Design and Analysis of Distributed Interacting Systems

Lecture 12 – MSDs and Controller Synthesis, and Introduction to Graph Transformation Systems (GTS)

Prof. Dr. Joel Greenyer

July 4, 2013
Agenda

• Environment & System Controllers (✓)
• Realizability (✓)
• Environment Assumptions (✓)
• The Play-Out Algorithm (✓)
• Advanced concepts:
  – Dynamic Object Systems and Dynamic Lifeline Binding (✓)
  – Parametized Messages (✓)
• Controller Synthesis
Realizability – Example

• what now? realizable?

I think so...

this can be very hard to “see”, i.e. to check manually

<<EnvironmentAssumption>>
ArmAMoveFromTable-ToPressAssumption

c:Controller
a:ArmA

moveToTable
arrivedAtTable

forbidden
arrivedAtPress
moveToTable
moveToPress

<<EnvironmentAssumption>>
ArmAMoveFromTable-ToPressAssumption

c:Controller
a:ArmA

moveToPress
arrivedAtPress

forbidden
arrivedAtTable
moveToTable
moveToPress

<<EnvironmentAssumption>>
NoBlankArrivesBefore-ArmAReturnedToTable

ts:TableSensor
c:Controller
a:ArmA

blankArrived
pickUp
moveToPress
arrivedAtPress
releaseBlank
moveToTable
arrivedAtTable

blankArrived
moveToPress
arrivedAtTable
Realizability – Example

- what now? realizable?

I think so...

dis can be very hard to “see”, i.e. to check manually

we have to find an implementation or show that there is none...

... can we do this automatically?
Reformulate “Does there exist an implementation?” as
“Can the system win the following game?”:
- The environment can send arbitrary sequences of events
- The system reacts to each environment event by sending finite sequences of system events (synchrony assumption)
- There must be no safety or liveness violation in any requirement MSDs
- Or there is a safety or liveness violation in an assumption MSD
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

system events: controllable

env. events: uncontrollable

safety violation in assumptions

safety violation in requirements
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

System events:
- controllable
- uncontrollable

System events:
- ArmATransportBlankToPress
- ArmAMoveFromTableToPressAssumption

Environment events:
- NoBlankArrivesBeforeArmAReturnedToTable
- ArmAMoveFromTableToPressAssumption

Forbidden actions:
- arrivedAtPress
- releasedBlank

Safety violations:
- assumptions
- requirements

NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
ArmATransportBlankToPress (-1, 2, 1)
ArmAMoveFromTableToPressAssumption (2, 2)
NoBlankArrivesBeforeArmAReturnedToTable (2, 3, 2)
ArmATransportBlankToPress (-1, 4, 3)
ArmAMoveFromTableToPressAssumption (2, 2)
The Game Graph

• LTS of reachable configuration of cuts of active MSDs

![Diagram showing system events: controllable vs. env. events: uncontrollable]

- safety violation in assumptions
- safety violation in requirements

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
- ArmATransportBlankToPress (-1, 2, 1)
- ArmAMoveFromTableToPressAssumption (2, 2)
- ArmATransportBlankToPress (2, 3, 2)
- arrivedAtPress
- blankArrived, arrivedAtTable
- ...
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- System events: controllable
- Env. events: uncontrollable

- safety violation in assumptions
- safety violation in requirements

- The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- System events: controllable
- Env. events: uncontrollable

- safety violation in assumptions
- safety violation in requirements

- The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- System events: controllable
- Env. events: uncontrollable

- safety violation in assumptions
- safety violation in requirements
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- Safety violation in assumptions

- Safety violation in requirements
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

![Diagram of the Game Graph]

- System events: controllable
- Env. events: uncontrollable

- Safety violation in assumptions
- Safety violation in requirements
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- System events: controllable
- Env. events: uncontrollable

- safety violation in assumptions
- safety violation in requirements

- blankArrived, arrivedAtTable
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- system events: controllable
  - env. events: uncontrollable

- blankArrived, arrivedAtTable
- safety violation in assumptions
- safety violation in requirements

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
  - ArmATransportBlankToPress (-1, 2, 1)

- ArmAMoveFromTableToPressAssumption (2, 2)

- arrivedAtPress
- control in system events: controllable
- control in env. events: uncontrollable

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
  - ArmATransportBlankToPress (-1, 3, 2)

- ArmAMoveFromTableToPressAssumption (2, 3, 2)

