First-Order Logic

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2023/2024

Roadmap

- Review of First-Order Logic
- Many-sorted First-Order Logic
- Modeling with First-Order Logic

(Classical) First-Order Logic

Introduction

First-order logic (FOL) is a richer language than propositional logic. Its lexicon contains not only the symbols $\wedge$, $\vee$, $\neg$, and $\rightarrow$ (and parentheses) from propositional logic, but also the symbols $\exists$ and $\forall$ for "there exists" and "for all" that range over a domain (universe) of discourse, along with various symbols to represent variables, constants, functions, and relations.

There are two sorts of things involved in a first-order logic formula:

- *terms*, which denote the objects that we are talking about;
- *formulas*, which denote truth values.

Examples:

"Not all birds can fly."
"Every mother is older than her children."
"John and Peter have the same maternal grandmother."

Syntax

The alphabet of a first-order language is organised into the following categories.

- *Variables:* $x, y, z, \ldots \in \mathcal{X}$ (arbitrary elements of an underlying domain)
- *Constants:* $a, b, c, \ldots \in \mathcal{C}$ (specific elements of an underlying domain)
- *Functions:* $f, g, h, \ldots \in \mathcal{F}$ (every function f has a fixed arity, $\mathsf{ar}(f)$)
- *Predicates:* $P, Q, R, \ldots \in \mathcal{P}$ (every predicate P has a fixed arity, $\mathsf{ar}(P)$)
- *Logical connectives:* $\top$, $\bot$, $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\forall$ (*for all*), $\exists$ (*there exists*)
- *Auxiliary symbols*: ".", "(" and ")".

We assume that all these sets are disjoint. $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{P}$ are the non-logical symbols of the language. These three sets constitute the *vocabulary* $\mathcal{V} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$.
(Alternatively, one could see constants as 0-ary functions.)

Syntax

Terms

The set of *terms* of a first-order language over a vocabulary $\mathcal{V}$ is given by the following abstract syntax

$$\mathbf{Term}_{\mathcal{V}} \ni t \ ::= \ x \mid c \mid f(t_1, \ldots, t_{\mathsf{ar}(f)})$$

Formulas

The set $\mathbf{Form}_{\mathcal{V}}$, of *formulas* of FOL, is given by the abstract syntax

$$\mathbf{Form}_{\mathcal{V}} \ni \phi, \psi \ ::= \ P(t_1, \ldots, t_{\mathsf{ar}(P)}) \mid \bot \mid \top \mid (\neg\phi) \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \rightarrow \psi) \mid (\forall x.\, \phi) \mid (\exists x.\, \phi)$$

Syntax

Convention

We adopt some syntactical conventions to lighten the presentation of formulas:

- Outermost parenthesis are usually dropped.
- In absence of parentheses, we adopt the following convention about precedence. Ranging from the highest precedence to the lowest, we have respectively: $\neg$, $\wedge$, $\vee$ and $\rightarrow$. Finally we have that $\rightarrow$ binds more tightly than $\forall$ and $\exists$.
- All binary connectives are right-associative.
- Nested quantifications such as $\forall x. \forall y. \phi$ are abbreviated to $\forall x, y.\, \phi$.
- $\forall \overline{x}. \phi$ denotes the nested quantification $\forall x_1, \ldots, x_n.\, \phi$.

Modeling with FOL

"Not all birds can fly."

We can code this sentence assuming the two unary predicates B and F expressing

$B(x)$ – x is a bird
$F(x)$ – x can fly

The declarative sentence "Not all birds can fly" can now be coded as

$$\neg(\forall x.\, B(x) \rightarrow F(x))$$

or, alternatively, as

$$\exists x.\, B(x) \wedge \neg F(x)$$

Modeling with FOL

"Every mother is older than her children."
"John and Peter have the same maternal grandmother."

