LiComR 2003 Summer

Visual Formal Methods Survey

고려대학교 컴퓨터학과 정형기법 연구실
김진현
jhkim@formal.korea.ac.kr

2003. 8.19

Contents

- Introduction
  - What is Visual Formalism?
  - Pros vs Cons
  - Statecharts
  - SyncCharts
  - GCSR(Graphical Communication Shared Resource)
  - UPPAL
Visual Formalism (1/2)

- **Visual Formalism**
  - Graphical notations
  - Formal semantics
    - Formal specification
    - Formal verifications
  - Class
    - State-based specification
    - Communication-based specification
    - Etc

Visual Formalism (2/2)

- **Merit vs demerit**
  - **Merit**
    - Comprehensive
    - Effectiveness
    - Readability
  - **Demerit**
    - Complexity...
    - Verifiability
State-based Specification (1/2)

- Describe behavior of a system in terms of possible states and transitions
  - Capture the condition of a system in terms of its state
- State-based specification techniques have:
  - Explicit description and complete coverage of states and transitions
  - Specification language and formal reasoning
  - Precise mathematics
  - Analysis tools

State-based Specification (2/2)

- Benefits
  - Objectives
    - Clarify requirements
    - Locate and correct inconsistency and non-determinism
    - Refine requirements consistently into design
    - Decomposition of system
    - Prove or disprove assertions about system behavior
  - End Result
    - Predictably safe systems, more deterministic behavior
    - Links between customer needs and system design
State-based Specification - Concepts (1/2)

- **State**
  - A state represents the computational readiness of a system at that instance of time during its lifetime
- **Initial state**
  - The state in which a system starts
- **Condition**
  - Represents an existing on-going state of the system
- **Quantification**
  - The application of a condition over more than one variable

State-based Specification - Concepts (2/2)

- **Transition**
  - A transition is a change in the state of the system
- **Event**
  - An event is an instantaneous change in the environmental or internal condition of a system.
- **Action/activity**
  - The function that is carried out or the event that is emitted upon the transition of the system from one state to another
- **Invariant**
  - A global or local property that holds true for the duration of the system lifetime
State-based Specification-Techniques and Methods

- Finite state machines
- Augmented transition networks
- SDL
- Petri nets
- Sequence diagrams
- Statecharts
- SyncCharts
- GCSR
- UPPAAL
- Computational tree logic
- UML state diagrams
Intro. to Statecharts (1/3)

- Developed by David Harel in 1987
- Provide behavioral description of reactive systems
  - Specification and design of discrete-event systems
  - Hardware, Instrument and Control system, Embedded Systems, Automobile, Cellular phone, Missile, Avionics...

"When event α occurs in state A and condition C is true, transfer state to B"

Intro. to Statecharts (2/3)

- Features
  - An Extension of State Transition Diagram
  - Hierarchy, concurrency and communication
  - Uses area and location of graphical objects

Statecharts = State diagrams + Hierarchy + Orthogonality + Broadcast-communication
Intro. to Statecharts (3/3)

- Clustering and refinement
  - States and transitions
  - Arrow-labeled event and optionally a parenthesized condition
  - Clustered two events to one
  - D is an abstraction of A and C
  - D can be refined to consist of A and C
  - Zooming in and out of D (in latter A and C are not shown)

Syntax of Statecharts (1/6)

- States: Represent the mode of operation of an activity
  - Hierarchy of states is represented by encapsulation of states.
- Connectors
  - Conditional: Shows branches to various possible states.
  - History: Remembers previously visited state.
- Default transitions
  - Specifies the initial state at each hierarchical level (assuming that an explicit transition was not taken).
- Transition: Shows the possible paths from one state to another state.
Syntax of Statecharts (2/6)

- **Transitions Labels.**
  - Define the criteria needed for a transition to be taken from one state to another.
  - Specifies what action to perform when taking the transition.

- **Trigger/Action syntax**
  - Trigger: causes the movement from one state to another.
    - Syntax of trigger:
      - Event_expression and/or[condition_expression]
  - Action: specifies what to do as a result of moving from one state to another.
    - Syntax of Action
      - Action_expression1; Action_expression2;...

