Embedded Hybrid Systems

Prof. T. John Koo

Embedded Computing Systems Laboratory
Institute for Software Integrated Systems
Department of Electrical Engineering and Computer Science
Vanderbilt University
USA
Collaborators

- **Researchers**
  - Gabor Karsai (Vanderbilt University), Janos Sztipanovits (Vanderbilt University), Shankar Sastry (UC Berkeley), Edward Lee (UC Berkeley), Tom Henzinger (UC Berkeley/EPFL), Alberto Sangiovanni-Vincentelli (UC Berkeley), Claire Tomlin (UC Berkeley/Stanford University), George Pappas (University of Pennsylvania), Ian Mitchel (University of British Columbia), Thao Deng (VERIMAG), Datta N. Godbole (Honeywell Technology Center), Zhi Han (Mathworks) and Sandeep Neema (ESCHER)

- **Students (Current/Former)**
  - Jie Chen, Chip Clifton, Abhishek Dubey, Graham Hemingway, Hang Su and Xianbin W
Funding Support

■ Current
  ■ National Science Foundation Faculty Early Career Development (CAREER) Program
  ■ National Science Foundation Information Technology Research Center Program
  ■ Department of Electrical Engineering and Computer Science, Vanderbilt University

■ Former
  ■ DARPA Information Technology Office Software Enabled Control (SEC) Program
  ■ DARPA Information Technology Office Model Based Integration of Embedded Software (MoBIES) Program
Outline

- Embedded Systems: Overview
- Embedded Systems: Modeling
- Embedded Hybrid Systems: Analysis
- Embedded Hybrid System: Design Tool
- Conclusion
Embedded Systems: Overview

- Embedded Systems
  - are specific purpose computing systems that interact with the environment in which the dynamics are governed by the laws of physics
  - can provide high level of automation and performance
  - have some unique characteristics
    - real-time
    - ubiquitous
    - heterogeneous
Embedded Systems: Overview

- **Embedded Systems**
  - = Embedded Hardware + Embedded Software

- Embedded systems close the feedback loops between the physical world and the information world
- Embedded software defines the physical behavior of complex systems
Embedded Systems: Overview

- Examples on how embedded software defines physical behavior of a complex system:

  - BMW 745i Sedan
    - there are more than 70 microprocessors
    - the computation is embedded in the Transmission Electronic Control Unit (ECU), Engine ECU and more
Embedded Systems: Overview

- Examples on how embedded software defines physical behavior of a complex system:

- **Boeing 777**
  - 1280 CPUs onboard
  - 7 million lines of code
Embedded Systems: Overview

- Examples on how embedded software defines physical behavior of a complex system:
  - Boeing 777
    - 1280 CPUs onboard
    - 7 million lines of code
  - International Space Station
    - 2.6 million lines of code on the ground
    - 1.5 million lines of code for flight software
Examples on how embedded software defines physical behavior of a complex system:

- Boeing 777
  - 1280 CPUs onboard
  - 7 million lines of code
- International Space Station
  - 2.6 million lines of code on the ground
  - 1.5 million lines of code for flight software
Embedded Systems: Overview

Examples on how embedded software defines physical behavior of a complex system:

Boeing 777
- 1280 CPUs onboard
- 7 million lines of code
- $4 billion development effort!!
- > 50% system integration & validation cost!!
Embedded Systems: Overview

- Embedded Systems
  = Embedded Hardware + Embedded Software

- Embedded hardware development is getting more mature
- However, embedded software lags embedded hardware as a reliable component of products
Embedded Systems: Overview

Computing systems show no signs of becoming less complex. In fact, they are becoming more complex faster and faster.

- **Windows Operating System**
  - Windows NT (1992): 4 million lines of code
  - Windows NT 4.0 (1996): 16.5 million lines of code
  - Windows NT 5.0 (1998): 20 million lines of code

- **Solaris Operating System**
  - 8 million lines of code

- **Linux Operating System**
  - 5 million lines of code
Embedded Systems: Overview

- Complexity of developing, testing, and supporting large scale software systems continues to escalate

13% of projects are on time! 18% of time spent on coding, 35% debugging

Source: Capers Jones, Estimating Software Costs, pg. 140
Capers Jones, Patterns of Software Systems Failure & Success
Figure: Microsoft Research
Embedded Systems: Overview

Failure in Embedded Software Design

- 2005: Toyota recalls 75,000 Prius hybrids due to a software defect.
- 2004: Pontiac recalls the Grand Prix since the software didn't understand leap years. 2004 was a leap year.
- 2003: A BMW trapped a Thai politician when the computer crashed. The door locks, windows, A/C and more were inoperable. Responders smashed the windshield to get him out.
- 2002: BMW recalls the 745i since the fuel pump would shut off if the tank was less than 1/3 full.
- 2001: 52,000 Jeeps recalled due to a software error that can shut down the instrument cluster.

