

Some Turing-Complete Extensions of First-Order Logic

Antti Kuusisto*

Institute of Computer Science
University of Wrocław
Poland

antti.j.kuusisto@uta.fi

We introduce a natural Turing-complete extension of first-order logic FO. The extension adds two novel features to FO. The first one of these is the capacity to *add new points* to models and *new tuples* to relations. The second one is the possibility of *recursive looping* when a formula is evaluated using a semantic game. We first define a game-theoretic semantics for the logic and then prove that the expressive power of the logic corresponds in a canonical way to the recognition capacity of Turing machines. Finally, we show how to incorporate generalized quantifiers into the logic and argue for a highly natural connection between oracles and generalized quantifiers.

1 Introduction

We introduce a natural Turing-complete extension of first-order logic FO. This extension essentially adds two features to basic FO. The first one of these is the capacity to *add new points* to models and *new tuples* to relations. The second one is the possibility of *looping* when a formula is evaluated using a semantic game.

Logics with different kinds of recursive looping capacities have been widely studied in the context of finite model theory [16]. Typically such logics are fragments of second-order predicate logic. A crucial weakness in the expressivity of k -th order predicate logic is that only a finite amount of information can be encoded by a finite number of quantified relations over a finite domain. Intuitively, there is *no infinitely expandable memory* available. Thus k -th order logic is not Turing-complete. To overcome this limitation, we add to first-order logic operators that enable the addition of new elements to the domains of models and new tuples to relations. Coupling this feature with the possibility of recursive looping leads to a very natural Turing-complete extension of first-order logic.

In addition to operators that enable the extension of domains and relations, we also consider an operator that enables the deletion of tuples of relations. It would be natural to also include to our framework an operator that enables the deletion of domain points. This indeed could (and perhaps should) be done, but for purely technical reasons, we ignore this possibility.

We provide a game-theoretic semantics for the novel logic. A nice feature of the logic—let us simply call it \mathcal{L} , or *logic* \mathcal{L} —is that it *simulates halting as well as diverging* computations of Turing machines in a very natural way. This behavioural correspondence between Turing machines and the logic \mathcal{L} stems from the appropriate use of game-theoretic concepts. Let us have a closer look at this matter.

Let \mathfrak{A} be a model and φ a formula of first-order logic. Let f be an *assignment function* that interprets the free variables of φ in the domain A of \mathfrak{A} . The semantic game $G(\mathfrak{A}, f, \varphi)$ is played between the two players \exists and \forall in the usual way (see, e.g., [11, 18]). If the verifying player \exists has a winning strategy

*This work was carried out during a tenure of the ERCIM “Alain Bensoussan” Fellowship Programme. The reported research has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 246016; the author has been employed as Marie Curie Fellow during the research period.

in the game $G(\mathfrak{A}, f, \varphi)$, we write $\mathfrak{A}, f \models^+ \varphi$ and say that φ is *true* in (\mathfrak{A}, f) . If, on the other hand, the falsifying player \forall has a winning strategy, we write $\mathfrak{A}, f \models^- \varphi$ and say that φ is *false* in (\mathfrak{A}, f) . Since φ is a first-order formula, we have $\mathfrak{A}, f \models^+ \varphi$ iff it is not the case that $\mathfrak{A}, f \models^- \varphi$.

If φ is a formula of *IF logic* [12] or *dependence logic* [21], for example, the situation changes. It is then possible that *neither player has a winning strategy* in the semantic game. This results in a third truth value (*indeterminate*). Turing machines of course exhibit analogous behaviour: on an input word w , a Turing machine can halt in an accepting state, halt in a rejecting state, or diverge.

