Synthesizing Multi-View Models of Software Systems

Lambeau Bernard

ICTeam institute
Université catholique de Louvain

March 2011

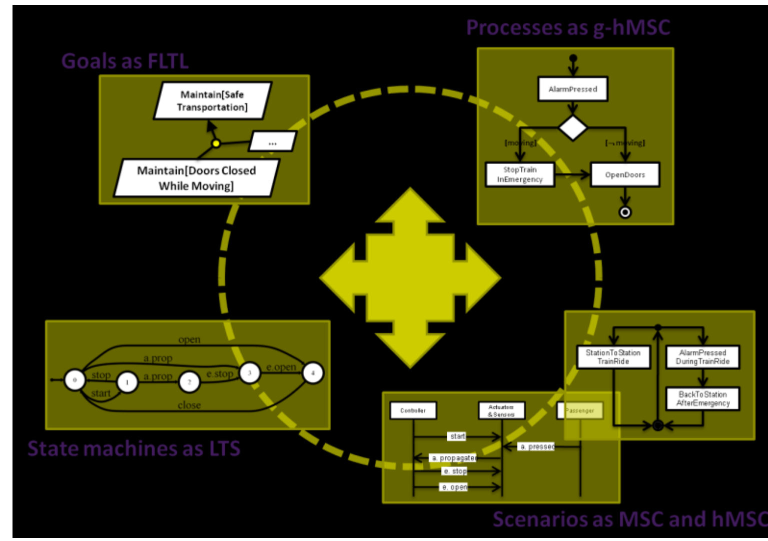
Introduction

- Why Modeling Software Systems ?
  - Elaborating requirements and exploring system design [Avl09]
  - Reasoning about, verifying and documenting systems
  - Generating code, prototypes, ...

- Difficult for complex systems
  - How to check adequacy / correctness of big models ?
  - How to ensure consistency among multiple views ?
  - Need for tool-supported techniques
    - To build software models
    - To check them

Multi-View Modeling

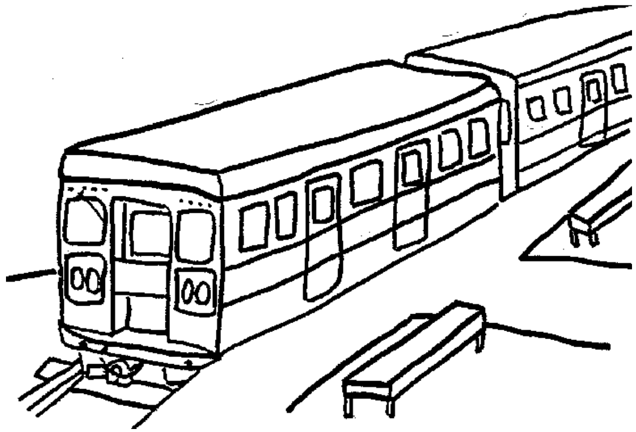
- Different models have complementary focusses
  - single agent vs. multi agent
  - declarative vs. operational
  - partial vs. exhaustive coverage



- Inter-model consistency rules
  - Consistent usage of models and their intent
  - Can be enforced through formal semantics
- Multi-View formal frameworks
  - Offer model building and checking opportunities
  - Only a few available in practice

Running Example
The Little Train System

- The system is composed of three agents
  - a train controller,
  - a train actuator/sensor,
  - and passengers

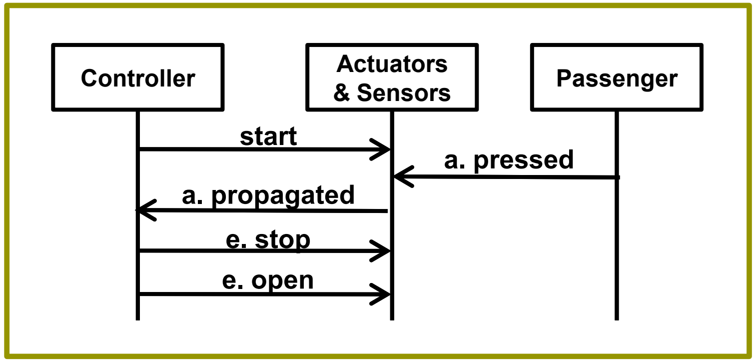


- The train controller controls operations such as start, stop, open doors, and close doors

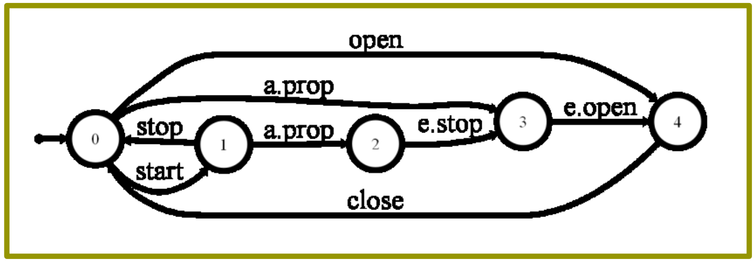
- A safety goal requires train doors to remain closed while the train is moving
  - If the train is not moving and a passenger presses the alarm button, the controller must open the doors in emergency.
  - When the train is moving and the passenger presses the alarm button, the controller must stop the train first and then open the doors in emergency.

Multi-View Models
Example: a Golden Triangle

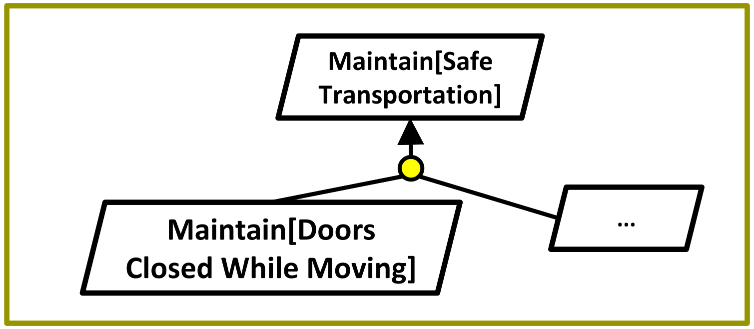
- Scenarios
  - Interactions between the software-to-be and environment agents
  - Multi-agent, Operational



- State machines
  - Classes of agent behaviors in terms of states & events firing transitions
  - Single-agent, Operational

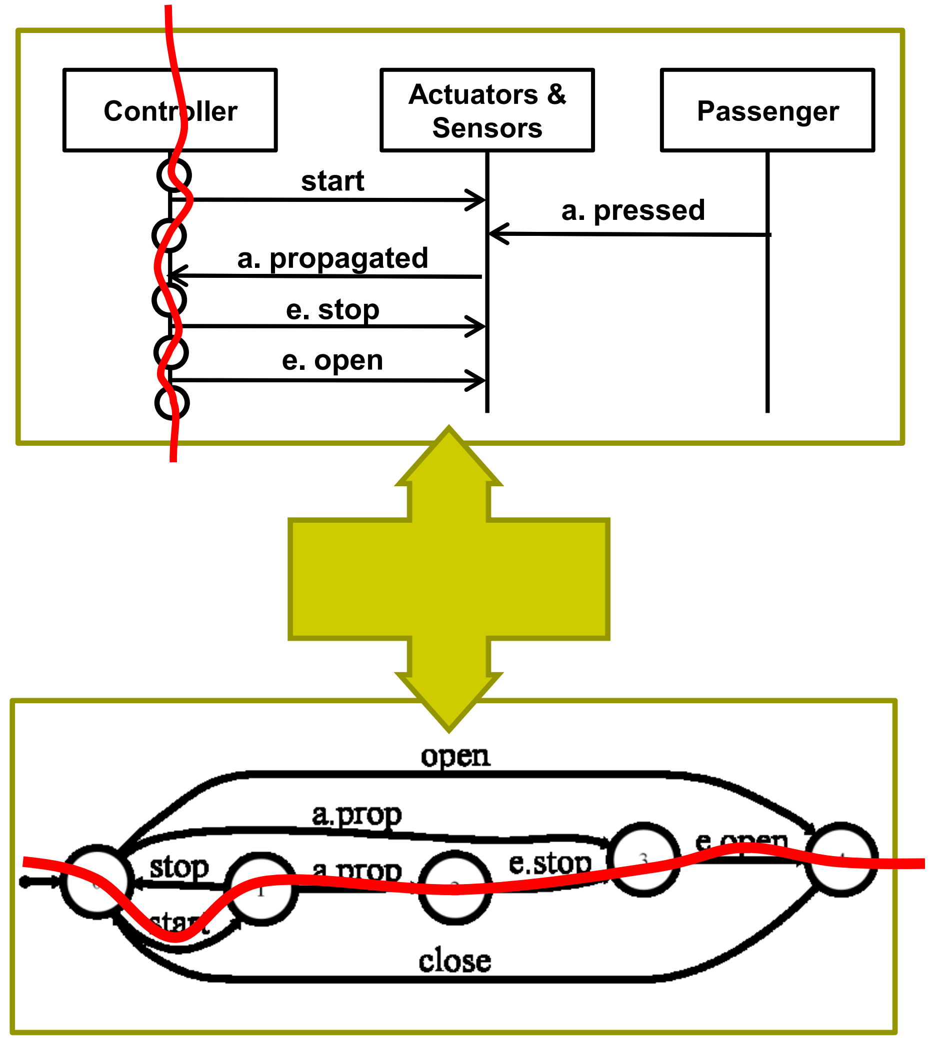


- Goals
  - Prescriptive statements of intent whose satisfaction requires cooperation among all agents
  - Single & Multi-agent, Declarative



Multi-View Models
Scenarios & State machine synergies

- A scenario defines
  - a path in each local state machine of corresponding agents
  - a path in the state machine of the composed system



- Enforced in [Uch03]
  - Scenarios as Message Sequence Charts (MSC)
  - State machines as Labeled Transition Systems (LTS)
  - LTS Composition operator, as in [Mag99]

Multi-View Models
Analysis vs. Synthesis opportunities

- Analysis of Multi-View Models
  - Detect violations of inter-consistency rules
    - Formal semantics used to precisely define these inter-consistency rules and check them on model artifacts
  - Example: <u>model checking</u>
    - *Given formal semantics, check that traces admitted by state machines respect stated goals*

- Synthesizing Multi-View Models
  - Synthesize new models artifacts from existings ones
    - Formal semantics used to automate the synthesis process from precise inter-consistency rules
  - Example: <u>controller synthesis</u>
    - *Given a set of scenarios, derive state machines so that semantics is preserved*

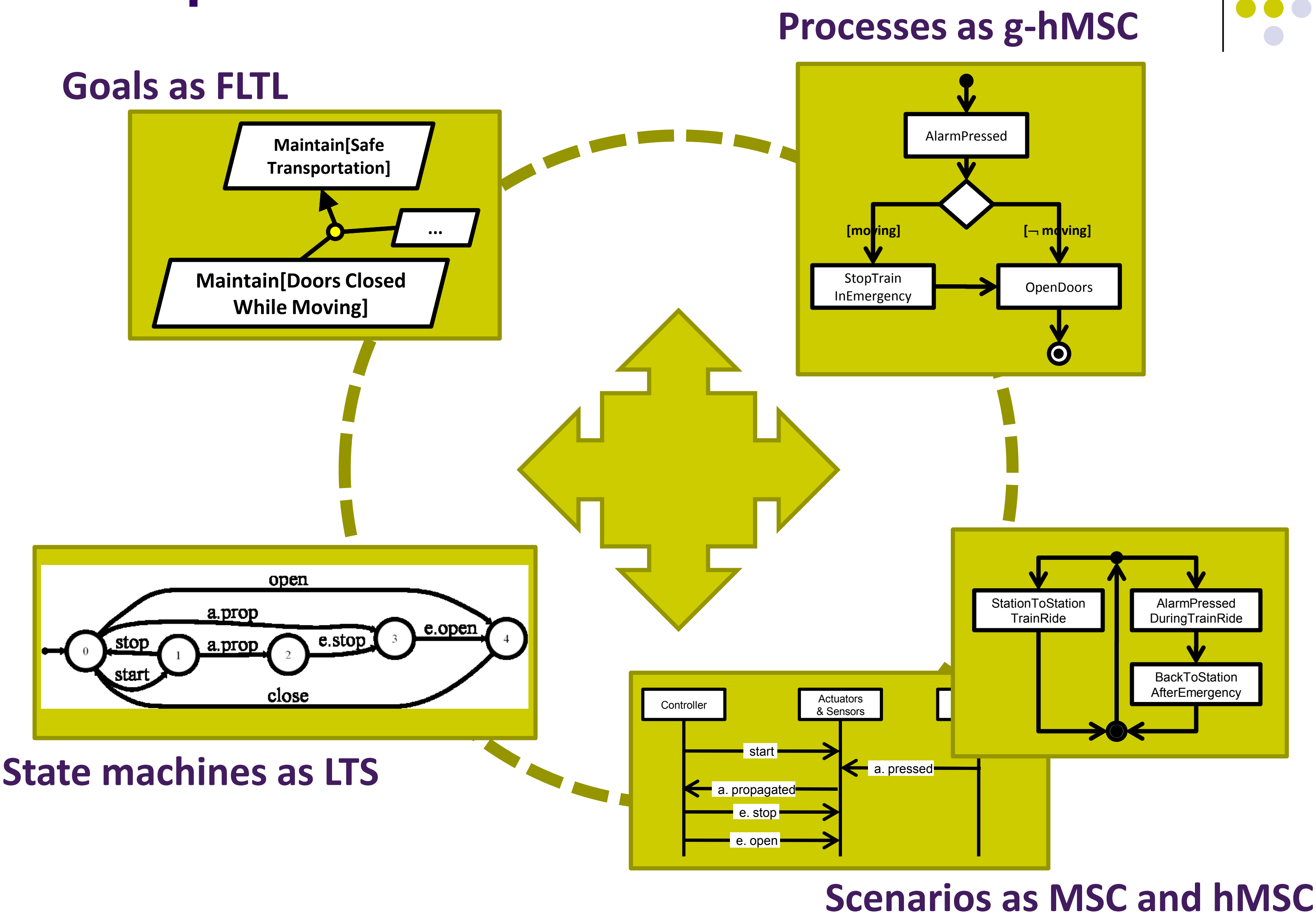
Thesis overview
Focus on model synthesis

- Multi-View formal modeling framework (chapter 2)
  - Scenarios, State machines, Processes & Goals
  - Event-based & State-based abstractions
  - Overview of existing semantic links & synthesis techniques
- New synthesis bricks (chapters 3 & 4)
  - From processes (g-hMSC) to state machines (LTS)
    - a step towards analyzable process models
  - From scenarios (hMSC) to state machines (LTS)
    - behavior generalization via grammar induction
    - under user supervision & control of other models (e.g. goals)
- Evaluation & Tool Support (chapters 5 & 6)
  - Both case studies & synthetic data

Today
A quick digression

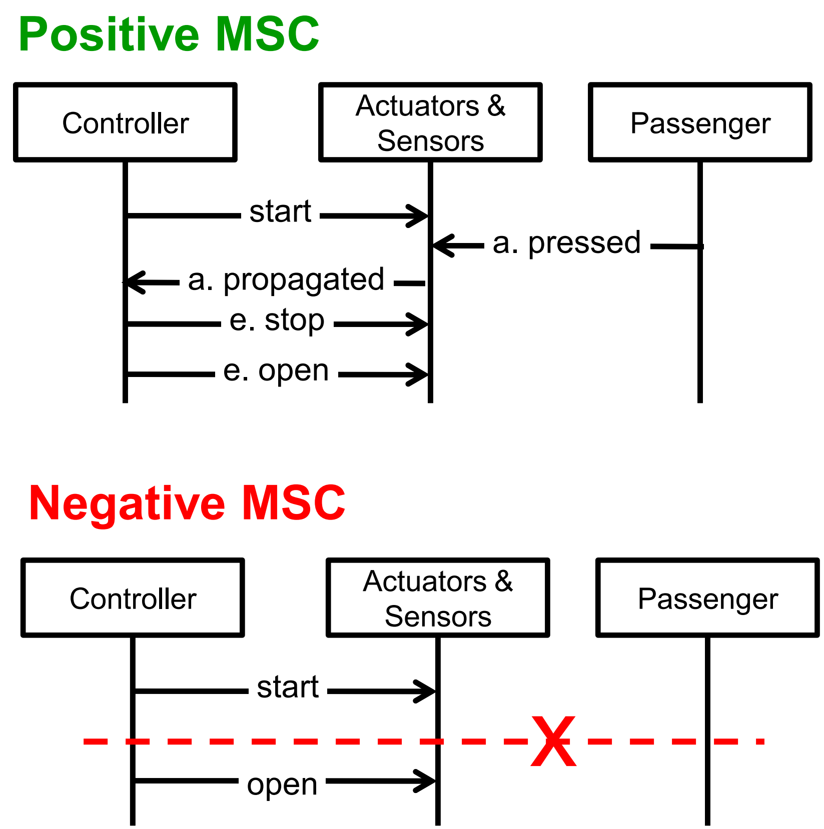
- Available slides
  - Deep technical details for almost all chapters
  - Only if required !?
- Progress report: what was requested last time ?
  - Stamina contest - Done
  - Gisele tool support - Done
- What would help me
  - Is the overal message clear (multi-model synthesis) ?
  - Shouldn't we move specific parts (see later) ?
  - How much about Stamina & Gisele tool support ?

