First Order Logic

Mahesh Viswanathan

Fall 2018

First order logic is a formal language to describe and reason about *predicates*. Modern efforts to study this logic grew out of a desire to study the foundations of mathematics in number theory and set theory. It has a creaful treatment of functions, variables, and quantification. First order logic deals with predicates as opposed to propositions — which are declarative statements that are either true or false — which is the subject of study in propositional logic. A predicate is a group of words that *qualify* on or more names of *individuals*. Examples of predicates include "is a man", "is less than", "is instructor of". Applied to individuals, it forms statements that may be true or false — "*John* is a man", "*4* is less than *3*", "*Mahesh* is instructor of *CS498MV*". *Constants* are names of individuals to whome predicates can be applied. The distiguishing features of first order logic is the use of *quantifiers*, that allow one to state whether something holds for some or all individuals.

1 Syntax

First order logic formulas are defined over a *vocabulary* or *signature* that identifies the predicates and constants that can be used in the formulas.

Definition 1. A *vocabulary* or *signature* is $\tau = (\mathcal{C}, \mathcal{R})$, where $\mathcal{C} = \{c_1, c_2, \ldots\}$ is a set of *constant* symbols, and $\mathcal{R} = \{\mathcal{R}^k\}_k$ is a collection of sets with $\mathcal{R}^k = \{R_1^k, R_2^k, \ldots\}$ is a set of k-ary relation symbols.

Note that any of the above sets of constants, or k-ary relation symbols can be empty, finite, or infinite. A signature will be called *relational* if there are no constant symbols in the signature. A signature is *finite* if the toal number of symbols in the signature is finite.

We will typically consider signatures that are finite. When the arity of a relation symbol is clear from the context, we will drop the superscript.

Formulas in first-order logic over signature τ are sequences of symbols, where each symbols is one of the following.

1. The symbol $\bot$ called *false* and the symbol $=$
2. An element of the infinite set $\mathcal{V} = \{x_1, x_2, x_3, \ldots\}$ of *variables*
3. Constant symbols and relation symbols in τ
4. The symbol $\rightarrow$ called *implication*
5. The symbol $\forall$ called the *universal quantifier*
6. The symbols (and) called *parenthesis*

As always, not all such sequences are formulas; only *well formed* sequences are formulas in the logic. This si defined as follows.

Definition 2. A *well formed formula (wff)* over signature τ is inductively defined as follows.

1. $\bot$ is a wff.
2. If t_1, t_2 are either variables or constant symbols in τ then $t_1 = t_2$ is a wff.
3. If t_i is either a variable or a constant for $1 \leq i \leq k$ and R is a k-ary relation symbol in τ then $Rt_1t_2 \cdots t_k$ is a wff.
4. If φ and ψ are wffs then $(\varphi \to \psi)$ is a wff.
5. If φ is a wff and x is a variable then $(\forall x\varphi)$ is a wff.

Example 3. Consider signature $\tau = \{R\}$ where R is a binary relation symbol. The following are formulas over this signature.

- *Reflexivity:* $\forall xRxx$
- *Irreflexivity:* $\forall x(Rxx \to \bot)$
- *Symmetry:* $\forall x(\forall y(Rxy \to Ryx))$
- *Anti-symmetry:* $\forall x(\forall y(Rxy \to (Ryx \to x = y)))$
- *Transitivity:* $\forall x(\forall y(\forall z(Rxy \to (Ryz \to Rxz))))$

Non-examples of formulas include Rx (R expects two arguments); x (a variable is not a formula); $(Rxy \to Rzx$ (mismatched parentheses); $\forall x$ (x is quantified but there is no formula provided as argument).

It is useful to introduce logical operators in addition to those in Definition 2. These operators can be "syntactically" defined in terms of the operators in Definition 2. As in propositional logic, we can define the Boolean connectives *negation* as $\neg\varphi = (\varphi \to \bot)$, *disjunction* as $\varphi \vee \psi = ((\neg\varphi) \to \psi)$, *conjunction* as $\varphi \wedge \psi = \neg((\neg\varphi) \vee (\neg\psi))$, and *true* as $\top = \neg\bot$. Finally, we can define *existential quantification* as $(\exists x\varphi) = \neg(\forall x(\neg\varphi))$.

To avoid the clutter of parenthesis, and at the same time have an unambiguous interpretation of formulas, we will assume the following precedence of operators (from increasing to decreasing): $\neg$, $\wedge$, $\vee$, $\to$, $\forall$. Thus $\forall x \forall yx = y \to \neg Rxy$ means $(\forall x(\forall yx = y \to (\neg Rxy)))$.

