Institute for Formal Models and Verification
Johannes Kepler University Linz

JKU
JOHANNES KEPLER
UNIVERSITY LINZ

VL Logik (LVA-Nr. 342208), Winter Semester 2015/2016

General Introduction

Version 2015.1

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

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.

Example: A map contains information about the streets and about spots of interest, but no details which people live there, which trees grow there, etc.

Modelling Languages (1/3)

- Purposes of models:
  - construction of new systems
  - analysis of complex systems

Question: What is a good language to describe a model?

- *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?

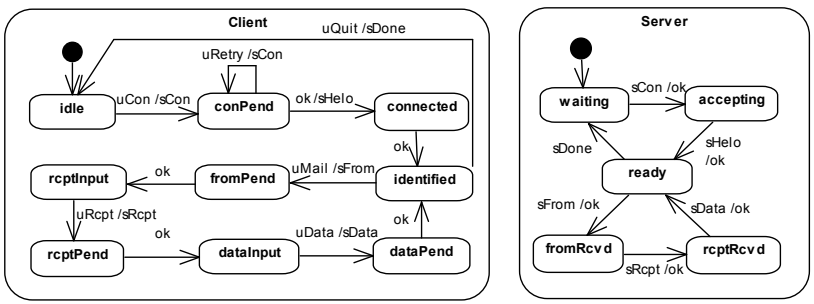
- *Modelling languages* have been introduced which are
  - artificially constructed
  - restricted in expressiveness
  - often specific to a domain
  - formally defined with concise semantics

Modelling Languages (2/3)

Examples of modelling languages in computer science:

- programming languages
- finite automata, regular expression
- languages for software designs (e.g., UML)
- logic-based languages

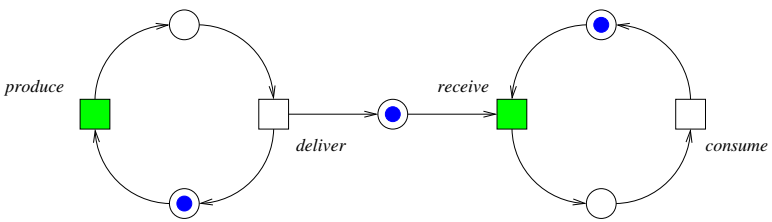
UML State Machines



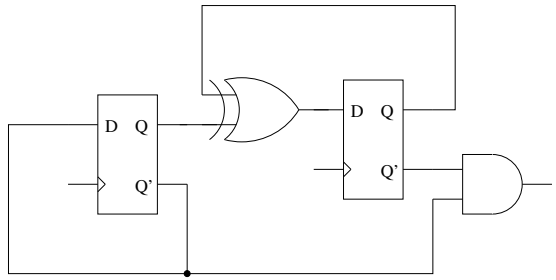
CSP

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

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

A language definition consists of rules defining the

- *syntax of the language*
  how do expressions look?
  - sequences of symbols forming words
  - rules for composing sentences (grammar); checked by parser
  - sometimes multiple (equivalent) representations with 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: $0, s(0), s(s(0)), \ldots$

- *semantics of the language*
  what do expressions mean?
  - meaning of the words
  - meaning of combinations of words

Example

The word $s(0)$ has the meaning 1, the word $s(s(s(0)))$ has the meaning 3.

Backus-Naur Form (BNF)

- notation technique for describing the syntax of a language
- elements:
  - symbols enclosed in brackets $\langle\rangle$ are variables (non-terminal symbols)
  - the symbol ::= 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:

$$\langle \textit{number} \rangle \;::=\; \langle \textit{integer} \rangle \text{ “.” } \langle \textit{integer} \rangle$$
$$\langle \textit{integer} \rangle \;::=\; \langle \textit{digit} \rangle \;\mid\; \langle \textit{digit} \rangle \;\langle \textit{integer} \rangle$$
$$\langle \textit{digit} \rangle \;::=\; 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 0$$

Some words: 0.0, 1.1, 123.546, 01.10000, . . .

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, which is consistent with rules 1-4?

Syntax of Propositional Logic

In BNF-like form:

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

where

- $\top$ 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: $\text{inviteAlice} \leftrightarrow \text{inviteBob}, \text{inviteCecile} \leftrightarrow \text{inviteDavid}$
  2. if Alice then Cecile: $\text{inviteAlice} \rightarrow \text{inviteCecile}$
  3. either David or Eva: $\neg\,(\text{inviteEva} \leftrightarrow \text{inviteDavid})$
  4. invite Bob and Fred: $\text{inviteBob} \wedge \text{inviteFred}$

- *encoding in propositional logic*:

$$(\text{inviteAlice} \leftrightarrow \text{inviteBob}) \wedge (\text{inviteCecile} \leftrightarrow \text{inviteDavid}) \wedge$$
$$(\text{inviteAlice} \rightarrow \text{inviteCecile}) \wedge \neg\,(\text{inviteEva} \leftrightarrow \text{inviteDavid}) \wedge$$
$$\text{inviteBob} \wedge \text{inviteFred}$$

Syntax of First-Order Logic: Terms

In BNF-like form:

$$\langle term \rangle \ ::= \ \langle constant \rangle \ | \ \langle variable \rangle \ | \ \langle fun\_sym \rangle \text{ ‘(’ } \langle term \rangle \ \ (\text{‘,’ } \langle term \rangle \ )* \text{ ‘)’}$$

where

- 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.
- Let ‘remainder’ be a function symbol with arity 2, a and b constants. Then remainder(a, b) and remainder$(a, s(a))$ are terms.
- Let ‘openInterval’ be a function symbol with arity 2, a and b constants. Then openInterval(a, b) is a term.

Syntax of First-Order Logic: Formulas

In BNF-like form:

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

where

- $\forall$ 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.

General pattern:

(modus ponens)

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

---

y holds.

- x and y are arbitrary propositions.
- From true premises, we can derive true conclusions.
- From false premises, we can derive everything.

Example

A comes to the party.
If A comes to the party, then also B comes. — *Premises*

B comes to the party. — *Conclusion*

Some Remarks on Inferences

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.

- A system is inconsistent, if we can infer that a statement holds and that a statement does not hold at the same time.
- 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 atomic propositions, logic connectives
  - low expressiveness
  - low complexity (satisfiability checking is exponential in the worst case)
  - very successful in industry (e.g., verification)
- *first-order logic (predicate logic)*
  - rich language: predicates, functions, terms, quantifiers, logical connectives
  - great power of expressiveness
  - high complexity (satisfiability checking is undecidable in general)
  - many applications in mathematics and system specifications
- *satisfiability modulo theories (SMT)*
  - customizable language: user decides on the included language concepts
  - as much expressiveness as required
  - as much complexity as necessary
  - very popular and successful in industry