Logic(s) for Computer Science - Week 10
Semantics of First-Order Logic

December 10, 2018

1 Introduction

The syntax of first-order logic tells us what are, from a syntactic point of view, the formulae of first-order logic. The semantics of the logic tells us what the *meaning* of each formula is. The semantics of a formula (or its meaning) shall be a truth value. In general, just as for propositional logic, the truth value of a formula depends not only on the formula itself, but also on the *mathematical structure* over which the formula is evaluated and also on an assignment of values to the *free variables* of the formula.

2 The Variables of a Formula

We will denote by $vars(\varphi)$ the variables of the formula φ. For example, we shall have $vars(\forall x.(P(x,y))) = \{x,y\}$. We next define the function $vars : \texttt{LP1} \to 2^{\mathcal{X}}$.

First of all, we define a function $vars : \mathcal{T} \to 2^{\mathcal{X}}$ as being the function that associates to a term (from the set $\mathcal{T}$) the set of variables occurring in that term. All following definitions are defined inductively, as the corresponding syntactic definitions are. We call them recursive definitions and we do not explicitly differentiate the base case and the inductive cases. We recall that the set $2^{\mathcal{X}}$ is the set of all subsets of $\mathcal{X}$.

Definition 2.1. *The function $vars : \mathcal{T} \to 2^{\mathcal{X}}$ is defined recursively as follows:*

1. *$vars(c) = \emptyset$ (if $c \in \mathcal{F}_0$ is a constant symbol);*
2. *$vars(x) = \{x\}$ (if $x \in \mathcal{X}$ is a variable);*
3. *$vars(f(t_1, \ldots, t_n)) = \bigcup_{i \in \{1,\ldots,n\}} vars(t_i)$.*

We can now define the homonymous (extended) function $vars : \texttt{LP1} \to 2^{\mathcal{X}}$, which associates to any first-order formula the set of variables of the formula:

Definition 2.2. *The function $vars : \texttt{LP1} \to 2^{\mathcal{X}}$ is define recursively as follows:*

1. $vars(P(t_1, \ldots t_n)) = vars(t_1) \cup \ldots \cup vars(t_n)$;
2. $vars(\neg\varphi) = vars(\varphi)$;
3. $vars(\varphi_1 \wedge \varphi_2) = vars(\varphi_1) \cup \varphi(\varphi_2)$;
4. $vars(\varphi_1 \vee \varphi_2) = vars(\varphi_1) \cup \varphi(\varphi_2)$;
5. $vars(\varphi_1 \rightarrow \varphi_2) = vars(\varphi_1) \cup \varphi(\varphi_2)$;
6. $vars(\varphi_1 \leftrightarrow \varphi_2) = vars(\varphi_1) \cup \varphi(\varphi_2)$;
7. $vars(\exists x.\varphi) = vars(\varphi) \cup \{x\}$;
8. $vars(\forall x.\varphi) = vars(\varphi) \cup \{x\}$.

Observe that the variable x is added to the set of variables even if it appears only as next to the quantifier $\exists$ or $\forall$.

Example 2.1. *Consider the formula* $\varphi = \Big(\forall x.\big(P(x,y) \wedge \exists y.\big(P(z, f(x,y)) \wedge P(x,y)\big)\big)\Big) \wedge P(x,x)$.

We have that $vars(\varphi) = \{x, y, z\}$.

2.1 The Scope of a Quantifier - Analogy with Programming Languages

In a programming language, we may declare several variables having the same name. For example, in C, we could have the following code:

```
/*  1:*/ int f()
/*  2:*/ {
/*  3:*/   int s = 0;
/*  4:*/   for (int x = 1; x <= 10; ++x) {
/*  5:*/     for (int y = 1; y <= 10; ++y) {
/*  6:*/       s += x * y * z;
/*  7:*/       for (int x = 1; x <= 10; ++x) {
/*  8:*/         s += x * y * z;
/*  9:*/       }
/* 10:*/     }
/* 11:*/   }
/* 12:*/   return s;
/* 13:*/ }
```

In the code fragment above, there are three variables, two of them having the same name, `x`. The scope of the variable `x` declared on line 4 is between the lines $4 - 11$, and the scope of the variable `x` declared on line 7 is the lines $7 - 9$. This way, any occurence of the name `x` between lines $7 - 9$ refers to the second

declaration of the variable, while any occurrence of the name `x` between lines $4-11$ (except lines $7-9$) refers to the second declaration of `x`. For example, the occurrence of `x` on line 6 refers to the variable x declared on line 4. The occurrence of `x` on line 8 refers to the variable x declared on line 7.