2 Semantics

The semantics of formulas in any logic is defined with respect to a *model*. In the context of propositional logic, models were nothing but truth assignments to the propositions. For first order logic, models will objects that help identify the interpretation of constants and relation symbols. Such models are typically called *structures*.

Definition 4. A structure $\mathcal{A}$ of signature τ is $\mathcal{A} = (A, \{c^{\mathcal{A}}\}_{c\in\tau}, \{R^{\mathcal{A}}\}_{R\in\tau})$ where

- A is a non-empty set called the *domain/universe* of the structure,
- For each constant symbol $c \in \tau$, $c^{\mathcal{A}} \in A$ is its interpretation,
- For each k-ary relation symbol $R \in \tau$, $R^{\mathcal{A}} \in A^k$ is its interpretation.

The structure $\mathcal{A}$ is said to be *finite* if the universe A is finite. The universe of a structure $\mathcal{A}$ will be denoted by $u(\mathcal{A})$.

Many mathematical objects can be studied through the lens of logic. Let us look at some example signatures and structures that will play an important role in the rest of the course.

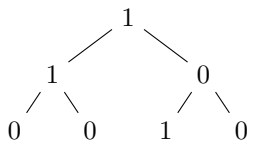

Figure 1: Example of labeled binary tree

Example 5. Consider the signature $\tau = \{E\}$, where E is a binary relation. We use this signature to study graphs. A graph $H = (V, E)$ modeled as a structure is $\mathcal{G} = (G, E^{\mathcal{G}})$, where the universe G is the set of vertices V, and for a pair of vertices $u, v \in G\ (= V)$, $E^{\mathcal{G}}uv$ [1] holds iff $(u, v) in E$.

Example 6. Let $\tau_O = \{<, S\}$ where $<$ and S are binary relation symbols. A finite order structure is $\mathcal{O} = (O, <^{\mathcal{O}}, S^{\mathcal{O}})$, where O is the universe of elements, $<$ is interpreted to be an ordering relation, and S as the "successor" relation.

Example 7. Words over an alphabet Σ can be viewed as a first order structure. Let $\tau_W = \{<, S, \{Q_a\}_{a\in\Sigma}\}$ where $<, S$ are binary relation symbols, and Q_a is unary relation symbol for every a in alphabet Σ. A word structure is $\mathcal{W} = (W, <^{\mathcal{W}}, S^{\mathcal{W}}, (Q_a^{\mathcal{W}})_{a\in\Sigma})$, where W is the set of positions in the word, $<$ is an order on positions, S is successor relation, Q_a holds in all positions that are labeled a.

For example, let us consider $\Sigma = \{0, 1\}$ and the word $w = 010110$. The w can be represented as a structure as follows. The signature (since $\sigma = \{0, 1\}$) is $\tau = \{<, S, Q_0, Q_1\}$. The structure for w is

$$\begin{aligned}\mathcal{W} = (\quad & \{1,2,3,4,5,6\},\\ & <^{\mathcal{W}} = \{(1,2),(1,3),\ldots(1,6),(2,3),\ldots(2,6),(3,4),(3,5),(3,6),(4,5),(4,6),(5,6)\},\\ & S^{\mathcal{W}} = \{(1,2),(2,3),(3,4),(4,5),(5,6)\},\\ & Q_0 = \{1,3,6\},\\ & Q_1 = \{2,4,5\}).\end{aligned}$$

Notice that though we took the positions of the word to be $1, 2, 3, 4, 5, 6$, we could have taken the positions to be any set of 6 elements.

Example 8. Labeled binary trees, where vertices are labeled by elements of Σ, can be represented as a structure. Let $\tau_T = \{<, S_0, S_1, (Q_a)_{a\in\Sigma}\}$ where $<, S_0, S_1$ are binary relation symbols, Q_a is a unary relation symbol. A tree (labeled by symbols in Σ) is a structure $\mathcal{T} = (T, <^{\mathcal{T}}, S_0^{\mathcal{T}}, S_1^{\mathcal{T}}, (Q_a^{\mathcal{T}})_{a\in\Sigma})$ where elements of T are called vertices, $<$ is a ancestor relation, S_0 and S_1 are the left and right child relations, respectively, and Q_a holds in all vertices labeled by a.

For example, consider the binary tree shown in Figure 1. Let us see how this tree is represented as a structure. The universe will consist of the vertices of the tree. We could use any names for the vertices. But it is convenient to name them in a manner that makes the edge relation explicit — the root will be ε, and for a vertex w, its left child will be $w0$, while its right child will be $w1$. Given this, the tree in Figure 1 corresponds to the following structure. $\mathcal{T} = (\{\varepsilon, 0, 1, 00, 01, 10, 11\}, <^{\mathcal{T}} = \{(u, uv) \mid v \neq \varepsilon\}, S_0^{\mathcal{T}} = \{(u, u0) \mid u \in \{\varepsilon, 0, 1\}\}, S_1^{\mathcal{T}} = \{(u, u1) \mid u \in \{\varepsilon, 0, 1\}\}, Q_0 = \{1, 00, 01, 11\}, Q_1 = \{\varepsilon, 0, 10\})$.

