Reasoning with Uncertainty and Inconsistency

Prof. Steve Easterbrook

Dept of Computer Science
University of Toronto

sme@cs.toronto.edu

Outline of Talk

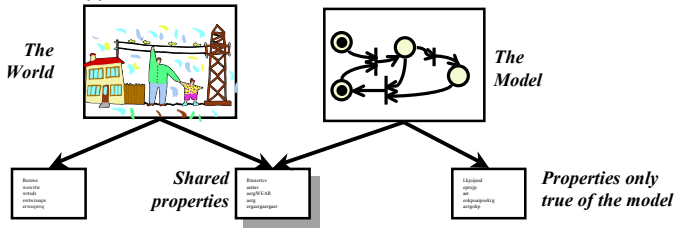
- → Modeling Requirements
  - Current modeling languages are too constraining
  - We'd like to be able to express our uncertainty, our disagreements, etc.
- → Models that express inconsistency and incompleteness
  - Use of multi-valued logics
  - truth orders and knowledge orders
  - MV model checking
- → Potential Applications
  - Abstraction
  - Disagreement & negotiation support
  - Reasoning about relative priority / criticality
  - Query Checking

Modeling…

- Modeling can guide elicitation:
  - Does the modeling process help you figure out what questions to ask?
  - Does the modeling process help to surface hidden requirements?
    - i.e. does it help you ask the right questions?
- Modeling can provide a measure of progress:
  - Does completeness of the model imply completeness of the elicitation?
    - i.e. if we've filled in all the pieces of the model, are we done?
- Modeling can help to uncover problems
  - Does inconsistency in the model reveal interesting things…?
    - e.g. inconsistency could correspond to conflicting or infeasible requirements
    - e.g. inconsistency could mean confusion over terminology, scope, etc
    - e.g. inconsistency could reveal disagreements between stakeholders
- Modeling can help us check our understanding
  - Can we test that the model has the properties we expect?
  - Can we reason over the model to understand its consequences?
  - Can we animate the model to help us visualize/validate the requirements?

But formal notations are inflexible

- Formal modeling languages demand:
  - preciseness
  - completeness
  - consistency
- but all models are approximations:
  - phenomena in the model that are not present in the application domain
  - phenomena in the application domain that are not in the model



  - …and we want to express our uncertainty, disagreement, priorities, etc.

Incompleteness and Inconsistency

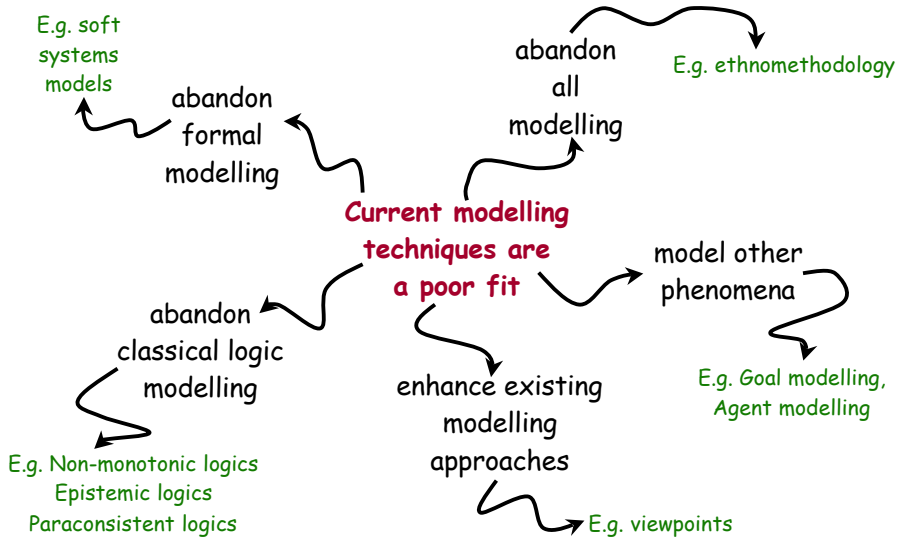
→ Most models are incomplete and/or inconsistent most of the time

→ Sources of incompleteness
- aspects of the model undeveloped
- properties are not fully understood
- complete model has been abstracted

→ Sources of inconsistency
- multiple stakeholders
  - affects both models and properties

→ Goal:
- want to reason about software systems at all stages of software development
- … and to do so automatically (e.g. tools such as model-checking)

How can we make progress?



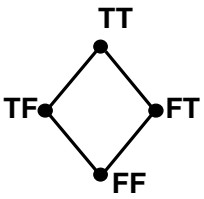
Example Multi-Valued Logics

→ 3-valued logic (Kleene logic)

- Middle value represents "maybe".
- Useful for:
  - Partial models - some behaviours unknown.
  - Open systems - some behaviours will be defined by other components or features.
  - Abstracted systems - some behaviours are elided to reduce the state space.