The lines $4-11$ represent the scope of the first declaration of `x`, and the lines $7-9$ represent the scope of the second declaration of `x`. The variable z is a global variable.

This is similar to first-order formulae. For example, in the formula $\forall x.\forall y.(P(x,y) \wedge P(x,z) \wedge (\exists x.P(x,y)))$, the variable x is quantified twice (the first time universally, the second time existentially). A quantification of a variable is called *binding* (ro. *legare*). A *binding* is similar, from the point of view of the scope of the variable, to defining a variable in a programming language.

This way, the scope of the variable x that is quantified universally is $\forall y.(P(x,y) \wedge P(x,z) \wedge (\exists x.P(x,y)))$, while the scope of the variable x that is quantified existentially is $P(x,y)$. The occurrences of a variable in the scope of a quantifier that binds the variable are called *bound occurrences*, while the occurrences of a variable outside of the scope of any quantifier that binds the variable are called *free occurrences*.

The following definitions establish formally the notion of free/bound occurrence and of free/bound variable. The free occurrences of a variable in first-order logic are, as an analogy, similar to global variables in a programming language.

2.2 Free and Bound Occurrences of Variables

Recall the definition of the *syntax abstract tree* associated to a first order formula and consider that the sentence *"on the path to the root"* has a formal definition.

Definition 2.3. *A* free occurrence of a variable x in a formula φ *is a node in the tree of the formula φ labeled by x and having the property that there is no node labeled $\forall x$ or $\exists x$ on the path to the root.*

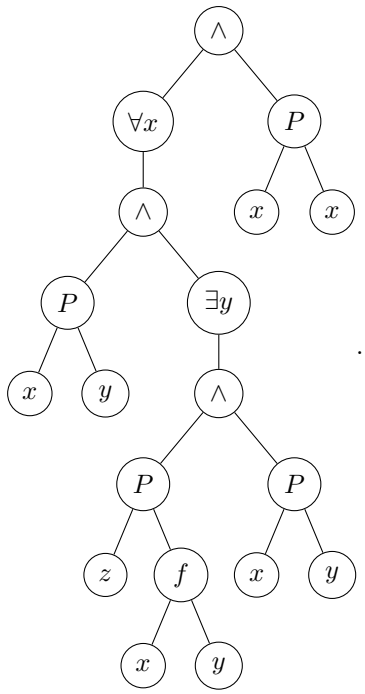
Definition 2.4. *A* bound occurrence of a variable x in a formula φ *is a node in the tree of the formula labeled by x and having the property that, on the path towards the root, there is at least a node labeled by $\forall x$ or by $\exists x$.*

The closest such node (labeled by $\forall x$ or by $\exists x$) is the quantifier that binds *that particular occurrence of the variable x.*

Example 2.2. *We next consider the formula*

$$\varphi = \Big(\forall x.\Big(P(x,y) \wedge \exists y.\big(P(z,f(x,y)) \wedge P(x,y)\big)\Big)\Big) \wedge P(x,x).$$

Its abstract syntax tree is:

$ast(\varphi) =$ 

.

In the formula φ above, the variable x has two free occurrences. The variable y has one free occurrence. The variable z has one free occurrence. All the free occurrences of the variables in the formula φ are thickened in the following identity:

$$\varphi = \Big(\forall x.\Big(P(x,\boldsymbol{y}) \wedge \exists y.\big(P(\boldsymbol{z}, f(x,y)) \wedge P(x,y)\big)\Big)\Big) \wedge P(\boldsymbol{x},\boldsymbol{x}).$$

All the bound occurrences of the variables in the formula φ are marked with a thickened font in the following identity:

$$\varphi = \Big(\forall x.\Big(P(\boldsymbol{x},y) \wedge \exists y.\big(P(z, f(\boldsymbol{x},\boldsymbol{y})) \wedge P(\boldsymbol{x},\boldsymbol{y})\big)\Big)\Big) \wedge P(x,x).$$

Remark 2.1. *Some authors also consider that the nodes labeled by $\forall x$ or by $\exists x$ are bound occurrences of the variable x. In this course, we shall not consider the nodes $\forall x$ and respectively $\exists x$ as being occurrences of the variable x, but simply as binding sites for the variable x.*

2.3 Free Variables and Bound Variables

The set of variables that have at least one free occurrence in a formula φ is denoted $\mathit{free}(\varphi)$.