In order to define the semantics of a first order logic formula, we need a structure, and an *assignment*. An assignment maps every variable to an element in the universe of the structure.

Definition 9. For a τ-structure $\mathcal{A}$, an *assignment* over $\mathcal{A}$ is a function $\alpha : \mathcal{V} \to u(\mathcal{A})$ that assigns every variable $x \in \mathcal{V}$ a value $\alpha(x) \in u(\mathcal{A})$. If t is a constant symbol c, we will take $\alpha(t)$ to be $c^{\mathcal{A}}$.

For an assignment α over $\mathcal{A}$, $\alpha[x \mapsto a]$ is the assignment

$$\alpha[x \mapsto a](y) = \begin{cases} \alpha(y) & \text{for } y \neq x \\ a & \text{when } x = y \end{cases}$$

[1] For a relation symbol R, we will sometimes write $R^{\mathcal{A}}a_1a_2\cdots a_n$ instead of $(a_1, a_2, \ldots a_n) \in R^{\mathcal{A}}$.

We now have all the elements to define the semantics of a formula. The satisfaction relation will be a ternary relation — $\mathcal{A} \models \varphi[\alpha]$ to be read as "φ is true/holds in $\mathcal{A}$ under assignment α". The relation will be defined inductively on the structure of the formula. In defining the relation, we will also say $\mathcal{A} \not\models \varphi[\alpha]$ to mean that $\mathcal{A} \models \varphi[\alpha]$ does not hold.

Definition 10. The relation $\mathcal{A} \models \varphi[\alpha]$ is inductively defined as follows.

- $\mathcal{A} \not\models \bot[\alpha]$ for all $\mathcal{A}$ and α
- $\mathcal{A} \models t_1 = t_2[\alpha]$ iff $\alpha(t_1) = \alpha(t_2)$
- $\mathcal{A} \models Rt_1 \cdots t_n[\alpha]$ iff $(\alpha(t_1), \alpha(t_2), \ldots \alpha(t_n)) \in R^{\mathcal{A}}$
- $\mathcal{A} \models (\varphi \to \psi)[\alpha]$ iff $\mathcal{A} \not\models \varphi[\alpha]$ or $\mathcal{A} \models \psi[\alpha]$
- $\mathcal{A} \models (\forall x \varphi)[\alpha]$ iff for every $a \in u(\mathcal{A})$, $\mathcal{A} \models \varphi[\alpha[x \mapsto a]]$

Example 11. Consider $\mathcal{G} = (\{1,2,3,4\}, E^{\mathcal{G}} = \{(1,2),(2,1),(2,3),(3,2),(3,4),(4,3)\})$. For any assignment α, $\mathcal{G} \models \forall x \forall y (Exy \to Eyx)[\alpha]$ because

$$
\begin{array}{l}
\mathcal{G} \models \forall y (Exy \to Eyx)[\alpha[x \mapsto 1]] \text{ because} \\
\quad \mathcal{G} \models (Exy \to Eyx)[\alpha[x \mapsto 1][y \mapsto 1]] \text{ because} \\
\quad \mathcal{G} \not\models Exy[\alpha[x \mapsto 1][y \mapsto 1]] \text{ moreover} \\
\quad \mathcal{G} \models (Exy \to Eyx)[\alpha[x \mapsto 1][y \mapsto 2]] \text{ because} \\
\quad \vdots \\
\mathcal{G} \models \forall y (Exy \to Eyx)[\alpha[x \mapsto 2]] \text{ because} \\
\vdots
\end{array}
$$

Notice that in Example 11, the actual assignment to variables x and y did not matter when determining the satisfaction of the formula in the graph. This is because they are *bound* by the universal quantifiers in the formula φ. This leads us to the important notion of bound and free variables in a formula. We begin by defining the scope of a quantifier.

Definition 12. For a wff $\varphi = \forall x \psi$, ψ is said to be the *scope* of the quantifier $\forall x$.

Definition 13. Every occurrence of the variable x in $\forall x \psi$ is called a *bound occurrence* of x in φ.

Any occurrence of x which is not bound is called a *free occurrence* of x in φ.

The free variables in wff φ will be denoted by free(φ). The notation $\varphi(x_1, x_2, \ldots x_n)$ will be used to indicate that free(φ) $\subseteq \{x_1, \ldots x_n\}$.

