VL LOGIK: GENERAL INTRODUCTION

WS 2016/2017 (342.208)

Armin Biere, FMV (biere@jku.at)
Martina Seidl, FMV (martina.seidl@jku.at)

Version 2016.1

JOHANNES KEPLER
UNIVERSITY LINZ

Abstractions and Modelling

Definition (Model)

A *model* is a simplified reflection of a natural or artificial entity describing only those aspects of the "real" entity relevant for a specific purpose.

Examples for models:

- geography: map
- architecture: construction plan
- informatics: almost everything (e.g., a software system)

A model is an abstraction hiding irrelevant aspects of a system. This allows to focus on the important things.

Modelling Languages (1/3)

- Purposes of models:
  - construction of new systems
  - analysis of complex systems
- *Natural Language* is
  - universal
  - expressive

  but also
  - complex, ambiguous, fuzzy.

Example

We saw the man with the telescope.

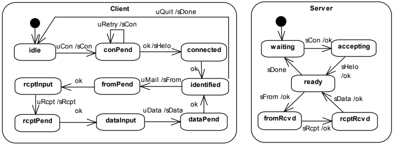
- Did the man have a telescope?
- Did we have a telescope?

- *Modelling languages* have been introduced which are
  - artificially constructed
  - restricted in expressiveness
  - often specific to a domain
  - formally defined with concise semantics

Modelling Languages (2/3)

Examples of modelling languages in computer science:

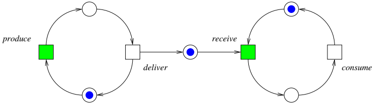
State Machines



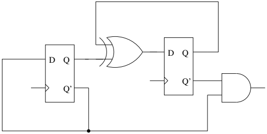
CSP

$$
\begin{aligned}
Road &= car.up.ccross.down.Road \\
Rail &= train.darkgreen.tcross.red.Rail \\
Signal &= darkgreen.red.Signal + \\
&\quad up.down.Signal \\
Crossing &= (Road \mid\mid Rail \mid\mid Signal)
\end{aligned}
$$

Petri Net



Circuit



Modelling Languages (3/3)

Modelling languages are distinguishable (amongst other properties) w.r.t.

- universality and expressiveness
- degree of formalization
- representation (graphical, textual)

Definition (Formal Modelling)

Translation of a (possibly ambiguous) specification to an unambiguous specification in a formal language

Languages of logic provide a very powerful tool for formal modeling.

Defining a Language: Syntax

- what do expressions (words, sentences) of a language look like?
  - sequences of symbols forming words
  - rules for composing sentences (grammar)
    - checked by parser
  - sometimes multiple (equivalent) representations
    - different goals (user-friendliness, processability)

Example

Definition of natural numbers:

- 0 is a natural number.
- For every natural number n, there is a natural number $s(n)$.

Some words: $0, s(0), s(s(0)), \ldots$

Defining a Language: Syntax

- what do expressions mean?
  - meaning of the words
  - meaning of combinations of words (sentences)
  - logic-based languages have a concise semantics

Example

Interpretation as natural numbers:

- 0 is interpreted as zero
- $s(0)$ is interpreted as *one*
- $s(s(0))$ is interpreted as *two*
- ...

Backus-Naur Form (BNF)

- notation technique for describing the syntax of a language
- elements:
  - non-terminal symbols (variables): enclosed in brackets $\langle\rangle$
  - $::=$ indicates the definition of a non-terminal symbol
  - the symbol $|$ means "or"
  - all other symbols stand for themselves (sometimes they are quoted, e.g., "`->`")

Example

Definition of the language of *decimal numbers* in BNF:

$$\begin{aligned} \langle number \rangle &::= \langle integer \rangle \text{ ``.'' } \langle integer \rangle \\ \langle integer \rangle &::= \langle digit \rangle \mid \langle digit \rangle \; \langle integer \rangle \\ \langle digit \rangle &::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 0 \end{aligned}$$

Logic-Based Languages (Logics)