A Multi-View Modeling Framework
i.e. Chapter 2

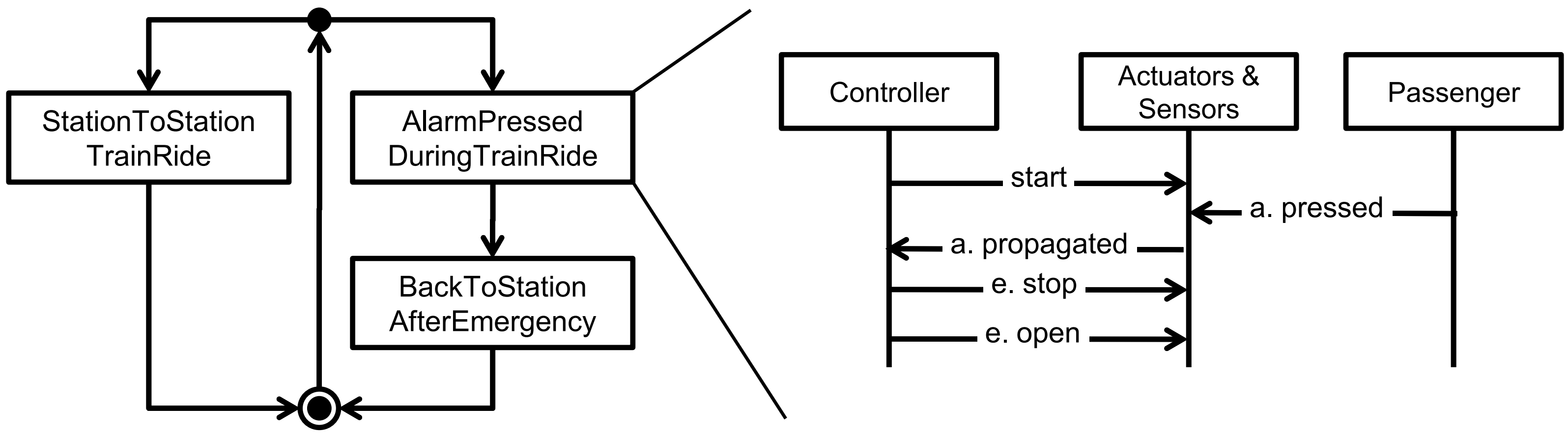


Scenarios
as Message Sequence Charts (MSC)

- Sequences of interactions between the software-to-be and agents in the environment
  - Positive & Negative
- Pro & Cons
  - Informal description easily given by stakeholders
  - Effective means for elicitating software requirements
  - Partial
  - Leave required properties about the intended system implicit



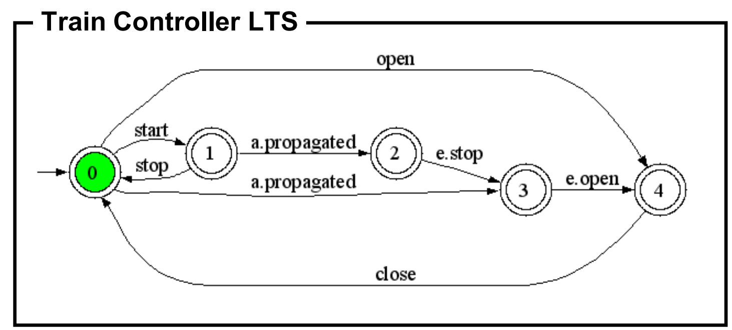
Scenarios
Flowcharting with high-level MSC (hMSC)

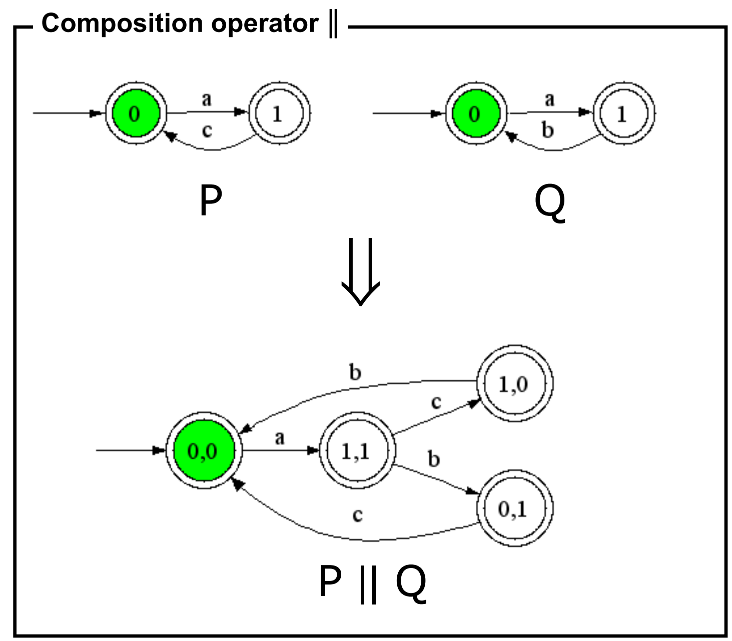


- Advantages
  - Reuse scenarios within a specification
  - Control information: sequence, loops, alternatives, ...

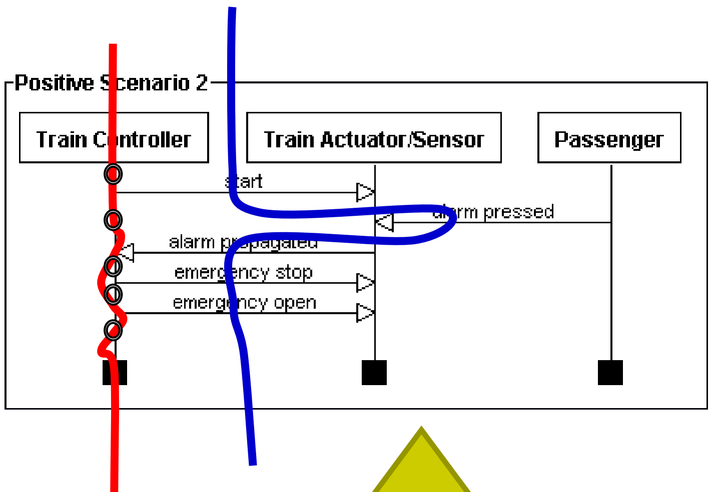
State Machines
as Labelled Transition Systems (LTS)

- A system is modeled as a set of LTS, one per agent
- Pro & Cons
  - Visual abstraction of system behavior
  - Executable
  - Rich opportunities for analysis and code generation
  - Hard to build
- The system can be modeled as the composition of agent behaviors [Mag99]
  - Each agent behaves asynchronously but synchronizes on shared events
  - composition operator ||

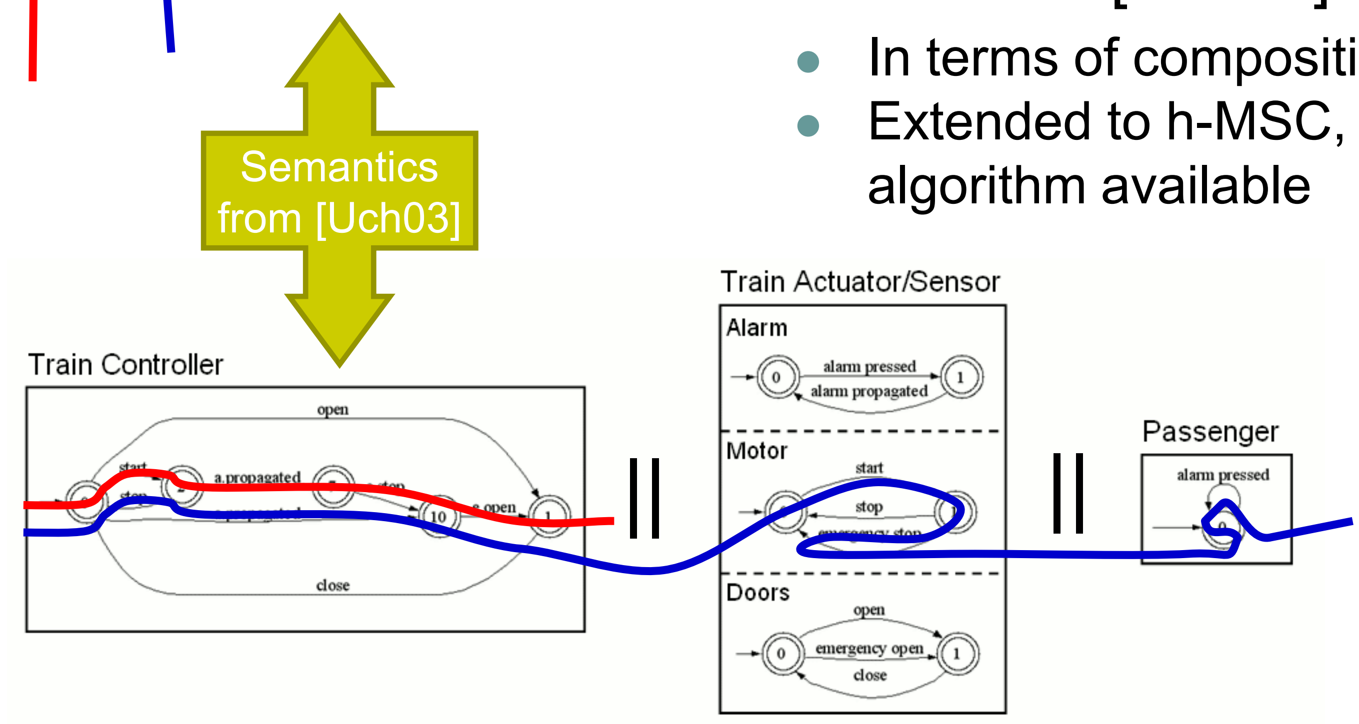




MSC & LTS
Semantics



- A scenario defines
  - a path in each local state machine of corresponding agents
  - a path in the state machine of the composed system
- Semantics [Uch03]
  - In terms of composition operator ||
  - Extended to h-MSC, synthesis algorithm available



State variables
as Fluents

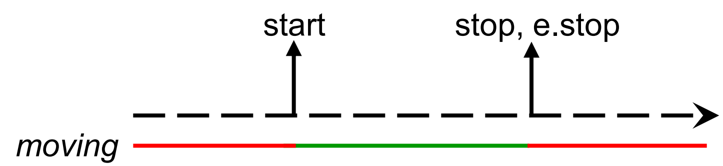
- Agent state variables are modeled with fluents

- fluent $F = \langle I_{Fl}, T_{Fl} \rangle$ initially $Initially_{Fl}$
  - I_{Fl} is a set of initiating events
  - T_{Fl} a set of terminating events
  - $Initially_{Fl}$ (boolean) is the initial value of F

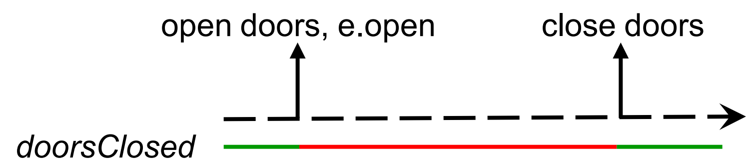
- A fluent is
  - monitored by an agent if it "monitors" or performs all initiating and terminating events
  - controlled by an if it performs all initiating and terminating events

fluent examples

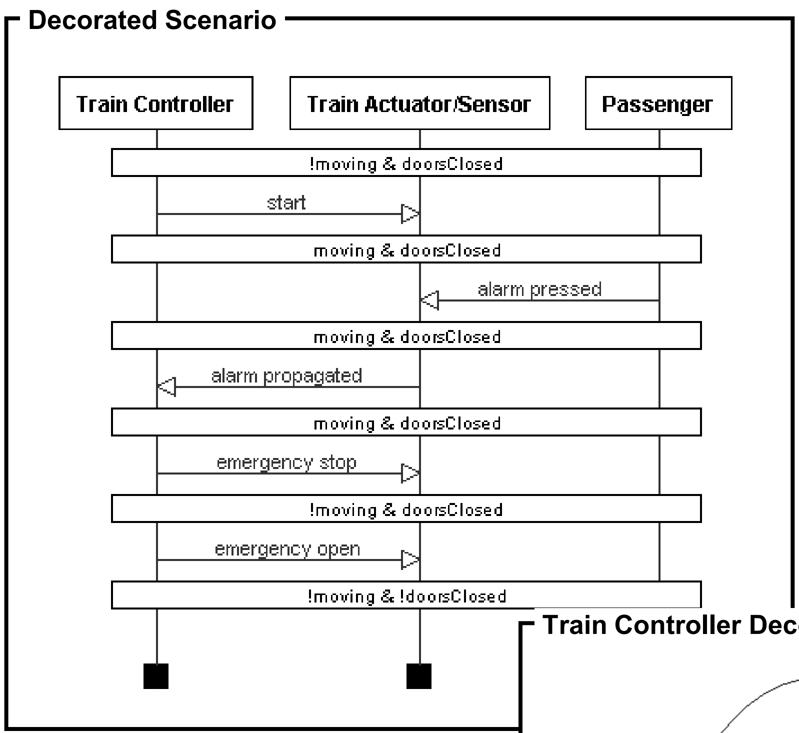
fluent *moving* = <
{*start*},
{*stop*, *emergency stop*} >
initially *false*