Source: Total Recall, Jack Ganssle
Embedded.com. Feb. 06
“Okay, Houston, we’ve had a problem!”

- Detailed Chronology of Events Surrounding the Apollo 13 Accident
  http://www.hq.nasa.gov/office/pao/History/Timeline/apollo13chron.html
Embedded Systems: Overview

- Embedded System Design (common practice)
  - The laws of physics are not well respected
  - Hardware and software are developed independently
  - Software is developed by using higher level language
  - Validation and verification are performed individually

Pyramid at Medum, Egypt
Embedded Systems: Overview

- **Embedded System Design (current trend)**
  - The laws of physics are partially considered
  - Development of hardware and software are coupled
  - Some software is synthesized from models
  - Validation and verification are performed tightly

Pyramid at Giza, Egypt
Embedded Systems: Overview

- Embedded System Design (model integrated)
  - The laws of physics are well captured in models
  - Hardware and software are co-designed based on models
  - Software is synthesized from models
  - Validation and verification are model based

Pyramid at Louvre, France
Outline

- Embedded Systems: Overview
- Embedded Systems: Modeling
- Embedded Hybrid Systems: Analysis
- Embedded Hybrid System: Design Tool
- Conclusion
Embedded Systems: Modeling

- Embedded Systems:
  - Embedded systems are complex and heterogeneous.
  - Various types of models are used for designing components within an embedded system.
  - These components interact with each other concurrently.
In embedded system design, in many cases

- Hardware and software are developed independently
- Software is developed by using higher level language
- Validation and verification are performed individually
The current trend in embedded systems design,
- Development of hardware and software are coupled
- Some software is synthesized from models
- Validation and verification are performed tightly
Embedded Systems: Modeling

- Digital Signal Processing
  - Design of Digital Signal Processing (DSP) applications can now be performed by using Data Flow (DF) models and they can be implemented onto FPGA.
Embedded Systems: Modeling

- Digital Signal Processing
  - Models are used for design and simulation at the system level for DSP applications.
  - Models enable design engineers to focus on the application rather than dealing with detailed knowledge of FPGA architecture.
  - Models can be transformed automatically and flawlessly onto VHDL for rapid prototyping.
  - Mixing implementations with models allows hardware-in-the-loop simulation.

**XtremeDSP Design Flow**

1. DSP System Modeling
   - The MathWorks MATLAB® / Simulink®

2. System Generation
   - Xilinx System Generator for DSP

3. HDL Synthesis
   - Choose From...
     - Synplicity Sympify Pro
     - Mentor Graphics Leonardo Spectrum
     - Xilinx XST

4. Simulation (optional)
   - ModelSim
   - Mentor Graphics MXE

5. FPGA Implementation
   - Xilinx ISE

6. In-System Debug
   - Xilinx ChipScope ILA
Control Logic

- In many programming languages, if-then-else statement is used to control the execution sequence of tasks.
- Finite state machines (FSM) can be used to model this type of execution and can be expressed graphically for visualization.
Continuous Dynamics

For many mechanical systems, their dynamics can be derived by using the laws of physics and the ordinary differential equations (ODE) can be used to specify the dynamics.