Let us look at an example to understand the subtle definition of bound and free variables.

Example 14. Consider $\varphi = P\mathbf{xy} \to (\forall x(\forall y Rxy) \to Q x\mathbf{y})$ the free variables are shown in **bold**. Notice that a variable may occur both bound and free. As we will establish soon, we can change the names of bound variables without affecting the meaning of formulas. Thus $\psi = P\mathbf{xy} \to (\forall u(\forall v Ruv) \to Qu\mathbf{y})$ is an equivalent formula. Therefore, we will typically assume that bound and free variables are disjoint. In addtion, since bound variables can be renamed without affecting its meaning, we can also assume that every bound variable is in the scope of a unique quantifier. Thus, instead of $P\mathbf{xy} \to (\forall u(\forall v Ruv) \to (\forall y Quy))$, we will consider the equivalent formula $P\mathbf{xy} \to (\forall u(\forall v Ruv) \to (\forall z Quz))$

The satisfaction of a formula in a structure $\mathcal{A}$ under assignment α only depends on the values α assigns to the free variables; the values assigned to the bound variables in α are unimportant.

Theorem 15. *For a formula φ and assignments α_1 and α_2 such that for every $x \in$ free(φ), $\alpha_1(x) = \alpha_2(x)$, $\mathcal{A} \models \varphi[\alpha_1]$ iff $\mathcal{A} \models \varphi[\alpha_2]$.*

Theorem 15 can be proved by induction on the structure of the formula φ. The proof is left as an exercise for the reader. Theorem 15 suggest that if a formula has no free variables, its truth is independent of the assignment. Formulas without any free variables (i.e., those all of whose variables are bound) are an important class of formulas and have special name.

Definition 16. A *sentence* is a formula φ none of whose variables are free, i.e., $\text{free}(\varphi) = \emptyset$.

An immediate consequence of Theorem 15 is that the truth of sentences is independent of the assignment.

Proposition 17. *For a sentence φ, and any two assignments α_1 and α_2, $\mathcal{A} \models \varphi[\alpha_1]$ iff $\mathcal{A} \models \varphi[\alpha_2]$.*

Proposition 17 is an immediate consequence of Theorem 15. Thus, we say $\mathcal{A} \models \varphi$ whenever $\mathcal{A} \models \varphi[\alpha]$ for some α.

Definition 18. For a sentence φ, $\mathcal{A}$ is said to be a *model* of φ iff $\mathcal{A} \models \varphi$. We will denote by $[\![\varphi]\!]$ the set of all models of φ.

Satisfiability and validity/tautologies are defined in a manner similar to that for propositional logic — a formula is satisfiable if there is some model and assignment in which it is true, and it is valid if it is true in all models and assignments.

Definition 19. A formula φ over signature τ is said to be *satisfiable* iff for some τ-structure $\mathcal{A}$ and assignment α, $\mathcal{A} \models \varphi[\alpha]$.

A formula φ over signature τ is said to be *logically valid* iff for every τ-structure $\mathcal{A}$ and assignment α, $\mathcal{A} \models \varphi[\alpha]$. We will denote this by $\models \varphi$.

We can also define with a formula φ is a *logical consequence* of a set of formulas Γ in exactly the same way as we defined it for propositional logic.

Definition 20. For a set of formulas Γ, we say $\mathcal{A} \models \Gamma[\alpha]$ iff for every $\varphi \in \Gamma$, $\mathcal{A} \models \varphi[\alpha]$.

We say φ is a *logical consequence* of Γ, denoted by $\Gamma \models \varphi$, if and only if for every $\mathcal{A}$ and α, $\mathcal{A} \models \Gamma[\alpha]$ implies that $\mathcal{A} \models \varphi[\alpha]$. Thus, if $\emptyset \models \varphi$ then $\models \varphi$.

The following observation is an immediate consequence of the definition of logical consequence.

Proposition 21. *$\Gamma \cup \{\varphi\} \models \psi$ iff $\Gamma \models \varphi \to \psi$*

Finally two formulas are (semantically) equivalent, if the hold in exactly the same set of structures and assignments.

Definition 22. Formulas φ and ψ are said to be *logically equivalent* (denoted $\varphi \equiv \psi$) if for every $\mathcal{A}$ and assignment α, $\mathcal{A} \models \varphi[\alpha]$ iff $\mathcal{A} \models \psi[\alpha]$.