Using constants symbols J and P for John and Peter, and predicates $=$, $mother$ and $older$ expressing that

$x = y$ – x and y are equal (the same)
$mother(x,y)$ – x is mother of y
$older(x,y)$ – x is older than y

these sentences could be expressed by

$$\forall x. \forall y.\ mother(x,y) \rightarrow older(x,y)$$

$$\forall x, y, u, v.\ mother(x,y) \wedge mother(y,J) \wedge mother(u,v) \wedge mother(v,P) \rightarrow x = u$$

Modeling with FOL

"Every mother is older than her children."
"John and Peter have the same maternal grandmother."

A different and more elegant encoding is to represent y'mother in a more direct way, by using a function instead of a relation. We write $m(y)$ to mean y'mother. This way the two sentences above have simpler encondings.

$$\forall x.\ older(m(x), x)$$

$$m(m(J)) = m(m(P))$$

Modeling with FOL

Assume further the following predicates and constant symbols

$flower(x)$ – x is a flower
$sport(x)$ – x is a sport
A – Anne
$likes(x,y)$ – x likes y
$brother(x,y)$ – x is brother of y

- "Anne likes John's brother." $\exists x.\ brother(x,J) \wedge likes(A,x)$
- "John likes all sports." $\forall x.\ sport(x) \rightarrow likes(J,x)$
- "John's mother likes flowers." $\forall x.\ flower(x) \rightarrow likes(m(J),x)$
- "John's mother does not like some sports." $\exists y.\ sport(y) \wedge \neg likes(m(J),y)$
- "Peter only likes sports." $\forall x.\ likes(P,x) \rightarrow sports(x)$
- "Anne has two children."

$$\exists x_1, x_2.\ mother(A,x_1) \wedge mother(A,x_2) \wedge x_1 \neq x_2 \wedge \forall z.\ mother(A,z) \rightarrow z = x_1 \vee z = x_2$$

Free and bound variables

- The *free variables* of a formula ϕ are those variables occurring in ϕ that are not quantified. FV(ϕ) denotes the set of free variables occurring in ϕ.
- The *bound variables* of a formula ϕ are those variables occurring in ϕ that do have quantifiers. BV(ϕ) denote the set of bound variables occurring in ϕ.

Note that variables can have both free and bound occurrences within the same formula. Let ϕ be $\exists x.\ R(x,y) \wedge \forall y.\ P(y,x)$, then

$$\text{FV}(\phi) = \{y\} \quad \text{and} \quad \text{BV}(\phi) = \{x, y\}.$$

- A formula ϕ is *closed* (or *a sentence*) if it does not contain any free variables.
- If FV(ϕ) $= \{x_1, \ldots, x_n\}$, then
  - its *universal closure* is $\forall x_1. \ldots \forall x_n. \phi$
  - its *existential closure* is $\exists x_1. \ldots \exists x_n. \phi$

Substitution

Substitution

- We define $u[t/x]$ to be the term obtained by replacing each occurrence of variable x in u with t.
- We define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable x in ϕ with t.

Care must be taken, because substitutions can give rise to undesired effects!

Substitution

Let us illustrate the problem...

Let ϕ be $\exists x.\, \mathit{likes}(x,y) \wedge \forall y.\, \mathit{older}(y,x)$, and t be the term $m(x)$.

- Now let us perform the substitution $\phi[t/y]$

  $(\exists x.\, \mathit{likes}(x,y) \wedge \forall y.\, \mathit{older}(y,x))[(m(x)/y] \;=\; \exists x.\, \mathit{likes}(x,m(x)) \wedge \forall y.\, \mathit{older}(y,x)$

  which is **wrong!**

  - The meaning of the formula has completly changed.
  - The free variable x in $m(x)$ was captured inadvertently by the $\exists x$.

- This can be fixed if we change the name of the bounded varible in $\exists x$, by renaming it to a fresh variable.