Syntax of Statecharts (3/6)

- **Basic State**
  - state_name

- **AND State**
  - state_name

- **Default Transition**
  - Conditional Connector
  - History Connector
  - Deep History Connector

- **Transition**
  - transition_label
Syntax of Statecharts (4/6)

- **Default states**
  - Equivalent to start states for FSM

- **History**
  - (H) Record of the last state at the current level
  - (H+) Deep history stores last state at all levels current and below

Syntax of Statecharts (5/6)

- **Orthogonality**
  - AND combination of states
  - Concurrency and synchronization
    - Simultaneous transitions in component states
  - Independence
    - Independent transition in one of the component state
  - Orthogonality = concurrency + independence
  - Communication among states by common events
  - Exits
    - Synchronized, independent and ε-transition exit
**Syntax of Statecharts (6/6)**

- **Orthogonality**

---

**Semantic Characteristics of Statecharts (1/3)**

- **Semantics of Statecharts in STATEMATE.**
  - how the model is executed.

- **Step semantics**
  - Run: a set of possible behavior
    - A series of snapshots of system responses to external environment stimuli
  - Status: a snapshot of system situation
  - Step: transition between two snapshots
Semantic Characteristics of Statecharts (2/3)

- Events are sensed one step after it happened
  - Ex(S) is sensed one step after S was existed
- Reactions to events, and changes occurred within a step, can be sensed only after the step
- Events live in the step following their occurrence, for one step only.
- Calculations are based on situation at the beginning of the step
- A maximal subset of nonconflicting transactions and SRs is always executed

Semantic Characteristics of Statecharts (3/3)

- **Time Scheme**
  - **Synchronous time scheme**
    - assumes that the system executes a single step every time unit.
    - modeling electronic digital systems, where execution is synchronized with clock signals,
    - external changes can occur between any two steps. The execution
  - **Asynchronous time scheme**
    - allows several steps to take place within a single point in time.
    - In general, external changes can occur at any moment between steps and several such changes can occur simultaneously
Verification in STATEMATE

- **Tools for validation**
  - Simulation

- **Tools for verification**
  - ModelChecker
  - ModelCertifier
    - which includes the whole functionality of ModelChecker.

- **Verification properties**
  - Non-Determinism
  - Write–Write Race
  - Write–Write Race (different values)
  - Read–Write Race
  - Drive-to State
  - Drive-to Configuration
  - Drive-to Property

---

**ModelChecker**

- Main Menu
- Action Bar
- Hierarchical Chart View
- Dictionary and Execution Queries
- Message area

![ModelChecker Interface]

- Name
- Type
- Desired Scope
- Write-Write Race
- Write-Write Race (different values)
- Read-Write Race
- Drive-to State
- Drive-to Configuration
- Drive-to Property
SyncCharts

Intro. to SyncCharts

- Specification and design of large and complex reactive systems
  - Telephone, Automobiles, Communication, SOC...
- Provides for hierarchy, orthogonality, Synchrony hypothesis

\[
\text{SyncCharts} = \text{FSM} + \\
\quad \text{Depth(hierarchy)} + \\
\quad \text{Concurrency(orthogonality)} + \\
\quad \text{Broadcast-communication}
\]
Syntax of SyncCharts (1/7)

- **Hierarchy**
  - State-levels
  - State: basic state, macro state

Syntax of SyncCharts (2/7)

- **Concurrency**
  - Synchronization and independence.
Syntax of SyncCharts (3/7)

- Sequential Automata
- Trigger event
- Effect: signal emission

Syntax of SyncCharts (4/7)

- Halt points (remembering state)
  - Halt, await S, pause

- Signal emission and test
  - emit S, present S
Syntax of SyncCharts (5/7)

- **Control flow**
  - `;` (sequence) and `||` (parallel)
  - loop, sustain, every

Syntax of SyncCharts (6/7)

- **Control flow (cont’d)**
  - abort, weak abort, suspend
Syntax of SyncCharts (7/7)

- **in()**
  - when a state is being executed

- **en(), ex()**

  ![Diagram of State A active]

  - State A active
  - \( in(A) \)
  - \( en(A) \)
  - \( ex(A) \)

- **Strong vs weak abortion**

Verification of SyncCharts

**Verification**

- Model Checking
- The Design Verifier using Model Checking validates the design by proving that the required system properties and assertions hold in all the possible cases.