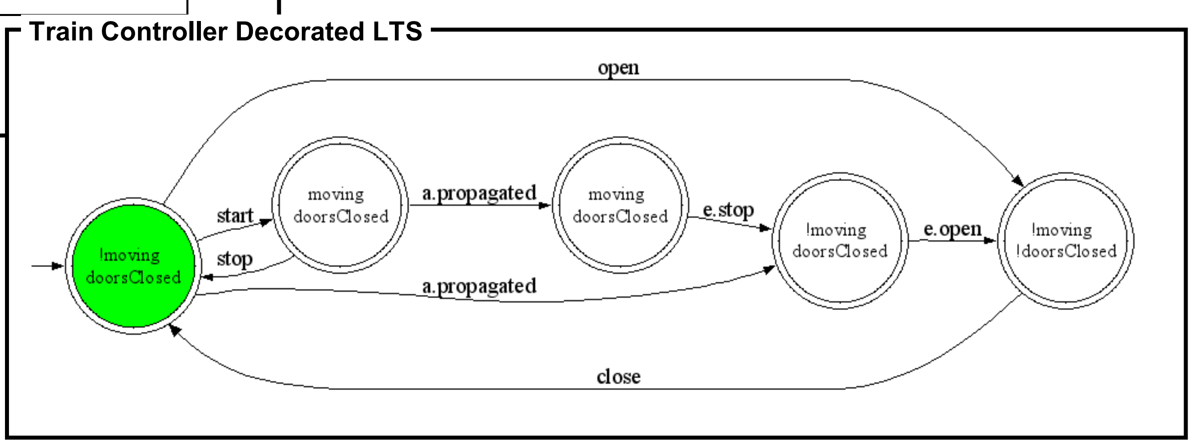
fluent *doorsClosed* = <
{*close doors*},
{*open doors*, *emergency open*} >
initially *true*



Decorations on behavior models

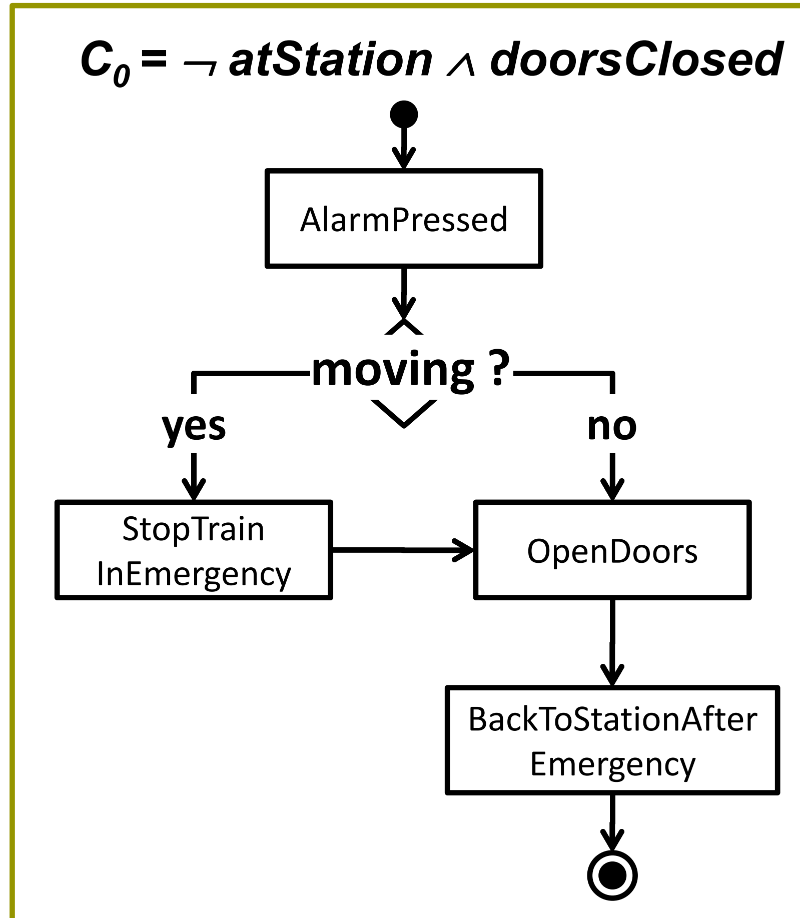


- Fluents can be used to decorate scenarios and state machines with state assertions [Dam05]
- Extension to other kinds of decorations in [Dam10]
  - Used for analysing medical models with respect to cost, time & dosage constraints



Process models
Introducing guarded hMSC (g-hMSC)

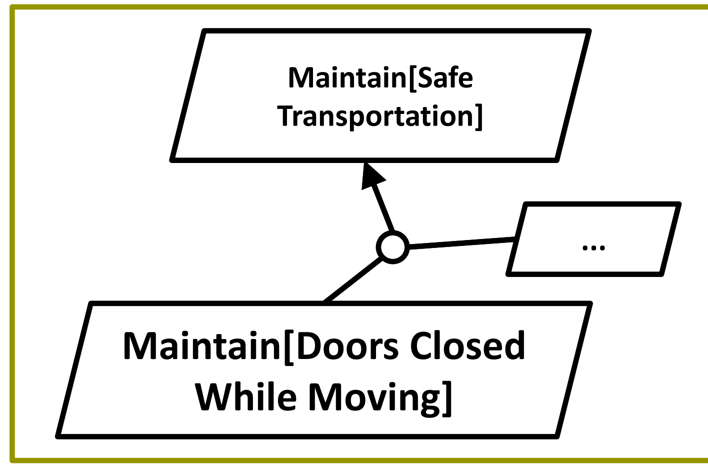
- hMSC + decision nodes
  - Outgoing transitions guarded with boolean conditions on fluents
- Initial state
  - Relaxes the assumption of an initial value being known for each fluent
  - Initial state may be constrainted with an initial condition C_0
- Semantics & Synthesis defined in chapter 3



Goals
as Fluent Linear Temporal Logic (FLTL)

- Goals are objectives that the system should achieve through cooperation of agents [Avl09]

- Pro & Cons
  - recognized paradigm for eliciting, elaborating, structuring, specifying, analyzing, and modifying software requirements
  - Sometimes too abstract
  - Hard to formalize for end-user



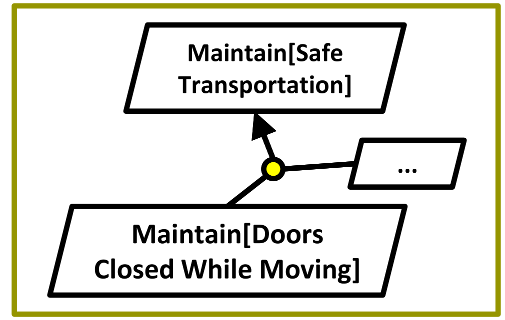
- Maintain[Doors Closed While Moving]
  - Def: The train should always move with doors closed
  - FormalDef: (moving $\rightarrow$ doorsClosed)

Goals and LTS
Synthesis & Semantics

- A goal declaratively define a set of histories to be accepted and/or rejected by the system
- A state machine typically captures a set of system histories

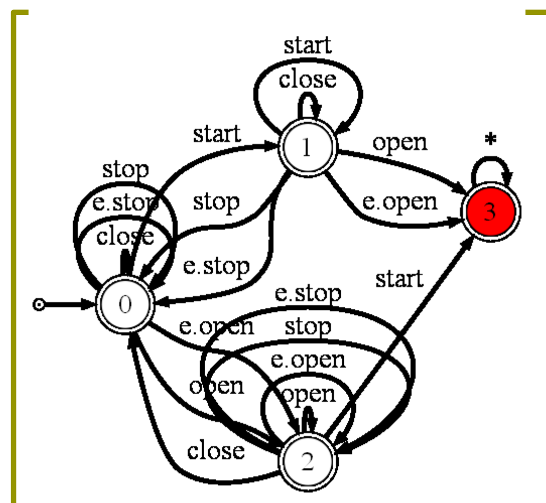
- Available synthesis techniques
  - A *Tester LTS* capturing event traces violating a FLTL safety property can be synthesized using [Gia03]
  - Also, a *Property LTS* capturing event traces NOT violating it [Let08]

Goals in FLTL



[Gia03, Let08]

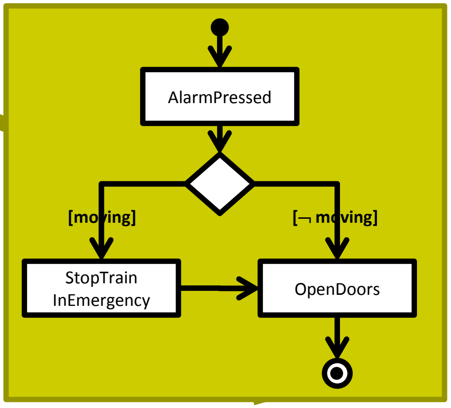
Tester LTS

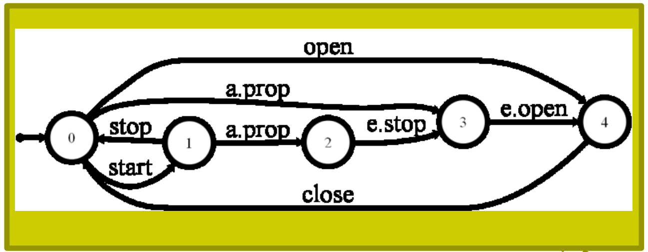


From g-hMSC to LTS
Chapter 3

What is the set of event traces admitted by a guarded hMSC ?

Processes as g-hMSC





State machines as LTS

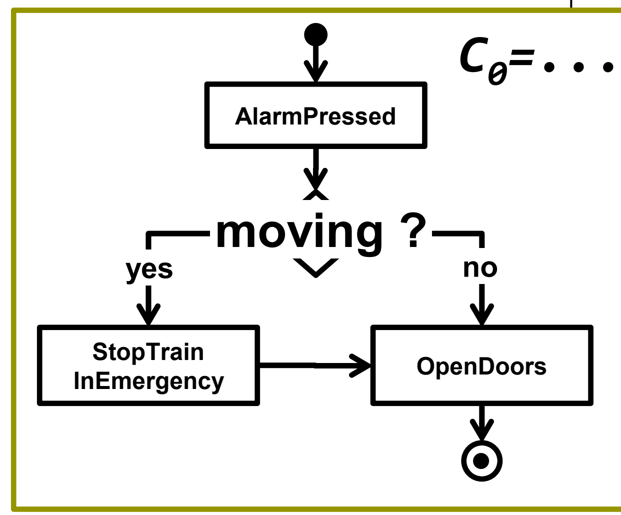
From g-hMSC to LTS
Problem Statement

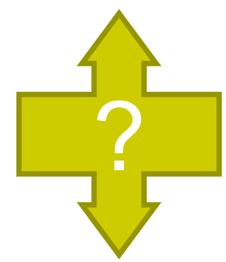
- What is the set of admissible event traces of a guarded hMSC ?

- Needed for
  - Precising model semantics
  - Trace based model-checking (LTSA)
  - Decoration-driven model analysis [Dam10]

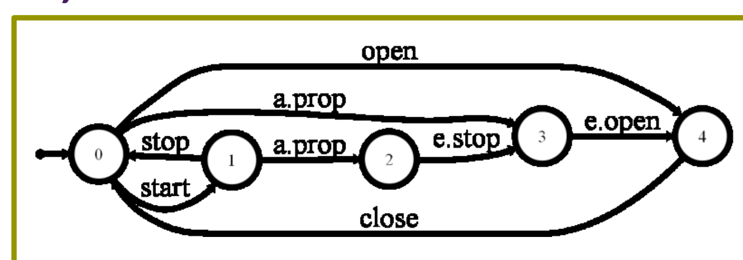
- Roundtrip ?
  - Support friendly analysis feedback

Guarded hMSC





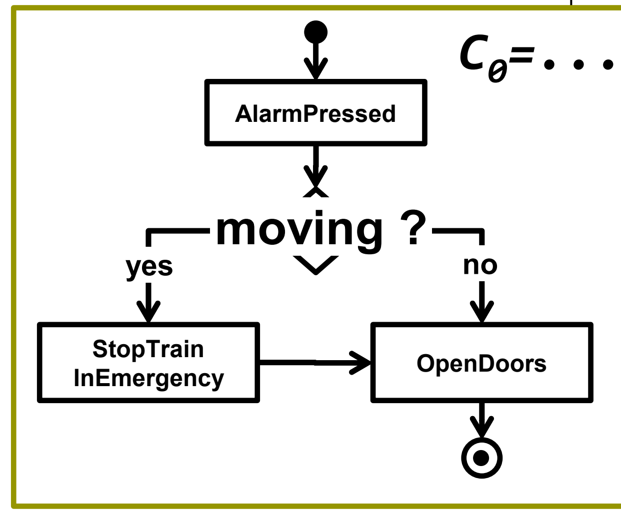
LTS, set of event traces



From g-hMSC to LTS
Solution overview

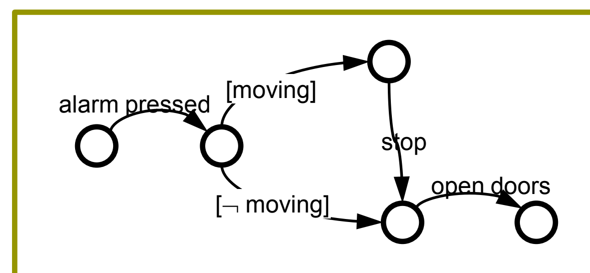
- Guarded LTS
  - Guards or events on transitions
  - C_0 condition, as in g-hMSC
  - Structured form of LTS avoiding state explosion
- Synthesis sub bricks
  - From g-hMSC to g-LTS
  - From g-LTS to pure LTS

Guarded hMSC



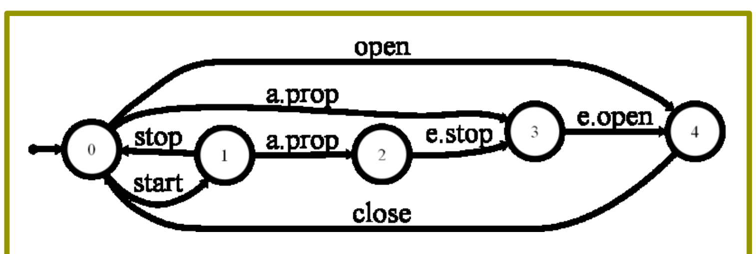
1

Guarded LTS, intermediate level



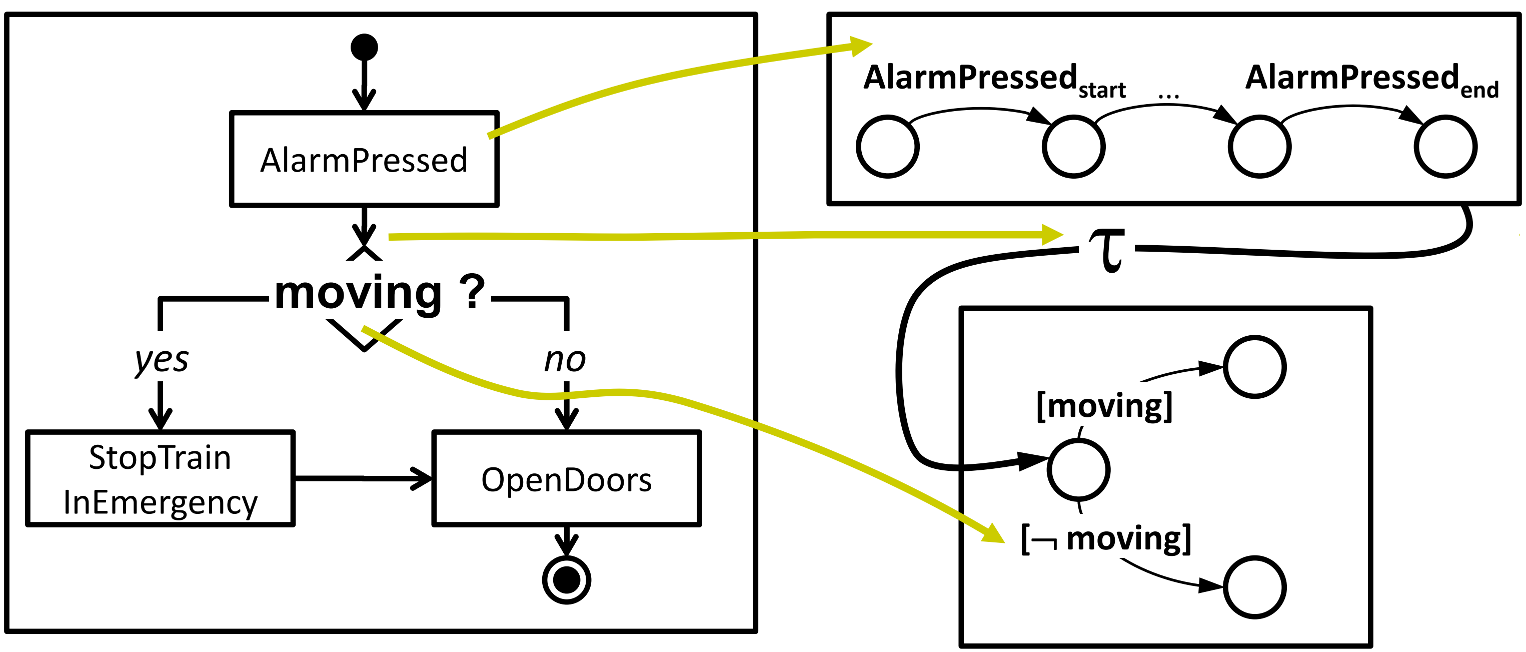
2

LTS, set of event traces



From g-hMSC to g-LTS
Synthesis algorithm

- Bricks connected with unobservable τ transitions
  - Extension of [Uch03] from h-MSC to LTS
  - Introduction of *start* and *end* events to support more accurate fluent definitions
  - Mapping preserved for feedback of g-LTS-driven analysis



