Representing Hierarchical State Machines in SMT-LIB

Nancy A. Day and Amirhossein Vakili

Waterloo Formal Methods Lab
Cheriton School of Computer Science
University of Waterloo

MiSE May 2016

Motivation

- Goal: Effective automated analysis of behavioural models early in the development process

Motivation

- Goal: Effective automated analysis of behavioural models early in the development process
- What should abstract behavioural models consist of?
  - Data abstractions:
    - Abstract datatypes (e.g., sets, relations, integers, uninterpreted types)

Motivation

- Goal: Effective automated analysis of behavioural models early in the development process
- What should abstract behavioural models consist of?
  - Data abstractions:
    - Abstract datatypes (e.g., sets, relations, integers, uninterpreted types)
  - Control abstractions:
    - State machines with hierarchy and concurrency

Motivation

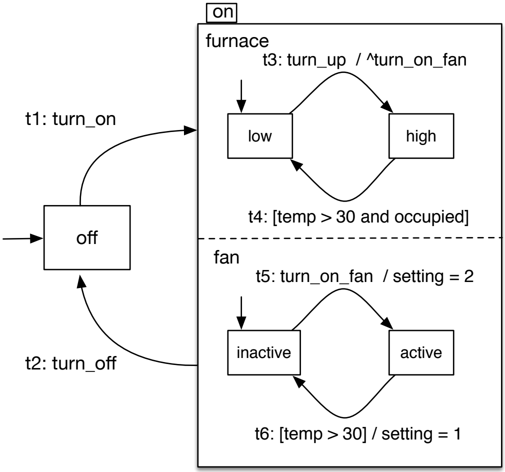
- Goal: Effective automated analysis of behavioural models early in the development process
- What should abstract behavioural models consist of?
  - Data abstractions:
    - Abstract datatypes (e.g., sets, relations, integers, uninterpreted types)
  - Control abstractions:
    - State machines with hierarchy and concurrency
- Behavourial analysis options:
  - Model checking: control-oriented models
  - Automated first-order logic (FOL) provers: data-oriented models
  - Recent work on model checking in FOL

Motivation

- Goal: Effective automated analysis of behavioural models early in the development process
- What should abstract behavioural models consist of?
  - Data abstractions:
    - Abstract datatypes (e.g., sets, relations, integers, uninterpreted types)
  - Control abstractions:
    - State machines with hierarchy and concurrency
- Behavourial analysis options:
  - Model checking: control-oriented models
  - Automated first-order logic (FOL) provers: data-oriented models
  - Recent work on model checking in FOL
- Question: Can we model check abstract models that include both data and control abstractions using automated FOL solvers?

Hierarchical State Machines (HSMs)

Simple Heating
System HSM Model

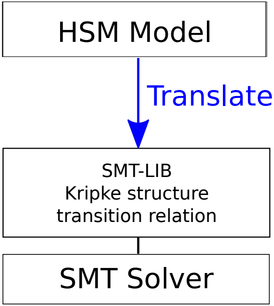


Examples: Statecharts, Stateflow, UML StateMachines
Variety in the semantics of these languages for similar syntax.

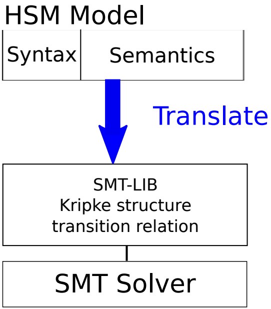
Automated FOL Solvers: SMT-LIB

- SMT-LIB = standard notation for SMT solvers
- SMT = Satisfiability Modulo Theories
- SMT solver = automated FOL theorem prover + standard interpretations for built-in types
- SMT-LIB contains S-expressions:
  - Declare types
  - Declare functions
  - Define functions
  - Assertions

Translation from HSM to SMT Solvers



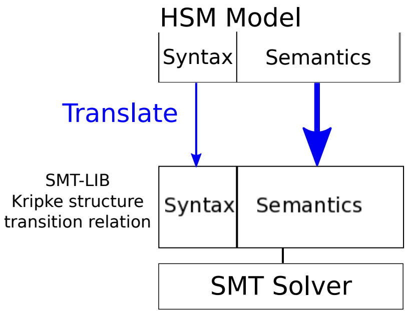
Translation from HSM to SMT Solvers



But . . .

- How do we support the variety of semantics of HSMs?

