CHESS

CENTER FOR HYBRID & EMBEDDED SOFTWARE SYSTEMS

COLLEGE OF ENGINEERING, UC BERKELEY

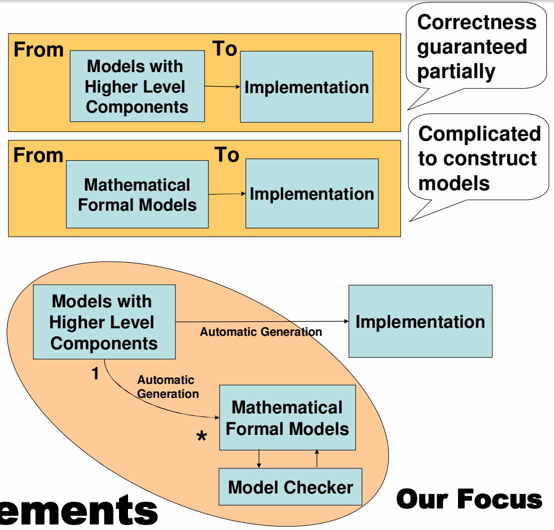
Specification and Formal Verification of Real Time Systems under Ptolemy II

Chihhong Cheng
Edward A. Lee
{patrickj,eal}@eecs

http://chess.eecs.berkeley.edu/

Background

In the context of embedded software design, one challenge is to mediate the gap between design facilitation and verification complication. We found that existing theories and practices in verification are powerful, but when applying formal techniques, it would greatly release the burden of system designers if detailed and complex mathematical models used for verification are hidden; construction of such models may be time consuming and error prone.

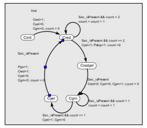

From
Models with Higher Level Components
To
Implementation
From
Mathematical Formal Models
To
Implementation
Correctness guaranteed partially
Complicated to construct models
Models with Higher Level Components
Automatic Generation
Implementation
1
Automatic Generation
*
Mathematical Formal Models
Model Checker
Our Focus

Goals and Achievements

We provide an automatic mapping from higher level components (actors) commonly used to lower level mathematical models; the conversion preserves behavior semantics. With this methodology, the productivity of designers as well as the correctness of designs can be maintained simultaneously.

Automatic conversion from FSMActors to Kripke Structures in the SR domain

FSMActors can be viewed as extended state machines, which enable the existence of inner variables. We are to convert multiple FSMActors in the SR (Synchronous Reactive) domain into Kripke structures accepted by model checker NuSMV [3]. We can also deal with ModalModels with state refinements.



FSMActors in Ptolemy II

```
MODULE CarLightNormal( Sec_isPresent )
VAR
   state : {Credyel,Cgrn,Cinit,Cred,Cyel};
   count : { ls,0,1,2,gt };
ASSIGN
   init(state) := Cinit;
   next(state) :=
      case
         state=Cinit & count=ls :{ Cred };
         state=Cinit & count=0 :{ Cred };
...
...
```

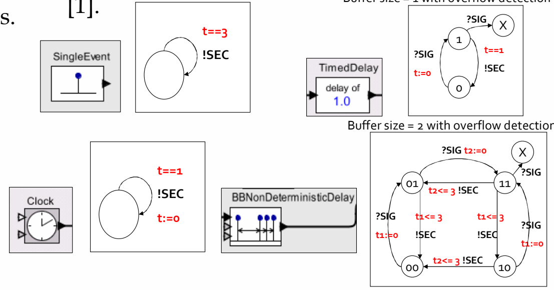
.smv format of the equivalent FSMActor

Incorporating FSMActors with Other Actors in Libraries under the DE domain

The DE (Discrete Event) domain is one of the most useful MoCs capturing the behavior of real time systems. We study the parts of the actors that can not be represented by simple FSMs; we are able to characterize their interplay using the theory of Timed Automata [1].