From g-LTS to pure LTS

Declarative trace semantics

2

A trace $(Init,\ \langle l_0, \ldots \rangle)$ is accepted from state q_0 by a guarded LTS $(Q, \Sigma, \Phi, \delta, q_0, C_0)$ iff for every *i:*

- trace inclusion

  $\exists\ q_{i+1} \in Q : (q_i,\ l_i,\ q_{i+1}) \in \delta$
- admissible start

  $Init \models C_0$
- guard satisfaction

  $S_i \models l_i$ if $l_i \in 2^{\Phi}$

S_i : state invariant after i-th event in trace $(S_0 = Init)$

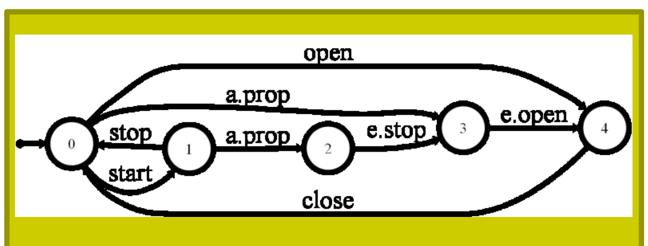
From g-LTS to pure LTS
Synthesis algorithm

2

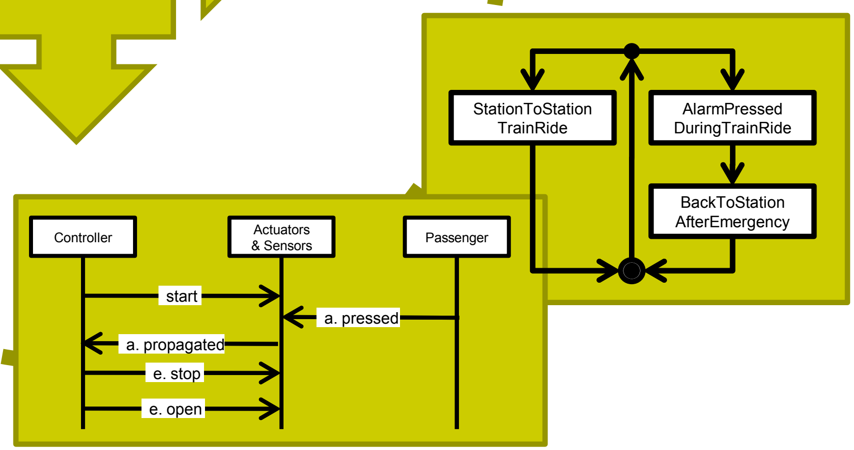
- Provides the set of event traces of a g-LTS
  - No roundtrip, no state mapping preserved
  - Allows model-checking g-hMSC and g-LTS against LTL safety properties
- Algorithm for composing multiple automata
  - Super LTS
    - guarded LTS where guards are replaced by special events
    - to meet the *trace inclusion* condition
  - Initializer LTS
    - to meet the *admissible start* condition
  - Fluent LTS (one per fluent)
    - to meet the *guard satisfaction* condition

From MSC and hMSC to LTS
Chapter 4

How to generalize a set of MSC scenarios as LTS state machines?



State machines as LTS



Scenarios as MSC and hMSC

From MSC to LTS
Problem statement

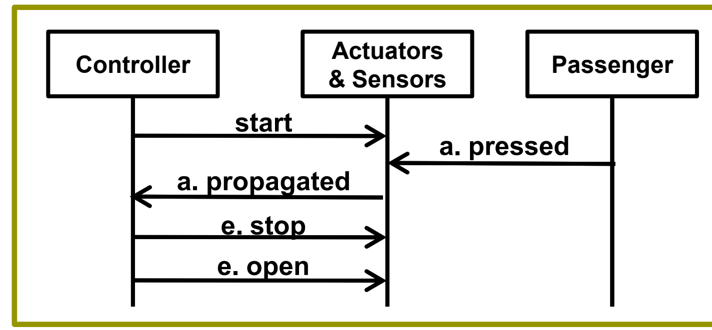
- Given a scenario collection $S_c=(S_+,S_-)$ showing typical examples of the system usage,

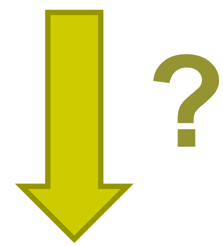
  synthesize the system as a composition of agent LTS

  $$S = A_{lts1} \,||\, \ldots \,||\, A_{ltsn}$$

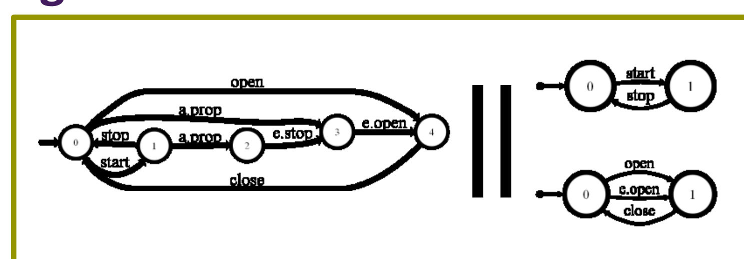
  such that it correctly accepts S_+ and rejects S_-

Positive & Negative MSCs

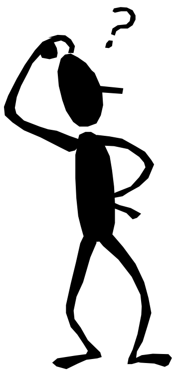




Agent LTS



Synthesis requirements



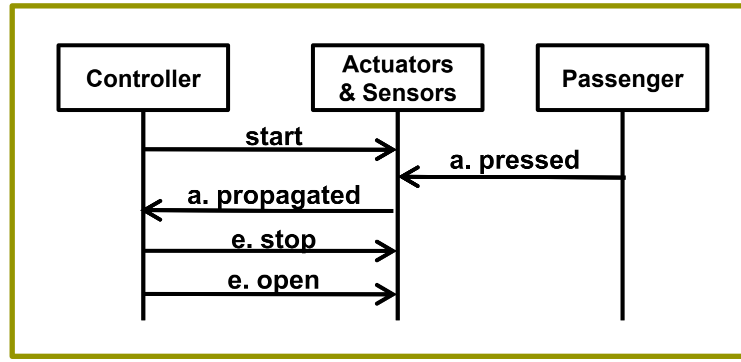
- Because of end-user involvement, the solution should work
  - From end-user positive and negative scenarios …
  - … and scenarios only : avoid hMSC, state variables annotations, operations formalization (pre/post), etc.
  - but such additional information could/should be used when available
- Initial scenario collection being incomplete, the approach
  - should support the elicitation of additional, “interesting” positive/negative scenarios
  - should be incremental : the models should be incrementally refinable as further scenarios become available

Solution overview

- **Generalization**
  - Interactive automaton induction
  - Guided by generated scenario questions classified as positive or negative by the end-user
    - Query-Driven State Merging (QSM)
  - Constrained by additional state information when available (state variables, legacy components, goals, etc.)
- **Decomposition**
  - Standard automaton algorithms (ε-moves, determinization, minimization)

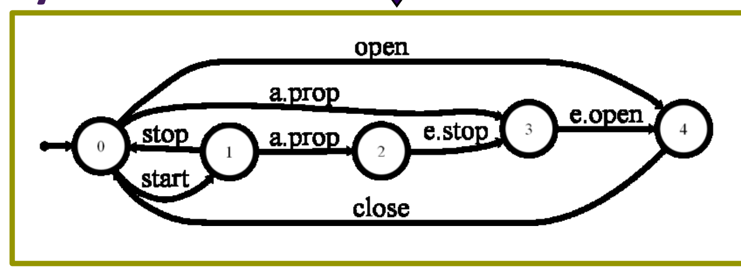
C. Damas, B. Lambeau, P. Dupont and A. van Lamsweerde, *Generating Annotated Behavior Models from End-User Scenarios*, IEEE Transactions on Software Engineering, Special Issue on Interaction and State-based Modeling, Vol. 31, No. 12, pp. 1056-1073, 2005 .

Positive & Negative MSCs



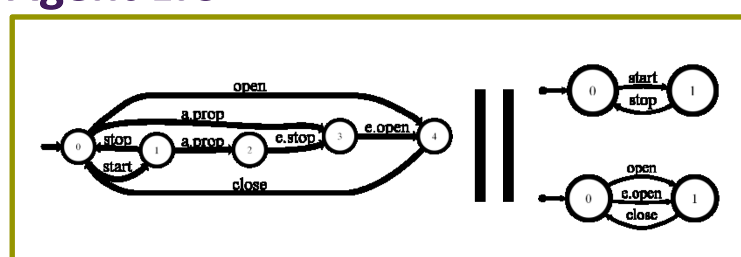
Generalization

System LTS



Decomposition

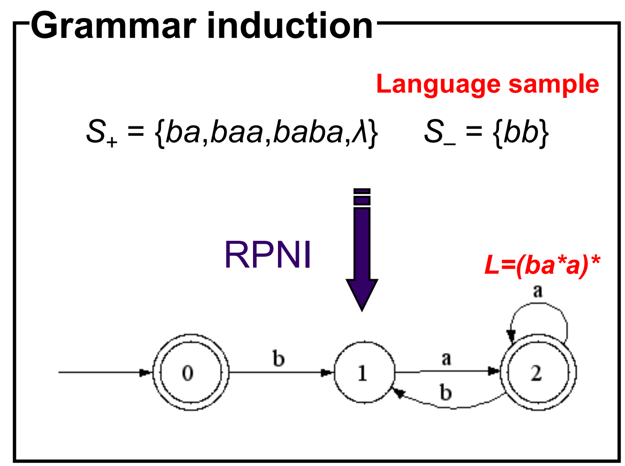
Agent LTS

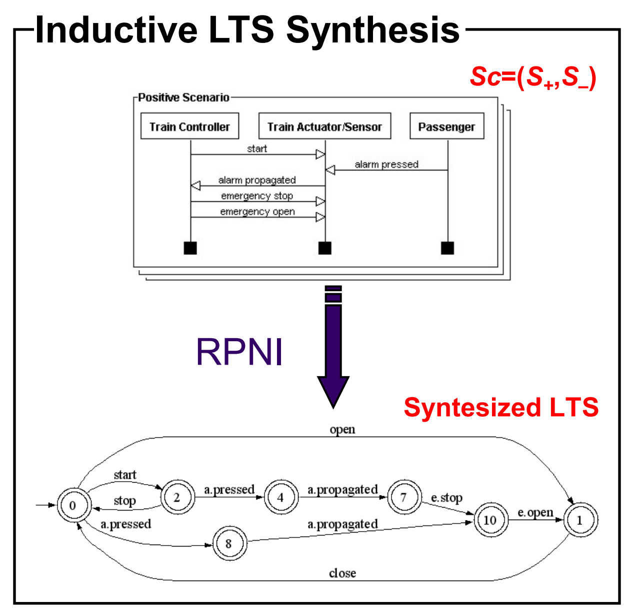


From grammar induction to LTS synthesis

- Grammar induction
  - Learning a regular language $L=(ba^*a)^*$
  - Represented by a DFA
  - From positive and negative strings
    - $S_+ = \{ba, baa, baba, \lambda\}$ and $S_- = \{bb\}$
  - Regular Positive and Negative Inference (RPNI + variants)

- LTS Synthesis
  - Learning a System
  - Represented by a LTS (a DFA)
  - From positive and negative scenarios

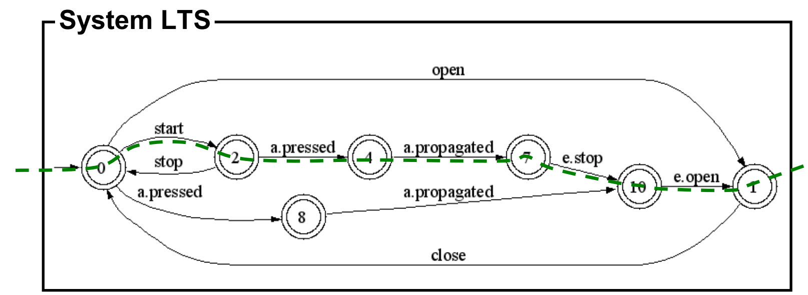


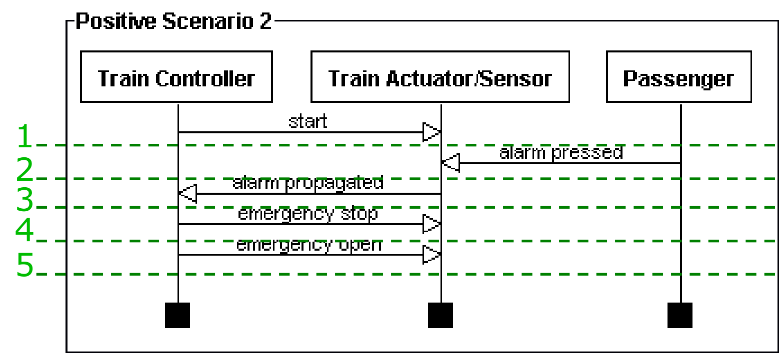


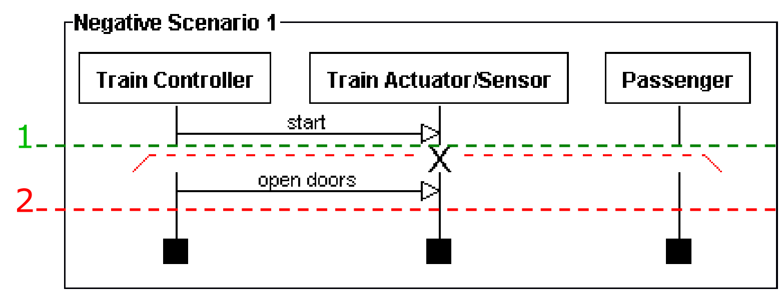
Scenarios as strings

- The system LTS represents the learned language *L*
- Scenarios are observed strings on *L*