The logic \mathcal{L} incorporates each of these three options in its semantics in a canonical way. For each Turing machine TM, there exists a sentence of φ_{TM} such that TM *accepts* the encoding of a *finite* model \mathfrak{A} iff $\mathfrak{A}, f \models^+ \varphi_{\text{TM}}$, and furthermore, TM *rejects* the encoding of \mathfrak{A} iff $\mathfrak{A}, f \models^- \varphi_{\text{TM}}$. Therefore TM *diverges* on the encoding of \mathfrak{A} iff neither the verifying nor the falsifying player has a winning strategy in the game involving \mathfrak{A} , f and φ . For the converse, for each formula χ of the logic \mathcal{L} , there exists a Turing machine TM_χ such that a similar full symmetry exists between semantic games involving χ and computations involving TM_χ . By Turing-completeness of a logic we mean exactly this kind of a behavioral equivalence between Turing machines and formulae.

The moves in the semantic games for \mathcal{L} are exactly as in first-order logic in positions involving the first-order operators $\exists x, \vee, \neg$. In positions of the type $Ix\varphi$, a fresh point is inserted into the domain of the model investigated, and the variable x is interpreted to refer to the fresh point. There are similar operators for the insertion (deletion) of tuples into (from) relations. The recursive looping is facilitated by operators such as the ones in the formula $1(P(x) \vee 1)$, where the player ending up in a position involving the novel atomic formula 1 can *jump back* into a position involving $1(P(x) \vee 1)$. Semantic games are played for at most omega rounds¹ and can be won only by moving to a position involving a first-order atomic formula. Winning and losing in positions involving first-order atoms is determined exactly as in first-order logic.

Operators that bear a resemblance to the ones used in the logic \mathcal{L} have of course been considered in logical contexts before. Lauri Hella has suggested (personal communication) extending first-order logic with recursive looping constructors that resemble those investigated in this article. His idea involves studying fixed point logics using a game-theoretic semantics with somewhat different kinds of winning conditions than the ones we shall formulate below. The framework does not involve modifying the domains of structures. The insertion (deletion) of tuples to (from) relations is an important ingredient in dynamic complexity (see, e.g., [8, 19]), although motivated and used there in a way that is quite different from the approach in this article. Logics that involve jumping into different model domains include for example the *sort logic* of Väänänen ([20],[22]), a logic which can in a sense be regarded as the strongest possible model theoretic logic. Other systems with similar constructors or motivations to those considered in this article include for example BGS logic [2], WHILE languages [1] and abstract state machines [10, 3]. See also the articles [5] and [14].

The reason we believe that the logic \mathcal{L} is particularly interesting lies in its simplicity and *exact behavioural correspondence* with Turing machines on one hand, and in the fact that it provides a *canonical unified perspective* on logic and computation on the other hand. The logic \mathcal{L} canonically extends ordinary first-order logic to a Turing-complete framework, and thereby serves not only as a novel logic, but also as a novel model of computation. It is also worth noting that the fresh operators of \mathcal{L} nicely capture two classes of constructors that are omnipresent in the practise of mathematics: scenarios where fresh points are added to investigated constructions (or fresh lines are drawn, etc.) play a central role in geometry, and recursive looping operators are found everywhere in mathematical practise, often indicated

¹Omega of course refers to the smallest infinite ordinal.

with the help of the famous three dots (...).

The structure of the paper is as follows. In Section 2 we define some preliminary notions and give a formal account of the syntax \mathcal{L} . In Section 3 we develop the semantics of \mathcal{L} . In Section 4 we establish the Turing-completeness of \mathcal{L} in restriction to the class of word models. In Section 5 we use the results of Section 4 in order to establish Turing-completeness of \mathcal{L} in the class of all finite models. In Section 6 we show how to extend \mathcal{L} with generalized quantifiers. We also briefly discuss the conceptual link between oracles and generalized quantifiers.

2 Preliminaries

Let \mathbb{Z}_+ denote the set of positive integers. Let $\text{VAR} := \{v_i \mid i \in \mathbb{Z}_+\}$ be the set of variable symbols used in first-order logic. We mainly use *metavariables* x, y, z, x_i, y_i, z_i , etc., in order to refer to the variables in VAR . Let $k \in \mathbb{Z}_+$. We let $\text{VAR}_{\text{SO}}(k)$ be a countably infinite set of k -ary relation variables. We let $\text{VAR}_{\text{SO}} = \bigcup_{k \in \mathbb{Z}_+} \text{VAR}_{\text{SO}}(k)$.