Definition 2.5. *The function $\mathit{free} : \mathtt{LP1} \to 2^{\mathcal{X}}$ is defined recursively as follows:*

1. $\mathit{free}(P(t_1, \ldots, t_n)) = \mathit{vars}(t_1) \cup \ldots \cup \mathit{vars}(t_n)$;
2. $\mathit{free}(\neg\varphi) = \mathit{free}(\varphi)$;
3. $\mathit{free}(\varphi_1 \wedge \varphi_2) = \mathit{free}(\varphi_1) \cup \mathit{free}(\varphi_2)$;
4. $\mathit{free}(\varphi_1 \vee \varphi_2) = \mathit{free}(\varphi_1) \cup \mathit{free}(\varphi_2)$;
5. $\mathit{free}(\varphi_1 \to \varphi_2) = \mathit{free}(\varphi_1) \cup \mathit{free}(\varphi_2)$;
6. $\mathit{free}(\varphi_1 \leftrightarrow \varphi_2) = \mathit{free}(\varphi_1) \cup \mathit{free}(\varphi_2)$;
7. $\mathit{free}(\forall x.\varphi) = \mathit{free}(\varphi) \setminus \{x\}$;
8. $\mathit{free}(\exists x.\varphi) = \mathit{free}(\varphi) \setminus \{x\}$.

Example 2.3. *Continuing the previous example, for the formula*

$$\varphi = \Big(\forall x.\Big(P(x,y) \wedge \exists y.\big(P(z, f(x,y)) \wedge P(x,y)\big)\Big)\Big) \wedge P(x,x),$$

we have that

$$\mathit{free}(\varphi) = \{x, y, z\}.$$

By $\mathit{bound}(\varphi)$ we denote the set of variables bound in a formula, that is the set of those variables x with the property that there exists in the formula at least one node labeled by $\forall x$ or by $\exists x$.

Definition 2.6. *The function* $\mathit{bound} : \texttt{LP1} \to 2^{\mathcal{X}}$ *is defined recursively as follows:*

1. $\mathit{bound}(P(t_1, \ldots, t_n)) = \emptyset$;
2. $\mathit{bound}(\neg\varphi) = \mathit{bound}(\varphi)$;
3. $\mathit{bound}(\varphi_1 \wedge \varphi_2) = \mathit{bound}(\varphi_1) \cup \mathit{bound}(\varphi_2)$;
4. $\mathit{bound}(\varphi_1 \vee \varphi_2) = \mathit{bound}(\varphi_1) \cup \mathit{bound}(\varphi_2)$;
5. $\mathit{bound}(\varphi_1 \to \varphi_2) = \mathit{bound}(\varphi_1) \cup \mathit{bound}(\varphi_2)$;
6. $\mathit{bound}(\varphi_1 \leftrightarrow \varphi_2) = \mathit{bound}(\varphi_1) \cup \mathit{bound}(\varphi_2)$;
7. $\mathit{bound}(\forall x.\varphi) = \mathit{bound}(\varphi) \cup \{x\}$;
8. $\mathit{bound}(\exists x.\varphi) = \mathit{bound}(\varphi) \cup \{x\}$.

Definition 2.7 (Bound Variables of a Formula)**.** *The* bound variables of a formula φ *are the elements of* $\mathit{bound}(\varphi)$.

Definition 2.8 (Free Variables of a Formula)**.** *The* free variables of a formula φ *are the elements of the set* $\mathit{free}(\varphi)$.

Remark 2.2. *The sets free(φ) and bound(φ) could in general have common elements.*

Remark 2.3. *A variable can have several occurrences in a formula. Some of the occurrences could be free in the formula, while others could be bound.*

We must make the distinction between a free occurrence of a variable in a formula *and a* free variable of a formula*. The free occurrence is a node in the abstract syntax tree of the formula, while the free variable is an element of the set $\mathcal{X}$.*

We must also make the distinction between a bound occurrence of a variable in a formula *and a* bound variable of a formula*. The bound occurrence is a node in the abstract syntax tree, while the bound variable is an element of the set $\mathcal{X}$.*

2.4 The Scope of a Bound Variable and Brackets

Now that we have understood the scope of a bound variable, we may clarify a subtlety related to the order of priority of logical connectives. Up to this point, we have established that the order of priority is: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists$. However, the quantifiers, $\forall$ and $\exists$, interact subtly with the other logical connectives, making parathesizing a formula more complicated.