  $(\exists z.\, \mathit{likes}(z,y) \wedge \forall y.\, \mathit{older}(y,z))[(m(x)/y] \;=\; \exists z.\, \mathit{likes}(z,m(x)) \wedge \forall y.\, \mathit{older}(y,z)$

  which is **OK!**

Substitution

Given a term t, a variable x and a formula ϕ, we say that *t is free for x in ϕ* if no free x in ϕ occurs in the scope of $\forall z$ or $\exists z$ for any variable z occurring in t.

From now on we will assume that all substitutions satisfy this condition. That is when performing the $\phi[t/x]$ we are always assuming that t is free for x in ϕ.

Semantics

$\mathcal{V}$-structure

Let $\mathcal{V}$ be a vocabulary. A *$\mathcal{V}$-structure* $\mathcal{M}$ is a pair $\mathcal{M} = (D, I)$ where D is a nonempty set called the *interpretation domain*, and I is an *interpretation function* that assigns constants, functions and predicates over D to the symbols of $\mathcal{V}$ as follows:

- for each constant symbol $c \in \mathcal{C}$, the interpretation of c is a constant $I(c) \in D$;
- for each $f \in \mathcal{F}$, the interpretation of f is a function $I(f) : D^{\mathsf{ar}(f)} \to D$;
- for each $P \in \mathcal{P}$, the interpretation of P is a function $I(P) : D^{\mathsf{ar}(P)} \to \{0, 1\}$. In particular, 0-ary predicate symbols are interpreted as truth values.

$\mathcal{V}$-structures are also called *models* for $\mathcal{V}$.

Semantics

Assignment

An *assignment* for a domain D is a function $\alpha : \mathcal{X} \to D$.

We denote by $\alpha[x \mapsto a]$ the assignment which maps x to a and any other variable y to $\alpha(y)$.

Given a $\mathcal{V}$-structure $\mathcal{M} = (D, I)$ and given an assignment $\alpha : \mathcal{X} \to D$, we define an *interpretation function for terms*, $\alpha_\mathcal{M} : \textbf{Term}_\mathcal{V} \to D$, as follows:

$$
\begin{array}{lcl}
\alpha_\mathcal{M}(x) & = & \alpha(x) \\
\alpha_\mathcal{M}(c) & = & I(c) \\
\alpha_\mathcal{M}(f(t_1, \ldots, t_n)) & = & I(f)(\alpha_\mathcal{M}(t_1), \ldots, \alpha_\mathcal{M}(t_n))
\end{array}
$$

Semantics

Satisfaction relation

Given a $\mathcal{V}$-structure $\mathcal{M} = (D, I)$ and given an assignment $\alpha : \mathcal{X} \to D$, we define the *satisfaction relation* $\mathcal{M}, \alpha \models \phi$ for each $\phi \in \textbf{Form}_\mathcal{V}$ as follows:

$$
\begin{array}{lll}
\mathcal{M}, \alpha \models \top & & \\
\mathcal{M}, \alpha \not\models \bot & & \\
\mathcal{M}, \alpha \models P(t_1, \ldots, t_n) & \text{iff} & I(P)(\alpha_\mathcal{M}(t_1), \ldots, \alpha_\mathcal{M}(t_n)) = 1 \\
\mathcal{M}, \alpha \models \neg\phi & \text{iff} & \mathcal{M}, \alpha \not\models \phi \\
\mathcal{M}, \alpha \models \phi \wedge \psi & \text{iff} & \mathcal{M}, \alpha \models \phi \text{ and } \mathcal{M}, \alpha \models \psi \\
\mathcal{M}, \alpha \models \phi \vee \psi & \text{iff} & \mathcal{M}, \alpha \models \phi \text{ or } \mathcal{M}, \alpha \models \psi \\
\mathcal{M}, \alpha \models \phi \to \psi & \text{iff} & \mathcal{M}, \alpha \not\models \phi \text{ or } \mathcal{M}, \alpha \models \psi \\
\mathcal{M}, \alpha \models \forall x.\, \phi & \text{iff} & \mathcal{M}, \alpha[x \mapsto a] \models \phi \text{ for all } a \in D \\
\mathcal{M}, \alpha \models \exists x.\, \phi & \text{iff} & \mathcal{M}, \alpha[x \mapsto a] \models \phi \text{ for some } a \in D
\end{array}
$$

Validity and satisfiability

When $\mathcal{M}, \alpha \models \phi$, we say that $\mathcal{M}$ *satisfies* ϕ *with* α.

