
CS/SE 3RA3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada
Most important graphical notations

- Conceptual structures: **entity-relationship diagrams**
- Information flows: **context diagrams** and **dataflow diagrams**
- System behaviors: **state machine diagrams**
- System operations: **use case diagrams**
- Modeling scenarios and workflows: **activity diagrams**
- You will have to use all of them in Assignment 2, some of them for the midterm and all of them for the exam.
What are Formal Methods?

- **Formal** = Mathematical
- **Methods** = Structured Approaches, Strategies
- Using mathematics in a structured way to analyze and describe a problem.
What kind of Formal Methods?

- Standard Discrete Mathematics and Predicate Logic
- Temporal Logic
- Tabular Expressions
Useful Mathematics

- Set theory
- Functions and Relations
- First-order predicate logic
- *Before-After predicates*, this is the only thing you might not know!
Set theory

“All humans are male or female.”

Humans = Male ∪ Female

“Nobody is male and female at the same time.”

Male ∩ Female = ∅
Functions and Relations

“Every customer must have a personal attendant.”

attendant : Customers → Employees

“Every customer has a set of accounts.”

AccountsOf: Customers → P(Accounts)
First-order Predicate Logic

“Everybody who works on a Sunday needs to have a special permit.”

\[ \forall p \in \text{Employee}: \text{workOnSunday}(p) \implies \text{hasPermit}(p) \]

“Every customer must at least have one account.”

\[ \forall c \in \text{Customers}: \exists a \in \text{Accounts}: a \in \text{AccountsOf}(c) \]
“People can enter the building if they have their ID with them. When entering, they have to leave their ID card at the registration desk.”

\[\text{hasAuthorization} : \text{PEOPLE} \rightarrow \{\text{true, false}\}\]
\[\text{carriesPassport} : \text{PEOPLE} \rightarrow \{\text{true, false}\}\]
\[\text{passportOf} : \text{PEOPLE} \rightarrow \text{PASSPORTS}\]
\[\text{inBuilding} : \text{PEOPLE} \rightarrow \{\text{true, false}\}\]
\[\text{peopleInBuilding} : \mathcal{P} (\text{PEOPLE}) \text{ (or set}(\text{PEOPLE}))\]
\[\text{passportsAtDesk} : \mathcal{P} (\text{PASSPORTS}) \text{ (or set}(\text{PASSPORTS}))\]

\[\text{hasAuthorization}(p) \land \text{carriesPassport}(p) \land \neg \text{inBuilding}(p) \Rightarrow \]
\[\text{peopleInBuilding}' = \text{peopleInBuilding} \cup \{p\} \land \]
\[\text{passportsAtDesk}' = \text{passportsAtDesk} \cup \{\text{passportOf}(p)\} \land \]
\[\text{inBuilding}(p) \land \neg \text{carriesPassport}(p)\]

\[\neg (\text{hasAuthorization}(p) \land \text{carriesPassport}(p) \land \neg \text{inBuilding}(p)) \Rightarrow \]
\[\text{peopleInBuilding}' = \text{peopleInBuilding} \land \]
\[\text{passportsAtDesk}' = \text{passportsAtDesk}\]
The advantages of using math for any analytical problem

- **Short** notation
- Forces you to be **precise**
- Identifies **ambiguity**
- Clean form of **communication**
- Makes you **ask the right questions**
Compare:

“For every ticket that is issued, there has to be a single person that is allowed to enter. This person is called the owner of the ticket.”

with

*TicketOwner* : *IssuedTickets* $\rightarrow$ *Person*
“On red traffic lights, people normally stop their cars.”

What does “normally” mean? How should we build a system based on this statement? What are the consequences? What happens in the exceptional case?

- Formalization Fails! We cannot formalize until the above questions are answered!
"When the temperature is too high, the ventilation has to be switched on or the maintenance staff has to be informed."