→ 4-valued logic

- Results from combining two sources of information.
  - TF and FT represent disagreement between the sources.
- Useful for:
  - Feature interaction
  - Viewpoint integration



Formally … Quasi-Boolean logics

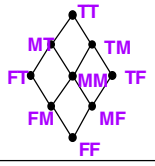
→ Truth values form a (finite) distributive lattice

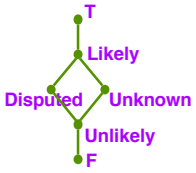
- True at the top, False at the bottom
- conjunction is lattice meet (greatest lower bound)
- disjunction is lattice join (least upper bound)
- negation is chosen to preserve involution, i.e. ¬¬A = A
- implication is material, i.e. A→ B ≡ ¬A ∨ B

→ Properties:

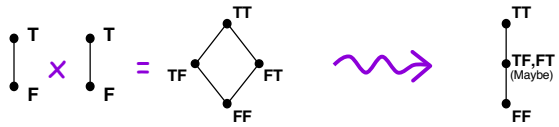
- associativity, idempotency, distributivity, and De Morgan's laws
- But not (necessarily):
  - Law of excluded middle: A ∨ ¬A = T
  - Law of non-contradiction: A ∧ ¬A = ⊥

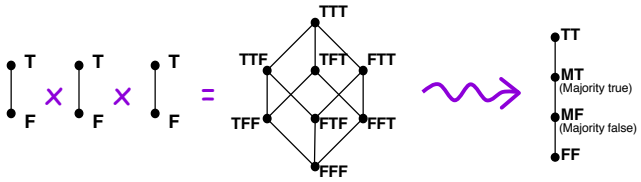
→ More examples:

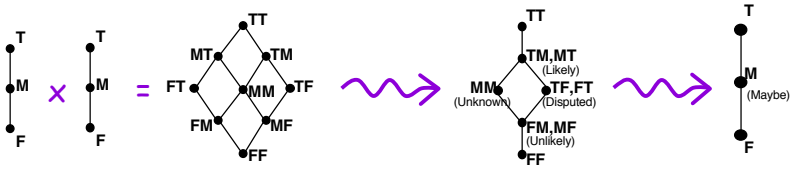




Other variants are possible

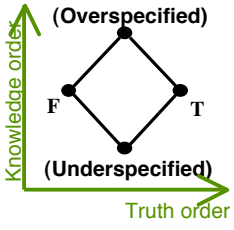


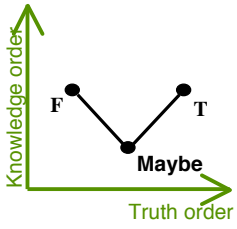


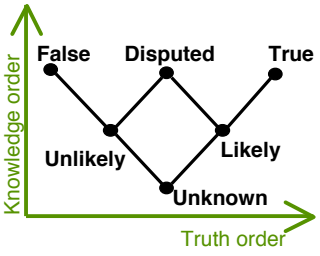


Truth Order vs. Knowledge Order

- Can distinguish two partial orders on truth values:
  - a truth order - gives the logics we use in the model checker
    - Must be a quasi-boolean lattice to give us a coherent logic
  - an information order - for reasoning about refinement of models
    - Doesn't need to be a lattice
- Related to bilattices
  - as developed by Belnap, Ginsberg, Fitting



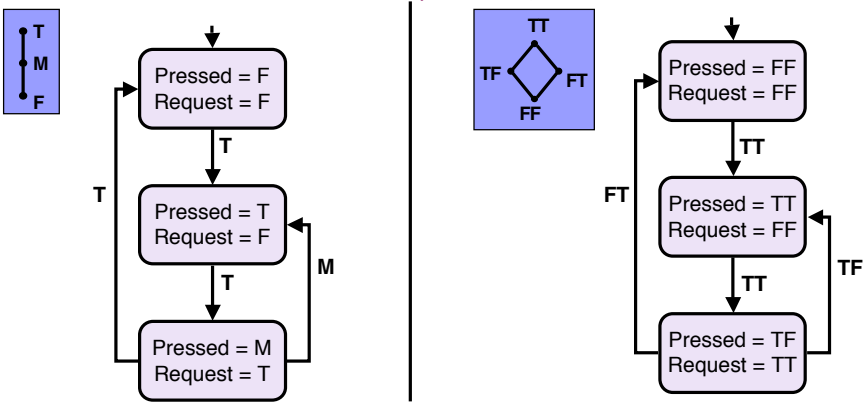




Multi-Valued Models

→ Generalize conventional state machines:

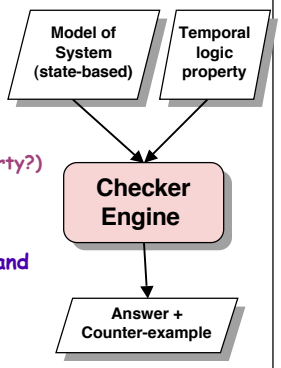
- variables take any value from the logic
- transitions between states take any value from the logic
  - False transitions are not shown (by convention)



What is MV model checking?

