Supporting Multi-View Models of Software Systems: Synthesis Techniques

Lambeau Bernard

UCL/EPL/INGI

dec 2009

Outline

1. Introduction
2. Background
   - Multi-View Formal Framework
   - Event-based Behavior Models
   - State-based abstractions
   - Goals as intentional models
3. Deductive LTS synthesis from guarded hMSC
4. Inductive LTS synthesis from MSC and hMSC
   - Short background on Grammar Induction
   - Grammar Induction for LTS Synthesis
   - Interactive LTS Synthesis from MSC
   - Pruning the induction with state decorations
   - Pruning the induction with goals
   - Pruning the induction with control information
5. Evaluation and Tool Support
6. References

Introduction

Why Modeling Software Systems?

- Elaborating requirements and exploring system design
- Reasoning about, verifying and documenting systems

Why Multi-View Models?

- Different models => different but complementary focusses
- Example based or rule based?
- Agent interactions or agent internals?
- Declarative or operational?

How Modeling with Multi-View Models?

- Multi-view framework => inter-model consistency rules
- Opportunity for synthesis-driven system modeling

Multi-View Models: Golden Triangle

Scenarios

Typical examples or counterexamples of system behavior through sequences of interactions among agents

- Example-based, interactions, operational

Goals

Prescriptive statements of intent whose satisfaction requires cooperation among the agents forming the system

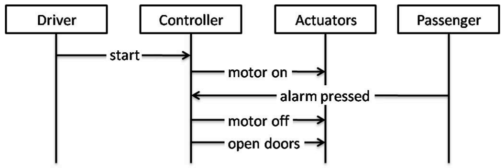
- Rule-based, interactions as well as agent internals, declarative

State machines

Classes of required agent behaviors in terms of states and events firing transitions

- Rule-based, agent internals, operational

Example of synthesis-driven modeling



Synthesize Controller's state machine

- Accepting at least the sequence of events shown in the example
- Under the control of descriptive properties: train doors are either opened or closed but not both in the same state
- Under the control of prescriptive properties: train doors must stay closed when the train moves

Background: Multi-View Formal Framework

Event-based Behavior Models

- Scenarios as Message Sequence Charts (MSC)
- State machines as Labeled Transition Systems (LTS)

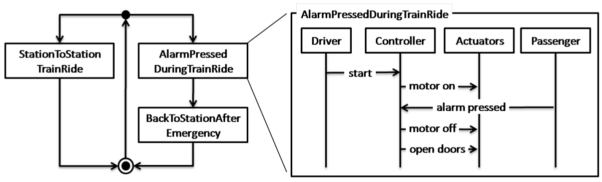
State-based abstractions

- System state through Fluents
- Guards in behavior models (g-LTS and g-hMSC)
- Decorations on behavior models

Goals as intentional models

- Goals and Fluent Linear Temporal Logic (FLTL)
- Linking FLTL and LTS: property and tester automata

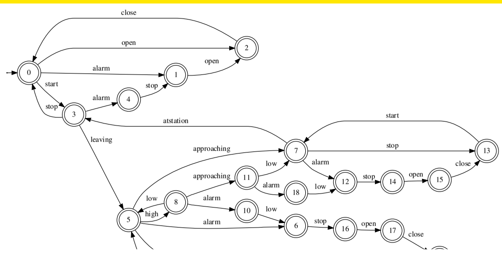
Message Sequence Charts: agent interaction examples



MSC (right) and high-level MSC (left)

- Syntax of MSC and hMSC is described in [ITU96]
- Semantics of MSC and hMSC is defined in terms of Labeled Transition Systems, following [UKM03]
- We also allow a hMSC node to be refined as a finer-grained hMSC

Labeled Transition Systems (LTS) for agent behaviors



Labeled Transition Systems

- Syntax and Semantics defined in [MK99]
- Each agent behavior is defined by a LTS. The system behavior is defined by LTS composition [MK99]
- MSCs are admissible paths in the system LTS [UKM03]

Capturing state information with Fluents

Fluents capture the system state through the occurrence of events

$$fluent\ Fl = < init_{Fl}, term_{Fl} > initially\ Init_{Fl}$$

where $init_{Fl}$ and $term_{Fl}$ are disjoint set of events rendering the fluent *true* and *false*, respectively

Example

$fluent\ moving = < start, \{stop, emergency\ stop\} > initially\ false$
$fluent\ doors\_closed = < close, \{open, emergency\ open\} > initially\ true$

A fluent is ...