May we do both? *(standard logical “or”)*

\[ \text{TemperatureIsHigh} \implies (\text{NotifyStaff} \lor \text{VentilationOn}) \]

or *(exclusive “or”, i.e. “xor”, often denoted as \(\oplus\) or \(\vee\))*

\[ \text{TemperatureIsHigh} \implies (\text{NotifyStaff} \lor \text{VentilationOn}) \land \neg(\text{NotifyStaff} \land \text{VentilationOn}) \]

or equivalently

\[ \text{TemperatureIsHigh} \implies (\text{NotifyStaff} \bigtriangleup \text{VentilationOn}) \]
Asking the Right Questions

“Every customer has is either trusted or untrusted.”

\[ \forall c \in \text{CUSTOMER} : \text{trusted}(c) \lor \text{untrusted}(c) \]

“Upon internet purchase, a person is automatically registered as a new customer.”

\[ \text{InternetPurchase} : \text{CUSTOMER} \times \text{CMS} \rightarrow \text{CMS}, \text{ where } \text{CMS} = \mathcal{P}(\text{CUSTOMER}) \text{ and } \mathcal{P} \text{ denotes the powerset.} \]

\[ \text{InternetPurchase}(c) : \text{customers}' = \text{customers} \cup \{c\} \]

- Is the new customer trusted or untrusted ?!
Clean Form of Communication

- Every mathematical notation has a precise semantic definition.
- New constructs can be added defined in terms of old constructs.
- Math does not need language skills and can be easily understood in an international context.
Tiny Example Problem: Temperature control

“The software should control the temperature of the room. It can read the current temperature from a thermometer. Should the temperature fall below a lower limit, then the heater should be switched on to raise the temperature. Should it rise above an upper limit, then the cooling system should be switched on to lower the temperature.”

[...]

“Safety concern: the heater and the cooler should never be switched on at the same time.”
Formal Specification: Temperature control

\( \text{currentTemperature} : \text{INTEGER} \quad \leftarrow \text{do you remember the concept of type in discrete math?} \)

\( \text{lowerLimit} : \text{INTEGER} \)
\( \text{upperLimit} : \text{INTEGER} \)
\( \text{coolingSystem} : \{ \text{on}, \text{off} \} \)
\( \text{heatingSystem} : \{ \text{on}, \text{off} \} \)

\[(\text{coolingSystem} = \text{on}) \implies (\text{heatingSystem} = \text{off})\]
\[(\text{heatingSystem} = \text{on}) \implies (\text{coolingSystem} = \text{off})\]

\[(\text{coolingSystem} = \text{off} \land \text{currentTemperature} > \text{upperLimit}) \implies \text{coolingSystem} = \text{on}\]
\[(\text{currentTemperature} \leq \text{upperLimit}) \implies \text{coolingSystem} = \text{off}\]
Train movement and door problem: from the textbook

measuredSpeed : Reals (or Integers? what precision?)
doorState : {closed, open, closing, opening, almost_closed, ...}
TrainSpeed : Reals (or Integers? what precision)
DoorClosed : {true, false}
TrainMoving : {true, false}, and much more...

measuredSpeed \neq 0 \iff doorState = closed
measuredSpeed = 0 \iff doorState \in \{open, closing, opening\}, ...
measuredSpeed \neq 0 \iff trainSpeed \neq 0,
|measuredSpeed - trainSpeed| < 2km/h
DoorClosed \iff doorState = ‘closed’
TrainMoving \iff trainSpeed \neq 0, and much more...

From the above we can derive that for example

TrainMoving \implies DoorClosed

and many more...
Elevator: Some Functional Requirements

- **Requirement:**
  The doors must not open when the elevator is moving.

- **Formal Specification:**
  - Domain and Notation.
    
    \[
    \begin{align*}
    ELEVATOR & : \text{set of all elevators in the building} \\
    elevatorMoving & : \{\text{true, false}\} \\
    doorState & : ELEVATOR \to \{\text{closed, open}\} \\
    e & \in ELEVATOR
    \end{align*}
    \]

  - Statement.
    
    \[
    \begin{align*}
    elevatorMoving(e) = \text{true} & \Rightarrow \text{doorState}(e) = \text{closed} \\
    \text{doorState}(e) = \text{open} & \Rightarrow elevatorMoving(e) = \text{false}
    \end{align*}
    \]
**Requirement:**

Pressing a floor number button on the elevator will set the elevator to move in that direction to the indicated floor after the doors close.