Clarification: a formula should be paranthesized such that the scope of a bound variable extends as far to the right as possible.

For example, the formula:

$$\forall x.P(x,x) \vee \neg\exists y.P(x,y) \wedge P(x,x)$$

should be paranthesized as follows:

$$(\forall x.(P(x,x) \vee (\neg(\exists y.(P(x,y) \wedge P(x,x)))))).$$

3 The Semantics of First-Order Formulae

We recall that a signature $\Sigma = (\mathcal{P}, \mathcal{F})$ is a pair consisting of predicate symbols $\mathcal{P}$ and functional symbols $\mathcal{F}$, each symbol having an associated natural number called its arity.

Example 3.1. *In the following examples, we shall work over the signature $\Sigma = (\{P\}, \{f, i, e\})$, where P is a predicate symbol of arity 2, and f, i and e are functional symbols of arity 2, 1 and respectively 0.*

Considering a fixed signature Σ, we denote by $\mathcal{P}_n$ the set of predicate symbols of arity n in the signature, and by $\mathcal{F}_n$ the set of functional symbols of arity n in the signature.

Example 3.2. *Continuing the previous example, $\mathcal{P}_2 = \{P\}, \mathcal{P}_1 = \emptyset, \mathcal{P}_0 = \emptyset, \mathcal{F}_2 = \{f\}, \mathcal{F}_1 = \{i\}, \mathcal{F}_0 = \{e\}$.*

We recall that, if $\Sigma = (\mathcal{P}, \mathcal{F})$ is a signature, by Σ-structure we understand any tuple $S = (D, Pred, Fun)$ with the property that:

1. D is a non-empty set called the domain of the structure S;
2. for every predicate symbol $P \in \mathcal{P}$, there is a predicate $P^S \in Pred$ with the same arity as P;
3. for every functional symbol $f \in \mathcal{P}$, there is a function $f^S \in Fun$ with the same arity as f;

Example 3.3. *Continuing the previous examples, where* $\Sigma = (\{P\}, \{f, i, e\})$, *consider the following* Σ*-structures:*

1. $S_1 = (\mathbb{Z}, \{=\}, \{+, -, 0\})$;
2. $S_2 = (\mathbb{R}^+, \{=\}, \{\times, \cdot^{-1}, 1\})$;
3. $S_3 = (\mathbb{N}, \{=\}, \{+, s, 0\})$;
4. $S_4 = (\mathbb{N}, \{<\}, \{+, s, 0\})$.
5. $S_5 = (\mathbb{Z}, \{<\}, \{+, -, 0\})$.

The structure S_1 *has the domain* $\mathbb{Z}$ *(the set of integers), the predicate associated to the predicate symbol* P *is* $=$ *(the equality predicate for integers), the function* $+$ *is the addition function for integers,* $-$ *is the unary minus function, and the constant symbol* e *has* 0 *as the associated constant.*

The structure S_2 *has the domain* $\mathbb{R}^+$ *(the set of positive reals), the predicate associated to the predicate symbol* P *is* $=$ *(the equality predicate over positive reals), the function* $\times$ *is the multiplication function over positive reals,* $\cdot^{-1}$ *is the unary function that computes the inverse of a positive real number (e.g.* $5^{-1} = \frac{1}{5}$, *and* $\frac{1}{10}^{-1} = 10$*), and the constant* 1 *is associated to the constant symbol* e.

The structure S_3 *has the domain* $\mathbb{N}$ *(the set of naturals), the predicate associated to the predicate symbol* P *is* $=$ *(the equality predicate for natural numbers), the function* $+$ *is the addition function for naturals,* s *is the* successor *function (which associates to a natural number the next natural number – e.g.,* $s(7) = 8$*), and the constant* 0 *is associated to the constant symbol* e.

The structure S_4 *has the domain* $\mathbb{N}$ *(the set of naturals), the predicate associated to the symbol* P *is* $<$ *(the* smaller than *relation over naturals), the function* $+$ *is the addition function for naturals,* s *is the successor function, and the constant* 0 *is associated to the constant symbol* e.

The structure S_5 *is similar to* S_1*, but the predicate symbol* P *is associated with the* less than *relation instead of* equality.

Using the notation above, we have that $P^{S^4} =<$, $f^{S_2} = \times$, *and* $e^{S_1} = 0$.