- ... controlled by an agent if the agent controls (aka emits) all initiating and terminating events of the fluent
- ... monitored by an agent if the agent controls or monitors (aka receives) all initiating and terminating events of the fluent

Guarded Behavior Models

Summary

Guards can be formally used in hMSC and LTS, leading to guarded hMSC (g-hMSC) and guarded LTS (g-LTS)

- A guard is a boolean expression on fluents
- Structured forms for hMSC and LTS, avoiding state/trace explosion
- Relax the assumption of fluent initial values being known for all instances

Open question, i.e. not discussed in the paper

- Architectural semantics of guards, i.e. what about guards and agents, guards and LTS composition, guard monitorability and controllability ?

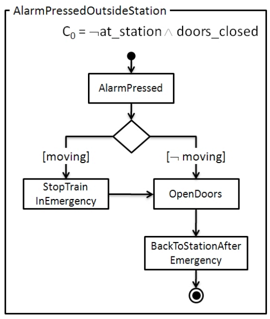
Related publications

Damas C., Lambeau B., Roucoux F. and van Lamsweerde A., *Analyzing Critical Process Models through Behavior Model Synthesis*, in Proc. ICSE'2009: 31th International Conference on Software Engineering, Vancouver, Canada, May 16-24, 2009.

Guards in hMSC, i.e. g-hMSC

Summary

- *Decision nodes*: outgoing transitions are labeled by boolean expressions on fluents
- Initial condition C_0 stating an invariant on the initial state
- Trace semantics through guarded LTS and LTS
- Automated checking of guards: *non overlapping*, *completeness* and *reachability*

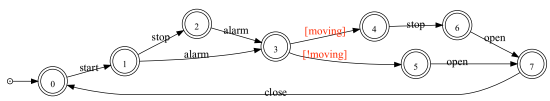


Guards in LTS, i.e. g-LTS

Summary

- A g-LTS transition is labeled by an event or a guard
- Initial condition C_0 stating an invariant on the initial state
- A trace is accepted by a g-LTS if three conditions hold: *trace inclusion*, *admissible start* and *guard satisfaction*

Example



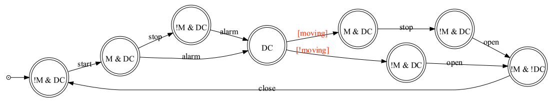
- $C_0 = \neg moving \wedge doors\_closed$
- The event trace (*start alarm open*) is not accepted due to the *guard satisfaction* condition

Decorations on behavior models

Summary

- [DLDvL05] proposes a decoration algorithm for generating fluent invariants on LTS states
- In [DLRvL10] the algorithm is generalized in order to
  - support additional decorations (e.g. cost, doses, time)
  - support additional transition systems (guarded LTS in particular)

Example



- where *M* stands for *moving* and *DC* stands for *doors_closed*

Goals expressed in FLTL

Goals

- Descriptive or prescriptive properties about the system [vL09]
- Can be structured in AND/OR goal graphs

Fluent Linear Temporal Logic (FLTL)

- Linear Temporal Logic (LTL, [MP92]) where propositions are Fluents
- FLTL is used by [GM03] to model check LTS against (state-based) temporal properties

Example

- Maintain[DoorsClosedWhileMoving] : Train doors must always remain closed when the train is moving
- FormalDef: $\Box(moving \implies doors\_closed)$

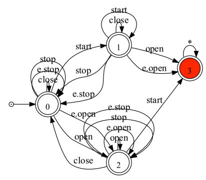
Linking FLTL and LTS

Tester and Property LTS

- A *Tester LTS* can be synthesized from a FLTL safety property, as explained in [GM03]. The error state captures all event traces violating the property
- The *Property LTS* obtained by removing the error state captures all event traces not violating the property