**Notes:**

- The hold direction does not need to be included because it is encompassed by the floorsPressed set being empty.
- There are two predicates from this requirement.
  1. The first is if a button is pressed, and that floor is above where they currently are, doors close, and the elevator begins to move up.
  2. The second is parallel to it, but if the floor is below the current floor, then the elevator goes down.
- The requirement doesn’t state that the elevator had to be moving in that direction originally (if the elevator is going up, in practice, it should only service floors pressed that are above the current floor), so those conditions were not added.
- Instead, whatever is pressed, it will go to, regardless of previous direction.
Formal Specification:

Domain and Notation.

$$FLOOR : \text{set of all floors served by elevators}$$
$$ELEVATOR : \text{set of all elevators in the building}$$
$$\text{elevatorDirection} : ELEVATOR \rightarrow \{ \text{up, down, hold} \}$$
$$\text{floorsPressed} : \text{set(MAX_FLOORS)}$$
$$\text{isPressed} : FLOOR \rightarrow \{ \text{true, false} \}$$
$$\text{curFloor} : \rightarrow FLOOR$$

$$f \in FLOOR;$$
$$e \in ELEVATOR$$

Statement.

$$(\text{isPressed}(f) \land (f > \text{curFloor})) \Rightarrow$$
$$(\text{floorsPressed}' = \text{floorsPressed} \cup \{ f \} \land \text{doorState}(e) = \text{closed} \land \text{elevatorMoving} = \text{true} \land \text{elevatorDirection} = \text{up})$$

$$(\text{isPressed}(f) \land (f < \text{curFloor})) \Rightarrow$$
$$(\text{floorsPressed}' = \text{floorsPressed} \cup \{ f \} \land \text{doorState}(e) = \text{closed} \land \text{elevatorMoving} = \text{true} \land \text{elevatorDirection} = \text{down})$$
In Requirements Engineering, Predicate Calculus is more often used for describing requirements than to prove the desired properties.

Proving properties in formal way is usually done for safety critical systems or safety critical parts of systems.
Elevator movement and door problem: An example of a proof

- **Requirement:** "For safety reasons, in any passenger lift, if the lift is moving, not responding to a request, or out of service, then the doors must be closed."

- **Mathematical Domains:**

  - measuredSpeed : \( \mathbb{R} \) (Real numbers)
  - doorState : \{closed, open\}
  - elevatorSpeed : \( \mathbb{R} \) (Real numbers)
  - doorClosed : \{true, false\}
  - elevatorMoving : \{true, false\}
  - elevatorDirection : \{up, down, holding\}
  - elevatorState : \{active, outService\}
  - emergButton : \{pressed, notPressed\}
  - hasRequest : \{true, false\}
Elevator movement and door problem: SFOREQ, ASM, DOM, SYSREQ

**SOFREQ:**
\[
\text{measuredSpeed} \not= 0 \Rightarrow \text{doorState} = \text{closed} \lor \\
\text{elevatorDirection} = \text{holding} \Rightarrow \text{doorState} = \text{closed} \lor \\
\text{elevatorState} = \text{outService} \Rightarrow \text{doorState} = \text{closed}
\]

**ASM:**
\[
\text{measuredSpeed} \not= 0 \Leftrightarrow \text{elevatorSpeed} \not= 0 \lor \\
\text{doorClosed} \Leftrightarrow \text{doorState} = \text{closed} \lor \\
\text{emergButton} = \text{pressed} \Rightarrow \text{elevatorState} = \text{outService} \lor \\
\text{hasRequest} = \text{false} \Leftrightarrow \text{elevatorDirection} = \text{holding}
\]

**DOM:**
\[
\text{elevatorMoving} = \text{true} \Leftrightarrow \text{elevatorSpeed} \not= 0 \lor \\
\text{measuredSpeed} \not= 0 \Leftrightarrow \text{elevatorDirection} \in \{\text{up, down}\} \lor \\
\text{elevatorDirection} = \text{holding} \Rightarrow \text{elevatorMoving} = \text{false} \lor \\
\]

**SYSREQ:**
\[
(\text{elevatorMoving} \lor \text{emergButton} = \text{pressed} \lor \text{hasRequest} = \text{false}) \Rightarrow \text{doorClosed}
\]
Proof of \((\text{SOFREQ, ASM, DOM} \models \text{SYSREQ})\)