Let τ denote a complete relational vocabulary, i.e., τ is the union $\bigcup_{k \in \mathbb{Z}_+} \tau_k$, where τ_k is a countably infinite set of k -ary relation symbols. Let $\sigma \subseteq \tau$. Define the language $\mathcal{L}^*(\sigma)$ to be the smallest set S such that the following conditions are satisfied.

1. If x_1, \dots, x_k are variable symbols and $R \in \sigma$ a k -ary relation symbol, then $R(x_1, \dots, x_k) \in S$.
2. If x_1, \dots, x_k are variable symbols and $X \in \text{VAR}_{\text{SO}}(k)$ a k -ary relation variable, then $X(x_1, \dots, x_k) \in S$.
3. If x, y are variable symbols, then $x = y \in S$.
4. If $k \in \mathbb{N}$ is (a symbol representing) a natural number, then $k \in S$.
5. If $\varphi \in S$, then $\neg\varphi \in S$.
6. If $\varphi, \psi \in S$, then $(\varphi \wedge \psi) \in S$.
7. If x is a variable symbol and $\varphi \in S$, then $\exists x \varphi \in S$.
8. If x is a variable symbol and $\varphi \in S$, then $Ix \varphi \in S$.
9. If x_1, \dots, x_k are variable symbols, $R \in \sigma$ a k -ary relation symbol and $\varphi \in S$, then $I_{R x_1, \dots, x_k} \varphi \in S$.
10. If x_1, \dots, x_k are variable symbols, $X \in \text{VAR}_{\text{SO}}$ a k -ary relation variable symbol and $\varphi \in S$, then $I_{X x_1, \dots, x_k} \varphi \in S$.
11. If x_1, \dots, x_k are variable symbols, $R \in \sigma$ a k -ary relation symbol and $\varphi \in S$, then $D_{R x_1, \dots, x_k} \varphi \in S$.
12. If x_1, \dots, x_k are variable symbols, $X \in \text{VAR}_{\text{SO}}$ a k -ary relation variable symbol and $\varphi \in S$, then $D_{X x_1, \dots, x_k} \varphi \in S$.
13. If $\varphi \in S$ and $k \in \mathbb{N}$, then $k\varphi \in S$.

While we could develop a sensible semantics for the language $\mathcal{L}^*(\sigma)$, we shall only consider a sublanguage $\mathcal{L}(\sigma) \subseteq \mathcal{L}^*(\sigma)$ that avoids certain undesirable situations in semantic games. Let $\varphi \in \mathcal{L}^*(\sigma)$ be a formula. Assume that φ contains an atomic subformula $k \in \mathbb{N}$ and another subformula $k\psi$. Assume that k is *not* a subformula of $k\psi$. Then we say that φ has a *non-standard jump*. Note that we define that *every instance* of the syntactically same subformula of φ is a *distinct* subformula: for example, the formula $(P(x) \wedge P(x))$ is considered to have *three* subformulae, these being the left and right instances of $P(x)$ and the formula $(P(x) \wedge P(x))$ itself. Thus for example the formula $(k(P(x) \wedge k) \wedge k(P(x) \wedge k))$ has a non-standard jump. We define $\mathcal{L}(\sigma)$ to be the largest subset of $\mathcal{L}^*(\sigma)$ that does not contain formulae with non-standard jumps.

The reason we wish to avoid non-standard jumps is simple and will become entirely clear when we define the semantics of $\mathcal{L}(\sigma)$ in Section 3. Let us consider an example that demonstrates the undesirable situation. Consider the formula $(k \wedge \exists x k P(x))$ of $\mathcal{L}^*(\sigma)$. As will become clear in Section 3, it is possible to end up in the related semantic game in a position involving the atomic formula $P(x)$ *without* first visiting a position involving the formula $\exists x k P(x)$. This is undesirable, since a related *variable assignment function* will then not necessarily give any value to the variable x . For this reason we limit attention to the fragment $\mathcal{L}(\sigma)$ containing only formulae without non-standard jumps.