3 A Proof System

Is there an "algorithm" to determine if a first order logic formula φ is valid? This is the *entscheidungsproblem* ("the decision problem") posed by David Hilbert in 1928. The question predated a mathematical definition of computation, as in the Turing machine model or related computational models, and so when Hilbert posed this question, the notion of algorithm was an informal one. The answer to this question is not obvious, unlike the case of propositional logic. In propositional logic, thanks to the fact satisfaction only depends on the truth assignments to the variables appearing in the formula, we have an algorithm based on the truth table method where need to consider only *finitely many* models before we can sure that a formula is valid. This no longer holds for first order logic. There are sentences that are only satisfiable in infinite models. Consider for example the formula $\varphi = (\forall x \neg Rxx) \wedge (\forall x \forall y \forall z Rxy \wedge Ryz \to Rxz) \wedge (\forall x \exists y Rxy)$ which says that R is a irreflexive, transitive binary relation, that is total is only true in an infinite model. Therefore, the only

Axioms from Propositional Logic

$$\frac{}{\varphi \to (\psi \to \varphi)} \qquad \frac{}{(\varphi \to (\psi \to \rho)) \to ((\varphi \to \psi) \to (\varphi \to \rho))} \qquad \frac{}{((\varphi \to \bot) \to \bot) \to \varphi}$$

Axioms for Equality

$$\frac{}{x = x}$$ for any variable x

$$\frac{}{(t_1 = t_2) \to ((\varphi \to \psi) \wedge (\psi \to \varphi))}$$ where t_1, t_2 are variables or constant symbols, and ψ is the result of replacing in φ one occurrence of t_1 by t_2 such that this occurrence of t_1 is not in the well-formed part of φ of the form $\forall t_1 \varphi'$ or $\forall t_2 \varphi'$.

Axioms for Quantifiers

$$\frac{}{(\forall x \varphi) \to \varphi_t^x}$$ where t is a constant symbol or variable that is free in φ

$$\frac{}{(\forall x(\varphi \to \psi)) \to (\varphi \to (\forall x \psi))}$$ where x is a variable that is not free in φ.

Rules of Inference

$$\frac{\varphi \qquad \varphi \to \psi}{\psi}$$ Modus Ponens

$$\frac{\varphi}{\forall x \varphi},\ x \text{ is a variable}$$ Generalization

Figure 2: A sound and complete proof system for first order logic.

way to determine that $\neg\varphi$ is not valid is to check its truth in infinite models. Thus, the model enumeration process encapsulated in the truth table method will not yield a decision procedure for first order logic.

Gödel, in his doctoral dissertation, settled the question. He established the soundness and completeness of a proof system for first order logic. One such proof system for first order logic is shown in Figure 2.

In order to explain the axioms in the proof system, we need to introduce the notion of substitution.

Definition 23. For a wff φ, variable x and variable/constant symbol t, φ_t^x is the result of replacing every free occurrence of x in φ by t.

Example 24. Let us look at an example to better understand Definition 23. $(Pxy \to (\forall x(\forall y Rxy) \to Qxy))_z^x$ is the formula $(Pzy \to (\forall x(\forall y Rxy) \to Qxy))$.

On the other hand, $(Pxy \to (\forall x(\forall y Rxy) \to Qxy))_z^y$ if the formula $(Pxz \to (\forall x(\forall y Rxy) \to Qxz))$.

The axiom schemas in our proof system (Figure 2) can be divided into 3 classes — those which pertain to propositional reasoning, those which pertain to equality, and those which relate to quantifiers. The first axiom for universal quantifiers, uses the notion of substitution introduced in Definition 23. In addition to these axiom schemas, the proof system has two rules of inference. One is Modus Ponens which we have seen in the proof system for propositional logic, and then second is generalization that allows one to introduce universal quantifiers in the conclusion.

Proofs in our proof system are similar to those for propositional logic — they are sequence of formulas such that each line in the proof either is a hypothesis, an axiom or follows by a rule of inference. We give a precise definition for completeness.

Definition 25. A *proof of φ from a set of hypotheses* Γ is a sequence of wffs $\psi_1, \psi_2, \ldots \psi_m$ such that $\psi_m = \varphi$ and for every $k \in \{1, 2, \ldots m\}$

- $\psi_k \in \Gamma$, or
- ψ_k is an axiom, or
- ψ_k follows from earlier lines in the proof by a rule of inference

If φ has a proof from Γ, then φ is said to be *provable from* Γ in the proof system, and this is denoted by $\Gamma \vdash \varphi$.

As for propositional logic, we would like to argue that the proof system we have presented for first order logic is sound and complete, i.e., the proof system only proves "true" facts, and that all true facts can be proved in the system. It turns out this holds and was first proved by Gödel in his doctoral dissertation.

Theorem 26. *For a set of formulas Γ and a formula φ, $\Gamma \models \varphi$ if and only if $\Gamma \vdash \varphi$.*

The proof of the soundness and completeness theorem for first order logic is a bit more complicated than that for propositional logic. We will unfortunately not present the proof. However, Theorem 26 has an important consequence for the entscheidungsproblem.