Translation from HSM to SMT Solvers



But . . .

- How do we support the variety of semantics of HSMs?
- Opportunity: Can we write FOL axioms/decision procedures for the semantics? (deductive analysis)

Representing HSMs in SMT-LIB

Goal: Represent the state machine syntax explicitly in SMT-LIB.

Challenges:

- Can we stay within a decidable fragment of FOL (at least for the syntax)?
- Can we use the rich datatypes native in SMT-LIB (transition guards and actions)?
- Can we support variable semantics for the control states?
  - FOL axioms to describe semantics of state hierarchy

Representing HSMs in SMT-LIB

Goal: Represent the state machine syntax explicitly in SMT-LIB.

Challenges:

- Can we stay within a decidable fragment of FOL (at least for the syntax)?
- Can we use the rich datatypes native in SMT-LIB (transition guards and actions)?
- Can we support variable semantics for the control states?
    - FOL axioms to describe semantics of state hierarchy

Contribution: A standard way to write HSM syntax in SMT-LIB.

Representing the State Hierarchy

```
(declare-sort _State 0)
(declare-fun _root () _State)

; declare every state name
(declare-fun off () _State)
(declare-fun on () _State)
(declare-fun furnace () _State)
...
```

Required types/functions begin with underscores.

Representing the State Hierarchy

```
(declare-sort _Kind 0)
(declare-fun _basic () _Kind)
(declare-fun _and () _Kind)
(declare-fun _or () _Kind)

; represent the state hierarchy
(define-fun _kind ((s _State)) _Kind
  (ite (or (= s _root) (= s furnace) (= s fan)) _or
  (ite (= s on) _and
  ...

(define-fun _parent ((s _State)) _State
  (ite (= s _root) _no_state   ; to represent a partial fcn
  (ite (or (= s off) (= s on)) _root
  ...
```

Representing States: Design Decision #1

Definitions vs Axioms

- State hierarchy is formalized using definitions of accessor functions.
- Options considered:
  - Recursive datatypes - not yet fully supported in all SMT solvers
  - Axioms rather than definitions, i.e.,

```
(assert (= (_kind (_root)) _or))
```

  - The use of axioms requires a quantifier to express the default case.
  - By not using quantifiers, we stay within a decidable fragment of FOL (logic of uninterpreted functions).

Representing Transitions: Design Decision #2

Deep vs Shallow Embeddings

- We have a deep embedding of the state hierarchy (its own new datatype) so we can write axioms/decision procedures about the semantics of the state hierarchy.
  - This is where languages vary in their semantics: which set of transitions are taken in a step?
- But, we want a shallow embedding of the transition labels (use native SMT-LIB datatypes).
  - Separate semantic axioms cannot access the contents of the guard or action.
  - Instead the semantic axioms must rely on a description of the effect of the guard or action relevant for determining the overall meaning of the model.

Representing Transitions

The semantics of an HSM is a transition relation between two vectors of configuration elements.

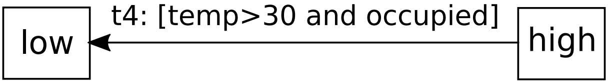
Representing Transitions

The semantics of an HSM is a transition relation between two vectors of configuration elements.

To write these axioms, we need to know the following about the transition labels:

- _guard : takes a transition name and a configuration and returns true if the guard is true in that configuration

Representing Transitions




```
; declare constants for transition names
...

; true or false in a configuration
(define-fun _guard ((t _Tran)
                    ; configuration elements
                    (temp Int) (occupied Bool) (setting Int))
                    Bool
  (or (and (= t t4) (and (> temp 30) occupied))
      (and (= t t6) (> temp 30))
      (not (or (= t t4) (= t t6)))))
```

This will be cleaner when SMT-LIB supports records.

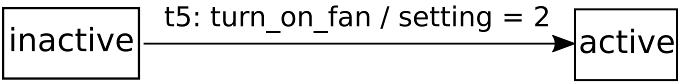
Representing Transitions

The semantics of an HSM is a transition relation between two vectors of configuration elements.