We write $\mathcal{M} \models \phi$ iff $\mathcal{M}, \alpha \models \phi$ holds for every assignment α.

A formula ϕ is

- *valid* iff $\mathcal{M}, \alpha \models \phi$ holds for all structure $\mathcal{M}$ and assignments α. A valid formula is called a *tautology*. We write $\models \phi$.
- *satisfiable* iff there is some structure $\mathcal{M}$ and some assigment α such that $\mathcal{M}, \alpha \models \phi$ holds.
- *unsatisfiable* iff it is not satisfiable. An unsatisfiable formula is called a *contradiction*.
- *refutable* iff it is not valid.

Consequence and equivalence

Given a set of formulas Γ, a model $\mathcal{M}$ and an assignment α, $\mathcal{M}$ is said to *satisfy* Γ *with* α, denoted by $\mathcal{M}, \alpha \models \Gamma$, if $\mathcal{M}, \alpha \models \phi$ for every $\phi \in \Gamma$.

Γ *entails* ϕ (or that ϕ is a *logical consequence* of Γ), denoted by $\Gamma \models \phi$, iff for all structures $\mathcal{M}$ and assignments α, whenever $\mathcal{M}, \alpha \models \Gamma$ holds, then $\mathcal{M}, \alpha \models \phi$ holds as well.

ϕ is *logically equivalent* to ψ, denoted by $\phi \equiv \psi$, iff $\{\phi\} \models \psi$ and $\{\psi\} \models \phi$.

Deduction theorem

$\Gamma, \phi \models \psi$ iff $\Gamma \models \phi \to \psi$

Consistency

The set Γ is *consistent* or *satisfiable* iff there is a model $\mathcal{M}$ and an assigment α such that $\mathcal{M}, \alpha \models \phi$ holds for all $\phi \in \Gamma$.

We say that Γ is *inconsistent* iff it is not consistent and denote this by $\Gamma \models \bot$.

Proposition

- $\{\phi, \neg\phi\} \models \bot$
- If $\Gamma \models \bot$ and $\Gamma \subseteq \Gamma'$, then $\Gamma' \models \bot$.
- $\Gamma \models \phi$ iff $\Gamma, \neg\phi \models \bot$

Substitution

- Formula ψ is a *subformula* of formula ϕ if it occurs syntactically within ϕ.
- Formula ψ is a *strict subformula* of ϕ if ψ is a subformula of ϕ and $\psi \neq \phi$

Substitution theorem

Suppose $\phi \equiv \psi$. Let θ be a formula that contains ϕ as a subformula. Let θ' be the formula obtained by safe replacing (i.e., avoiding the capture of free variables of ϕ) some occurrence of ϕ in θ with ψ. Then $\theta \equiv \theta'$.

Some equivalences

Renaming of bound variables

If y is free for x in ϕ and $y \notin \mathsf{FV}(\phi)$, then the following equivalences hold.

- $\forall x.\phi \equiv \forall y.\phi[y/x]$
- $\exists x.\phi \equiv \exists y.\phi[y/x]$

The following equivalences hold in first-order logic.

$$\forall x.\phi \wedge \psi \equiv (\forall x.\phi) \wedge (\forall x.\psi) \qquad \exists x.\phi \vee \psi \equiv (\exists x.\phi) \vee (\exists x.\psi)$$
$$\forall x.\phi \equiv (\forall x.\phi) \wedge \phi[t/x] \qquad \exists x.\phi \equiv (\exists x.\phi) \vee \phi[t/x]$$
$$\neg\forall x.\phi \equiv \exists x.\neg\phi \qquad \neg\exists x.\phi \equiv \forall x.\neg\phi$$