Corollary 27. *The set of valid formulas is recursively enumerable.*

Proof. We will present a nondeterministic algorithm that given an input formula φ will accept if and only if φ is valid. The nondeterministic algorithm will guess a proof (as sequence of formulas) along with which rule of the proof system was used. It will then check if the proof is valid and if so, it will accept the input string φ; other wise it will reject it. The important observation is that given a line and whether it is an axiom or which previous lines of the proof it follows from, it is easy to check if that is indeed the case. The correctness of this algorithm follows from the completeness theorem. □

Gödel's result establishes that the set of valid first order logic formula is in RE. Is it decidable? In other words, is the complement of the validity problem also recursively enumerable. This is equivalent to asking if the satisfiability problem is recursively enumerable — checking if φ is *not* valid is equivalent ot checking if $\neg\varphi$ is satisfiable! It turns out that this isn't true. We will establish this in the next section.

4 The Church-Turing Theorem

In this section, we will prove the following result.

Theorem 28 (Church-Turing 1936)**.** *Given a first order logic formula φ, the problem of determining if φ is valid is RE-complete.*

Before proving Theorem 28, let us examine the consequences of this observation. Observe that, we had previously proved that if A is RE-hard then A is not decidable. Thus, we can conclude that the validity problem for first order logic is undecidable. Because validity is undecidable but recursively enumerable, it means that the complement of the validity problem (namely, to determine if a given formula φ is not valid) is not recursively enumerable. Determining the non-validity of φ is the same as determining the satisfiability of $\neg\varphi$. Thus, the satisfiability problem for first order logic is not recursively enumerable.

Corollary 29. *The validity problem for first order logic is not decidable. The satisfiabiity problem for first order logic is not recursively enumerable.*

Proof of Theorem 28. Let $\mathsf{Valid} = \{\langle\varphi\rangle \mid \varphi \text{ is valid}\}$. Membership in RE follows from Corollary 27. So all we have to establish is RE-hardness.

Consider the language MP and the universal Turing machine U recognizing MP. Without loss of generality we can assume that U has one worktape (and an input tape); we can ignore the output tape since we are not computing a function. We can also assume that the input alphabet is $\Sigma = \{0, 1\}$ and the tape alphabet is $\Gamma = \{0, 1, \triangleright, \sqcup\}$. Let the set of states of U to be Q with initial state q_0 and accepting state q_{acc}, and let δ be its transition function. A configuration of U can be represented as (q, u, v, w, x) with $u, v, w, x \in \Gamma^*$ where

- q is the control state of the configuration,
- the input tape is $u^R v \sqcup^\omega$ with the input head scanning the leftmost symbol of v, and
- the worktape is $w^R x \sqcup^\omega$ with the worktape head scanning the leftmost symbol of x.

In the above, u^R and w^R are the strings obtained by *reversing* the strings u and w, respectively. Given configurations $\mathrm{C}_1 = (q_1, u_1, v_1, w_1, x_1)$ and $\mathrm{C}_2 = (q_2, u_2, v_2, w_2, x_2)$, we will say $\mathrm{C}_1 \longmapsto \mathrm{C}_2$ to indicate that U can move from configuration C_1 to C_2 in one step.

We will now prove that $MP \leq_m \mathsf{Valid}$; the RE-hardness of MP will enable us to conclude the RE-hardness of Valid. Consider the signature $\tau = (\varepsilon, \{q\}_{q\in Q} 0, 1, \sqcup, \triangleright, c)$, where ε and $\{q\}_{q\in Q}$ are constant symbols, $0, 1, \sqcup, \triangleright$ are binary relation symbols, and c is a 5-ary relation symbol. We will construct a sentence $F(\alpha)$ over signature τ such that U accepts α if and only if $F(\alpha)$ is valid. Since U recognizes MP, we have $\alpha \in MP$ iff $F(\alpha)$ is valid. The function F will be computable, and therefore it is a reduction from MP to Valid establishing the RE-hardness of Valid.

Before explaining the construction, it will be useful to give some intuition behind how we "view" the symbols in τ. ε is the constant that corresponds to the empty string, q is the constant that corresponds to the state q of U, and the binary relation symbols $a \in \{0, 1, \triangleright, \sqcup\}$ will be thought of as follows — if $a(x, y)$ holds then y is a followed by x. Finally $c(q, u, v, w, x)$ indicates that the configuration (q, u, v, w, x) is reached at some point during the computation of U on α. This overall intuition is useful to keep in mind as we describe the construction of $F(\alpha)$. However, it is important to note that an individual τ-structure may interpret these symbols in any way.