Buffer size = 1 with overflow detection

Buffer size = 2 with overflow detection



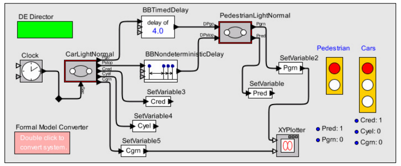
Case Study: Traffic Light System in SR

We construct a simplified traffic light system in the SR domain, where the system contains of one car light and one pedestrian light. Our design should make it impossible to have the car light and pedestrian light be green at the same time (this might lead to accidents). We use a model checker to test whether our design satisfies the specification.

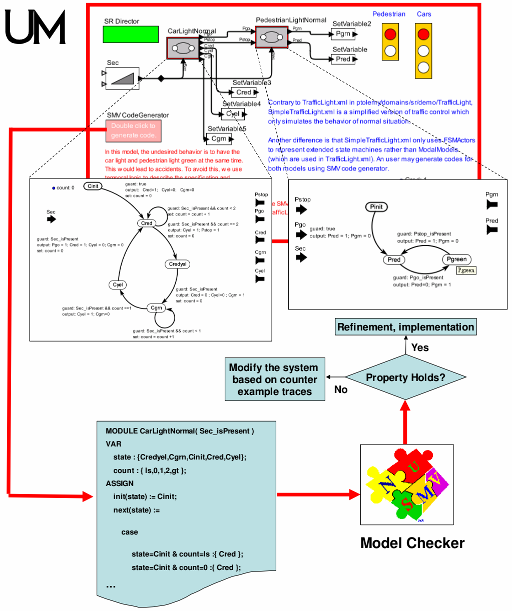
Formula: !EF (CarLightNormal.state = Cgrn & PedestrianLightNormal.state = Pgreen)

Case Study: Buffer Overflow Detection in DE

Buffer overflow detection has been an important issue in the design of embedded systems. Our conversion process enables us to configure parameters of buffer size for each component, and use model checkers to detect the undesired behavior. Also we can describe more interesting properties using TCTL [2].



Port name	Buffer Size
CarLightNormal_Sec	2
BBTimedDelay_Pgo	2
BBNondeterministicDelay_Pstop	3
PedestrainLightNormal_DPgo	2
PedestrainLightNormal_DPstop	2



In this model, the undesired behavior is to have the car light and pedestrian light green at the same time. This would lead to accidents. To avoid this, we use temporal logic to describe the specification and

Contrary to TrafficLight.xml in ptolemy/domains/sr/demo/TrafficLight, SimpleTrafficLight.xml is a simplified version of traffic control which only simulates the behavior of normal situation.

Another difference is that SimpleTrafficLight.xml only uses FSMActors to represent extended state machines rather than ModalModels (which are used in TrafficLight.xml). An user may generate codes for both models using SMV code generator.

```
MODULE CarLightNormal( Sec_isPresent )
VAR
   state : {Credyel,Cgrn,Cinit,Cred,Cyel};
   count : { ls,0,1,2,gt };
ASSIGN
   init(state) := Cinit;
   next(state) :=

      case

         state=Cinit & count=ls :{ Cred };
         state=Cinit & count=0 :{ Cred };

...
```

Conclusion and Future Work

In this work, we offer design with verification by translating component models into the mathematical models needed for verification. This work opens interesting problems for future study in parametric safety analysis and rate control analysis.

References

[1] R. Alur and D.L. Dill, *A Theory of Timed Automata*. Theoretical Computer Science, 1994. **126**(2): p. 183-235.

[2] R. Alur, C. Courcoubetis, and D. L. Dill, “Model checking for realtime systems,” in *Proc. 5th Annu. IEEE Symp. Logic in Computer Science*, 1990, pp. 414–425.

[3] A. Cimatti, et al. *Nusmv 2: An opensource tool for symbolic model checking*. in *Computer Aided Verification*. 2002: Springer-Verlag.