Before defining the semantics of the language $\mathcal{L}(\sigma)$, we make a number of auxiliary definitions. Let \mathfrak{A} , \mathfrak{B} , etc., be models. We let A , B , etc., denote the domains of the models in the usual way. A function f that interprets a finite subset of $\text{VAR} \cup \text{VAR}_{\text{SO}}$ in the domain of a model \mathfrak{A} is called an *assignment*. Naturally, if $X \in \text{VAR}_{\text{SO}} \cap \text{Dom}(f)$ is a k -ary relation variable, then $f(X) \subseteq A^k$, and if $x \in \text{VAR} \cap \text{Dom}(f)$, then $f(x) \in A$. We let $f[x \mapsto a]$ denote the valuation with the domain $\text{Dom}(f) \cup \{x\}$ defined such that $f[x \mapsto a](x) = a$ and $f[x \mapsto a](y) = f(y)$ if $y \neq x$. We analogously define $f[X \mapsto S]$, where $X \in \text{VAR}_{\text{SO}}$ is a k -ary relation variable and $S \subseteq A^k$. We will also construct valuations of, say, the type $f[x \mapsto a, y \mapsto b, X \mapsto S]$. The interpretation of these constructions is clear.

We define the set of free variables $\text{free}(\varphi)$ of a formula $\varphi \in \mathcal{L}(\sigma)$ as follows.

1. If $R \in \sigma$, then $\text{free}(R(x_1, \dots, x_k)) = \{x_1, \dots, x_k\}$.
2. If $X \in \text{VAR}_{\text{SO}}(k)$, then $\text{free}(X(x_1, \dots, x_k)) = \{X\} \cup \{x_1, \dots, x_k\}$.
3. $\text{free}(x = y) = \{x, y\}$.
4. $\text{free}(k) = \emptyset$.
5. $\text{free}(\neg\varphi) = \text{free}(\varphi)$.
6. $\text{free}((\varphi \wedge \psi)) = \text{free}(\varphi) \cup \text{free}(\psi)$.
7. $\text{free}(\exists x\varphi) = \text{free}(\varphi) \setminus \{x\}$.
8. $\text{free}(Ix\varphi) = \text{free}(\varphi) \setminus \{x\}$.
9. $\text{free}(I_{R_{x_1, \dots, x_k}}\varphi) = \text{free}(\varphi) \setminus \{x_1, \dots, x_k\}$.
10. $\text{free}(I_{X_{x_1, \dots, x_k}}\varphi) = \text{free}(\varphi) \setminus \{X, x_1, \dots, x_k\}$.
11. $\text{free}(D_{R_{x_1, \dots, x_k}}\varphi) = \text{free}(\varphi) \setminus \{x_1, \dots, x_k\}$.
12. $\text{free}(D_{X_{x_1, \dots, x_k}}\varphi) = \text{free}(\varphi) \setminus \{X, x_1, \dots, x_k\}$.
13. $\text{free}(k\varphi) = \text{free}(\varphi)$.

A formula φ of $\mathcal{L}(\sigma)$ is a *sentence* if $\text{free}(\varphi) = \emptyset$.

3 A Semantics for $\mathcal{L}(\sigma)$

In this section we define a game-theoretic semantics for the language $\mathcal{L}(\sigma)$. The semantics extends the well-known game-theoretic semantics of first-order logic (see, e.g., [18]). The semantic games are played by two players \exists and \forall .

Let φ be a formula of $\mathcal{L}(\sigma)$. Let \mathfrak{A} be a σ -model, and let f be an assignment that interprets the free variables of φ in A . Let $\# \in \{+, -\}$ be simply a symbol. The quadruple $(\mathfrak{A}, f, \#, \varphi)$ defines a semantic game $G(\mathfrak{A}, f, \#, \varphi)$. The set of *positions* in the game $G(\mathfrak{A}, f, \#, \varphi)$ is the smallest set S such that the following conditions hold.