Numerical methods implemented in some programming languages are used for solving the ODE.

\[
\begin{align*}
\dot{x}_1(t) &= f_1(x_1(t), x_2(t), \ldots, x_n(t), t) \\
\dot{x}_2(t) &= f_2(x_1(t), x_2(t), \ldots, x_n(t), t) \\
& \vdots \\
\dot{x}_n(t) &= f_n(x_1(t), x_2(t), \ldots, x_n(t), t)
\end{align*}
\]
Embedded Systems: Modeling

- Embedded systems are designed and built as “systems of systems.”
How can the models be used together to form the system-level representation of a system design for performance elevation before actual implementation?
Embedded Systems: Modeling

Current Design Practice

- Communication systems
- DSP systems
- Control systems

Application

- VHDL program
- C++ program
- Java Program

Embedded software

- FPGA configurations
- PowerPC assembly code
- x86 assembly code

Embedded code

- FPGAs
- Power PC
- x86

Embedded hardware
Edward Lee and Alberto Sangiovanni-Vincentelli propose that embedded system design should be based on the use of formal mathematical descriptions to describe the behaviors of the system at a high level of abstraction.

These mathematical descriptions are called Models of Computation (MOCs).
Each MOC provides a formal framework for reasoning about certain aspects of embedded systems.

MOCs can be thought as the “laws of physics” governing both the physical and information worlds.

There are many MOCs such as
- Continuous time (ODEs), Spatial/temporal (PDEs), Finite State Machine (FSM), Discrete Event (DE), Synchronous Dataflow (SDF), Synchronous/Reactive (SR) and ...
Embedded Systems: Modeling

- System-Level Design

  - Communication systems
  - DSP systems
  - Control systems
  - Dataflow models
  - Continuous models
  - FSM models
  - System models

- Integrating Models of Computation
  - We need to understand these MoCs relative to one another, and understand their interaction when combined in a single system design.
Embedded Systems: Modeling

- Models of Computation: Notion of Time

- Continuous time
- Discrete time
- Totally-ordered discrete events
- Multirate discrete time
- Partially-ordered discrete events
- Synchronous/reactive

Salvador Dali, *The Persistence of Memory*, 1931
Embedded Systems: Modeling

- Hybrid Modeling of Embedded Systems
  - Hybrid system is built from atomic discrete components and continuous components by parallel and serial composition, arbitrarily nested.
Embedded Systems: Modeling

Hybrid Modeling of Embedded Systems
- Hybrid system is built from atomic discrete components and continuous components by parallel and serial composition, arbitrarily nested.
- The behaviors and interactions of components are governed by models of computation (MOCs).
  - Discrete Components
    - Finite State Machine (FSM)
    - Discrete Event (DE)
    - Synchronous Data Flow (SDF)
  - Continuous Components
    - Ordinary Differential Equation (ODE)
    - Partial Differential Equation (PDE)
- Hybrid system models have been proposed by [Alur-Dill], [Henzinger], [Lygeros and Sastry] and many others.
Embedded Systems: Modeling

- In control engineering, differential equations have been used for modeling physical systems.
- In computer science, discrete systems have been used to encapsulate logical behavior of machines.

**Hybrid Systems Theory**

- Ordinary differential equations (ODE)
- Partial differential equation (PDE)
- Differential algebraic equations (DAE)

- Finite state machines (FSM)
- Discrete event (DE)
- Synchronous data flow (SDF)

**Control Engineering**

\[ \dot{x} = f(x) + g(x)u \]

**Computer Science**
**Embedded Systems: Modeling**

**Why Hybrid Systems?**
- Modeling abstraction of
  - Continuous systems with phased operation
    - walking robots, mechanical systems with collisions, circuits with diodes
  - Continuous systems controlled by discrete inputs
    - Switches, valves, digital computers
- Coordinating processes
  - multi-agent systems
- Important in applications
  - Computer Aided Design
  - Manufacturing Automation
  - Communication Networks
  - Multimedia
  - Medical Systems
  - And more

*Workshop on Hybrid and Embedded Systems 2006*
Embedded Systems: Modeling

- Hybrid Automaton (Lygeros, 2003)

\[ H = (Q, X, Init, f, D, E, G, R) \]

- \( Q \) is a finite collection of discrete state variables;
- \( X \) is a finite collection of continuous state variables;
- \( Init \subseteq Q \times X \) is a set of initial states;
- \( f : Q \times X \rightarrow \mathbb{R}^n \) is a vector field;
- \( D : Q \rightarrow 2^X \) assigns to each \( q \in Q \) a domain;
- \( E \subseteq Q \times Q \) is a collection of discrete transition;
- \( G : E \rightarrow 2^X \) assigns to each \( e \in E \) a guard; and,
- \( R : E \times X \rightarrow 2^X \) defines a reset relation.