- LTS contains only accepting states
  - A positive scenario defines a set of positive strings
  - A negative scenario defines a set of positive strings and one negative string







Query-Driven State Merging (QSM)

Input: A non-empty initial scenario collection $Sc = (S+, S-)$

Output: An automaton A consistent with an ex[...] collection $Sc = (S+, S-)$

$A \leftarrow$ `Initialize`(S_+)

while $(q, q') \leftarrow$ `ChooseStatePairs`(A) **do**

  $A_{new} \leftarrow$ `Merge`(A, q, q')

  if `Compatible`(A_{new}, S_-) **then**

    ok $\leftarrow$ true

    while $Q \leftarrow$ `GenerateQuestion`(A, A_{new}) **do**

      if `CheckWithEndUser`(Q) **then**

        $S_+ \leftarrow S_+ \cup Q$

      else

        $S_- \leftarrow S_- \cup Q$

        ok $\leftarrow$ false

        break

    if ok **then**

      $A \leftarrow A_{new}$

return A

1. Build an initial solution A

2. Generalize $A \rightarrow A_{new}$

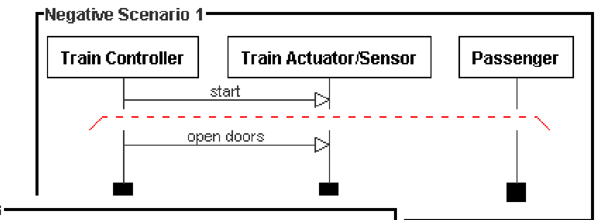
3. With user involvement

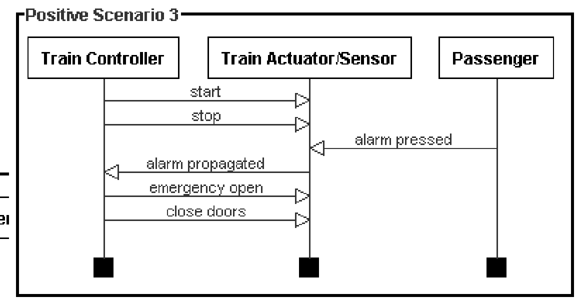
4. Keep it when correct $A_{new} \rightarrow A$

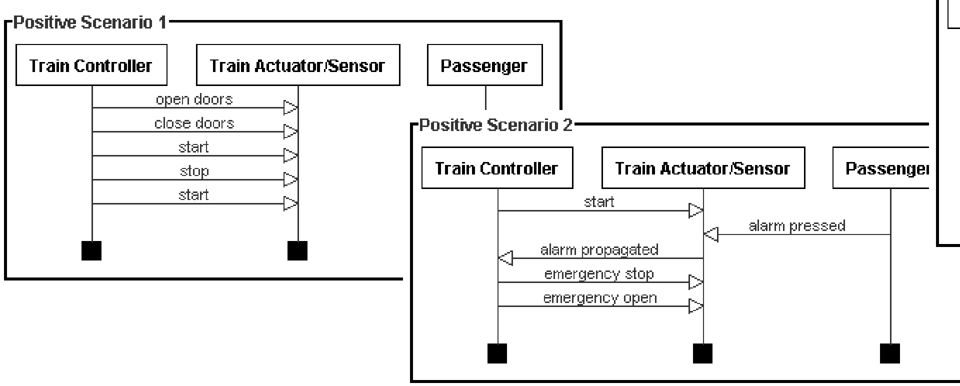
5. Return synthesized system

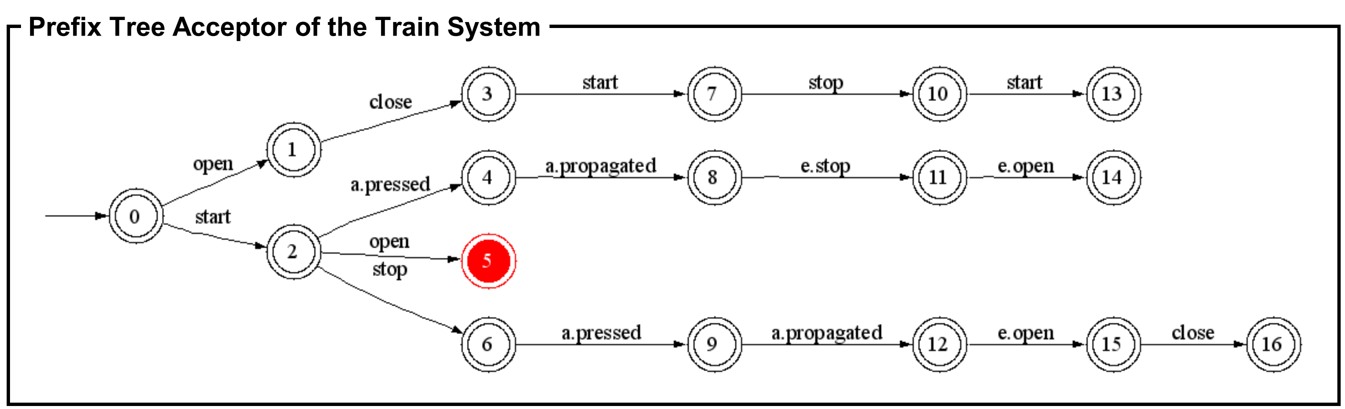
Initial solution, the PTA

- Prefix Tree Acceptor (PTA)
  - Largest DFA accepting S_+ while rejecting S_-
  - No generalization of observed behaviors
  - Red states are error states

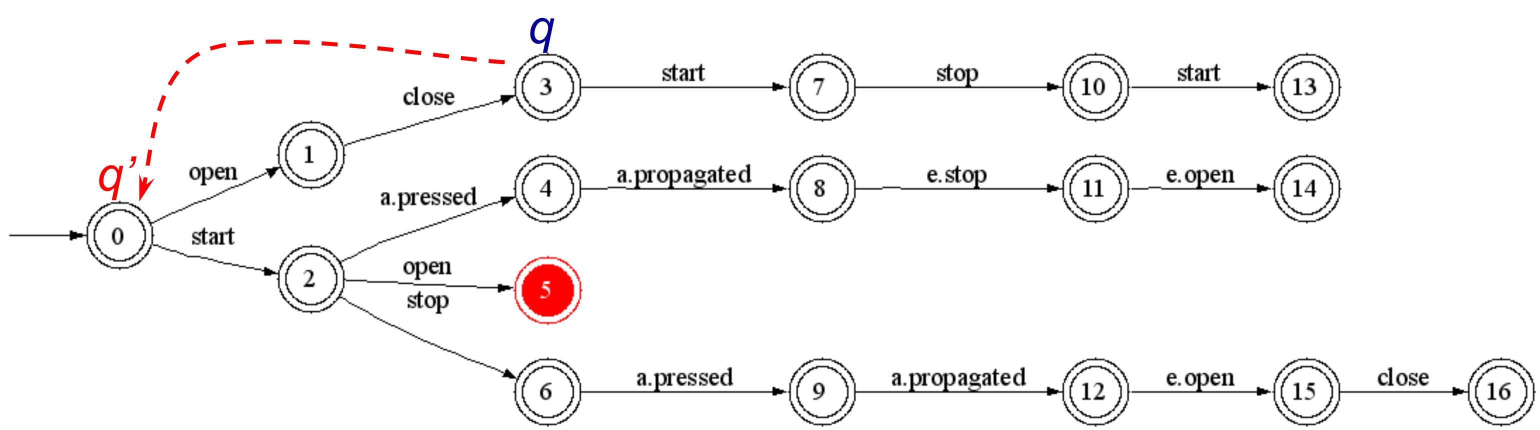






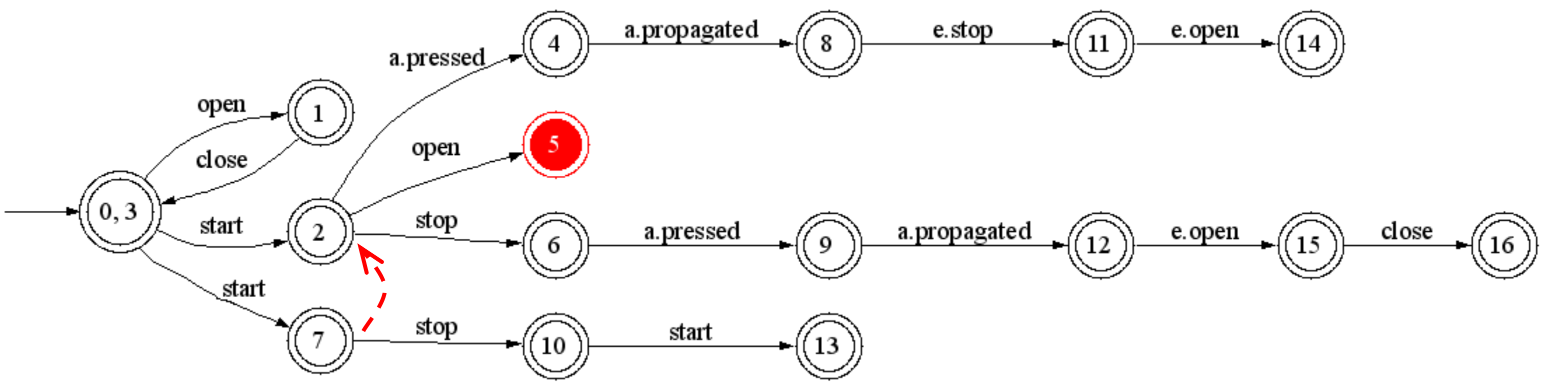


Generalization - merge



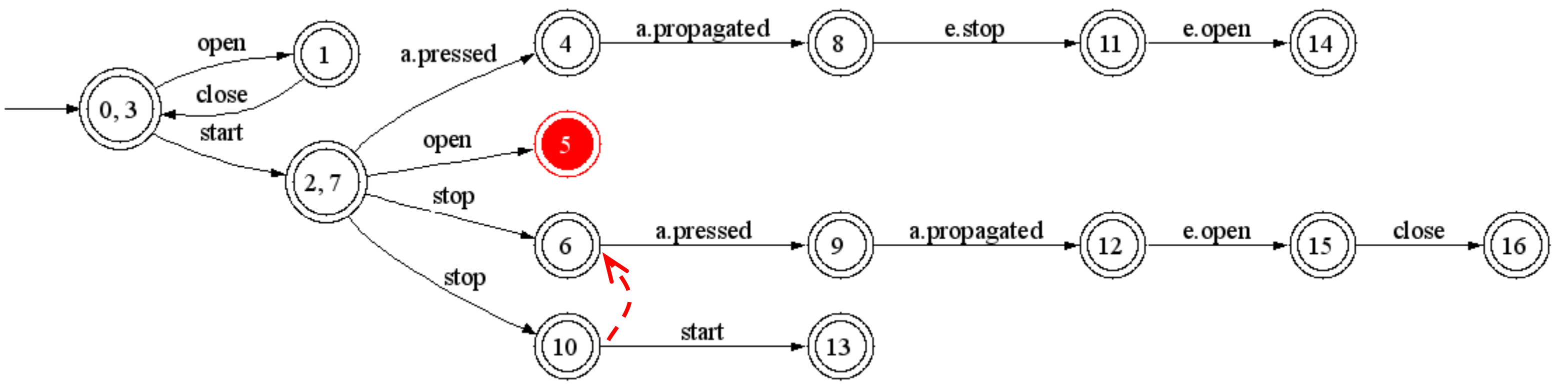
- Consider merging of *q*=*3* with *q'*=*0*

Generalization - determinize



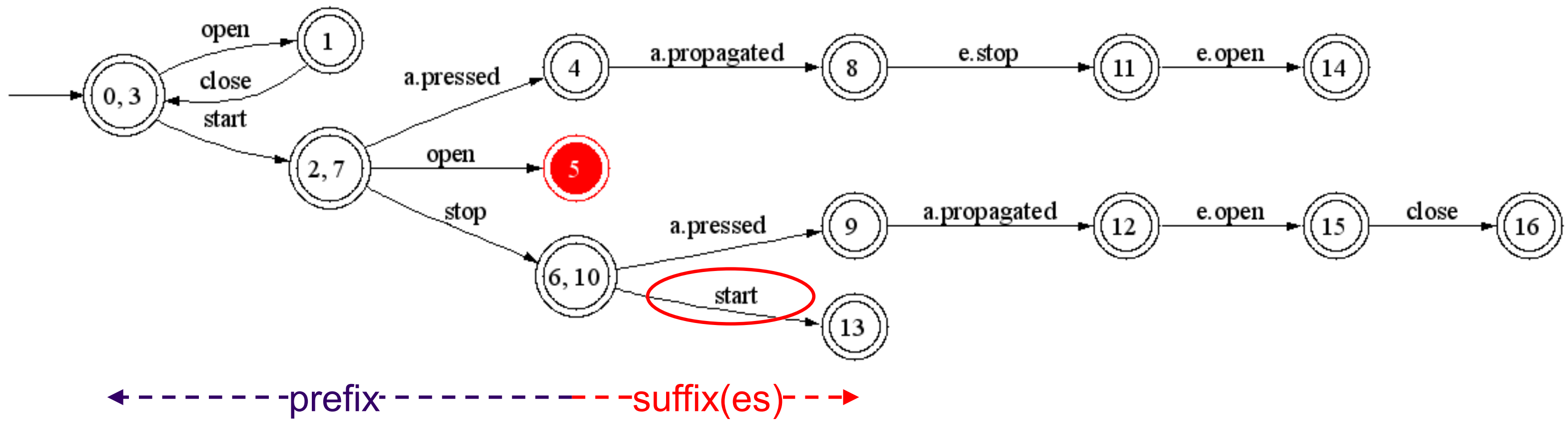
- State 0 contains two outgoing transitions labeled « start »
  - merge 7 and 2 for determinization

Generalization - determinize

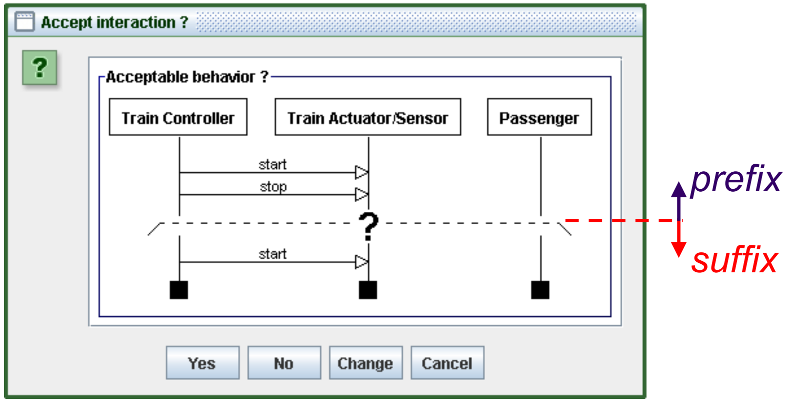


- State 2 contains two outgoing transitions labeled « stop »
  - merge 10 and 6 for determinization

Generalization - questions



- State 6 gains an outgoing transition labeled "start"
  - Leads to generated scenario questions
    - Used prefix is the shortest history leading to state 6
    - Suffixes are the gained continuations of 6 (only one here)