Example

- $\square(moving \implies doors\_closed)$

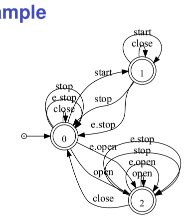


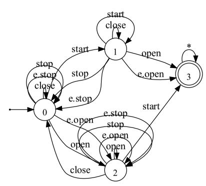
Aside: a regular language point of view on Tester and Property LTS

Error state vs. accepting and non accepting states...

- Property (left) and Tester (right) automata are language complement of each other...
- The tester automaton (resp property) accepts all traces violating (resp. not violating) the safety property

Example





Deductive LTS synthesis from guarded hMSC

Question addressed

- What is the set of event traces accepted by a guarded hMSC?
- How to model-check a guarded model?

Main results

- Adaptation of [UKM03] to synthesize a guarded LTS from a guarded hMSC
- LTS composition algorithm for synthesizing a pure LTS from a guarded LTS
- Adaptation of [GM03] to model-check FLTL properties on guarded-LTS

Deductive LTS synthesis: what's done, what remains to be done?

Work in progress

- Full state tracability through the g-hMSC, g-LTS, LTS synthesis (work in progress)
- Model-checker feedback in terms of the g-LTS and g-hMSC instead of the pure LTS

Open questions

- Only applied on the whole system so far, not with different agents in mind (lack of g-LTS composition/decomposition semantics)

Related publications

Damas C., Lambeau B., Roucoux F. and van Lamsweerde A, *Analyzing Critical Process Models through Behavior Model Synthesis*, in Proc. ICSE'2009: 31th International Conference on Software Engineering, Vancouver, Canada, May 16-24, 2009.

Inductive LTS synthesis from MSC and hMSC

Limitation of deductive approaches

- Scenarios are known to be inherently partial, they only provide typical examples of system the usage
- Therefore, deductive techniques (e.g. [UKM03]) can only result in partial LTS
- Generalizing observed behaviors looks interesting in practice

About our inductive approach

- Relies on Grammar Induction (GI), which provides a sound mathematical background
- But how to
  - Avoid poor behavior generalizations?
  - Ensure consistency with other models (state variables, goals, ...)?
  - Prune the induction process?

Short background on Grammar Induction

In a few words

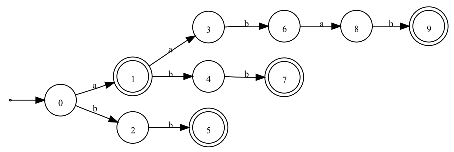
- *Grammar induction* aims at learning a regular language L from a set of positive and negative strings, i.e. respectively belonging and not belonging to the language
- Also known as *Automaton Induction* when L is represented by a (Deterministic) Finite Automaton $A(L)$

Known results ... among others

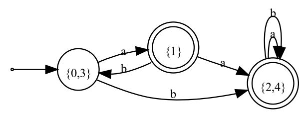
- RPNI algorithm (Regular Positive and Negative Inference)
- Necessary condition for convergence: *structural completeness* of the sample (each transition and accepting state of the automaton is used at least once in the input sample)
- Sufficient condition for convergence: the input sample contains a *characteristic sample* for $A(L)$

RPNI algorithm (and variants) in one slide

From a Prefix Tree Acceptor, accepting the positive sample only, ...



... induce a DFA by successively merging well-chosen state pairs (BlueFringe heuristics), under the control of the negative sample



Grammar Induction for LTS Synthesis ?

Applicability

- A LTS is a DFA with all accepting states
- LTS define a subclass of regular languages (prefix-closed languages)
- A positive MSC is an accepting trace in the system LTS (obtained by composition of all agent LTS)
- A positive (resp. negative) MSC can be seen as a positive (resp. negative) string of the language accepted by the system LTS

The idea

- Learn the system LTS from positive and negative scenarios using RPNI
- Synthesize agent LTS by projecting the system LTS on their monitored events (standard automaton algorithms)

Grammar Induction for LTS Synthesis: a short summary

Question addressed

- Importance of negative strings; are they initially available from end-users?
- How to ensure consistency with other models?
- Assumption that all MSC start in the same system state, what about loops and reuse?