3.1 The Notion of Assignment

As in the propositional logic, in order to obtain the truth value of a formula in a structure, we have to start by fixing some concrete values to the syntactic symbols over which the formula is built. In the case of first order logic, we start with the variables.

Definition 3.1 (Assignment)**.** *Let Σ be a signature and let S be Σ-structure with the domain D.*

An S-assignment is any function

$$\alpha : \mathcal{X} \to D.$$

Example 3.4. *Consider the S_1-assignment $\alpha_1 : \mathcal{X} \to \mathbb{Z}$ defined as follows:*

1. $\alpha_1(x_1) = 5$;
2. $\alpha_1(x_2) = 5$;
3. $\alpha_1(x_3) = 6$;
4. $\alpha_1(x) = 0$ *for any* $x \in \mathcal{X} \setminus \{x_1, x_2, x_3\}$.

Example 3.5. *Consider the S_1-assignment $\alpha_2 : \mathcal{X} \to \mathbb{Z}$ defined as follows:*

1. $\alpha_2(x_1) = 6$;
2. $\alpha_2(x_2) = 5$;
3. $\alpha_2(x_3) = 6$;
4. $\alpha_2(x) = 0$ *for any* $x \in \mathcal{X} \setminus \{x_1, x_2, x_3\}$.

Definition 3.2 (The Value of a Term in an Assignment)**.** *Given an S-assignment α and a term $t \in \mathcal{T}$ over the signature Σ,* the value of the term t in the assignment α *is an element of the domain D, denoted by $\overline{\alpha}(t)$, and computed as follows:*

1. $\overline{\alpha}(c) = c^S$ *if $c \in \mathcal{F}_0$ is a constant symbol;*
2. $\overline{\alpha}(x) = \alpha(x)$ *if $x \in \mathcal{X}$ is a variable;*
3. $\overline{\alpha}(f(t_1, \ldots, t_n)) = f^S(\overline{\alpha}(t_1), \ldots, \overline{\alpha}(t_n))$ *if $f \in \mathcal{F}_n$ is a function symbol of arity n, and $t_1, \ldots, t_n$ are terms.*

Example 3.6. *Continuing the previous examples, we have that*

$$\begin{aligned}
\overline{\alpha_1}(f(i(x_1), e)) &= \overline{\alpha_1}(i(x_1)) + \overline{\alpha_1}(e) \\
&= -(\overline{\alpha_1}(x_1)) + e^{S_1} \\
&= -(\alpha_1(x_1)) + 0 \\
&= -5 + 0 \\
&= -5.
\end{aligned}$$

Therefore, the value of the term $f(i(x_1), e)$ in the assignment α_1 is -5.

Definition 3.3 (Updating an Assignment)**.** *Given an assignment α, a variable $x \in \mathcal{X}$ and a value $u \in D$, we denote by $\alpha[x \mapsto u]$ a new assignment, obtained from α by updating the value of the variable x to u, formally defined as follows:*

$$\alpha[x \mapsto u] : \mathcal{X} \to D, \text{ s.t.}$$

1. $(\alpha[x \mapsto u])(x) = u$;
2. $(\alpha[x \mapsto u])(y) = \alpha(y)$, *for any* $y \in \mathcal{X} \setminus \{x\}$.

Example 3.7. *For example, the assignment $\alpha_1[x_1 \mapsto 6]$ is exactly the same as the assignment α_2 defined earlier.*

The value of the term $f(i(x_1), e)$ in the assignment $\alpha_1[x \mapsto 6]$ is $\overline{\alpha_1[x \mapsto 6]}(f(i(x_1), e)) = -6$.

3.2 The Truth Value of a Formula

Let Σ be a signature, let S be a Σ-structure and let α be an S-assignment. Having fixed the elements above, we may compute the truth value of a first-order formula constructed over the signature Σ.

By writing $S, \alpha \models \varphi$ we denote the fact that the formula φ is true in the structure S with the assignment α. By writing $S, \alpha \not\models \varphi$ we denote the fact that the formula φ is not true (is false) in the structure S with the assignment α.

The notation $S, \alpha \models \varphi$ can also be read as *S satisfies φ with the assignment α*, and $S, \alpha \not\models \varphi$ can also be read as *S does not satisfy φ with the assignment α*.