**Assertions**

- "a FIFO full never overflows"
- "Bus access can be granted to 1 only peripheral"

"Design for Verification and Save Verification Time & Effort With Esterel Studio"
Intro. to GCSR

- **Features**
  - Graphical notations
  - Scalability
    - Modular and hierarchical structures through nesting and modularity.
  - Resource
  - Priority
  - Formal Semantics
  - Equivalence
    - ACSR equivalence relation
  - Executable
**GCSR Language**

**GCSR Syntax (1/2)**

- **Node**
  - instantaneous: undelayed communication
  - execution
  - time-consuming
  - parallel
  - halting process
  - refer to process
  - Reference: events
  - Close: resource

- **Action nodes**
- **Resource**
- **Process nodes**
**GCSR Syntax (2/2)**

**Edge**
- **Normal Edge**
  - That is externally controlled by an interacting process
- **Exceptional Edge**
  - That is triggered internally through voluntary release of control by raising an exception.

**GCSR Process**
- **GCSR process**
  - is a tuple \(<N, I, E, L, R>\)
    - \(L\) is a set of event names
    - \(R\) is a set of resource names
    - \((N, E)\) is directed graph with initial node \(I \subseteq N\)
    - \(E\) is a set of labeled edges \(\subseteq N \times L \times N\)
    - \(L\) is set of labels \((\varepsilon) \cup (L \times N) \cup (N \cup \infty)\)
- **Hierarchical**
  - hierarchical function \(p\)
  - \(p(n) = (G_1, G_2, ..., G_n)\)
    - the processes \(G_1, G_2, ..., G_n\) inside node \(n\)
GCSR Semantics (1/3)

Informal Semantics
- A GCSR process represents a system.
  - A system can sequentially execute communication events or time and resource consuming actions
  - execution with instantaneous communication events
  - execution with resource and time consuming
  - execution with compound nodes

GCSR Semantics (2/3)

Informal Semantics (cont'd)
- instantaneous node no time consume

Stop by
1. after t-time
2. external events
3. internal events

Synchronizing their execution with private events
GCSR Semantics (3/3)

- Simple GCSR to ACSR process
  - $T$ represents translation form GCSR specification to ACSR processes

Analysis Techniques

- ACSR has operational semantics that make GCSR possible to execute a GCSR specification.
- This allows designers to test a GCSR specification for unintended behaviors before attempting to prove correctness.
- With algebraic semantics of GCSF, we can verify the strong and weak equivalence of two GCSR specifications.
Example: Airport with taxi-way

Timed Automata using UPPAAL
Intro. to UPPAAL (1/3)

- Developed by Kim Larsen, Wang Yi and Paul Petterssoon ... in 1996
- Real-Time Verification and Validation Tools.
  - RT-SPIN – Real-Time extensions to SPIN.
  - UPPAAL – Toolbox for validation and verification of real-time systems.
- Real-Time Communication.

Intro. to UPPAAL (2/3)

- UPPAAL consists of three main parts:
  - a description language,
  - a simulator, and
  - a model checker.
- The description language is a non-deterministic guarded command language with data types.
  - describe a system as a network of timed automata
- The simulator enables examination of possible dynamic executions of a system during the early modeling stages.
- The model checker exhaustively checks all possible states.
UPPAAL Model = 
Networks of Timed Automata

Intro. to UPPAAL(3/3)

checkta - syntax checker
simta - simulator
verifyta - model checker
process B {
    state B0 ( x<=4 ), B1, B2, B3;
    commit B1;
    init B0;
    trans B0 -> B1 { 
        guard x>=2;
        sync a?
        assign n:=5, x:=0;
    },
    B1 -> B2 {
        assign n:=n+1;
    },
    B2 -> B3 {
    },
    B3 -> B0;
}

system A, B;
Timed Automata (1/2)