- arrivedAtTable
- control in system events: controllable
- control in env. events: uncontrollable

- NoBlankArrivesBeforeArmAReturnedToTable (2, 3, 2)
  - ArmATransportBlankToPress (-1, 4, 3)

- ArmAMoveFromTableToPressAssumption (2, 2)

- blankArrived, arrivedAtTable
- safety violation in assumptions
- safety violation in requirements

- NoBlankArrivesBeforeArmAReturnedToTable (2, 3, 2)
  - ArmATransportBlankToPress (-1, 5, 4)
### The Winning Condition

**Goal states:**

- There must be no safety violation or executed message enabled in any active requirement MSDs.
- Or there is a safety violation or executed message enabled in an assumption MSD.
The Winning Condition

• **Goal states:**
  - There must be no safety violation or executed message enabled in any active requirement MSDs.
  - Or there is a safety violation or executed message enabled in an assumption MSD.

Or the environment did something that was forbidden or it has currently **unfulfilled obligation** to do something.

The system did nothing that was forbidden and it has currently **no unfulfilled obligation** to do anything.
The Winning Condition

- **Goal states:**
  - There must be no safety violation or executed message enabled in any active requirement MSDs
  - Or there is a safety violation or or executed message enabled in an assumption MSD

- The system can win the game if it can guarantee that goal states are visited *infinitely often*, regardless of the moves of the environment
  - the system can choose controllable transitions (i.e., sending some system message)
  - but the system must assume that the environment can choose any uncontrollable transition (i.e., send any environment messages)
The Winning Condition

• **Goal states:**
  – There must be no safety violation or executed message enabled in any active requirement MSDs
  – Or there is a safety violation or or executed message enabled in an assumption MSD

• The system can win the game if it can guarantee that goal states are visited *infinitely often*, regardless of the moves of the environment
  – the system can choose controllable transitions (i.e., sending some system message)
  – but the system must assume that the environment can choose any uncontrollable transition (i.e., send any environment messages)

Games with a winning condition requiring that some condition must hold infinitely often are also called *Büchi Games*
Winning Strategy

- A winning strategy is a function that maps states to sets of controllable events that the system can choose in these states in order to win.
  - the set of controllable events is also called winning moves
  - empty set: the system cannot choose (it's the environment's turn)
  - more than one winning move: the strategy is non-deterministic, i.e., there are multiple ways to win
A Simple Example

• Game graph
  – controllable transitions are controlled by the system (the player)
    • shown as solid arrows
  – uncontrollable transitions are controlled by the environment (the opponent)
    • shown as dashed arrows
  – Goal states are shown with a double border

• A winning strategy for the system/player to visit goal states infinitely often?

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(wait)</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
</tr>
<tr>
<td>2</td>
<td>(wait)</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
</tr>
</tbody>
</table>
Check if Winning Strategy Exists: Algorithm Sketch

- Given: Game graph with initial state $q_0$ and goal states $Goal$
  - we assume that $q_0 \in Goal$

In the game graph corresponding to an MSD specification, the initial state is always a goal state.
Check if Winning Strategy Exists: Algorithm Sketch

• Given: Game graph with initial state \( q_0 \) and goal states \( \text{Goal} \)
  – we assume that \( q_0 \in \text{Goal} \)

• Two functions:
  – boolean \( \text{isGoalReachable(State } q) \)
    • checks if the system can guarantee to reach goal states from \( q \)

will not be covered here in detail: when exploring the game graph, the procedure checks for all explored states whether the system can reach a goal state from it.
Check if Winning Strategy Exists: Algorithm Sketch

- Given: Game graph with initial state \( q_0 \) and goal states \( Goal \)
  - we assume that \( q_0 \in Goal \)
- Two functions:
  - boolean \( \text{isGoalReachable}(\text{State } q, \text{Set}<\text{State}> G) \)
    - checks if the system can guarantee to reach states in \( G \) from \( q \)
  - boolean \( \text{solveBüchiGame}(\text{State } q) \) – called with \( q_0 \)

Set<State> \( G = Goal \)
Set<State> \( Win = \emptyset \)
foreach \( g \in G: \text{if } (\text{isGoalReachable}(g), G) \text{ then } Win\.add(g) \)
while \( G \neq Win \) do
  \( G = Win \)
  \( Win = \emptyset \)
  foreach \( g \in G: \text{if } (\text{isGoalReachable}(g), G) \text{ then } Win\.add(g) \)
endwhile
return \( q_0 \in G \)

ScenarioTools Controller Synthesis

- Game graph for the production cell example (only Arm A)

- Safety violation occurred in assumptions
- Safety violation occurred in requirements

- blue border: goal state
- green: winning state
— from here the system can guarantee to reach a goal state
Realizability – Example

- what now? realizable?

yes!