Definition 3.4. *The fact that a structure S satisfies a formula φ with an assignment α (or equivalently, that φ is true in the structure S with the assignment α) is defined inductively as follows:*

1. $S, \alpha \models P(t_1, \ldots, t_n)$ *if* $P^S(\overline{\alpha}(t_1), \ldots, \overline{\alpha}(t_n)) = 1$;
2. $S, \alpha \models \neg\varphi$ *if* $S, \alpha \not\models \varphi$;
3. $S, \alpha \models \varphi_1 \wedge \varphi_2$ *if* $S, \alpha \models \varphi_1$ *and* $S, \alpha \models \varphi_2$;
4. $S, \alpha \models \varphi_1 \vee \varphi_2$ *if* $S, \alpha \models \varphi_1$ *or* $S, \alpha \models \varphi_2$;
5. $S, \alpha \models \varphi_1 \to \varphi_2$ *if* $S, \alpha \not\models \varphi_1$ *or* $S, \alpha \models \varphi_2$;
6. $S, \alpha \models \varphi_1 \leftrightarrow \varphi_2$ *if (1) both* $S, \alpha \models \varphi_1$, *and* $S, \alpha \models \varphi_2$, *or (2)* $S, \alpha \not\models \varphi_1$ *and* $S, \alpha \not\models \varphi_2$;
7. $S, \alpha \models \exists x.\varphi$ *if there exists* $u \in D$ *such that* $S, \alpha[x \mapsto u] \models \varphi$;
8. $S, \alpha \models \forall x.\varphi$ *if for any* $u \in D$, *we have that* $S, \alpha[x \mapsto u] \models \varphi$.

Example 3.8. *We shall work over the signature* $\Sigma = (\{P\}, \{f, i, e\})$*, the* Σ*-structure* $S_1 = (\mathbb{Z}, \{=\}, \{+, -, 0\})$ *and the* S_1*-assignments* α_1, α_2*.*

We have that

$$\begin{aligned} S_1, \alpha_1 \models P(x_1, x_1) \text{ if } & \quad P^{S_1}(\overline{\alpha_1}(x_1), \overline{\alpha_1}(x_1)) = 1, \\ \text{if } & \quad \overline{\alpha_1}(x_1) = \overline{\alpha_1}(x_1), \\ \text{if } & \quad \alpha_1(x_1) = \alpha_1(x_1), \\ \text{if } & \quad 5 = 5. \end{aligned}$$

As $5 = 5$*, we have that* $S_1, \alpha_1 \models P(x_1, x_1)$*, meaning that the formula* $P(x_1, x_1)$ *is true in the structure* S_1 *with the assignment* α_1*. Equivalently, we say that* S_1 *satisfies* $P(x_1, x_1)$ *with the assignment* α_1*.*

Example 3.9. *Continuing the previous example, we have that*

$$\begin{aligned} S_1, \alpha_1 \models P(x_1, x_3) \text{ if } & \quad P^{S_1}(\overline{\alpha_1}(x_1), \overline{\alpha_1}(x_3)) = 1, \\ \text{if } & \quad \overline{\alpha_1}(x_1) = \overline{\alpha_1}(x_3), \\ \text{if } & \quad \alpha_1(x_1) = \alpha_1(x_3), \\ \text{if } & \quad 5 = 6. \end{aligned}$$

As $5 \neq 6$*, we have that* $S_1, \alpha_1 \not\models P(x_1, x_3)$*, meaning that* $P(x_1, x_3)$ *is false in the structure* S_1 *with the assignment* α_1*. Equivalently, we say that* S_1 *does not satisfy* $P(x_1, x_3)$ *with the assignment* α_1*.*

Example 3.10. *Continuing the previous example, we have that*

$$\begin{aligned} S_1, \alpha_1 \models \neg P(x_1, x_3) \text{ if } & \quad S_1, \alpha_1 \not\models P(x_1, x_3) \\ \text{if } & \quad P^{S_1}(\overline{\alpha_1}(x_1), \overline{\alpha_1}(x_3)) = 0, \\ \text{if } & \quad \overline{\alpha_1}(x_1) \neq \overline{\alpha_1}(x_3), \\ \text{if } & \quad \alpha_1(x_1) \neq \alpha_1(x_3), \\ \text{if } & \quad 5 \neq 6. \end{aligned}$$

As $5 \neq 6$*, we have that* $S_1, \alpha_1 \models \neg P(x_1, x_3)$*, meaning that the formula* $\neg P(x_1, x_3)$ *is true in the structure* S_1 *with the assignment* α_1*. Equivalently, we say that* S_1 *satisfies* $\neg P(x_1, x_3)$ *with the assignment* α_1*.*