Some equivalences

If x does not occur free in ψ, then the following equivalences hold.

$$(\forall x.\phi) \wedge \psi \equiv \forall x.\phi \wedge \psi \qquad \psi \wedge (\forall x.\phi) \equiv \forall x.\psi \wedge \phi$$
$$(\forall x.\phi) \vee \psi \equiv \forall x.\phi \vee \psi \qquad \psi \vee (\forall x.\phi) \equiv \forall x.\psi \vee \phi$$
$$(\exists x.\phi) \wedge \psi \equiv \exists x.\phi \wedge \psi \qquad \psi \wedge (\exists x.\phi) \equiv \exists x.\psi \wedge \phi$$
$$(\exists x.\phi) \vee \psi \equiv \exists x.\phi \vee \psi \qquad \psi \vee (\exists x.\phi) \equiv \exists x.\psi \vee \phi$$

The applicability of these equivalences can always be assured by appropriate renaming of bound variables.

Decidability

Given formulas ϕ and ψ as input, we may ask:

Decision problems

Validity problem:	"Is ϕ valid ?"
Satisfiability problem:	"Is ϕ satisfiable ?"
Consequence problem:	"Is ψ a consequence of ϕ ?"
Equivalence problem:	"Are ϕ and ψ equivalent ?"

These are, in some sense, variations of the same problem.

ϕ is valid	iff	$\neg\phi$ is unsatisfiable
$\phi \models \psi$	iff	$\neg(\phi \to \psi)$ is unsatisfiable
$\phi \equiv \psi$	iff	$\phi \models \psi$ and $\psi \models \phi$
ϕ is satisfiable	iff	$\neg\phi$ is not valid

Decidability

A *solution* to a decision problem is a program that takes problem instances as input and always terminates, producing a correct "yes" or "no" output.

- A decision problem is *decidable* if it has a solution.
- A decision problem is *undecidable* if it is not decidable.

Theorem (Church & Turing)

- The decision problem of validity in first-order logic is undecidable: no program exists which, given any ϕ, decides whether $\models \phi$.
- The decision problem of satisfiability in first-order logic is undecidable: no program exists which, given any ϕ, decides whether ϕ is satisfiable.

Semi-decidability

However, there is a procedure that halts and says "yes" if ϕ is valid.

A decision problem is *semi-decidable* if exists a procedure that, given an input,

- halts and answers "yes" iff "yes" is the correct answer,
- halts and answers "no" if "no" is the correct answer, or
- does not halt if "no" is the correct answer

Unlike a decidable problem, the procedure is only guaranteed to halt if the correct answer is "yes".

The decision problem of validity in first-order logic is semi-decidable.

Modeling with FOL

Use the predicates

$\text{admires}(x, y)$:	x admires y	$\text{Professor}(x)$:	x is a professor
$\text{attended}(x, y)$:	x attended y	$\text{Student}(x)$:	x is a student
		$\text{Lecture}(x)$:	x is a lecture

and the constant Mary to translate the following into FOL:

- *Mary admires every professor.*

  $$\forall x.\, \text{Professor}(x) \to \text{admires}(\text{Mary}, x)$$

- *Some professor admires Mary.*

  $$\exists x.\, \text{Professor}(x) \land \text{admires}(x, \text{Mary})$$

- *Mary admires herself.*

  $$\text{admires}(\text{Mary}, \text{Mary})$$

Modeling with FOL

Use the predicates

admires(x, y) :	x admires y	Professor(x) :	x is a professor
attended(x, y) :	x attended y	Student(x) :	x is a student
		Lecture(x) :	x is a lecture

and the constant Mary to translate the following into FOL:

- *No student attended every lecture.*

  $$\neg(\exists x.\, \text{Student}(x) \land (\forall y.\, \text{Lecture}(y) \to \text{attended}(x, y)))$$

- *No lecture was attended by every student.*

  $$\neg(\exists x.\, \text{Lecture}(x) \land (\forall y.\, \text{Student}(y) \to \text{attended}(y, x)))$$

- *No lecture was attended by no student.*

  $$\neg(\exists l.\, \text{Lecture}(l) \land \forall s.\, \text{Student}(s) \to \neg\text{attended}(s, l))$$

  or equivalently

  $$\forall l.\, \text{Lecture}(l) \to \exists s.\, \text{Student}(s) \land \text{attended}(s, l)$$

FOL with equality

There are different conventions for dealing with equality in first-order logic.