<<EnvironmentAssumption>>
ArmAMoveFromTable -ToPressAssumption

<<EnvironmentAssumption>>
ArmAMoveFromTable -ToPressAssumption

<<EnvironmentAssumption>>
NoBlankArrivesBefore-ArmAReturnedToTable
ScenarioTools Controller Synthesis

- The controller extracted from the game graph
ScenarioTools Controller Synthesis

- Example from the last assignment

not realizable if we assume a global acceptance condition
ScenarioTools Controller Synthesis

- Example from the last assignment

R-MSD Interaction2 (1, 2, 2)
  +OTFR(-1), +OTFR(0), +w(0)

R-MSD Interaction1 (2, 1, -1)
  +OTFR(-1), +OTFR(0), +w(0)

R-MSD Interaction1 (2, 1, -1)
  +OTFR(-1), +w(0)

R-MSD Interaction2 (1, 2, 2)
  +OTFR(-1), +OTFR(0), -OTFR(0)

R-MSD Interaction3 (2, 2)
  +OTFR(-1), +OTFR(0)

R-MSD Interaction4 (2, 2)
  +OTFR(-1), +OTFR(0)

绿色： winning state ——从这里开始系统可以保证达到目标状态

losing goal state

green: winning state ——from here the system can guarantee to reach a goal state

it turns out that the system cannot guarantee to reach a goal state infinitely often: no winning strategy and no controller exists
ScenarioTools Controller Synthesis

- Another technical example

```
<table>
<thead>
<tr>
<th>MSD1</th>
<th>MSD3</th>
</tr>
</thead>
<tbody>
<tr>
<td>e:Env a:A b:B</td>
<td>e:Env a:A b:B</td>
</tr>
<tr>
<td>e1</td>
<td>m1</td>
</tr>
<tr>
<td>m2</td>
<td>m4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MSD2</th>
<th>MSD4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e:Env a:A b:B</td>
<td>a:A b:B</td>
</tr>
<tr>
<td>e2</td>
<td>m1</td>
</tr>
<tr>
<td>m2</td>
<td>m3</td>
</tr>
<tr>
<td>m4</td>
<td></td>
</tr>
</tbody>
</table>
```

realizable
ScenarioTools Controller Synthesis

• The game graph
• The controller (layout is different, unfortunately)
Open Challenges

• Synthesis is computationally expensive
  – The game graph grows exponentially with the number of MSDs
  – Solving Büchi games takes quadratic time in the size of the game graph
    • although faster for many MSD controller synthesis problems

• Improvement?
  – Heuristics to improve search for goal states
  – On-the-fly algorithms that do not explore the entire game graph
  – Incremental synthesis

– Compositional synthesis: Break down the synthesis problem into smaller problems
Open Challenges

- Synthesis of distributed controllers
  - one controller for each object

- Synthesis and Realizability Checking for dynamic systems?
  - very difficult
MSDs & Co. : What you should “take home”

• Powerful visual formalism
  – precise semantics for a part of UML
• (Relatively) easy to understand
  – compared to a corresponding LTL formula or Büchi Automaton

• Maybe needs extension in the future
  – for describing particular properties in particular kinds of systems
• Many challenges
  – smart ideas for more efficient controller synthesis needed
  – controller synthesis for distributed controllers
• Many opportunities
  – find contradictions early
  – automated implementation (no more programming!)
  – automated test generation
Agenda

• Environment & System Controllers (✓)
• Realizability (✓)
• Environment Assumptions (✓)
• The Play-Out Algorithm (✓)
• Advanced concepts:
  – Dynamic Object Systems and Dynamic Lifeline Binding (✓)
  – Parametized Messages (✓)
• Controller Synthesis (✓)
Design and Analysis of Distributed Interacting Systems

Lecture 12 – MSDs and Controller Synthesis, and Introduction to Graph Transformation Systems (GTS)

Prof. Dr. Joel Greenyer

July 4, 2013
Graph Transformation Systems – Motivation

• Many objects with relationships that change over time
• How do we describe how the structure of these systems change over time?
  – their reconfiguration behavior

Describe Structural Changes

• Most children understand this way of describing structural changes:
Graph Transformations

• Idea: View the system as a graph
  – objects are nodes
  – edges are relationships

• Describe the reconfiguration behavior by rules that describe how and when a particular part of the graph can be modified
  – (similar to the Lego manual)
  – we use graph grammars
  – in this context also called graph transformation rules
View the System as a Graph

- Idea: View the system as a graph
  - objects are nodes
  - edges are relationships
Graph Reconfiguration Behavior

• Describe the reconfiguration behavior by rules that describe how and when a particular part of the graph can be modified.