Example 3.11. *Continuing the previous example, we have that*

$$\begin{aligned} S_1, \alpha_1 \models P(x_1, x_1) \wedge \neg P(x_1, x_3) \text{ if } & \quad S_1, \alpha_1 \models P(x_1, x_1) \text{ and } S_1, \alpha_1 \models \neg P(x_1, x_3), \\ \text{if } & \quad \ldots \text{ and } \ldots, \\ \text{if } & \quad 5 = 5 \text{ and } 5 \neq 6. \end{aligned}$$

As $5 = 5$ *și* $5 \neq 6$*, we have that* $S_1, \alpha_1 \models P(x_1, x_1) \wedge \neg P(x_1, x_3)$*.*

Example 3.12. *Continuing the previous example, we have that*

$$S_1, \alpha_1 \models P(x_1, x_3) \vee P(x_1, x_1) \text{ if } \quad S_1, \alpha_1 \models P(x_1, x_3) \text{ or } S_1, \alpha_1 \models P(x_1, x_1).$$

We have already seen that $S_1, \alpha_1 \models P(x_1, x_3)$*, so* $S_1, \alpha_1 \models P(x_1, x_3) \vee P(x_1, x_1)$ *(even if* $S_1, \alpha_1 \not\models P(x_1, x_1)$*).*

Example 3.13. *Continuing the previous example, we have that*

$$\begin{aligned}
S_1, \alpha_1 \models \exists x_3.P(x_1, x_3) \text{ if } & \text{ there exists } u \in D \text{ s.t. } S_1, \alpha_1[x_1 \mapsto u] \models P(x_1, x_3), \\
\text{if } & \text{ there exists } u \in D \text{ s.t. } P^{S_1}(\overline{\alpha_1[x_1 \mapsto u]}(x_1), \overline{\alpha_1[x_1 \mapsto u]}(x_3)) = 1, \\
\text{if } & \text{ there exists } u \in D \text{ s.t. } \overline{\alpha_1[x_1 \mapsto u]}(x_1) = \overline{\alpha_1[x_1 \mapsto u]}(x_3), \\
\text{if } & \text{ there exists } u \in D \text{ s.t. } \alpha_1[x_1 \mapsto u](x_1) = \alpha_1[x_1 \mapsto u](x_3), \\
\text{if } & \text{ there exists } u \in D \text{ s.t. } u = \alpha_1(x_3), \\
\text{if } & \text{ there exists } u \in D \text{ s.t. } u = 6.
\end{aligned}$$

As there exists u *(namely* $u = 6$*) so that* $u = 6$*, we have that* $S_1, \alpha_1 \models \exists x_3.P(x_1, x_3)$*.*

Example 3.14. *Continuing the previous example, we have that*

$$\begin{aligned}
S_1, \alpha_1 \models \forall x_1.\exists x_3.P(x_1, x_3) \text{ if } & \text{ for any } u \in D, \text{ we have } S_1, \alpha_1[x_1 \mapsto u] \models \exists x_3.P(x_1, x_3), \\
\text{if } & \text{ for any } u \in D, \text{ there exists } v \in D \text{ s.t. } S_1, \alpha_1[x_1 \mapsto u][x_3 \mapsto v] \models P(x_1, x_3), \\
\text{if } & \ldots \\
\text{if } & \text{ for any } u \in D, \text{ we have that there exists } v \in D \text{ s.t. } u = v.
\end{aligned}$$

As for any integer u*, there exists an integer* v *so that* $u = v$*, we have that* $S_1, \alpha_1 \models \forall x_1.\exists x_3.P(x_1, x_3)$*.*

Exercise 3.1. *Show that* $S_1, \alpha_1 \models \forall x_1.\exists x_3.P(x_1, i(x_3))$*.*

3.3 Semantical Notions

Satisfiability in a Fixed Structure

Definition 3.5 (Satisfiability in a Fixed Structure)**.** *A formula* φ is satisfiable in a structure S *if there exists an* S*-assignment* α *such that*

$$S, \alpha \models \varphi.$$

Example 3.15. *The formula* $P(x_1, x_3)$ *is satisfiable in the structure* S_1*, because there is an assignment, namely* α_2*, having the property* $S_1, \alpha_2 \models P(x_1, x_3)$*.*

The formula $\neg P(x_1, x_1)$ *is not satisfiable in the structure* S_1*, because for any assignment* α *we would choose,* $S_1, \alpha \not\models \neg P(x_1, x_1)$ *(check that this is indeed the case).*

