Formal analysis of object models with temporal constraints

Stefano Tonetta

Embedded System Unit, Fondazione Bruno Kessler

ETCS Workshop
Trento October 10, 2008
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Methodology recap

Informal Analysis  Formalization  Formal Validation
Methodology recap

Informal Analysis  Formalization  Formal Validation

Well Formedness Check  Domain Narrowing  Formal Analysis
Well formedness check

- Check syntax.
- Friendly error messages.
- CNL editor:
  - Syntax highlighting.
  - Completion.
- Check for completeness of a selection.
  - Example: if a constraint referring to train is selected, also the class train must be selected.
Property-based approach

- The requirement is the object of the analysis.
- Everything starts from the specification.
- Formalization enhances property-based analysis:
  - Every piece of specification has a formal independent counterpart.
  - If we focus on a subset of the specification, we consider the corresponding piece of formalism.
  - If we get a result on a subset of the formalism, we propagate the result to the corresponding piece of specification.
Narrowing

- For narrowing, we mean the choice on a subset of the specification.
- After formalization.
- Back to specification: the validation starts from the specification, not from the model.
- Stress on tool/methodology for specification validation, not for model analysis.
Different approaches

The property-based formalization allows for different approaches:

- **hierarchical approach:**
  - focus on high-level requirements;

- **modular approach:**
  - focus on specific module requirements;

- **incremental approach:**
  - add new requirements to previous validated set;

- **what-if analysis:**
  - add new requirements to previous validated set;
Problem creation

- Definition of the objects into play.
- Addition of assumptions in terms of constraints.
Formal analysis

- **Domain Independent Checks:**
  - Does the specification contain contradictions?
  
    1. Logical Consistency;

- **Domain Dependent Checks:**
  - Does the specification capture original intents?
  
    2. Scenario Compatibility:
      - Is the specification too strong?

    3. Property Checking:
      - Is the specification too weak?

- **Quality of the results:**
  - Traces;
  - Unsat Cores.
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
     - Scenario checking
     - Property checking
     - Feedback
     - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Consistency

- Domain independent.
  - No necessary input.
- Freedom from contradictions.
- Formally, does there exist a model that satisfies all requirements?
  \[ \Rightarrow \] Satisfiability of conjunction of requirements.
- Example of three inconsistent requirements:
  1. always (light=red or light=green)
  2. always (light=red implies (in the future light=green))
  3. always (light=green implies (in the future light=yellow))
Consistency as model intersection

- Specification: set of constraints.

- Inconsistent specification.
1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Scenario checking

- Domain dependent:
  - necessary to know the expected behaviors.
- The expected behavior can be specified with a constraint or a sequence diagram.
- Formally, does there exist a model that satisfies all requirements and satisfies also the scenario?
  \[ \Rightarrow \text{Satisfiability of conjunction of requirements and scenario.} \]
- Two possible specification language for scenario:
  1. CNL
  2. Sequence diagrams
- Can be seen as a generalized test case.
Sequence diagrams are translated to CNL regular expressions.
In CNL you can express more:

“SCENARIO: never train.moves”

---

Translation

**Traces**  \[ TRUE[*]; e_1; \]  \[ TRUE[*]; e_2; \]  \[ \ldots \]  \[ ; TRUE[*]; e_n; TRUE[*] \]

**Concatenation**  \[ \{ f_1; f_2 \} \]

**Alternate**  \[ \{ \{ b \text{ & } f_1 \} | \{ !b \text{ & } f_2 \} \} \]

**Conditional**  \[ \{ \{ !b \text{ | } f_1 \} \} \]

**Parallel**  \[ \{ f_1 \&\& f_2 \} \]

**Repetition**  \[ \{ f_1[*] \} \]

\[ \{ f_1[*k] \} \]

\[ \{ f_1[*i..j] \} \]
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Property checking

- Domain dependent:
  - necessary to know the guaranteed behaviors.
- The guaranteed behavior can be specified with a constraint.
- Formally, if a model satisfies all requirements, does the model satisfy also the property?
- Dually, does there exist a model that satisfies all requirements but does not satisfy the property?
- Finally, does there exist a model that satisfies all requirements and satisfies also the negation of the property?
  \[\Rightarrow\] Satisfiability of conjunction of requirements and negated property.
Summary type of behaviors

- **Specification:**
  - defines a set of required behaviors $R$;
  - defined by:
    - class diagram
    - constraints (CNL keywords: BEHAVIOR, INVAR, INIT)
    - state machines

- **Scenario:**
  - defines a set of possible behaviors $P$;
  - defined by:
    - constraints (CNL keywords: SCENARIO)
    - sequence diagrams
  - check: $R \cap P = \emptyset$?