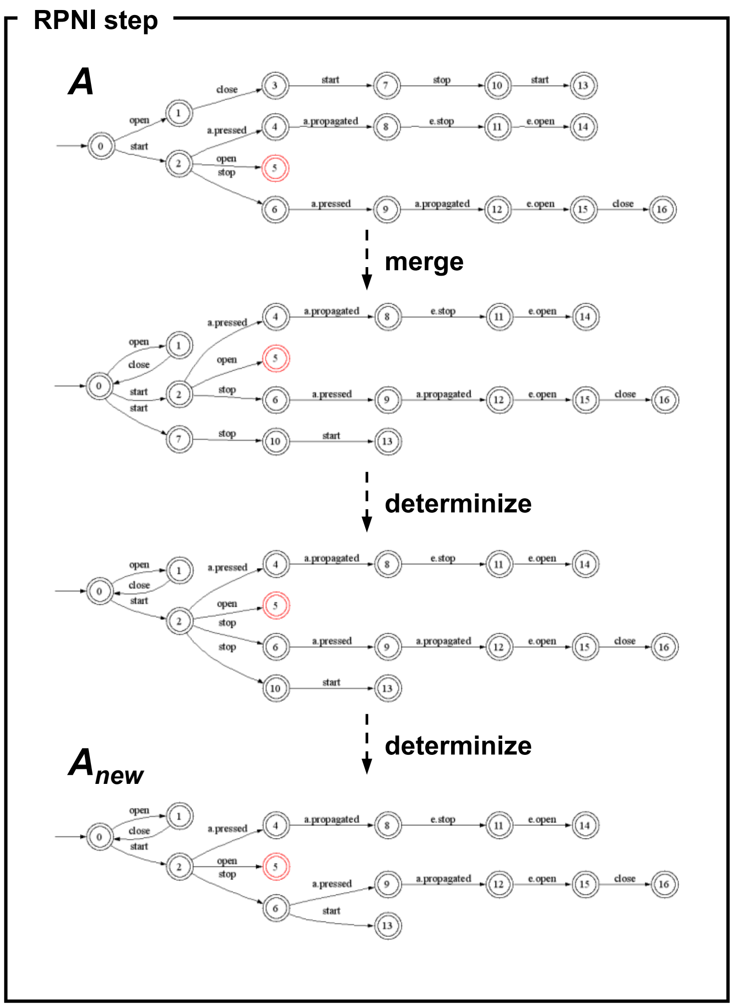


Generated scenarios

- Generated scenarios are
  - Classified by the user as positive or negative examples
  - Added to the initial scenario collection $S_c=(S_+,S_-)$
- Generation procedure
  - Constraints the generalization process with scenarios only
  - Relies on sound results from grammar induction
    - Notion of characteristic sample
  - Provides the elicitation of additional, “interesting” positive/negative scenarios

Intermediate solution

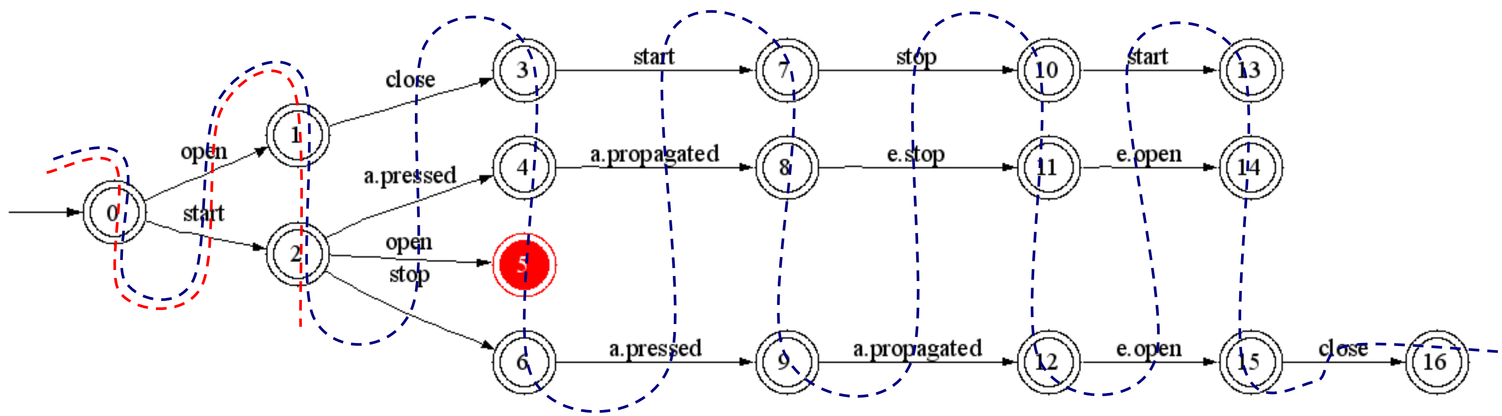
- States q=3 with q'=0 are merged, and stay merged forever ($A \leftarrow A_{new}$), if
  - The merging and determinization process does not lead to merge an error state with a non error one
  - All generated questions are classified as positive scenarios by the user
- Otherwise, the solution A_{new} is discarded
- The algorithm continues, choosing another merging pair (q, q'), until no pair is available



Order of state pair selection

- RPNI considers state pairs in order

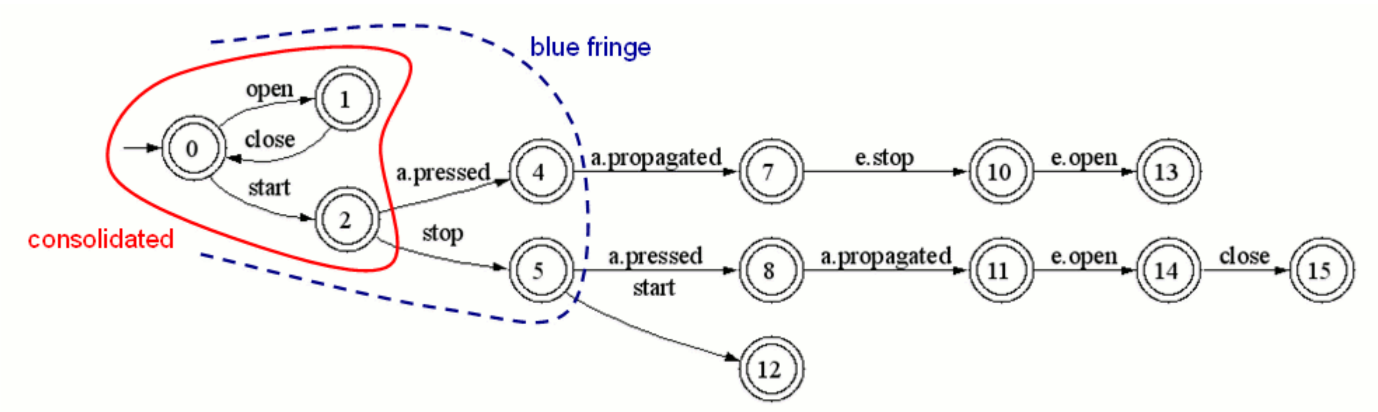
```
for each state q with rank i
    for each state q' with rank j<i
```



- QSM relies on the Blue-Fringe heuristics
  - Evidence-driven state merging
  - Evaluate candidate state pair and choose the best one
  - Reduces the number of submitted questions in practice

Evidence-driven merging

- Consider an intermediate solution *A*
  - Kernel states have been shown to be incompatible with each other
  - Fringe states are situated at one letter of a kernel state



- RPNI tries to merge a fringe state with a kernel one, in standard order
  - questions here : (4,0) ; (4,1) ; (4,2) rejected ; (5,0) accepted
- Blue-Fringe extension selects pairs (fringe,kernel) according to an evidence evaluation function, based on shared continuations
  - questions here : (5,2) rejected, (5,0) accepted
  - 3 questions only on the entire train example (2 accepted and 1 rejected)

Pruning the search space

- Risk of poor generalization
  - Without questions (if no user available)
  - Due to high importance of negative scenarios (non intuitive in the first place)

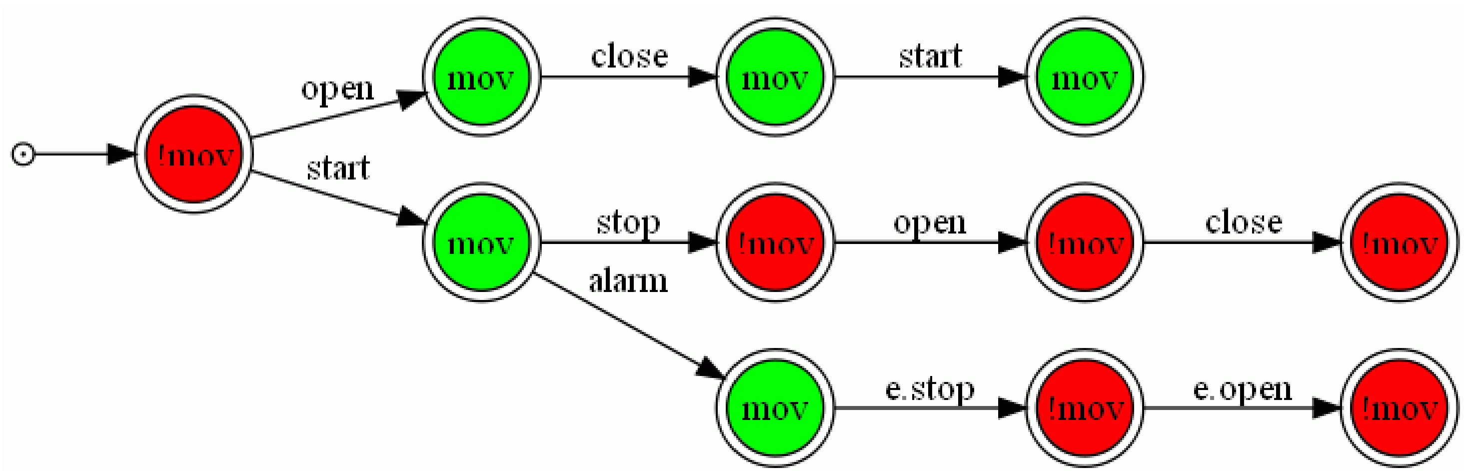
- Too many questions, often rejected
  - On the train example : 20 questions (3 to be accepted and 17 to be rejected)

- => Use available information to prune the search space
  - State information (e.g. fluents decorations)
  - Goals (safety properties in particular)
  - Control information (sequence, loops, ... from h-MSC)

Pruning with fluent decorations

- A PTA can be decorated using available fluent definitions

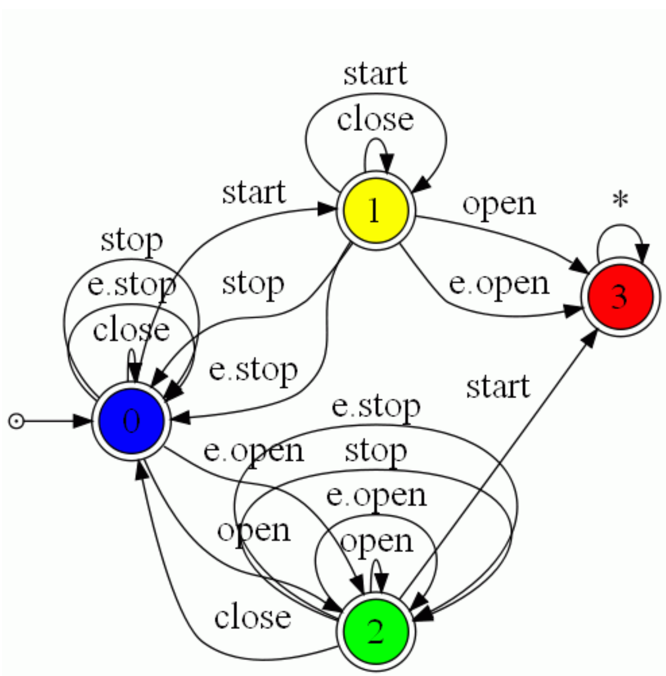
```
fluent mov(ing) = <{start},
                   {stop, e.stop}> initially false
```

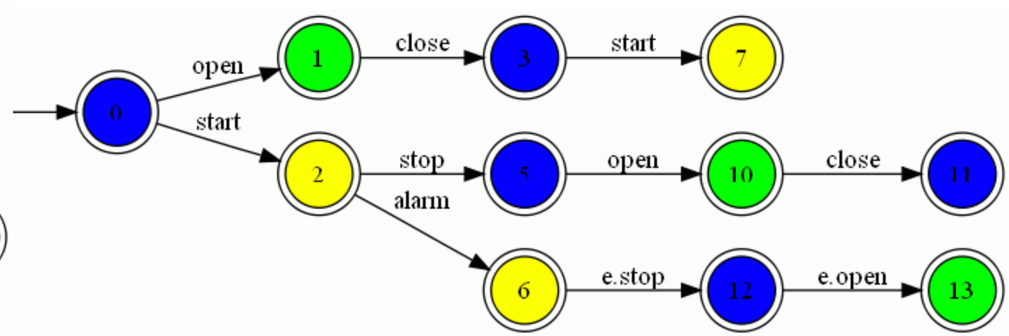


- Induction can be constrained ...
  - Avoid merging states with inconsistent decorations
  - e.g. avoid merging a state where the train is moving with another state where the train is not moving !

Pruning with goals

- *Tester LTS* for the safety property [Gia03]
  - Color PTA states with corresponding states
  - Avoid merging PTA states of different color
  - Enforces multi-model consistency by construction





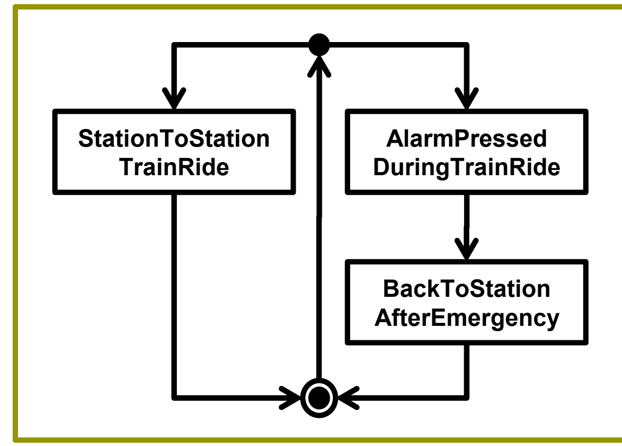
`Maintain[Doors Closed While Moving]`

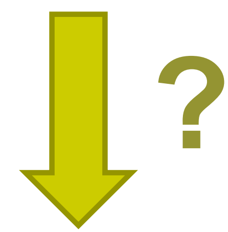
Pruning with control information

- What if PTA states are known to be equivalent ?
  - mandatory merge constraints
  - counterpart of state coloring

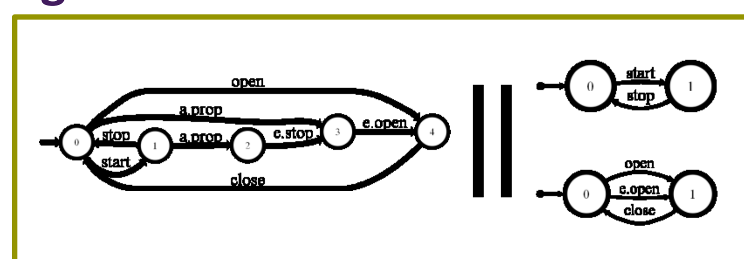
- Alternatively, how to use a h-MSC instead of set of MSCs as input ?
  - Relax assumption of MSCs starting in same state
  - Allow using sequence, loops, ...