- We have followed the approach of considering equality predicate $(=)$ as a non-logical symbol, treated in the same way as any other predicate. We are working with what are usually known as *"first-order languages without equality"*.
- An alternative approach, usually called *"first-order logic with equality"*, considers equality as a logical symbol with a fixed interpretation.

  In this approach the equality symbol $(=)$ is interpreted as the equality relation in the domain of interpretation. So we have, for a structure $\mathcal{M} = (D, I)$ and an assignment $\alpha : \mathcal{X} \to D$, that

  $$\mathcal{M}, \alpha \models t_1 = t_2 \quad \text{iff} \quad \alpha_{\mathcal{M}}(t_1) \text{ and } \alpha_{\mathcal{M}}(t_2) \text{ are the same element of } D$$

FOL with equality

To understand the significant difference between having equality with the status of any other predicate, or with a fixed interpretation as in first-order logic with equality, consider the formulas

- $\exists x_1, x_2 . \forall y.\ y = x_1 \lor y = x_2$

  With a fixed interpretation of equality, the validity of this formula implies that the cardinality of the interpretation domain is at most two – the quantifiers can actually be used to fix the cardinality of the domain, which is not otherwise possible in first-order logic.

- $\exists x_1, x_2 . \neg(x_1 = x_2)$

  The validity of this formula implies that there exist at least two distinct elements in the domain, thus its cardinality must be at least two.

Modeling in FOL with equality

- To formalize a problem in FOL, first of all, we have to identify
  - the domain of discourse (What are the entities we are dealing with?)
  - properties of the domain of discourse
  - basic relationships between entities (How entities relate to each other?)
  - properties of the relations
- Based on this analysis
  - We start by establish the logical language that we are going to use (i.e., the vocabulary of the language).
  - We write the set of logical formulas that describe the problem.

Modeling in FOL with equality

Graph coloring

Can one assign one of K colors to each of the vertices of graph $G = (V, E)$ such that adjacent vertices are assigned different colors?

- The domain of discourse has vertices and colors, which are different sorts of entities.

  So, we need a unary predicate for each entity (these predicates will act as the "type" of the entity, in a domain that is untyped):

  Vertex(-)   Color(-)

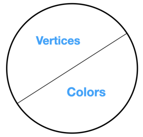
- There are edges between vertices, and each vertex is painted with a color. So, we need two binary predicates

  edge(-,-)   painted(-,-)

Modeling in FOL with equality

Predicates:

Vertex(-)   Color(-)
edge(-,-)   painted(-,-)



- Partitioning the universe of discourse with the types of the entities.

  $\forall x.\ \text{Vertex}(x) \leftrightarrow \neg\text{Color}(x)$

  Alternative: any element of the universe
  - at most is of one of these "types" $\forall x.\ \text{Vertex}(x) \rightarrow \neg\text{Color}(x)$
  - must be of one of these "types" $\forall x.\ \text{Vertex}(x) \vee \text{Color}(x)$

- Typing relations

  $\forall x, y.\ \text{edge}(x, y) \rightarrow \text{Vertex}(x) \wedge \text{Vertex}(y)$

  $\forall x, y.\ \text{painted}(x, y) \rightarrow \text{Vertex}(x) \wedge \text{Color}(y)$

Modeling in FOL with equality

- Each vertex is assigned exactly one color.
  - At least one color to each vertex:

    $\forall x.\ \text{Vertex}(x) \rightarrow \exists y.\ \text{painted}(x, y)$
  - At most one color to each vertex:

    $\forall x, y, z.\ \text{painted}(x, y) \wedge \text{painted}(x, z) \rightarrow y = z$

- Adjacent vertices must have different colors:

  $\forall x, y, a, b.\ \text{edge}(x, y) \wedge \text{painted}(x, a) \wedge \text{painted}(y, b) \rightarrow a \neq b$

Many-sorted FOL

- A natural variant of first-order logic that can be considered is the one that results from allowing different domains of elements to coexist in the framework. This allows distinct "sorts" or types of objects to be distinguished at the syntactical level, constraining how operations and predicates interact with these different sorts.

- Having full support for different sorts of objects in the language allows for cleaner and more natural encodings of whatever we are interested in modeling and reasoning about.

- By adding to the formalism of FOL the notion of sort, we obtain a flexible and convenient logic called *many-sorted first-order logic*, which has the same properties as FOL.

Many-sorted FOL

- A many-sorted vocabulary (signature) is composed of a set of sorts, a set of function symbols, and a set of predicate symbols.
  - Each function symbol f has associated with a type of the form $S_1 \times \ldots \times S_{\mathsf{ar}(f)} \to S$ where $S_1, \ldots, S_{\mathsf{ar}(f)}, S$ are sorts.
  - Each predicate symbol P has associated with it a type of the form $S_1 \times \ldots \times S_{\mathsf{ar}(P)}$.
  - Each variable is associated with a sort.
- The formation of terms and formulas is done only accordingly to the typing policy, i.e., respecting the "sorts".
- The domain of discourse of any structure of a many-sorted vocabulary is fragmented into different subsets, one for every sort.
- The notions of assignment and structure for a many-sorted vocabulary, and the interpretation of terms and formulas are defined in the expected way.

Modeling in many-sorted FOL with equality

Graph coloring

Can one assign one of K colors to each of the vertices of graph $G = (V, E)$ such that adjacent vertices are assigned different colors?

In a many-sorted FOL, to codify this problem we need

- two types: Vertex and Color
- two predicates:

  edge : Vertex × Vertex
  painted : Vertex × Color
- At least one color to each vertex:

  $\forall x : \mathsf{Vertex}. \exists y : \mathsf{Color}.\ \mathsf{painted}(x, y)$ (we can drop the types)
- At most one color to each vertex:

  $\forall x, y, z.\ \mathsf{painted}(x, y) \wedge \mathsf{painted}(x, z) \to y = z$
- Adjacent vertices must have different colors:

  $\forall x, y, a, b.\ \mathsf{edge}(x, y) \wedge \mathsf{painted}(x, a) \wedge \mathsf{painted}(y, b) \to a \neq b$

SMT solvers

- When judging the validity/satisfiability of first-order formulas we are typically interested in a particular domain of discourse, which in addition to a specific underlying vocabulary includes also properties that one expects to hold. We work with respect to some *background theory*.
- The *Satisfiability Modulo Theories (SMT) problem* is a variation of the SAT problem for first-order logic, with the interpretation of symbols constrained by (a combination of) specific theories.
- SMT solvers are tools that aim to answer the SMT problem.
  - The underlying logic of SMT solvers is many-sorted first-order logic with equality.
  - SMT solvers are the core engine of many tools for analyzing and verifying software, planning, etc.

SMT solvers

- SMT solvers in their implementation exploit propositional SAT technology. (We will see more about this topic later.)
- An SMT solver can also be used simply as a SAT solver. In this case one just use the Core theory, with the sort Bool and the basic Boolean operators.
- There are many different SMT solvers: Alt-Ergo, CVC4, CVC5, MathSAT 5, Yices 2, Z3, ...
- When using an SMT solver as a SAT solver, we do not need to convert the Boolean formulas to CNF. The tool does that internally.
- See the Colab Notebook with examples of using the Z3 API for Python with propositional logic.