1. $(\mathfrak{A}, f, \#, \varphi) \in S$.
2. If $(\mathfrak{B}, g, \#, \neg\psi) \in S$, then $(\mathfrak{B}, g, \#', \psi) \in S$, where $\#'' \in \{+, -\} \setminus \{\#\}$.
3. If $(\mathfrak{B}, g, \#, (\psi \wedge \psi')) \in S$, then $(\mathfrak{B}, g, \#, \psi) \in S$ and $(\mathfrak{B}, g, \#, \psi') \in S$.
4. If $(\mathfrak{B}, g, \#, \exists x\psi) \in S$ and $a \in B$, then $(\mathfrak{B}, g[x \mapsto a], \#, \psi) \in S$.
5. If $(\mathfrak{B}, g, \#, Ix\psi) \in S$ and $b \notin B$ is a fresh element², then $(\mathfrak{B} \cup \{b\}, g[x \mapsto b], \#, \psi) \in S$; we define $\mathfrak{B} \cup \{b\}$ to be the σ -model \mathfrak{C} where b is simply a fresh isolated point, i.e., the domain of \mathfrak{C} is $B \cup \{b\}$, and $R^{\mathfrak{C}} = R^{\mathfrak{B}}$ for each $R \in \sigma$.
6. If $(\mathfrak{B}, g, \#, I_{Rx_1, \dots, x_k} \psi) \in S$ and $b_1, \dots, b_k \in B$, then $(\mathfrak{B}^*, g^*, \#, \psi) \in S$, where \mathfrak{B}^* is obtained from \mathfrak{B} by defining $R^{\mathfrak{B}^*} := R^{\mathfrak{B}} \cup \{(b_1, \dots, b_k)\}$, and $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]$. For each relation symbol $P \in \sigma \setminus \{R\}$, we have $P^{\mathfrak{B}^*} := P^{\mathfrak{B}}$. The models \mathfrak{B} and \mathfrak{B}^* have the same domain.
7. Assume $(\mathfrak{B}, g, \#, I_{Xx_1, \dots, x_k} \psi) \in S$ and $b_1, \dots, b_k \in B$. If $X \in \text{Dom}(g)$, call $C := g(X)$. Otherwise let $C := \emptyset$. Then $(\mathfrak{B}, g^*, \#, \psi) \in S$, where $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k, X \mapsto (C \cup \{(b_1, \dots, b_k)\})]$.
8. If $(\mathfrak{B}, g, \#, D_{Rx_1, \dots, x_k} \psi) \in S$ and $b_1, \dots, b_k \in B$, then $(\mathfrak{B}^*, g^*, \#, \psi) \in S$, where \mathfrak{B}^* is obtained from \mathfrak{B} by defining $R^{\mathfrak{B}^*} := R^{\mathfrak{B}} \setminus \{(b_1, \dots, b_k)\}$, and $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]$. For each relation symbol $P \in \sigma \setminus \{R\}$, we have $P^{\mathfrak{B}^*} := P^{\mathfrak{B}}$. The models \mathfrak{B} and \mathfrak{B}^* have the same domain.
9. Assume $(\mathfrak{B}, g, \#, D_{Xx_1, \dots, x_k} \psi) \in S$ and $b_1, \dots, b_k \in B$. If $X \in \text{Dom}(g)$, call $C := g(X)$. If $X \notin \text{Dom}(g)$, define $C := \emptyset$. Then $(\mathfrak{B}, g^*, \#, \psi) \in S$, where $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k, X \mapsto (C \setminus \{(b_1, \dots, b_k)\})]$.
10. If $(\mathfrak{B}, g, \#, k\psi) \in S$, then $(\mathfrak{B}, g, \#, \psi) \in S$.

The game $G(\mathfrak{A}, f, \#, \varphi)$ is played as follows.