Main results

- Our interactive variant of RPNI, namely *QSM*, when few negative scenarios are initially provided
- Injecting fluent definitions, goals and legacy components ensures inter-model consistency and prunes the process
- Other contributed variants, namely *ASM* and *ASM**, support an hMSC as input, relaxing the assumption mentionned

Interactive LTS Synthesis from MSC: the QSM algorithm

Summary

- RPNI with BlueFringe heuristic for selecting state pairs to merge
- When two states are merged, scenario queries are submitted to an end-user for classification as positive or negative behaviors
- The end-user, aka Oracle, guides the induction process, avoiding poor generalizations
- Query generation relies on the definition of a *characteristic sample*, which provides a convergence criteria

Related publications

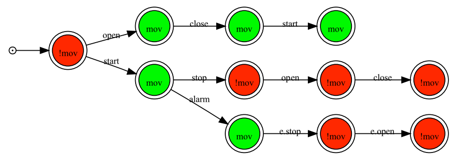
- Damas C., Lambeau B., and van Lamsweerde A, *Generating Annotated Behavior Models From End-User Scenarios*, IEEE Transactions on Software Engineering, Special Issue on Interaction and State-based Modeling, Vol. 31, No. 12, pp. 1056-1073, 2005.
- P. Dupont, B. Lambeau, C. Damas, and A. van Lamsweerde, *The QSM Algorithm and its Application to Software Behavior Model Induction*, Applied Artificial Intelligence, Vol. 22, 2008, 77-115.

Pruning the induction with fluents

Summary

- PTA states can be decorated with fluent values, using the decoration algorithm
- The induction process can be constrained to avoid merging non equivalent states

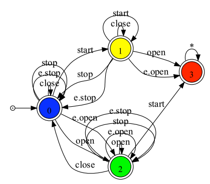
$fluent\ mov(ing) =< start, \{stop, emergency\ stop\} > initially\ false$

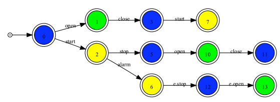


Pruning the induction with goals

Summary

- Color PTA states with corresponding states in the tester. Avoid merging two PTA states not sharing the color





Open questions

- Sound but too strong! This is related to another open issue on ASM^* about the negative language L^- (see later)

Pruning the induction though equivalence classes

State coloring as a generalization

- Equivalence relations can be defined on PTA states and induction process constrained to avoid merging non equivalent states.
- Legacy components can be used in a similar way, for example

Open questions

- Defining equivalence classes could be extended in the light of lattice-based decorations
- We are convinced that additional architectural constraints could be used (to avoid introducing implied scenarios, for example)

Related publications

- C. Damas, B. Lambeau and A. van Lamsweerde, *Scenarios, Goals, and State Machines: a Win-Win Partnership for Model Synthesis*, Proc. FSE'06: Intl. ACM Symposium on the Foundations of Software Engineering, Portland (OR), November 2006.

Pruning the induction with control information

Questions addressed

- Assumption that all MSCs start in the same initial state
- End-users would like to capture loops when known
- Would it be possible to take a hMSC as input instead of a collection of MSCs?