hMSC





Agent LTS

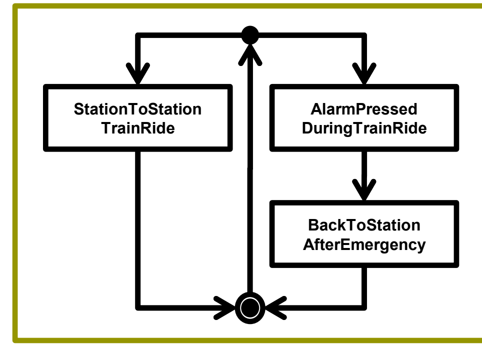


Pruning with control information

The Uch03 + ASM approach

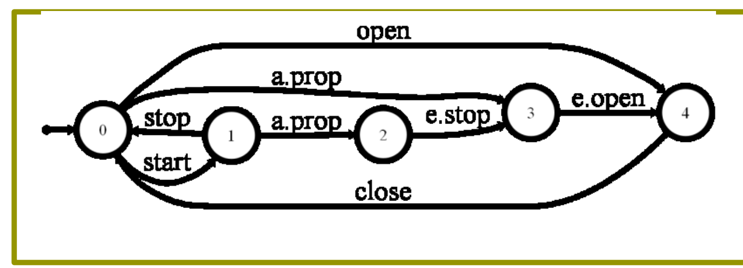
- Automaton State Merging (ASM)
  - Generalizes a DFA under control of a negative sample (not shown)
  - No oracle / user query support
  - But still compatible with previous pruning methods
- Also MSM & ASM* algorithms

hMSC



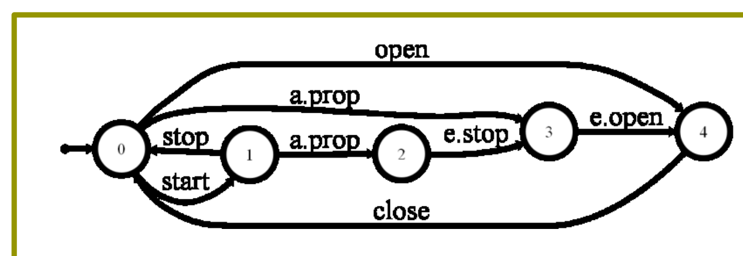
Using [Uch03]

Equivalent LTS

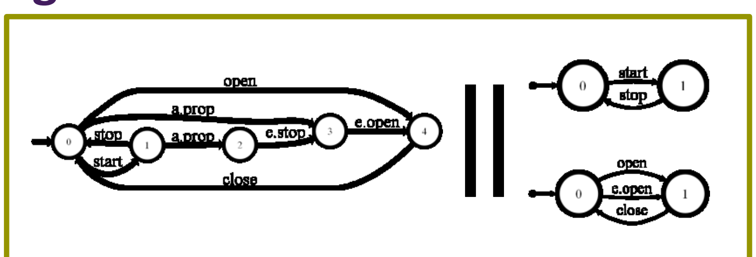


ASM

Generalized LTS



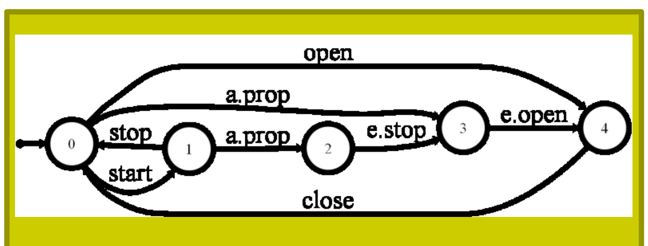
Agent LTS



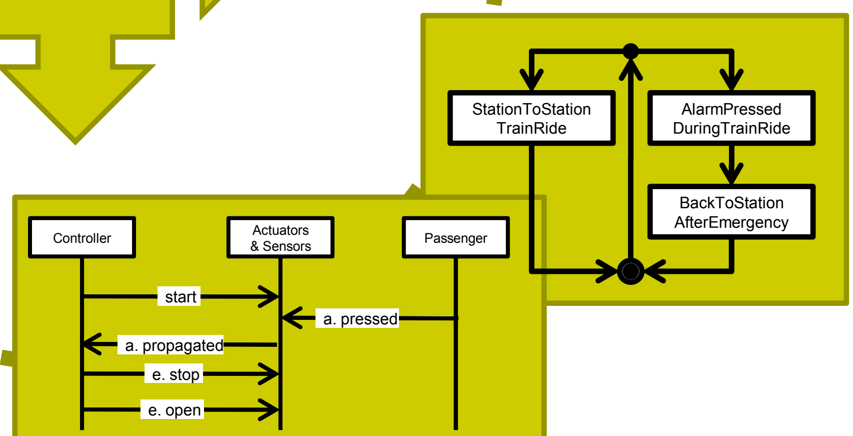
Evaluation
Chapter 5

How do the approach(es) perform

- ***On RE case studies ?***
- ***On synthetic data ?***



State machines as LTS



Scenarios as MSC and hMSC

RE case studies

- Guarded hMSC $\rightarrow$ LTS
  - Medical case-studies can be found in [Dam09, Dam11]
  - Technique implicitely used by the model-checker (see tool support)

- LTS Induction evaluated on three case studies
  - Small & medium sizes (mine pump, train, phone)
  - Blue-Fringe strategy required in practice
    - To avoid large number of user queries
    - To reach good model accuracy
  - Additional state information is very useful as well
    - Coloring & mandatory merge constraints help reaching better accuracy

Synthetic data

- QSM & ASM evaluated
  - Machines & Sample generation following the protocol of Abbadingo Contest
    - Different state machine sizes & sample sparsities
  - Comparison with RPNI & Blue-Fringe
    - How well do queries help (i.e. accuracy & convergence) ?
    - How well do mandatory merge constraints help ?
- In short, all experiments confirm what is expected
  - Inject all state information you can have
    - reaching better model accuracy
    - with less raw data as input (i.e. sample / scenarios)
    - and it goes faster

Discussion
Evaluation on synthetic data - issues

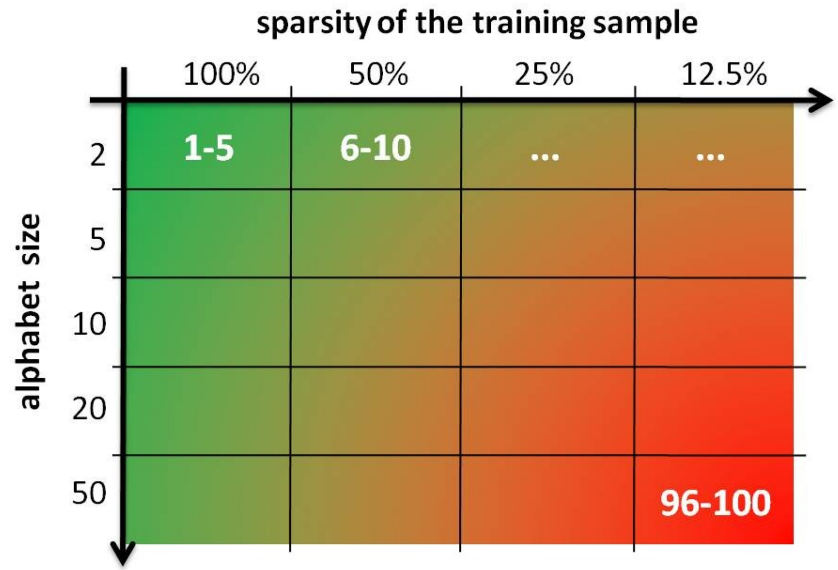
- Alphabet size
  - Binary alphabets only; state of the art in GI evaluations
  - In contrast, more than 30 events is commonly observed with behavior models
- State machines
  - No variance of state degree, as a consequence of binary alphabet
  - States with high in-/out degrees are commonly observed with behavior model (capturing system *idle*, *halt*, *immediate response*)
- Samples
  - Randomly generated from uniform distribution of strings
  - Not representative of what should be generated "from the
  - machine", especially on large alphabets

The Stamina Competition

- Online Regular Induction Contest
  - Extends former competitions, especially Abbadingo
  - Cross-fertilization between the machine learning and software engineering communities
- Previous issues adressed
  - Focus on the complexity of the learning with respect to the alphabet size
  - Adapted generation protocol for state machines and samples to mimic features of behavior models
- Not an evaluation of the thesis techniques *per se*
  - Unsupervised learning (i.e. no oracle, no queries)
  - No pruning with fluents, goals, control information

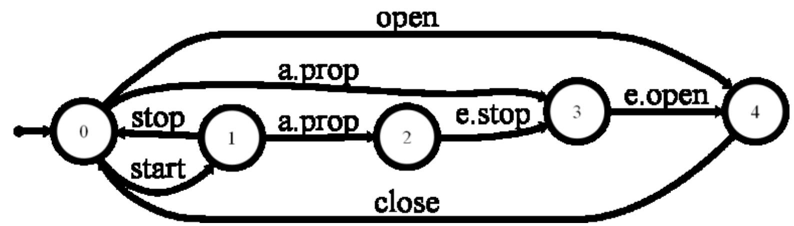
Competition overview

- 100 induction problems (20 cells of 5 problems)
- Two difficulty dimensions
  - alphabet size vs. sparsity of learning sample



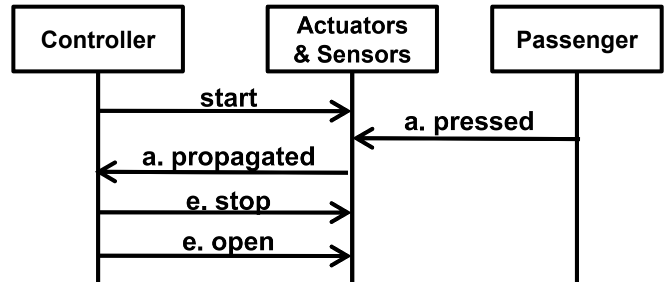
- Solving a problem
  - Download learning (labeled) and test (unlabeled) samples
  - Learn a model (typically a DFA)
  - Label the test sample using learned model
  - Submit labeling on the competition server

Scientific setup
State Machines



- Approach
  - Review of SE litterature to identify representative features of behavior models
  - Tuning of the Forest-fire algorithm to mimic these features
- Main features
  - Approximately 50 states (to avoid adding a third difficulty dimension to the competition)
  - Alphabet sizes ranging from 2 to 50 letters
  - Equal proportion of accepting vs. rejecting states
  - Large variance of degree distribution, to mimic behavior models

Scientific setup
Samples



- Approach
  - Generated by the target machine: random walk algorithm
  - Negative strings by randomly perturbing positive ones
    - three kinds of edit: substitution, insertion and deletion
  - Tuned to ensure good induction results using Blue-Fringe on the simplest problems
- Main features
  - Learning and test samples do not overlap
  - Learning samples may contain duplicates, as a consequence of the random walk generation
  - String length distribution: centered on *5 + depth(automaton)*

Scientific setup
Submission & Scoring

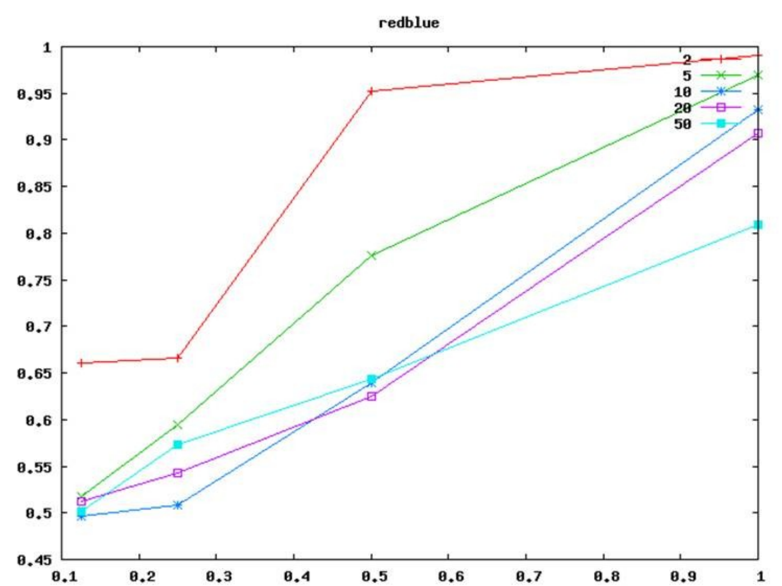
- Submission
  - Solutions submitted as binary strings labelling the test sample
  - Binary feedback (problem broken or not broken), to avoid hill-climbing
- Scoring
  - Balanced Classification Rate to place equal emphasis on accuracy in terms of positive and negative strings
  - Problem broken if BCR score >= 0.99
  - A cell is broken if all problems it contains are broken

Scientific setup
Baseline

- Problem grid empirically ajusted
  - To ensure good induction results using Blue-Fringe on the simplest problems
  - Without breaking the cell

sparsity of the training sample

alphabet size	100%	50%	25%	12.5%
2	0.989	0.95	0.67	0.66
5	0.96	0.77	0.60	0.51
10	0.93	0.64	0.50	0.50
20	0.90	0.63	0.54	0.50
50	0.80	0.64	0.57	0.50

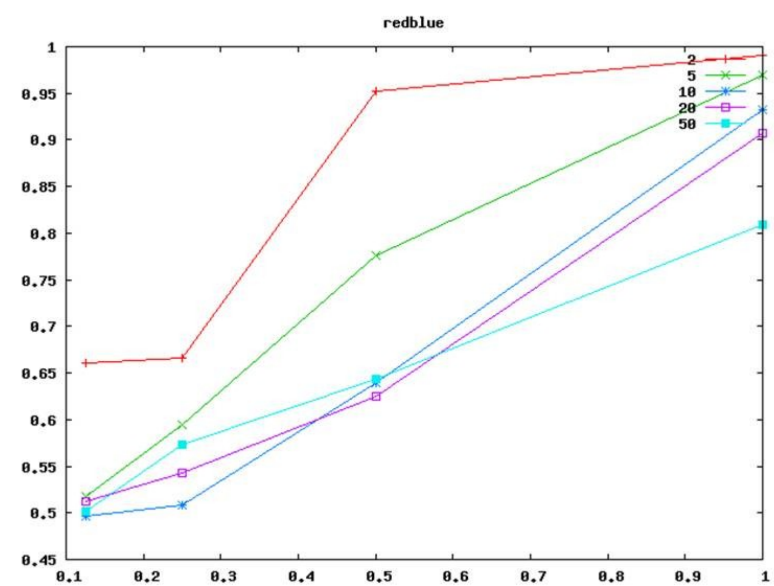


Scientific setup
Baseline: lessons learned

- RPNI and BlueFringe converge on largest alphabets
  - Theoretically expected, big samples needed in practice
- Size of the alphabet "hurts" convergence in practice
  - Confirms experimentally what we expected theoretically
  - Supports the interest of launching Stamina