Validity in a Fixed Structure

Definition 3.6 (Validity in a Fixed Structure)**.** *A formula* φ is valid in a structure S *if, for any* S*-assignment* α*, we have that*

$$S, \alpha \models \varphi.$$

Example 3.16. *The formula* $P(x_1, x_3)$ *is not valid in the structure* S_1*, because there exists an assignment, namely* α_1*, having the property that* $S_1, \alpha_1 \not\models P(x_1, x_3)$*.*

The formula $P(x_1, x_1)$ *is valid in the structure* S_1*, because for any assignment* α *we would choose,* $S_1, \alpha \models P(x_1, x_1)$ *(check that this is indeed the case).*

Satisfiability

Definition 3.7 (Satisfiability)**.** *A formula* φ is satisfiable *if there exists a structure* S *and an* S*-assignment* α *such that*

$$S, \alpha \models \varphi.$$

Example 3.17. *The formula* $\neg P(x_1, x_1)$ *is satisfiable, because there exists a structure (namely* S_5*) and an* S_5*-assignment (namely* α_1*) so that* $S_5, \alpha_1 \models \neg P(x_1, x_1)$ *(because* $5 \not< 5$*).*

Remark. Because S_5 *and* S_1 *have the same domain, the assignment* α_1 *is both an* S_1*-assignment as well as an* S_5*-assignment.*

Remark 3.1. *A formula could be unsatisfiable in a fixed structure (for example* $\neg P(x_1, x_1)$ *is not satisfiable in the structure* S_1*) but it could still be satisfiable (for example* $\neg P(x_1, x_1)$*, as we have seen above).*

Validity

Definition 3.8 (Validity)**.** *A formula* φ is valid *if, for any structure* S *and for any* S*-assignment* α*, we have that*

$$S, \alpha \models \varphi.$$

Example 3.18. *The formula* $P(x_1, x_1)$ *is not valid, because* $S_5, \alpha_1 \not\models P(x_1, x_1)$*.*

The formula $P(x_1, x_1) \to P(x_1, x_1)$ *is valid.*

Remark 3.2. *A formula could be valid in a fixed structure (for example* $P(x_1, x_1)$ *is valid in the structure* S_1*) and still not be valid (for example,* $P(x_1, x_1)$*).*

Semantical Consequence

Definition 3.9. *A formula* φ is a semantical (or logical) consequence of the formulae $\varphi_1, \ldots, \varphi_n$ in a fixed structure S*, denoted by* $\varphi_1, \ldots, \varphi_n \models_S \varphi$*, if, for any* S*-assignment* α *for which* $S, \alpha \models \varphi_1$*,* $S, \alpha \models \varphi_2$*, . . . ,* $S, \alpha \models \varphi_n$*, we also have that* $S, \alpha \models \varphi$*.*

Example 3.19. *We have $P(x,y) \models_{S_1} P(y,x)$, because, for any S_1-assignment α with the property that $S_1, \alpha \models P(x,y)$ (meaning that $\alpha(x) = \alpha(y)$), we also have $S_1, \alpha \models P(y,x)$ (meaning $\alpha(y) = \alpha(x)$).*

We have that $P(x,y) \not\models_{S_5} P(y,x)$, because, for the assignment $\alpha(x) = 5$, $\alpha(y) = 6$, we have $S_5, \alpha \models P(x,y)$ (that is, $5 < 6$), bu $S_5, \alpha \not\models P(y,x)$ ($6 \not< 5$).

Definition 3.10. *A formula φ is a semantical (or logicl) consequence of the formulae $\varphi_1, \ldots, \varphi_n$, denoted by $\varphi_1, \ldots, \varphi_n \models \varphi$, if*

$$\varphi_1, \ldots, \varphi_n \models_S \varphi$$

for any structure S.

Example 3.20. *We have $P(x,y) \not\models P(y,x)$, because there exists a structure (namely S_5) so that $P(x,y) \not\models_{S_5} P(y,x)$.*

On the other hand, we have $\forall x.\forall y.\forall z.(P(x,y) \wedge P(y,z) \rightarrow P(x,z)), P(x_1,x_2), P(x_2,x_3) \models P(x_1,x_3)$.

Of course, in the previous notations (as in the propositional logic), the list $\varphi_1, \varphi_2 ... \varphi_n$ denotes the set having as elements the enumerated formulae.