- Looking at all three individual components of the LHS, we will show all imply doorClosed.
- It will be derived through the axioms listed in SOFREQ, ASM, and DOM, which, when the premises are presumed to be true, will guarantee validity.
- Because the theorem can be proven deductively, it is complete.
- The validity of the solution indicates that when the elevator is moving, or the emergency button has been pressed, or the elevator does not have a request, it will have its doors closed.
- The completeness of the solution deductively shows that these premises must actually be true (and not assumed to be true).
Proof of \((SOFREQ, ASM, DOM \models SYSREQ)\)

(a) \(\text{elevatorMoving} \Rightarrow \text{doorClosed}\)

\[
\text{elevatorMoving} \\
\Leftrightarrow \langle \text{DOM} : \text{elevatorMoving} = \text{true} \Leftrightarrow \text{elevatorSpeed} \neq 0 \rangle \\
\text{elevatorSpeed} \neq 0 \\
\Leftrightarrow \langle \text{ASM} : \text{measuredSpeed} \neq 0 \Leftrightarrow \text{elevatorSpeed} \neq 0 \rangle \\
\text{measuredSpeed} \neq 0 \\
\Rightarrow \langle \text{SOFREQ} : \text{measuredSpeed} \neq 0 \Rightarrow \text{doorState} = \text{closed} \rangle \\
\text{doorState} = \text{closed} \\
\Leftrightarrow \langle \text{ASM} : \text{doorClosed} \Leftrightarrow \text{doorState} = \text{closed} \rangle \\
\text{doorClosed}
\]
Proof of \((\text{SOFREQ, ASM, DOM} \models \text{SYSREQ})\)

(b) \(\text{emergButton} = \text{pressed} \Rightarrow \text{doorClosed}\)

\[
\begin{align*}
\text{emergButton} = \text{pressed} \\
\Rightarrow \langle \text{ASM} : \text{emergButton} = \text{pressed} \Rightarrow \text{elevatorState} = \text{outService} \rangle \\
\text{elevatorState} = \text{outService} \\
\Rightarrow \langle \text{SOFREQ} : \text{elevatorState} = \text{outService} \Rightarrow \text{doorState} = \text{closed} \rangle \\
\text{doorState} = \text{closed} \\
\Leftrightarrow \langle \text{ASM} : \text{doorClosed} \Leftrightarrow \text{doorState} = \text{closed} \rangle \\
\text{doorClosed}
\end{align*}
\]
(c) \( \text{hasRequest} = \text{false} \Rightarrow \text{doorClosed} \)

\[
\begin{align*}
\text{hasRequest} &= \text{false} \\
\iff & \langle \text{ASM} : \text{hasRequest} = \text{false} \iff \text{elevatorDirection} = \text{holding} \rangle \\
\text{elevatorDirection} &= \text{holding} \\
\Rightarrow & \langle \text{SOFREQ} : \text{elevatorDirection} = \text{holding} \Rightarrow \text{doorState} = \text{closed} \rangle \\
\text{doorState} &= \text{closed} \\
\iff & \langle \text{ASM} : \text{doorClosed} \iff \text{doorState} = \text{closed} \rangle \\
\text{doorClosed} &
\end{align*}
\]
Problem with Predicate Logic

- First Order Predicate Logic is **undecidable**
- In particular $P \implies Q$ is **undecidable**
- Hence $SOFTREQ, ASM, DOM \models SESREQ$ is **undecidable**
- This means that a 'push button' universal automatic theorem prover will never be built!
- Propositional logic (i.e. no $\exists$, $\forall$) is decidable
- But expressive power of propositional logic (i.e. only $\lor, \land, \neg, \implies, \iff$) is rather weak!
- Are there other logics that are decidable, but with stronger expressive power?
After some training, most users do not have much problems with describing system properties with Predicate Calculus, however proving is another story.

Undecidability of First Order Predicate Calculus makes automatic theorem proving problematic and difficult.

In Requirements Engineering it is mainly used to describe desired properties of crucial and critical parts of systems, and to prove crucial parts of safety critical systems.