- A *logic* consists of
  - ☐ a set of symbols (like $\vee, \wedge, \neg, \top, \bot, \forall, \exists \ldots$)
  - ☐ a set of variables (like $x, y, z, \ldots$)
  - ☐ concise syntax: well-formedness of expressions
  - ☐ concise semantics: meaning of expressions
- Logics support *reasoning* for
  - ☐ derivation of “new” knowledge
  - ☐ proving the truth/falsity of a statement (satisfiability checking)
- Different logics *differ* in their
  - ☐ truth values: binary (true, false), multi-valued (true, false, unknown), fuzzy (between 0 and 1, e.g., $[0, 1]$ as subset of the real numbers)
  - ☐ expressiveness (what can be formulated in the logic?)
  - ☐ complexity (how expensive is reasoning?)

Example: Party Planning

We want to plan a party.

Unfortunately, the selection of the guests is not straight forward.

We have to consider the following rules.

1. If two people are married, we have to invite them both or none of them. Alice is married to Bob and Cecile is married to David.
2. If we invite Alice then we also have to invite Cecile. Cecile does not care if we invite Alice but not her.
3. David and Eva can't stand each other, so it is not possible to invite both.
4. We want to invite Bob and Fred.

Question: Can we find a guest list?

Syntax of Propositional Logic

In BNF-like form:

$$
\begin{aligned}
\langle \textit{formula} \rangle &::= \top \mid \bot \mid \langle \textit{variable} \rangle \mid \langle \textit{connective\_f} \rangle \\
\langle \textit{connective\_f} \rangle &::= \langle \textit{conn1} \rangle \ \langle \textit{formula} \rangle \mid \langle \textit{formula} \rangle \ \langle \textit{conn2} \rangle \ \langle \textit{formula} \rangle \\
\langle \textit{conn1} \rangle &::= \neg \\
\langle \textit{conn2} \rangle &::= \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow
\end{aligned}
$$

- $\top$ is the truth constant which is always true
- $\bot$ is the truth constant which is always false
- a propositional variable can take the values true and false
- $\neg$ is the negation
- $\wedge$ is the conjunction (logical and)
- $\vee$ is the disjunction (logical or)
- $\rightarrow$ is the implication
- $\leftrightarrow$ is the equivalence

Party Planning with Propositional Logic

- *propositional variables*:
  inviteAlice, inviteBob, inviteCecile, inviteDavid, inviteEva, inviteFred
- *constraints*:
  1. invite married: inviteAlice $\leftrightarrow$ inviteBob, inviteCecile $\leftrightarrow$ inviteDavid
  2. if Alice then Cecile: inviteAlice $\rightarrow$ inviteCecile
  3. either David or Eva: $\neg$ (inviteEva $\leftrightarrow$ inviteDavid)
  4. invite Bob and Fred: inviteBob $\wedge$ inviteFred
- *encoding in propositional logic*:

  (inviteAlice $\leftrightarrow$ inviteBob) $\wedge$ (inviteCecile $\leftrightarrow$ inviteDavid) $\wedge$
  (inviteAlice $\rightarrow$ inviteCecile) $\wedge$ $\neg$ (inviteEva $\leftrightarrow$ inviteDavid) $\wedge$
  inviteBob $\wedge$ inviteFred

Syntax of First-Order Logic: Terms

In BNF-like form:

$\langle term \rangle ::= \langle constant \rangle \mid \langle variable \rangle \mid \langle fun\_sym \rangle$ '(' $\langle term \rangle$ (',' $\langle term \rangle$)$*$ ')'

- function symbols ($\langle fun\_sym \rangle$) have an arity (number of arguments).
- (',' $\langle term \rangle$)$*$ means zero or more repetitions of ", $\langle term \rangle$ ".

Example

- Let s be a function symbol with arity 1 and y a variable. Then $s(y)$ is a term.