The sentence $F(\alpha) = (\varphi_{\mathsf{init}} \wedge \varphi_{\mathsf{comp}}) \to \varphi_{\mathsf{acc}}$ where

- φ_{init} states that the initial configuration is reachable in the computation,
- φ_{comp} states that if configuration C_1 is reachable and $\mathrm{C}_1 \longmapsto \mathrm{C}_2$ then C_2 is reached in the computation, and
- φ_{acc} says that an accepting configuration is reached during the computation.

Let us see how these individual formulas are constructed.

To describe the construction of φ_{init} is useful to introduce the following helper formula. For a string $u \in \Gamma^*$, we define a formula $\mathsf{str}_u(x)$ with free variable x that says that "x is the string u". This can be defined inductively as follows.

$$\mathsf{str}_u(x) = \begin{cases} x = \varepsilon & \text{if } u = \varepsilon \\ \exists y.\ a(y,x) \wedge \mathsf{str}_v(y) & \text{if } u = av \end{cases}$$

Using $\mathsf{str}_u(x)$, we can define φ_{init} as

$$\exists v, x.\ c(q_0, \varepsilon, v, \varepsilon, x) \wedge \mathsf{str}_{\triangleright\alpha}(v) \wedge \mathsf{str}_{\triangleright}(x)$$

Essentially it is saying that the configuration $(q_0, \varepsilon, \triangleright\alpha, emptystr, \triangleright)$, which is the initial configuration on input α, is reached in the computation (which is the predicate c).

In order to describe the formula φ_{comp}, we to capture the effect of changes to the tape and head movement. To do this easily we need another helper formula. Recall that in our representation of our configuration, a pair of strings (u, v) represent a tape that holds $u^R v \sqcup^\omega$ with the head reading the leftmost symbol of v. We will define a formula $\mathsf{move}_{a,b,d}(u_1, v_1, u_2, v_2)$ with free variables u_1, v_1, u_2, v_2 that captures the fact that in "tape (u_1, v_1) symbol a is read, and the effect of writing b and moving the head in direction d is the tape (u_2, v_2)". There one technical issue we need to deal with. Observe that, in a tape (u, v), we are reading $\sqcup$ when either v is a string that begins with $\sqcup$ or when $v = \varepsilon$. To handle this uniformly, let us introduce a new predicate $\mathsf{hd}_a(x, y)$ defined as

$$\mathsf{hd}_a(x,y) = \begin{cases} a(x,y) & \text{if } a \neq \sqcup \\ a(x,y) \vee (x = \varepsilon \wedge y = \varepsilon) & \text{if } a = \sqcup \end{cases}$$

Let us first define moving right. If we move right when we read a and write b, we go from tape (u, av) to (bu, v); we encode this in our logic.

$$\mathsf{move}_{a,b,+1}(u_1, v_1, u_2, v_2) = \mathsf{hd}_a(v_2, v_1) \wedge b(u_1, u_2).$$

Notice the use of b as opposed to hd_b because after the step u_2 will be a string starting with b, even if b is $\sqcup$. Let us now consider moving left. Moving left when reading a and writing b, from tape (cu, v) we get the tape (u, cbv).

$$\mathsf{move}_{a,b,-1}(u_1, v_1, u_2, v_2) = \exists x \exists y \bigvee_{e \in \{0,1,\sqcup,\triangleright\}} (e(u_2, u_1) \wedge \mathsf{hd}_a(x, v_1) \wedge \mathsf{hd}_b(x, y) \wedge e(y, v_2))$$

Notice again the use of $e()$ and the use of $\mathsf{hd}_a()$ and $\mathsf{hd}_b()$. Our assumption on Turing machines is that you never move left of the leftmost cell because of the left endmarker; therefore u_1 cannot be ε when a move left step is executed. Next, it is possible that $v_1 = \varepsilon$ and the symbol b being written is $\sqcup$ and therefore v_2 is just the string e.

On the input tape, we never write anything and its moves can be described as if we read and write the same symbol, using move. We can define φ_{comp} as follows.

$$\begin{aligned} \varphi_{\mathsf{comp}} = \quad & \forall u_1 \forall v_1 \forall w_1 \forall x_1 \forall u_2 \forall v_2 \forall w_2 \forall x_2 \bigwedge_{(p,i,a,q,b,d_1,d_2):\delta(p,i,a)=(q,d_1,b,d_2)} \\ & ((c(p, u_1, v_1, w_1, x_1) \wedge \mathsf{move}_{i,i,d_1}(u_1, v_1, u_2, v_2) \wedge \mathsf{move}_{a,b,d_2}(w_1, x_1, w_2, x_2)) \rightarrow c(q, u_2, v_2, w_2, x_2)) \end{aligned}$$

φ_{comp} effectively says that if (p, u_1, v_1, w_1, x_1) is reached and a transition is enabled such that $(p, u_1, v_1, w_1, x_1) \longmapsto (q, u_2, v_2, w_2, x_2)$ then (q, u_2, v_2, w_2, x_2) is also reached.