1. Every *play* of the game begins from the position $(\mathfrak{A}, f, \#, \varphi)$.
2. If a position $(\mathfrak{B}, g, \#, \neg\psi)$ is reached in a play of the game, the play continues from the position $(\mathfrak{B}, g, \#', \psi)$, where $\#'' \in \{+, -\} \setminus \{\#\}$.
3. If a position $(\mathfrak{B}, g, \#, (\psi \wedge \psi'))$ is reached, then the play continues as follows. If $\#' = +$ (respectively, $\#' = -$), then the player \forall (respectively, \exists) picks a formula $\chi \in \{\psi, \psi'\}$, and the play continues from the position $(\mathfrak{B}, g, \#, \chi)$.
4. If a position $(\mathfrak{B}, g, \#, \exists x\psi)$ is reached, then the play continues as follows. If $\#' = +$ (respectively, $\#' = -$), then the player \exists (respectively, \forall) picks an element $b \in B$, and the play continues from the position $(\mathfrak{B}, g[x \mapsto b], \#, \psi)$.
5. If a position $(\mathfrak{B}, g, \#, Ix\psi)$ is reached, then the play continues from the position $(\mathfrak{B} \cup \{b\}, g[x \mapsto b], \#, \psi)$, where $\mathfrak{B} \cup \{b\}$ is the σ -model \mathfrak{C} , where b is simply a fresh isolated point³, i.e., the domain of \mathfrak{C} is $B \cup \{b\}$, and $R^{\mathfrak{C}} = R^{\mathfrak{B}}$ for each $R \in \sigma$.
6. Assume a position $(\mathfrak{B}, g, \#, I_{Rx_1, \dots, x_k} \psi)$ has been reached. The play of the game continues as follows. If $\#' = +$ (respectively, $\#' = -$), then the player \exists (respectively, \forall) chooses a tuple $(b_1, \dots, b_k) \in B^k$. The play of the game continues from the position $(\mathfrak{B}^*, g^*, \#, \psi)$, where \mathfrak{B}^* is obtained from \mathfrak{B} by redefining $R^{\mathfrak{B}^*} := R^{\mathfrak{B}} \cup \{(b_1, \dots, b_k)\}$, and $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]$. Other relations and the domain remain unaltered.

²To avoid introducing a proper class of new positions here, we assume $b = B$. Since $B \notin B$, the element $b = B$ is a fresh element. Only a single new position is generated.

³Recall that we let $b := B$ in order to avoid proper classes of new positions.

7. Assume a position $(\mathfrak{B}, g, \#, I_{X_{x_1, \dots, x_k}} \psi)$ has been reached. The play of the game continues as follows. If $\# = +$ (respectively, $\# = -$), then the player \exists (respectively, \forall) chooses a tuple $(b_1, \dots, b_k) \in B^k$. The play of the game continues from the position $(\mathfrak{B}, g^*, \#, \psi)$, where $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k, X \mapsto (C \cup \{(b_1, \dots, b_k)\})]$; here $C = g(X)$ if $X \in \text{Dom}(g)$, and otherwise $C = \emptyset$.
8. Assume a position $(\mathfrak{B}, g, \#, D_{R_{x_1, \dots, x_k}} \psi)$ has been reached. The play of the game continues as follows. If $\# = +$ (respectively, $\# = -$), then the player \exists (respectively, \forall) chooses a tuple $(b_1, \dots, b_k) \in B^k$. The play of the game continues from the position $(\mathfrak{B}^*, g^*, \#, \psi)$, where \mathfrak{B}^* is obtained from \mathfrak{B} by redefining $R^{\mathfrak{B}^*} := R^{\mathfrak{B}} \setminus \{(b_1, \dots, b_k)\}$, and $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]$. Other relations and the domain remain unaltered.
9. Assume a position $(\mathfrak{B}, g, \#, D_{X_{x_1, \dots, x_k}} \psi)$ has been reached. The play of the game continues as follows. If $\# = +$ (respectively, $\# = -$), then the player \exists (respectively, \forall) chooses a tuple $(b_1, \dots, b_k) \in B^k$. If $X \in \text{Dom}(g)$, call $C := g(X)$. Otherwise define $C := \emptyset$. The play of the game continues from the position $(\mathfrak{B}, g^*, \#, \psi)$, where $g^* := g[x_1 \mapsto b_1, \dots, x_k \mapsto b_k, X \mapsto (C \setminus \{(b_1, \dots, b_k)\})]$.
10. If a position $(\mathfrak{B}, g, \#, k\psi)$ is reached, then the play of the game continues from the position $(\mathfrak{B}, g, \#, \psi)$.
11. If a position $(\mathfrak{B}, g, \#, k)$ is reached, then the play of the game continues as follows. If $\# = +$ (respectively, $\# = -$) and there exists a subformula $k\psi$ of the original formula φ , then the player \exists (respectively, \forall) chooses some subformula $k\chi$ of φ , and the play continues from the position $(\mathfrak{B}, g, \#, k\chi)$. If no subformula $k\psi$ exists, the play of the game ends.
12. If ψ is an atomic formula $R(x_1, \dots, x_k)$, $X(x_1, \dots, x_k)$ or $x = y$, and a position $(\mathfrak{B}, g, \#, \psi)$ is reached, then the play of the game ends.