To write these axioms, we need to know the following about the transition labels:

- _guard : takes a transition name and a configuration and returns true if the guard is true in that configuration
- _action: takes a transition name and two configurations and returns true if the action of the transition took place between these two configurations

Representing Transitions




```
; true or false in a pair of configurations
(define-fun _action
      ((t _Tran)
       ; configuration elements
       (temp Int) (occupied Bool) (setting Int)
       ; next values of config elements
       (temp_n Int) (occupied_n Bool) (setting_n Int)
       Bool
  (or (and (= t t5) (= setting_n 2))
      (and (= t t6) (= setting_n 1))
      (not (or (= t t4) (= t t6)))))
```

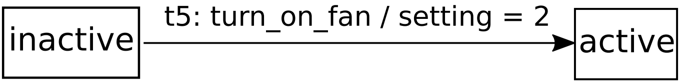
Representing Transitions

The semantics of an HSM is a transition relation between two vectors of configuration elements.

To write these axioms, we need to know the following about the transition labels:

- _guard : takes a transition name and a configuration and returns true if the guard is true in that configuration
- _action: takes a transition name and two configurations and returns true if the action of the transition took place between these two configurations
- _change_*conf_element*: returns true if a transition affects this conf element

Representing Transitions




```
; per configuration element
; does a transition constrain it?
(define-fun _change_setting ((t _Tran)) Bool
    (or (= t t5) (= t t6)))
```

Remaining Design Decisions

3. Modelling Events
   - An event is an instantaneous occurrence to which the system reacts.
   - Deeply embedded.
   - Modeller can add function to create them: entered(state).
4. Invariants
   - Invariants express parts of the model or its environment declaratively.
   - No need to make them an explicit element of the HSM in SMT-LIB:
     - Model them separately and conjunct with transition relation.

Contributions

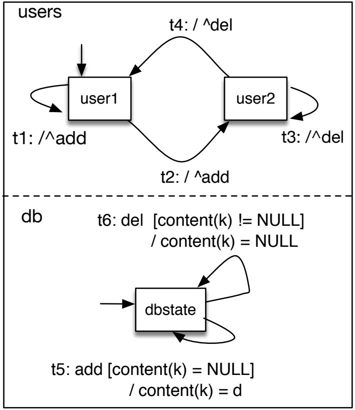
A representation of HSM syntax in SMT-LIB where:

- The state hierarchy is deeply embedded:
  - Semantics can be written separately, which accommodates variable semantics.
  - Axioms/decision procedures can be created for deductive reasoning about the state hierarchy.
- The transition guards and actions are shallowly embedded:
  - Model can use the rich datatypes native in SMT-LIB.

The above is accomplished within a decidable fragment of FOL.

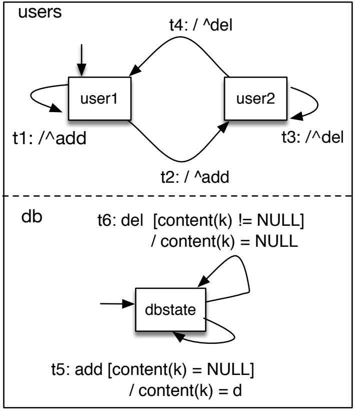
Example: Integrating Control with Rich Datatypes

Database Model



Example: Integrating Control with Rich Datatypes

Database Model




```
; DB represents the
; state of the database.
(declare-sort DB 0)

; Data represents the
; possible data that can be
; stored in the database
(declare-sort Data 0)

; Key represents the
; possible keys
(declare-sort Key 0)

; uninterpreted function
; represents the contents
; of the database
; content: DB x Key -> Data
(declare-fun content
   (DB Key) Data)
```

Related Work

On creating standard representations of the syntax of HSMs:

- OpenModel Modeling Language (OMML) - Hall and Zisman, 2004
- Composed Hierarchical State Machines (XML) - Niu, Atlee, and Day, 2005
- fUML, Alf - OMG, 2013

We differ from these approaches because our representation is an embedding within an existing logic and it allows a variable semantics for the control state hierarchy.

Contributions

A representation of HSM syntax in SMT-LIB where:

- The state hierarchy is deeply embedded:
  - Variable semantics can be written separately.
  - Axioms/decision procedures can be created for deductive reasoning about the state hierarchy.
- The transition labels are shallowly embedded:
  - Model can use the rich datatypes native in SMT-LIB.

The above was accomplished within a decidable fragment of FOL.

Future Work:

- Translators from a user-friendly form of models to this representation.
- Analysis: Write axioms for the semantics!

Goal: Model checking abstract behavioural models that include both data and control abstractions using automated FOL solvers!