The state of \( H \) is defined as \((q, x) \in Q \times X\).

Notice that in the definition the hybrid automaton has no input or output and the state transition only depends on the continuous state.
Embedded Systems: Modeling

Execution of Hybrid Automaton

\[ G(q_1, q_2) \]

\[ D(q_1) \]

\[ G(q_2, q_3) \]

\[ D(q_2) \]

\[ R((q_1, q_2), x) \]

\[ R((q_2, q_3), x) \]
Embedded Systems: Modeling

- Hybrid Automata: Research
  - Modeling & Simulation
    - Control: classify discrete phenomena, existence and uniqueness of execution, Zeno [Branicky, Brockett, van der Schaft, Astrom]
    - Computer Science: composition and abstraction operations [Alur-Henzinger, Lynch, Sifakis, Varaiya]
  - Analysis & Verification
    - Control: stability, Lyapunov techniques [Branicky, Michel], LMI techniques [Johansson-Rantzer]
  - Controller Synthesis
    - Control: optimal control [Branicky-Mitter, Bensoussan-Menaldi], hierarchical control [Caines, Pappas-Sastry], supervisory control [Lemmon-Antsaklis], safety specifications [Lygeros-Sastry, Tomlin-Lygeros-Sastry], control mode switching [Koo-Pappas-Sastry]
    - Computer Science: algorithmic synthesis [Maler et.al., Wong-Toi], synthesis based on HJB [Mitchell-Tomlin]
Embedded Systems: Modeling

- **Hybrid Automata: Computational Tools**
  - Model driven system development
    - Model integrated computing [Sztipanovits-Karsai 1997]
    - Model-integrated development of embedded software [Karsai et. al. 2003]
    - Generic modeling environment [Ledeczi et. al. 2001]
  - Modeling & Simulation
    - Generic Modeling Environment (GME) [Ledeczi et. al. 2001]
    - Matlab \ Simulink
    - Charon [Alur et. al. 2001]
    - Ptolemy II [Brooks-Lee et. al. 2005]
    - Modelica [Tiller 2001]
  - Verification, and/or Synthesis
    - UPPAAL [Bengtsson-Larsean et. al. 1996]
    - KRONOS [Yovine 1997]
    - HyTech [Henzinger-Ho et. al. 1997]
    - d/dt [Asarin-Dang et. al. 2002]
    - Level Set toolbox [Mitchell et. al.2005]
    - Checkmate [Chutinan-Krogh 1999]
    - ReachLab [Koo 2005]
Outline

- Embedded Systems: Overview
- Embedded Systems: Modeling
- Embedded Hybrid Systems: Analysis
- Embedded Hybrid System: Design Tool
- Conclusion
Embedded Hybrid Systems: Analysis

- **Embedded System Design Process**

  - Complex systems are often decomposed into manageable subsystems for design purposes.
  - Experiments on models can allow validation & verification of a system design before actual system is built.
  - Multiple iterations can be performed for system refinement and specification compliance.

*Workshop on Hybrid and Embedded Systems 2006*
Embedded Hybrid Systems: Analysis

“Even when not a matter of life and death, the financial consequences of a bug can be very serious.”

Embedded Hybrid Systems: Analysis

“This has been the Holy Grail of computer science for many decades, but now in some very key areas, for example, driver verification, we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.”

French Guyana, June 4, 1996
$800 million software failure
Embedded Hybrid Systems: Analysis

- Analysis: Simulation and Verification

- Given a specification
  - Simulation enables the analysis of hybrid automaton by generating a single trajectory from an initial condition, but
  - Verification can prove the correctness of a hybrid automaton under all possible initial conditions.
Embedded Hybrid Systems: Analysis

- Specification for Hybrid Automaton

- Given a set of safe states of system (F) determine if all the system states *always* stay within F

- Given a set of states of system (F) determine if all the system states *eventually* reach F
Embedd ed Hybrid Systems: Analysis

- Executions of a hybrid automaton

  Consider a hybrid automaton. To specify all possible executions that can be accepted by the hybrid automaton, we have to consider both continuous flows governed by differential equations and discrete jumps specified by the guards.
Embedded Hybrid Systems: Analysis

- Simulation of Hybrid Automaton
  - In Ptolemy II, a hybrid automaton can be modeled as a modal system in which a continuous model is hierarchically embedded in each discrete location of a FSM.
Simulation of Hybrid Automaton