Main results

- Introduction of *mandatory merge constraints* on PTA states, which is the logical counterpart of equivalence classes
- The *ASM* algorithm generalizes a positive language L^+ under the control of a negative sample S^-. *ASM** generalizes a positive language L^+ under the control of a negative language L^-
- Therefore, behaviors described in a hMSC can be generalized under the control of negative MSCs

Pruning the induction with control information

Open questions

- Three concurrent techniques about pruning with goals: equivalent classes (too strong), *ASM** (looks not applicable directly) and Coste's work in [CFKdlH04]
- *ASM* and *ASM** do not support the BlueFringe heuristics nor the interactive feature
- Theoretical GI questions because *ASM* and *ASM** do not perfectly fit the classical regular language learning framework

Related publications

- B. Lambeau, C. Damas and P. Dupont, *State-merging DFA Induction Algorithms with Mandatory Merge Constraints*, Lecture Notes in Artificial Intelligence No. 5278, Springer, pp. 139-153, 2008, 9th International Colloquium on Grammatical Inference, St Malo, France, September 22-24.
- No feedback in the RE community so far

Evaluation

Deductive synthesis evaluation

- Usage of the g-hMSC model-checker is illustrated on a case study in [DLRvL09]
- Additional case-studies: work in progress

Inductive synthesis evaluation

- Number of generated queries and convergence of QSM have been evaluated on RE case studies and synthetic data in [DLDvL05] and [DLDvL08]
- Evaluation of the pruning techniques on RE case studies in [DLvL06]
- Evaluation protocol for *ASM* in [BLD08], applied on one RE case study and synthetic data

Tool Support

Induction Toolkit

- Recent refactoring of all the automaton tools (still in progress)
- Light release will be provided during the Stamina induction competition (2011)
- Full release will be provided after the competition (few work remaining here)

A FLTL model-checker for g-hMSC models

- Work in progress (architecture and packaging of the tool support)
- Future collaboration with Antoine Cailliau for additional (F)LTL tools (master thesis)

References

C. Damas B. Lambeau and P. Dupont.
State-merging dfa induction algorithms with mandatory merge constraints.
In *Grammatical Inference, ICGI'08*, number 5278 in Lecture Notes in Artificial Intelligence, pages 139–153, St Malo, France, 2008. Springer Verlag.

F. Coste, D. Fredouille, C. Kermorvant, and C. de la Higuera.
Introducing domain and typing bias in automata inference.
In *Grammatical Inference: Algorithms and Applications*, number 3264 in Lecture Notes in Artificial Intelligence, pages 115–126, Athens, Greece, 2004. Springer Verlag.

C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde.
Generating annotated behavior models from end-user scenarios.
IEEE Transactions on Software Engineering, 31(12):1056–1073, 2005.

P. Dupont, B. Lambeau, C. Damas, and A. van Lamsweerde.
The QSM algorithm and its application to software behavior model induction.
Applied Artificial Intelligence, 22:77–115, 2008.

C. Damas, B. Lambeau, F. Roucoux, and A. van Lamsweerde.
Analyzing critical process models through behavior model synthesis.
In *ICSE'09: 31th International Conference on Software Engineering*, Vancouver, Canada, May 2009.

C. Damas, B. Lambeau, F. Roucoux, and A. van Lamsweerde.
Abstractions for analyzing decision-based process models.
In *Submitted to ICSE'10: 32th International Conference on Software Engineering*, Cape Town, South Africa, May 2010.

C. Damas, B. Lambeau, and A. van Lamsweerde.
Scenarios, goals, and state machines: a win-win partnership for model synthesis.
In *International ACM Symposium on the Foundations of Software Engineering*, pages 197–207, Portland, Oregon, November 2006.

References

- D. Giannakopoulou and J. Magee.
  Fluent model checking for event-based systems.
  In *International ACM Symposium on the Foundations of Software Engineering*, Helsinki, Finland, 2003.
- ITU.
  Message sequence charts, recommandation z.120.
  International Telecom Union, Telecommunication Standardization Sector, 1996.
- J. Magee and J. Kramer.
  Concurrency: State Models and Java Programs.
  Wiley, 1999.
- Z. Manna and A. Pnueli.
  The Temporal Logic of Reactive and Concurrent Systems.
  Addison Wesley, 1992.
- S. Uchitel, J. Kramer, and J. Magee.
  Synthesis of behavorial models from scenarios.
  IEEE Transactions on Software Engineering, 29(2):99–115, 2003.
- A. van Lamsweerde.
  Systematic Requirements Engineering: From System Goals to UML Models to Software Specifications.
  Wiley, 2009.