- **Property:**
  - defines a set of asserted behaviors $A$;
  - defined by:
    - constraints (CNL keywords: PROPERTY)
  - check: $R \subseteq A$?
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Traces

- A trace is generated in case of
  - consistent specification
  - compatible scenario
  - violated property
- peculiar feature of model checking techniques
- if trace is not satisfactory, further constraining is possible.
Unsat core

- subset of elements that caused the unsatisfiability
- it can represent
  - a subset that is inconsistent
  - a subset that is incompatible with the scenario
  - a subset that proves the property
if a problem has been found, look for causes;

different possibilities:
  - there is an evident flaw in the specification
    - **ACTION**: fix the specification
  - there is an evident flaw in the formalization
    - **ACTION**: fix the formalization
  - the formalization did not capture the real meaning of the specification
    - **ACTION**: disambiguate the specification and fix the formalization
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
If no problem has been found, continue with new tests until good confidence.

**Validation Loop**

1. select subset of requirements
2. define objects
3. define assumptions on objects
4. define scenarios and properties
5. check consistency
6. check scenario
7. check property
8. go to 1 or 2 or 3
Tool support

Create new problem

Model Checking

Requirements are consistent

Title: Start_Of_Mission_Problem

Trace 0

Type: counter-example
Description: BMC Counterexample
Loops: 1, at state(s): 2.
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Nothing but satisfiability
Nothing but satisfiability
Undecidability

Definition

A problem is undecidable if there cannot be an algorithm that solves it.

Sources of undecidability:

1. quantified formulas over objects and their relationships,
   - as undecidable as full theory of uninterpreted functions;
2. temporal logic with infinite domain constraints,
   - known to have very restricted decidable fragments.
A problem is undecidable if there cannot be an algorithm that solves it.

Sources of undecidability:

1. quantified formulas over objects and their relationships,
   - as undecidable as full theory of uninterpreted functions;
2. temporal logic with infinite domain constraints,
   - known to have very restricted decidable fragments.

Though, we cannot give up:
Undecidability

Definition
A problem is undecidable if there cannot be an algorithm that solves it.

Sources of undecidability:
1. quantified formulas over objects and their relationships,
   - as undecidable as full theory of uninterpreted functions;
2. temporal logic with infinite domain constraints,
   - known to have very restricted decidable fragments.

Though, we cannot give up:
1. The problem can be simplified.
2. Approximated algorithms can be used.
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Transition systems

- Transition system are a formal model for (infinite-)state machines.
- Described symbolically with variables and formulas to describe their evolution.

**Example**

```plaintext
VAR
  x: integer;
  b: boolean;
TRANS
  b -> next(x)=x+1
```
Objects

- We fix a set of objects for each class.
- The tool looks for a sequence of models over the chosen set of objects.
- If we find a model, then
  - the original problem is consistent,
  - the scenario is compatible,
  - the property is not satisfied.
- If we don't find a model, then
  - the original problem is consistent with this set of objects,
  - the scenario is compatible with this set of objects,
  - the property is satisfied by this set of objects.
Since the sets of objects are finite, we can remove quantifiers:

\[ \forall v : \tau. (\varphi) = \bigwedge_{o \in O_\tau} (\varphi[o/v]) \]

\[ \forall v \in e.a. (\varphi) = \bigwedge_{1 \leq i \leq e.a.max} (e.a.size \geq i \rightarrow \varphi[e.a[i]/v]) \]

\[ t \in e.a = \bigvee_{1 \leq i \leq e.a.max} (e.a.size \geq i \land t = e.a[i]) \]
Hybrid model

- Introduced real variable $\delta$ to represent the elapsed time.
  - discrete step: $\delta = 0$;
  - continuous step: $\delta > 0$.
- Derivatives substituted with linear definition:
  - $der(v) = (next(v) - v)/\delta$
Asynchronous model

- Introduced an input variable that selects at every step the active element.
- Frame conditions define when a variable can or cannot change.