Example: Movement of the RailCab
Graph Reconfiguration Behavior

- Describe the reconfiguration behavior by rules that describe how and when a particular part of the graph can be modified.

Example: Movement of the RailCab

- Not only this RailCab can move
- this one can, too

![Diagram showing movement of RailCabs and tracks]
Graph Transformation Rule

- Describe the context of the change and the changes in a rule

```
move

:RailCab
\[\times isOn\]\n
:Track \rightarrow next \rightarrow :Track

:RailCab
\[\times isOn\]\n
:Track \rightarrow next \rightarrow :Track

:Track \rightarrow next \rightarrow :Switch \rightarrow next \rightarrow :Track
```
Graph Transformation Rule

- Describe the context of the change and the changes in a rule

The intuition is clear (at least for this example), but what does this mean exactly?
• Introduction to Graph Transformations (✓)
• Models and Graphs
• Graph Grammars
• Graph Grammar Rules
• Rule Application
• Advanced Concepts
  – Negative Application Conditions
  – Attribute Constraints
• Analysis of Graph Transformation Systems
• Class Model vs. Object Model

![Diagram of Models and Graphs]

- RailCab isOn 0..1 next 1..1 isOn 0..1 turnNext
- Track
- Switch
- :RailCab
- :Track
- :Switch

instance of

:RailCab

:Track

:Track

:Track
Models and Graphs

- Type Graph vs. Typed Graph

Type Graph

Typed Graph (edges are typed, too)
Graph Grammar

- A graph grammar consists of
  - a set of graph grammar rules
  - a start graph (also called host graph)
  - a type graph

- A graph grammar describes a (possibly infinite) set of graphs
  - that can be constructed from the start graph by applying the graph grammar rules

- Synonym: Graph Rewriting- or Graph Transformation System
A graph grammar rule consists of two typed graphs – called left-hand side (LHS) and right-hand side (RHS).

**Graph Grammar Rule**

- **move**
  - LHS: 
    - :RailCab \( v_1 \)
    - :Track \( v_2 \)
    - :Track \( v_3 \)
    -.isOn
    - next
  - RHS:
    - :Track \( v_2 \)
    - :Track \( v_3 \)
    -.isOn

**short-hand notation:**

- LHS: 
  - :RailCab \( v_1 \)
  - :Track \( v_2 \)
  - :Track \( v_3 \)
  -.isOn
  - next
- RHS: 
  - :Track \( v_2 \)
  - :Track \( v_3 \)
  -.isOn

identities
1. Match LHS in host graph (find isomorphic subgraph)
Graph Grammar Rule Application

host graph:

2. Remove nodes and edges that are in the LHS, but not in the RHS
Graph Grammar Rule Application

host graph:

2. Create nodes and edges that are in the RHS, but not in the LHS
Graph Grammar Rule Application

• When to move which RailCab?
  – non-determinism!

\[
\text{:RailCab} \xrightarrow{\text{isOn}} \text{:Track} \xrightarrow{\text{next}} \text{:Track} \xrightarrow{\text{next}} \text{:Track} \xrightarrow{\text{next}} \text{:Track}
\]

\[
\text{move} \\
\text{:RailCab} \xrightarrow{\text{isOn}} \text{:Track} \xrightarrow{\text{next}} \text{:Track} \xrightarrow{\text{next}} \text{:Track} \\
\text{:::} = \\
\text{:RailCab} \xrightarrow{\text{isOn}} \text{:Track} \xrightarrow{\text{next}} \text{:Track} \xrightarrow{\text{next}} \text{:Track}
\]
Agenda

- Introduction to Graph Transformations (✓)
- Models and Graphs (✓)
- Graph Grammars (✓)
- Graph Grammar Rules (✓)
- Rule Application (✓)
- Advanced Concepts
  - Negative Application Conditions
  - Attribute Constraints
- Analysis of Graph Transformation Systems