : default output. Ignored by client.

Simplification # 3

- Server has another state variable valid_file
  - Boolean variable
- Models updates to the server by other clients
  - Which need not be modeled explicitly
- Suppose C.belief = suspect
  - Then C sends a validate message to Server
- If Server has received a file update message by another client C' by now
  - Then server deems cached copy of C as invalid
  - Else …

Modeling other clients

- Whether Server receives a new update from another client is modeled by the variable valid_file
  - Set non-deterministically since we do not model the other clients explicitly
- Modeled one client, and exploited the star topology to implicitly model the effects of other clients on the server
  - Drastically cuts down the state space.

Simplification #4 (uncommon)

- Our modeling considers only finite traces
  - "Final" states in Client and Server
  - Restricted modeling of a single session
  - But can be used for verifying invariant properties
    - AG(boolean formula of atomic propositions)
  - If we still find a violation of such properties they would have occurred in the generic modeling with infinite execution traces as well
    - The finite execution traces modeled are possible prefixes of the actual traces if the system was modeled in details

System description in SMV

```
MODULE main
VAR
     Client : client(Server.out);
     Server: server(Client.out);
```

- Client module already presented
- Let us look at the Server.

Server : SMV description

```
MODULE server(input)
VAR
     out         : {0, val, inval};
     belief      : {none, valid, invalid };
     valid_file  : boolean;
ASSIGN
     valid_file := {0,1};
     …
```

Server's Transition Relation

```
init(belief) := none;
next(belief):=
case
  (belief=none)&(input=fetch): valid;
  (belief=none)&(input=validate)&valid_file:valid;
  (belief=none)&(input=validate)& !valid_file:
                                            invalid;
  (belief=invalid)&(input=fetch): valid;
    1  : belief
esac
```

Server's Transition Relation

```
init(out) := 0;
next(out):=
case
  (belief=none)&(input=fetch): val;
  (belief=none)&(input=validate)&valid_file:
                                          val;
  (belief=none)&(input=validate)& !valid_file:
                                          inval;
  (belief=invalid)&(input=fetch): val;
    1  : 0
esac
```

Properties to Verify # 1

- If client C believes cached copy of file f is valid, it does not go to server
  - In this case, server also should believe that C's cached copy of f is valid
  - AG (C.belief=valid ⇒ S.belief = valid)
    - Always if C.belief=valid then S.belief = valid
      - **We will learn about Temporal Logics in the next lecture at a formal level.**
    - Verified to be true by CMU SMV

Property # 2

- We can also check
  - AG(S.belief = valid ⇒ C.belief = valid)
  - Otherwise, client might sometimes unnecessarily go to the server for validation
    - Inefficiency : additional traffic from client to server !!
  - SMV produces a counter-example.

SMV Counter-example

- **state 1.1**
- C.out =0, C.belief = nofile
- S.out =0, S.belief =none, S.valid_file = 0
- **State 1.2**
- C.out = fetch
- **State 1.3**
- S.out = val, S. belief = valid

SMV counterexample

- If client C does not have a copy of a shared file (cache miss), it requests from the server via fetch
- Server S sends a fresh copy and updates its belief about the status of cached copy at C
- Due to the transit delay between S and C, the client still has not updated its belief, but it will do in a few steps
  - This leads to the counter-example
  - Not a cause of concern in terms of additional traffic from the client to the server.

Some useful simplifications …

- … employed in today's case study
  - Do not model data of the data items
    - Files in this case !
  - Model only a single data item
    - Hence model a single server
  - Model only one client
    - Other clients modeled implicitly by considering their effect on the server.

Property specification

- We have not learnt about the Property Specification Language
  - Temporal Logics
  - In two flavors
    - Linear Time (LTL)
    - Branching Time (CTL)
- In next class …