**Example**

```
VAR
  x: real;
  selected: {time, env};
TRANS
  selected=env -> next(x)=x
```
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
     - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
The satisfiability of the temporal constraints is performed with an automata-theoretic approach.

Every formula can be automatically translated into a state machine with an equivalent language.

Translation is based on the so-called “tableau rule”:

$$\varphi_1 U \varphi_2 = \varphi_2 \lor (\varphi_1 \land X(\varphi_1 U \varphi_2))$$

The tableau rule allows to define the formula in term of current and next values.

“Fairness” conditions are necessary to guarantee that eventually $\varphi_2$ will be satisfied.

Non-determinism is necessary because it is not possible to predict the future.
Tableau construction: example
Propositional case

- In the propositional case, the translation is exponential.
- Every state represent a set of sub-formulas.
- One fairness condition for each $U$-formula.
- Advance techniques to translate the formulas.
Formally, a fairness condition is a set of states that must be visited infinitely often.

As for finite domain, a satisfying sequence must have a lasso shape counterexample.

Search involves nested reachability.

NuSMV known to have the best available compilation-search algorithm for satisfiability.
First-order case

The compilation in the more general case of temporal logic with infinite domain constraints follows three steps:

1. for every transition expression $\psi$ in $\varphi$, we introduce a new Boolean variable $a_\psi$;
2. we consider the Boolean abstraction $\varphi^A$ of $\varphi$ where every transition expression $\psi$ has been substituted with the corresponding Boolean constant $a_\psi$;
3. we translate $\varphi^A$ into $S_{\varphi^A}$;
4. we add $\bigwedge_{\psi \in \varphi} (a_\psi \leftrightarrow \psi)$ to the transition condition of $S_{\varphi^A}$. 
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Sound vs. complete

- **Sound**: if the algorithm returns a result, the result is correct.
- **Complete**: the algorithm returns always a result.
- Our approaches are sound but not complete.
- Note that the reachability problem for a transition system is undecidable.
Over- vs. under-approximations

- **Over approximation:**
  - it looks at a super-set of behaviors;
  - if no model is found, then it concludes that the original system has no model;
  - if a model is found, it must check that it is a model of the original behavior;
  - if not, the over-approximation is refined by reducing the super-set.

- **Under approximation:**
  - it looks at a sub-set of behaviors;
  - if a model is found, it is a model of the original system;
  - if no model is found, it must check that there cannot be a model of the original behavior;
  - if yes, the under-approximation is refined by increasing the sub-set.
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Concrete bounded model checking

- Case of under-approximation.
- It looks for behavior of length up to a given bound.
- Refined by increasing the bound.
- Very efficient to find models.
- Ideal for consistency and scenario checking.
- Approximated check for property.
Outline

1. Formal analysis of requirements
   - Methodology
   - Consistency checking
   - Scenario checking
   - Property checking
   - Feedback
   - Validation loop

2. Formal analysis of object models with temporal constraints
   - Undecidability
   - Translation into transition systems
   - Temporal compilation

3. Formal analysis of transition systems
   - Concrete bounded model checking
   - Abstract unbounded model checking
Abstract unbounded model checking

- Case of over-approximation.
- It considers a set of predicates that track important facts of the system.
- It builds an abstract system based on the predicates.
- Refined by adding new predicates.
- Efficient to prove properties.
Counter-Example Guided Abstraction Refinement:

\[ M \]

\[ \hat{M} \]

\[ \hat{M} \not\models \varphi \]

\[ \hat{M} \models \varphi \]

\[ M \models \varphi \]

\[ \gamma(\pi) = \emptyset \]

\[ \gamma(\pi) \neq \emptyset \]

\[ M \not\models \varphi \]
Both techniques use SMT solver as underlying subroutine.

An SMT solver solves the satisfiability of a fragment of FOL.

In case of satisfiability, it gives back an assignment to the variables.

The assignment represent a trace in the model.

The tool parses the trace and translate it in terms of the UML model.

The fragment handled by the solver defines some limitations on the expressions that can be used in the constraints.

Currently, for example, formulas over integers and reals must be linear.
Summary and discussion

- Very active research field.
- Tool can benefit from all improvement to verification engine.
- New functionalities in the tool:
  - guided search.
  - automatic creation of main.
- Different use:
  - generation of generalized test case
  - coverage of generated traces:
  - which requirements have been solicited?
  - different from typical case: coverage of statements, conditions, states, transitions, ...
Thank you!