Finally, we need to describe φ_{acc} which says that an accepting configuration is reached. This is simply

$$\varphi_{\mathsf{acc}} = \exists u \exists v \exists w \exists x.\ c(q_{\mathsf{acc}}, u, v, w, x).$$

Clearly the sentence $F(\alpha)$ can be computed by a Turing machine. To complete the proof we need to argue that $F(\alpha)$ is valid iff α is accepted by U. Let us assume that $F(\alpha)$ is valid. That means $F(\alpha)$ holds in all structures. We will construct a specific structure in which the fact that $F(\alpha)$ holds implies that U has an accepting computation on α. Consider the structure Comp defined as follows.

- The universe will be the set of all strings over Γ.
- ε will be interpreted as the empty string.
- Each q will interpreted by some unique element of the universe.
- $a(x, y)$ will hold if the string $y = ax$ for any $a \in \Gamma$.
- $c(q, u, v, w, x)$ will hold if the computation of U on input α reaches configuration (q, u, v, w, x) at some point.

Now in the structure Comp, since the initial configuration $(q_0, \varepsilon, \triangleright\alpha, \varepsilon, \triangleright)$ is reachable, $c(q_0, \varepsilon, \triangleright\alpha, \varepsilon, \triangleright)$ holds and therefore so does φ_{init}. Next, our interpretation of the predicate c in Comp ensures that φ_{comp} holds. Since $\mathsf{Comp} \models F(\alpha)$, it must be that $\mathsf{Comp} \models \varphi_{\mathsf{acc}}$. That means for some strings u, v, w, x, the predicate $c(q_{\mathsf{acc}}, u, v, w, x)$ holds in Comp. But that means $(q_{\mathsf{acc}}, u, v, w, x)$ is reached in the computation of U on α. In other words, α is accepted by U.

Let us now prove the converse. Suppose U accepts α. Let

$$(q_0, \varepsilon, \triangleright\alpha, \varepsilon, \triangleright) \longmapsto \cdots \longmapsto (q_i, u_i, v_i, w_i, x_i) \longmapsto \cdots \longmapsto (q_n, u_n, v_n, w_n, x_n)$$

be the accepting computation of U on α, where $q_n = q_{\mathsf{acc}}$. We need to show that $F(\alpha)$ is valid, i.e., holds in every τ-structure. Consider an arbitrary τ-structure $\mathcal{A}$. If $\mathcal{A} \not\models \varphi_{\mathsf{init}} \wedge \varphi_{\mathsf{comp}}$ then $\mathcal{A} \models F(\alpha)$. Therefore, the interesting case is when $\mathcal{A} \models \varphi_{\mathsf{init}} \wedge \varphi_{\mathsf{comp}}$. This case we will show (by induction) that if variables u, v, w, x "represent" the strings u_i, v_i, w_i, x_i, respectively, then $c(q_i, u, v, w, x)$ holds. That is

$$(\mathsf{str}_{u_i}(u) \wedge \mathsf{str}_{v_i}(v) \wedge \mathsf{str}_{w_i}(w) \wedge \mathsf{str}_{x_i}(x)) \rightarrow c(q_i, u, v, w, x)$$

holds in the structure $\mathcal{A}$. If we establish this result, then it follows that φ_{acc} holds.

The base case of the induction holds, because $\mathcal{A}$ satisfies φ_{init}. For the induction step, let us assume that the result holds for i. Consider any pair configurations $(q_1', u_1', v_1', w_1', x_1') \longmapsto (q_2', u_2', v_2', w_2', x_2')$ according to transition $\delta(q_1', i, a) = (q_2', d_1, b, d_2)$. Then one can prove that

$$\begin{aligned}(\mathsf{str}_{u_1'}(u) \wedge \mathsf{str}_{v_1'}(v) \wedge \mathsf{str}_{w_1'}(w) \wedge \mathsf{str}_{x_1'}(x) \wedge \mathsf{str}_{u_2'}(u') \wedge \mathsf{str}_{v_2'}(v') \wedge \mathsf{str}_{w_2'}(w') \wedge \mathsf{str}_{x_2'}(x')) \\ \rightarrow (\mathsf{move}_{i,i,d_1}(u, v, u', v') \wedge \mathsf{move}_{a,b,d_2}(w, x, w', x'))\end{aligned}$$

The induction step then follows from the fact that φ_{comp} holds. □