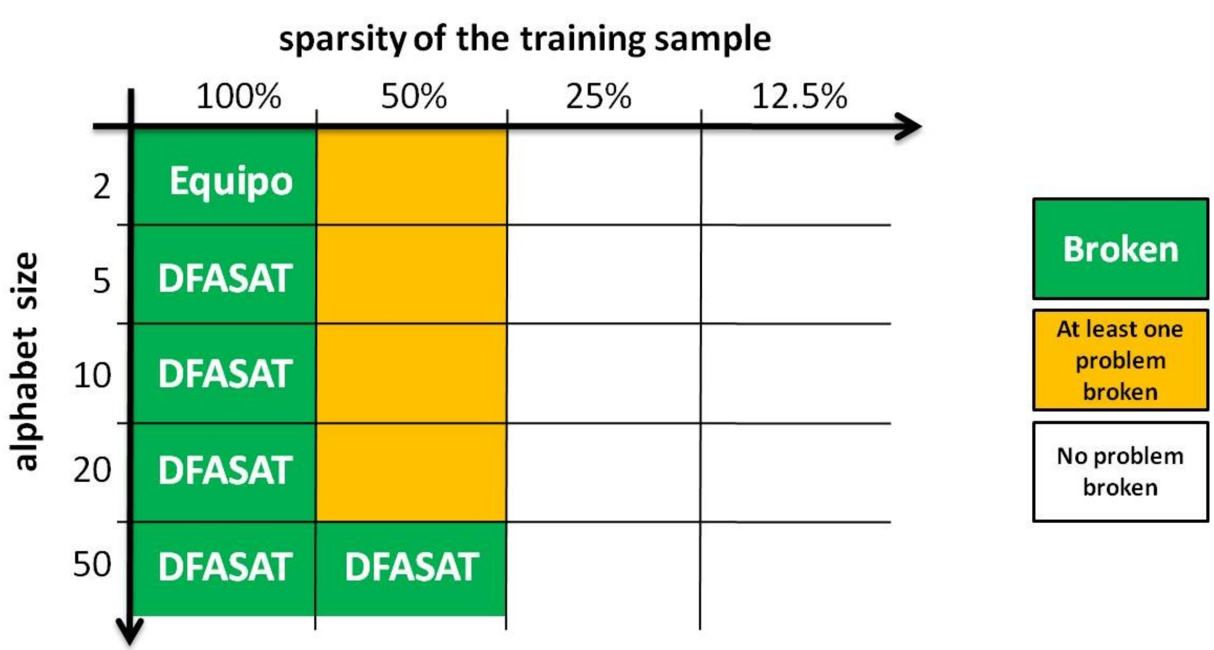
sparsity of the training sample

alphabet size	100%	50%	25%	12.5%
2	0.989	0.95	0.67	0.66
5	0.96	0.77	0.60	0.51
10	0.93	0.64	0.50	0.50
20	0.90	0.63	0.54	0.50
50	0.80	0.64	0.57	0.50



Participation overview

- Between march and december 2010 (official)
  - 1856 submissions made by 11 challengers
  - 65 winning submissions broke 42 problems
  - 6 cells broken, by 2 challengers (Equipo & **DFASAT**)



A big winner - DFASAT
Marijn Heule & Sicco Verwer

Blue-Fringe

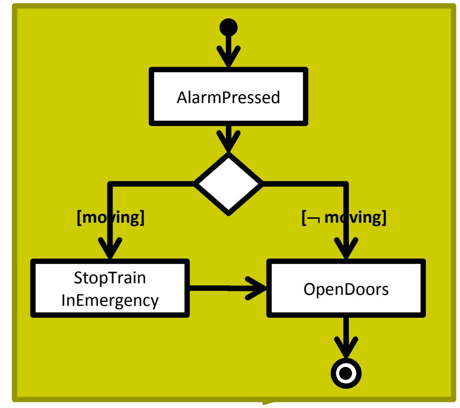
alphabet size	100%	50%	25%	12.5%
2	0.989	0.95	0.67	0.66
5	0.96	0.77	0.60	0.51
10	0.93	0.64	0.50	0.50
20	0.90	0.63	0.54	0.50
50	0.80	0.64	0.57	0.50

DFASAT

alphabet size	100%	50%	25%	12.5%
2	0.99	0.98	0.78	0.66
5	0.99	0.96		
10	0.99	0.97		
20	0.99	0.98		
50	0.99	0.99	0.96	

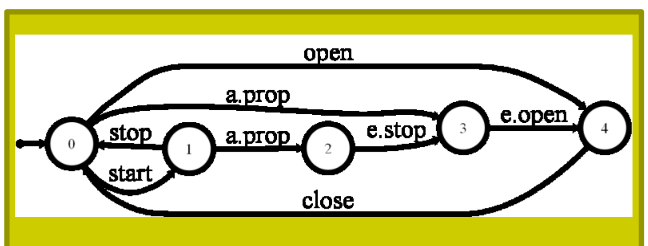
Tool Support
Chapter 6

Processes as g-hMSC



A model-checker for guarded hMSC (chapter 3)

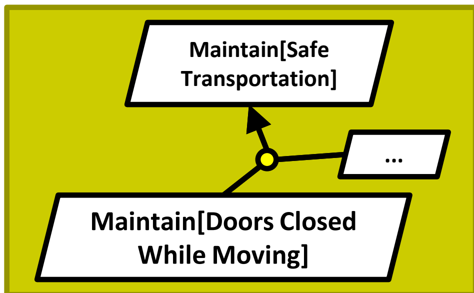
- *BDD-driven, efficient implementation of g-hMSC $\rightarrow$ g-LTS*



State machines as LTS

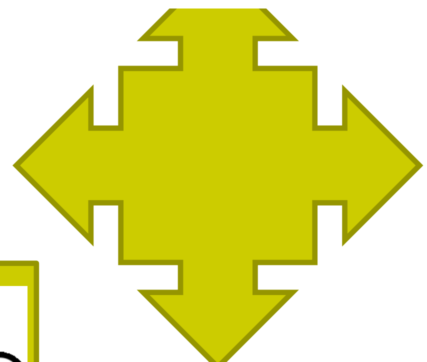
Tool Support
Chapter 6

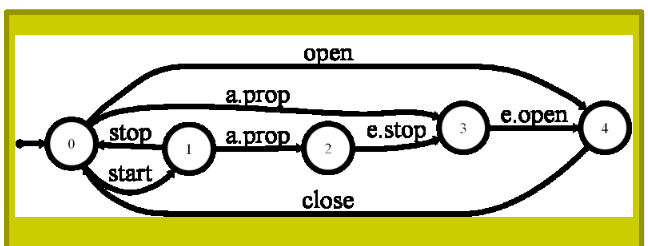
Goals as FLTL



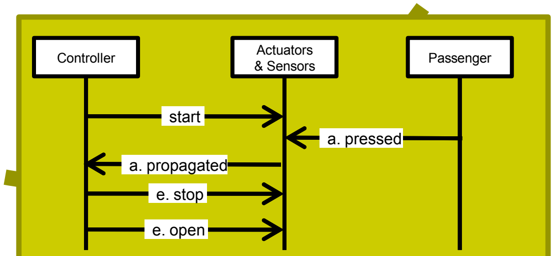
The ISIS Tool (chapter 4)

- *Interactive synthesis of LTS from positive & negative MSC*
- *Under control of fluents and goals*





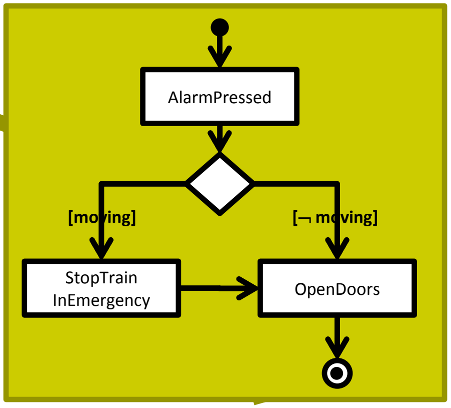
State machines as LTS



Scenarios as MSC and hMSC

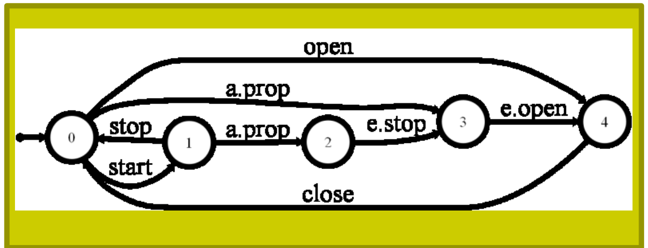
Tool Support
Chapter 6

Processes as g-hMSC



The Gisele Clinical Pathway Analyzer

- ***Implementation of decoration-driven model analysis (Damas thesis)***



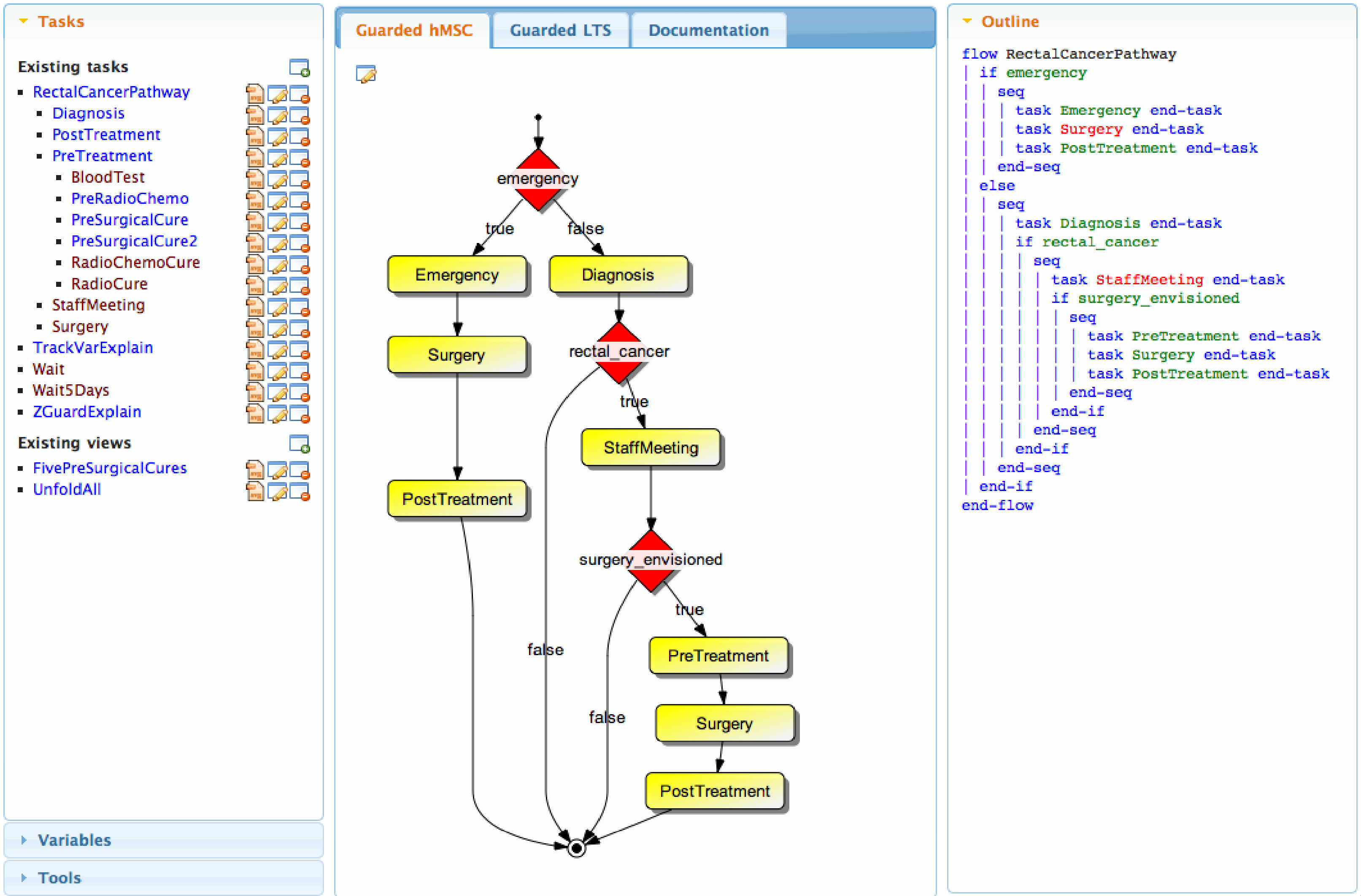
State machines as LTS

Context & Motivation
The Gisele Project

- Health-care processes are safety critical
  - Medical errors => 98,000 deaths every year in the US, over 1 million of non-lethal injuries
    - E.g. wrong patient, wrong dose, wrong timing, wrong channel
  - Major causes
    - complex coordination among clinical tasks, communication problems among actors, complex decisions

- Response
  - Clinical pathways, software support
  - Needed: models for analysis, error anticipation and enactment [Clarke08]

A Clinical Pathway Analyzer
Tool support of the Gisele project



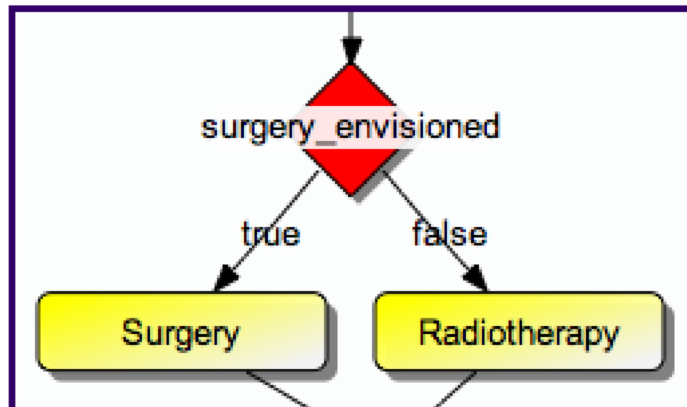
A Clinical Pathway Analyzer
Tool support of the Gisele project

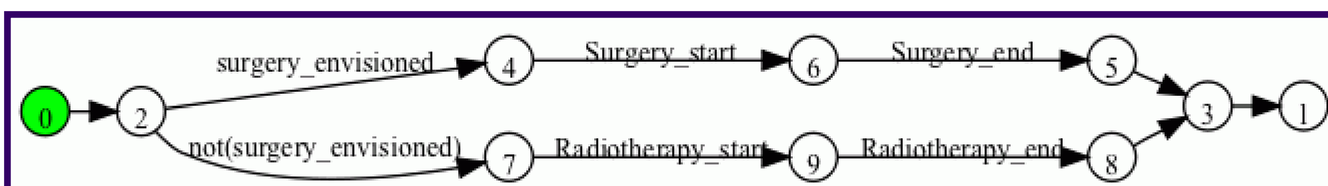
- Variety of analyses (cf. Damas thesis)
  - verification of functional & non-functional requirements
  - state-based, event-based
- Integrated within uniform formal framework and toolset
  - Guarded models (g-hMSC & g-LTS)
  - Fluents and tracking variables
  - Decoration-based analysis
- Incremental analysis throughout model building
  - according to model refinement structure
  - early, local, stepwise on model of varying granularity

Process modeling language

- Textual language
  - guarded commands
  - tasks and decisions
- Graphical counterpart
  - guarded High-Level Message Sequence Charts (g-hMSC)
- Semantics & analysis in terms of
  - guarded LTS (g-LTS), in turn defined by LTS (cf. chapter 3)

```
if surgery_envisioned
  Surgery
else
  Radiotherapy
end
```





Discussion

- Stamina
  - Related work or discussion in evaluation chapter ?
  - How far about presenting results ?
- Gisele tool
  - Medical case-studies suddently appear...
  - Almost about decoration-based analysis (the other thesis)
- Tool support
  - A quick poll: what are you interrested in?
- MSM & ASM
  - Not interactive, no queries
  - Shouldn't we move it to another chapter ?

Thesis Outline

- A Multi-View Modeling Framework
  - Event-based behavior models
  - State-based abstractions
  - Intentional models as goal graphs on uents
- Deductive synthesis of LTS models from guarded hMSCs
  - From guarded hMSC to guarded LTS
  - From guarded LTS to pure LTS
- Inductive synthesis of LTS models from MSC and hMSC models
  - From grammar induction to model induction
  - Interactive induction of LTS models from MSCs
  - Pruning the induction space with state information
  - Pruning the induction space with goals
  - Pruning the induction space with control information
- Evaluation
  - ...
- Tool Support
  - ...