Clocks: $x, y$

Guard
Boolean combination of comp with integer bounds

Reset
Action performed on clocks

State
$(\text{location}, x=v, y=u)$ where $v, u$ are in $\mathbb{R}$

Transitions

- $(n, x=2.4, y=3.1415) \xrightarrow{a} (m, x=0, y=3.1415)$
- $(n, x=3.5, y=4.2415)$

Timed Automata (2/2)

Clocks: $x, y$

Location Invariants

Transitions

- $(n, x=2.4, y=3.1415)$
- $(n, x=3.5, y=4.2415)$

Invariants ensure progress!!
UPPAAL Specification Language

- Temporal Logic

\[ \begin{align*}
A[\]p & \rightarrow (AG\,p) \\
E<>p & \rightarrow (EF\,p)
\end{align*} \]

\[ p ::= a.l \mid gd \mid gc \mid p\,\text{and}\,p \mid p\,\text{or}\,p \mid \text{not}\,p \mid p\,\text{imply}\,p \mid (p) \]

Syntax of UPPAAL(1/2)

- Labels and Transitions
  - The edges of the automata can be labeled with three different types of labels:
    - a guard expressing a condition on the values of clocks and integer variables that must be satisfied in order for the edge to be taken,
    - a synchronization action which is performed when the edge is taken
    - a number of clock resets and assignments to integer variables.
  - Nodes may be labeled with invariants
    - Conditions expressing constraints on the clock values in order for control to remain in a particular node
**Syntax of UPPAAL (2/2)**

- **Committed Locations**
  - A committed location must be left immediately.
  - A broadcast can be represented by two transitions with a committed state between sends.

![Diagram of committed locations and broadcasts](image)

**Semantics of UPPAAL (1/5)**

- **Transitions**
  - Delay transitions
  - Action transitions

- The above two types of transitions may be overruled by presence of urgent channels and committed locations in the following ways:
  - Urgent channels
  - Committed locations
Semantics of UPPAAL (2/5)

- Delay transitions – if none of the invariants of the nodes in the current state are violated, time may progress without making a transition; e.g., from \(((A_0, B_0), x=0, y=0, n=0)\), time may elapse 3.5 units to \(((A_0, B_0), x=3.5, y=3.5, n=0)\), but time cannot elapse 5 time units because that would violate the invariant on \(B_0\).

Semantics of UPPAAL (3/5)

- Action transitions – if two complementary edges of two different components are enabled in a state, then they can synchronize; also, if a component has an enabled internal edge, the edge can be taken without any synchronization; e.g., from \(((A_0, B_0), x=0, y=0, n=0)\) the two components can synchronize to \(((A_1, B_1), x=0, y=0, n=5)\).
Semantics of UPPAAL (4/5)

- When two components can synchronize on an urgent channel, no further delay is allowed; e.g., if channel $a$ is urgent, time could not elapse beyond 3, because in state $((A_0, B_0), x=3, y=3, n=0)$, synchronization on channel $a$ is enabled.

- If one of the components is in a committed node, no delay is allowed to occur and any action transition must involve the component committed to continue; e.g., in state $((A_1, B_1), x=0, y=0, n=5)$, $B_1$ is committed, so the next state of the network is $((A_1, B_2), x=0, y=0, n=6)$.
UPPAAL Model Construction

Plant
Continuous

Controller
(Control Program)
Discrete

Construct model of environment
(user-supplied)

Construct model of tasks
(automatic)

UPPAAL Model

Translation to UPPAAL

P1 :: while True do
T1 : wait(turn=1)
C1 : turn:=0
endwhile

P2 :: while True do
T2 : wait(turn=0)
C2 : turn:=1
endwhile

Mutual Exclusion Program
Conclusion

- **Visual Formalism**
  - State-based specification and its verification
    - Statecharts
    - SyncCharts
    - GCSR
    - UPPAAL
  - Verification methods
    - Model Checking
    - Bi-simulation

- **Future...**
  - Real-time
    - Time, Resource, Priority, Concurrency
  - ...You...