- Simulation can be performed to check if the specification is satisfied from a given initial condition.
Executions of a hybrid automaton

- Simulation can only generate at most a single execution that can be accepted by the hybrid automaton.
- Therefore, it is hard to capture all possible behaviors of the hybrid automaton by using simulation only.
Executions of a hybrid automaton

Verification considers all possible executions that can be accepted by the hybrid automaton.
Embedded Hybrid Systems: Analysis

Verification of Hybrid Automaton

- Given a specification, check if a given hybrid automaton can satisfy the specification for all possible initial conditions.

Formal verification of hybrid automata

- Theorem Proving
  - A given specification is inferred or deducted using a set of logical deductions.
  - The problem with these methods is that it usually requires human intervention.

- Model Checking
  - Exploration of various states of the system model to arrive at the set of states which will satisfy a given specification.
  - Advantage is that algorithms can be used to specify the manner in which state space should be explored.
Verification of Hybrid Automaton

Model Checking has been used extensively for verifying hardware and software systems. In model checking, the concept of reachable set plays an important role. Reachable sets are used for collecting possible system states that can be reached by the system from a given initial set.
Verification of Hybrid Automaton

Model checker can be used for checking against specifications formulated in some formally specified temporal logics such as Computational Tree Logic (CTL) which can be used to specify the safety and reachability properties.

- \( E<> \varphi \) (eventually)
  exists trajectory where \( \varphi \) eventually holds

- \( E[ ] \varphi \) (potentially always)
  exists trajectory where \( \varphi \) always holds

- \( A<> \varphi \) (eventually always)
  for all trajectories \( \varphi \) eventually hold

- \( A[ ] \varphi \) (invariantly)
  for all trajectories \( \varphi \) always hold

- \( \varphi \rightarrow > \psi \) (leads to)
  whenever \( \varphi \) holds, \( \psi \) eventually hold
### Verification of Hybrid Automaton

These are immediate predecessors of state $q_5$.

It is denoted by $Pre_d(\{q_2, q_3\}) = \{q_1\}$

These are immediate predecessors of state $q_5$.

It is denoted by $Pre_d(\{q_5\})$

- $F = \{q_5\}$

### Discrete System

If initial state is anywhere in $\{q_1, q_2, q_3\}$, the system will eventually have the state $\{q_5\}$
Embedded Hybrid Systems: Analysis

- **Verification of Hybrid Automaton**
  These are immediate predecessors of state $q_5$. It is denoted by $\text{Pre}_d(q_2, q_3) = \{q_1\}$

- **Discrete System**
  If initial state is anywhere in $\{q_1, q_2, q_3\}$, the system will eventually have the state $\{q_5\}$

- **Predecessor of a system set** collects all system states that can reach the system set in some time. It can be called as backward reachable set and there is also forward reachable set.

- **The concept of reachable sets** play a crucial role in model checking

---

Workshop on Hybrid and Embedded Systems 2006
Computing Reachable Sets

For hybrid automata, the reachable set contains both discrete and continuous states. However, due to any subset of a continuous state space containing infinitely many elements, it is difficult to exactly represent the set and to accurately compute the evolution of the set.
Computing Reachable Sets

In level set methods (LSM), the boundary of a set called interface is embedded as the zero level set of a higher-dimensional function. The interface evolution problem is transformed into the level set equation (LSE) which is a partial differential equation (PDE). The evolution of a the set is then computed by solving the LSE by using proper numerical methods.

The level set equation is defined by the level set equation which has the following form

\[ \varphi_t + f(x) \cdot \nabla \varphi(x) = 0, \]
\[ \varphi(x(0), 0) = 0. \]

At any time, the interface is implicitly defined by \( \varphi(x(t), t) = 0 \) isocontour.
Computing Reachable Sets

- LSM is developed by Sethian and Osher for solving the problem of interface evolution of dynamical systems.
- In LSM, the formulation of interface evolution is Eulerian, since the interface is captured by the implicit function. It makes gross changes to the interface topology, such as merging and pinching easier to handle, as opposed to the Lagrangian formulation, which typically requires ad hoc techniques to address mesh connectivity during merging and pinching.
- Tomlin and Sastry proposed a framework of using LSM for computing reachable sets for nondeterministic dynamical systems.
- Tomlin and Mitchell developed computational methods for computing reachable sets for nondeterministic dynamical systems.
- Mitchell develops a Matlab toolbox for LSM.
Embedded Hybrid Systems: Analysis