Syntax of First-Order Logic: Formulas

In BNF-like form:

$$
\begin{aligned}
\langle \textit{formula} \rangle &::= \top \mid \bot \mid \langle \textit{atomic\_f} \rangle \mid \langle \textit{connective\_f} \rangle \mid \langle \textit{quantifier\_f} \rangle \\
\langle \textit{atomic\_f} \rangle &::= \langle \textit{pred\_sym} \rangle \text{ `(' } \langle \textit{term} \rangle \ \ (\text{`,' } \langle \textit{term} \rangle\ )* \text{ `)'} \\
\langle \textit{connective\_f} \rangle &::= \langle \textit{conn1} \rangle\ \langle \textit{formula} \rangle \mid \langle \textit{formula} \rangle\ \langle \textit{conn2} \rangle\ \langle \textit{formula} \rangle \\
\langle \textit{conn1} \rangle &::= \neg \\
\langle \textit{conn2} \rangle &::= \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow \\
\langle \textit{quantifier\_f} \rangle &::= \langle \textit{quantifier} \rangle\ \langle \textit{variable} \rangle \text{ `:' } \langle \textit{formula} \rangle \\
\langle \textit{quantifier} \rangle &::= \forall \mid \exists
\end{aligned}
$$

- $\forall$ is the *universal quantifier*
  - $\forall x\ :\ p(x)$ is reads as “forall possible values of x, the unary predicate p is true.”
- $\exists$ is the *existential quantifier*
  - $\exists x\ :\ p(x)$ is reads as “there is a value of x such that the unary predicate p is true.”

Party Planning with First-Order Logic

- *objects (constants)*: alice, bob, cecile, david, eva, fred
- *relations (predicates)*: married/2, invited/1
- *background knowledge*: married(alice,bob), married(cecile,david)
- *constraints*:
  1. $\forall X, Y$ (married(X,Y) $\rightarrow$ (invited(X) $\leftrightarrow$ invited(Y))
  2. if Alice then Cecile: invited(alice) $\rightarrow$ invited(cecile)
  3. either David or Eva: $\neg$ (invited(eva) $\leftrightarrow$ invited(david))
  4. invite Bob and Fred: invited(bob) $\wedge$ invited(fred)
- *encoding in first-order logic*:

  $\forall X, Y$ (married(X,Y) $\rightarrow$ (invited(X) $\leftrightarrow$ invited(Y)) $\wedge$
  invited(alice) $\rightarrow$ invited(cecile) $\wedge$
  $\neg$ (invited(eva) $\leftrightarrow$ invited(david)) $\wedge$ invited(bob) $\wedge$
  invited(fred)

Automated Reasoning and Inferences

- Logical languages allow the inference of new knowledge ("reasoning").
- For reasoning, a logic provides various sets of *rules* (calculi).
- Reasoning is often based on certain syntactical patterns.

Example: (modus ponens)

x holds.
If x holds, then also y holds.

---

y holds.

Some Remarks on Inferences (1/2)

- A system is inconsistent, if we can infer that a statement holds and that a statement does not hold at the same time.

Example

Assume we have modelled the following system

- A comes to the party.
- B comes to the party.
- If A comes to the party, then B does not come to the party.

With the *modus ponens*, we can infer that B does not come to the party.

So, we have some inconsistency in our party model.

Some Remarks on Inferences (2/2)

- Sometimes we cannot infer anything.

Example

Assume we have modelled the following system:

- If A comes to the party, then B comes to the party.
- C comes to the party.

Then we cannot infer anything.

Logic in Practice

- *hardware and software industry*:
  - computer-aided verification
  - formal specification
- *programming*: basis for declarative programming language like Prolog
- *artificial intelligence*: automated reasoning (e.g., planning, scheduling)
- *mathematics*: reasoning about systems, mechanical proofs

Logics in this Lectures

In this lecture, we consider different logic-based languages:

- *propositional logic (SAT)*
  - simple language: only atoms and connectives
  - low expressiveness, low complexity
  - very successful in industry (e.g., verification)
- *first-order logic (predicate logic)*
  - rich language: predicates, functions, terms, quantifiers
  - great power of expressiveness, high complexity
  - many applications in mathematics and verification
- *satisfiability modulo theories (SMT)*
  - customizable language: user decides
  - as much expressiveness as required
    as much complexity as necessary
  - very popular and successful in industry