Expressing Design-Time Uncertainty in Software Models

Naama Ben-David

In Collaboration with Marsha Chechik, Alessio Di Sandro and Michalis Famelis
Expressing Design-Time Uncertainty in Software Models
• Large Project
• Lots of Documents
• Lots of Requirements
• Hard to get started

Make sense of this!
Abstraction

- Communicate effectively
- Understand the project better
Automation

- Mathematical structure allows verifying correctness
- Generate code
- Refine and make adjustments
Abstract concrete connect four game to a UML class diagram
Example

• Generate code from diagram
• Communicate and plan using diagram
Property Checking using Models

Generating code is great, but useless if that code isn’t correct.

Make sure that *every time a player makes a move, a new piece appears on the board!*

for all player: if makesMove(player) and isTurn(player) then hasNewPiece(Board)
Expressing Design-Time Uncertainty in Software Models
Which one should I use?
Partial Model
Partial Model

**May** element
The element may or may not exist

Concretizations of the partial model
Partial Model Advantages

• Can continue adding new features, while keeping track of uncertainties.
Partial Model Advantages

- Can continue adding new features, while keeping track of uncertainties
- Can make adjustments when new information is known (called *refinement*)

We haven’t decided yet, but we know this:
There *cannot be green elements* in the model!

Property

assert (not (exists green))
Partial Model Problems

Too many options!

... and more!
Partial Model Problems

Too many options!

Number of concretizations exponential in number of May elements!

... and more!
Solution: Limit with a propositional formula

\[
\begin{align*}
&+(L_1 \land L_2 \land C \land \neg L_3 \land \neg L_4 \land \neg T) \lor \\
&+(\neg L_1 \land \neg L_2 \land \neg C \land L_3 \land L_4 \land T) \lor \\
&+(\neg L_1 \land \neg L_2 \land \neg C \land \neg L_3 \land \neg L_4 \land \neg T)
\end{align*}
\]

Add element IDs in order to easily reason about them
Partial Model Problems

Too many options!

\[(L_1 \land L_2 \land C \land \neg L_3 \land \neg L_4 \land \neg T) \lor
\neg L_1 \land \neg L_2 \land \neg C \land L_3 \land L_4 \land T \lor
\neg L_1 \land \neg L_2 \land \neg C \land \neg L_3 \land \neg L_4 \land \neg T\]
Expressing Design-Time Uncertainty in Software Models
MU-MMINT
May Uncertainty Model Management INTeractive
Creating a Usable Tool for dealing with Uncertainty

- Check properties on Partial Models
- Do refinements
- Express Uncertainty and constraints effectively
  - Decision Points
Checking Properties

When checking properties on partial models, there are three possible truth values:

- **True** - if property true for ALL concretizations
- **False** - if property false for ALL concretizations
- **Maybe** - if property true for some, false for others

assert (not (exists green))

= MAYBE
Checking Properties

When checking properties on partial models, there are three possible truth values:

- **True** - if property true for ALL concretizations
- **False** - if property false for ALL concretizations
- **Maybe** - if property true for some, false for others

assert (not (exists green))
Doing Refinements

After getting a **Maybe** value:

- Grey out the part that does not satisfy the property
- Remove if necessary, reducing uncertainty in the model
- Eventually, by doing refinements, one final concretization can be reached

```
“assert (not (exists green))”
```
**Doing Refinements**

After getting a *Maybe* value:

- Grey out the part that does not satisfy the property
- Remove if necessary, reducing uncertainty in the model
- Eventually, by doing refinements, one final concretization can be reached

```
“assert (not (exists green))”
```
Propositional formulas can get confusing very quickly.

Note that there are **groups** of elements that have to appear together, and these groups are **mutually exclusive**.
Decision Points

Define **Decisions** and **Alternatives**
**Decision Points**

Define *Decisions* and *Alternatives*

Decision D1 has 3 alternatives:

- **A1** = \{L1, L2, C\}
- **A2** = \{L3, L4, T\}
- **A3** = \{

Now, old formula

\[(L_1 \land L_2 \land C \land \neg L_3 \land \neg L_4 \land \neg T) \lor

(\neg L_1 \land \neg L_2 \land \neg C \land L_3 \land L_4 \land T) \lor

(\neg L_1 \land \neg L_2 \land \neg C \land \neg L_3 \land \neg L_4 \land \neg T)\]

can be reduced to  \[A1 \oplus A2 \oplus A3\]
**Decision Points**

In larger examples, there can be multiple Decisions that can be independent from each other, or there may be a dependency.

D1 has 3 alternatives:
- \(A_1 = \{L_1, L_2, C\}\)
- \(A_2 = \{L_3, L_4, T\}\)
- \(A_3 = \{\}\\)

D2 has 2 alternatives:
- \(A_1 = \{L_5, R\}\)
- \(A_2 = \{\}\)
**Summary**

- **Abstraction**: Communicate effectively, understand the project better.

- **Automation**:
  - Mathematical structure allows verifying correctness
  - Generate code
  - Refine and make adjustments

---

**Checking Properties**

When checking properties on partial models, there are three possible truth values:

- **True** - if ALL concretizations satisfy the property
- **False** - if NONE of the concretizations satisfy the property
- **Maybe** - if some do and some don’t

---

**Decision Points**

Define **Decisions** and **Alternatives**

Decision D1 has 3 alternatives:

\[ A1 = \{L1, L2, C\} \]
\[ A2 = \{L3, L4, T\} \]
\[ A3 = \{\} \]

Now, old formula can be reduced to \[ A1 \oplus A2 \oplus A3 \]
Questions?