- Computing Reachable Sets

\[ \dot{x}_1 = x_2, \quad \dot{x}_2 = x_2 (1 - x_1^2) - x_1 \]
Embedded Hybrid Systems: Analysis

- Computing Reachable Sets

\[ \dot{x}_1 = x_2, \quad \dot{x}_2 = x_2(1 - x_1^2) - x_1 \]
Embedded Hybrid Systems: Analysis

Computing Reachable Sets

- We have implemented the LSM in a parallel manner on the Advanced Computing Center for Research & Education (ACCRE) cluster facility at Vanderbilt which allows using at most 1000 processors with Myrinet networking for computing reachable sets for continuous systems

- 672 PowerPC Processors
  - 2.2 GHz processors
  - 1.5 GB of memory
- 335 Xeon Processors
  - 2.0 GHz processors
  - 1-4 GB of memory
- 409 Opteron Processors
  - 2.0 GHz processors
  - 1-2 GB of memory
Computing Reachable Sets

By choosing a proper computation configuration and topology, the large-scale computation problem can be performed effectively by the processors via high-speed communication network.
Outline

- Embedded Systems: Overview
- Embedded Systems: Modeling
- Embedded Hybrid Systems: Analysis
- Embedded Hybrid System: Tool
- Conclusion
“Verification has become the dominant cost in design process. On current projects, verification engineers outnumber designers, with this ratio reaching two or three to one for the most complex design. Design conception and implementation are becoming mere preludes to the main activity of verification.”
Embedded Hybrid Systems: Tool

- Higher of automation is need to enhance the effectiveness and efficiency in the verification process.

- Example: Boeing 777
  - $4 billion development effort!!
  - > 50% system integration & validation cost!!
**Embedded Hybrid System: Tool**

- **Current Picture**
  - Many computation tools have been developed for hybrid system analysis, such as d/dt, the level set toolbox, HyVisual, Simulink, and CheckMate, with their own capabilities, purposes, theoretical foundations, computation methods and implementations.

- Therefore, it becomes difficult to develop some custom-defined model checking algorithms along with specific computation methods.
Embedded Hybrid System: Tool

- **ReachLab**
  - is Computation platform for the analysis and design of hybrid systems.
  - is based on model-based approach for the design of analysis algorithms and system models
  - enables the reuse of models of algorithms and systems
  - allows various implementation possibilities of the models due to computation needs

*Workshop on Hybrid and Embedded Systems 2006*
ReachLab

ReachLab is based on Model Integrated Computing (MIC) approach, and adopts meta-model based Domain Specific Modeling Language (DSML) as its Hybrid Analysis and Design Language (HADL).

MIC approach promotes model reuse and separates the analysis design from implementation details. Various computation tools have been enriched with common hybrid system analysis notions as computation kernels of ReachLab.

Workshop on Hybrid and Embedded Systems 2006
Embedded Hybrid System: Tool

- ReachLab
  - Use algorithms approaching for solving verification and synthesis problems.

- Examples:
  - Using algorithms for finding the initial state sets based on ◊F and □F properties for a given hybrid automaton.
Using algorithms for finding the initial state sets based on ◊ F property.

Discrete and continuous reachability operators are different.

Operators on state sets and algorithms for continuous systems differ from algorithms for discrete systems.

Discrete and continuous state sets have different representations.

Multidimensional lists can be made generic to store both discrete and continuous state sets.

Same control flow constructs can be used for algorithms for discrete systems and algorithms for continuous systems.

Discrete and continuous reachability operators are different.
Composition of algorithms

Since algorithms are specified as models, composing algorithms is equivalent to composing models of algorithms.

More complicated algorithms can be composed by using multiple algorithms.

Example:

Algorithm for $F$ can be derived by using $F = \neg \neg \neg \neg F_c$ where $F_c$ is the set complement of $F$.

Therefore, algorithm for finding initial state sets such that $F$ is true can be composed using algorithm for solving $\neg \neg \neg \neg F_c$ problem.