→ Classical Model Checking…

- Inputs:
  - a state machine model, M
  - a correctness property, P (in a suitable temporal logic)
- Calculates:
  - the value of M ⊨ p (does the model satisfy the property?)



→ Multi-Valued Model Checking…

- generalization to more truth values than just "true" and "false".
  - the meanings of these depend on the domain
- Inputs:
  - A state machine model, M (variables and transitions may be multi-valued)
  - A correctness property, P (in a multi-valued temporal logic)
- Calculates:
  - does the model satisfy the property?
  - …but now the value of M ⊨ p might be multi-valued

Correctness properties: □CTL

→ CTL (Computation Tree Logic)

- propositional temporal logic
- branching-time logic, allowing explicit quantification over possible futures
  - every atomic proposition is a CTL formula
  - T and F are CTL formulae
  - if p and q are CTL formulae, then so are: □p, p□q, p q,
  - EX p - p is true in some next states; AX p …in all next states
  - EF p - along some path, p is true in some future state; AF p along all paths…
  - E[p U q] - along some path, p holds until q holds; A[p U q] along all paths…
  - EG p - along some path, p holds in every state; AG p along all paths…

→ □CTL - multi-valued extension of CTL

- Each truth value in the logic is a CTL formula
- replace existential quantification by disjunction, universal quantification by conjunction, so

$$[EX\,\Box](s) = \bigsqcup_{t \sqsubseteq S} (R(s,t) \sqcap [\Box](t))$$

Implementation

→ Our model-checker is named □Chek

- Implemented in Java
- Symbolic model checker
  - uses decision diagrams to represent and manipulate the state space and transition relation

→ Input:

- Multi-valued model
  - expressed in XML, using a GXL derived language
  - or built directly using Java
- A specification of the multi-valued logic
  - expressed in XML

→ Visualising the output:

- Counter-example/witness generator (KEG)
- Interactive front-end for exploration and visualization of counter-examples and witnesses (KegVis)

Abstraction Example

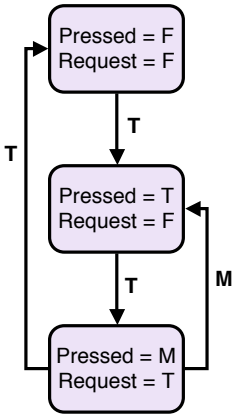
→ A simple elevator controller

- Logic: 3-valued
- Number of floors: 4
- A door that can be opened and closed
- Buttons inside the elevator and on floor landings
  - Pressed - button's physical state
  - Request - outstanding request

→ Abstraction

- We don't care about the state of 'Pressed' when there is a request pending for any of the buttons
- Two alternative implementations:
  - each button press just sends signal to controller
  - button remains latched once pressed

Model for each button:



Relative Priority

→ Uses total orders:

- Intermediate values of the logic represent relative criticality.

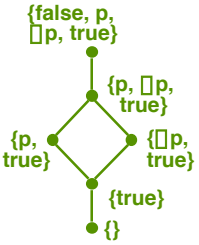
→ Basic example:

- 5-valued model can represent a set of layers
- each layer specifies values for properties left unspecified at previous layers.
- Analysis:
  - If a property is 'Must' in the model, it is true in the core layer
  - i.e. it doesn't matter what other layers do.
- We can reason about which properties are preserved if functionality at lower layers is lost.

Query-Checking

→ For discovering properties

- A query is a CTL formula containing unknowns, indicated by '?'
  - AG(send ⇒ AF ?)
  - "What condition always follows a 'send' event?"
- A solution is a propositional formulae that, when substituted for '?' results in a valid CTL formula
  - e.g. ? = receive ∨ ack ? = receive ∨
  - Often we just want the strongest such formula.
- Can restrict the placeholder to variables of interest:
  - AG(send ⇒ AF ?{receive})



→ As a multi-valued model checking problem:

- Uses upset lattices

Uses of Query Checking

→ Finding invariants

E.g. What is invariant for the whole model:
AG ?

→ General model exploration

- provide partial explanation when property holds
  - e.g. instead of AG (a ∨ b), ask AG ?{a, b}
  - answer a ⇒ b is stronger!
- provide diagnostic information when property fails
  - e.g. if AG (req ⇒ AF ack) fails - ask AG (req ⇒ AF ?)

→ Other

- Test case generation
- Guided simulation

Conclusions & Future Work

→ Motivation

- most of our models are incomplete or inconsistent most of the time
- It would be nice to reason about the incompleteness/inconsistency

→ We're investigating multi-valued logics

- Quasi-boolean logics - truth values form a lattice
- We've built a multi-valued model checker
- We're investigating applications:
  - abstraction, negotiation, layered systems, query checking

→ Future work:

- Apply these ideas to existing (real) modeling languages
- Framework for composing multi-valued models
- Framework for reasoning about refinement, using the knowledge order
- We're looking for more case studies
- We're looking for other possible applications