Furthermore, $\neg \neg \neg \neg F_c$ algorithm uses the algorithm for computing $cPre_{c[T]}(R_k)$.
For developing analysis and design algorithms by exploring the hybrid state space, we need:
- Representation of state sets
- Operators on state sets
- Reachability operators
- Control flow
- Multi-dimensional list

For developing this type of algorithms, computation tools exist but most of them provide different implementations of Representation of state sets, Operators on state sets, Reachability operators and lack Control flow and Multi-dimensional list;
- algorithm designed in these tools has to be tailored according to implementation detail; and
- it is hard to reuse algorithms for different classes of hybrid automata.
We have designed a metamodel based hybrid system analysis and design language (HADL) which can provide:

- Representation of state sets
- Operators on state sets
- Reachability operators
- Control flow
- Multi-dimensional list

Benefits:

- Enabling specifying hybrid automaton by using abstract data types.
- Providing hybrid automaton related operations.
- Supporting facilities for specifying algorithms models.
- Allowing hierarchical composition of algorithm models.
- All these constructs are abstract and are designed such that they could be translated to specific implementations for performing actual computation.
## Embedded Hybrid System: Tool

<table>
<thead>
<tr>
<th>Category</th>
<th>Syntactical Elements</th>
</tr>
</thead>
</table>
| Data                | ■ Primitive data types: integer, float, Boolean  
                   ■ Multi-dimensional lists                                                                                                                                                                                       |
| Hybrid Automaton    | ■ Discrete mode, associated with invariant;  
                   ■ Discrete transition, associated with guard and reset  
                   ■ Continuous set and initial continuous set  
                   ■ Analysis set, as a specialization of continuous set                                                                                                                                                    |
| Control Flow        | ■ Routine, hierarchical in nature  
                   ■ Looping: while loop  
                   ■ Branching: if-then-else                                                                                                                                                                                      |
| Operators           | ■ Primitive data operations: +, -, *  
                   ■ Reachable set operations: discrete successor and predecessor, (constraint) continuous successor and predecessor in a single step (in bounded time)  
                   ■ Operations on state sets: intersection, union, complement  
                   ■ Logical operations: equal, less than, and, or, not  
                   ■ Multidimensional list operators                                                                                                                                                                           |
Embedded Hybrid Systems

DataObjects

DataContainer

Classes with *italics* names are **abstract** classes, which cannot be instantiated.

Connections with solid filled diamond as their end adornment are used to represent **Composition Aggregation**. It is equivalent to saying DataContainer is contained in DataObjects.

Each class contains certain **attributes** with specific attribute types.

White triangle represents **inheritance**. For example, Discrete modes, GuardSet, Continuous Set are all different types of Set.

List of either numeric or set data types
Embedded Hybrid System: Tool

System Aspect

Programming Aspect

Data Aspect

Workshop on Hybrid and Embedded Systems 2006
Embedded Hybrid System: Tool

- Analysis Algorithm in ReachLab

System Aspect

Programming Aspect

Hierarchical Algorithm Design
Embedded Hybrid System: Tool

Workshop on Hybrid and Embedded Systems 2006
Embedded Hybrid Systems: Tool

- Model-Integrated Design Tool Chain

  - Specification
  - Design
  - Analysis
  - Simulation
  - Implementation
  - Testing
  - Production

ReachLab (Vanderbilt University)
Ptolemy II (UC Berkley)
Embedded Hybrid Systems: Tool

- Model-Integrated Design Tool Chain

Implementation Stages:
- Specification
- Design
- Analysis
- Implementation
- Testing
- Production
- Code Generation
- Target Specification
- Code Distribution
- System Execution
- Code Distribution
Conclusion

- Program is written by human beings
Conclusion

Program is synthesized by program
Conclusion

- Program is synthesized by program synthesized by program.
Conclusion

- Correct-by-Construction Design Paradigm
  - Use MOCs to describe the behaviors of systems at a high level of abstraction before making choice on hardware and software
  - Use verification or validation at higher levels of abstraction
  - Use automatic synthesis from the high level of abstraction to the implementation of the system which includes hardware and software
Conclusion

“This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning..”
- Sir Winston S. Churchill
THANK YOU

Contact Info.:

T. John Koo
john.koo@vanderbilt.edu
http://www.vuse.vanderbilt.edu/~kootj