A play of the game $G(\mathfrak{A}, f, \#, \varphi)$ is played up to omega rounds. If a play of the game continues for omega rounds, then neither of the two players wins the play. If a play of the game ends after a finite number of rounds, then one of the players wins the play. The winner is determined as follows.

1. If the play ends in a position $(\mathfrak{B}, g, \#, k)$, which may happen in the pathological case where there are no subformulae of φ of the type $k\psi$, then \exists wins if $\# = -$ and \forall wins if $\# = +$.
2. If the play ends in a position $(\mathfrak{B}, g, \#, \psi)$, where ψ is an atomic formula $R(x_1, \dots, x_k)$, $X(x_1, \dots, x_k)$ or $x = y$, then the winner of the play is determined as follows.
 - (a) Assume $\# = +$. Then \exists wins if $\mathfrak{B}, g \models \psi$. If $\mathfrak{B}, g \not\models \psi$, then \forall wins. Here \models is the semantic turnstile of ordinary first-order logic.
 - (b) Assume $\# = -$. Then \forall wins if $\mathfrak{B}, g \models \psi$. If $\mathfrak{B}, g \not\models \psi$, then \exists wins.

A *strategy* of \exists in the game $G(\mathfrak{A}, f, \#, \varphi)$ is simply a function that determines a unique choice for the player \exists in every position of the game $G(\mathfrak{A}, f, \#, \varphi)$ that requires \exists to make a choice. A strategy of \forall is defined analogously. A strategy of \exists (\forall) in the game $G(\mathfrak{A}, f, \#, \varphi)$ is a *winning strategy* if every play of the game where \exists (\forall) makes her moves according to the strategy, ends after a finite number of rounds in a position where \exists (\forall) wins.

We write $\mathfrak{A}, f \models^+ \varphi$ iff the player \exists has a winning strategy in the game $G(\mathfrak{A}, f, +, \varphi)$. We write $\mathfrak{A}, f \models^- \varphi$ iff \exists has a winning strategy in the game $G(\mathfrak{A}, f, -, \varphi)$. By duality of the rules of the game, it is easy to see that \exists has a winning strategy in $G(\mathfrak{A}, f, -, \varphi)$ iff \forall has a winning strategy in $G(\mathfrak{A}, f, +, \varphi)$. Similarly, \forall has a winning strategy in $G(\mathfrak{A}, f, +, \varphi)$ iff \exists has a winning strategy in $G(\mathfrak{A}, f, -, \varphi)$.

Let φ be a *sentence* of $\mathcal{L}(\sigma)$. We write $\mathfrak{A} \models^+ \varphi$ iff $\mathfrak{A}, \emptyset \models^+ \varphi$, where \emptyset denotes the empty valuation. Similarly, we write $\mathfrak{A} \models^- \varphi$ iff $\mathfrak{A}, \emptyset \models^- \varphi$.

4 Turing-Completeness

Let σ be a finite nonempty set of unary relation symbols. let $Succ$ be a binary relation symbol. A *word model* \mathfrak{A} over the vocabulary $\{Succ\} \cup \sigma$ is defined as follows.

1. The domain of \mathfrak{A} is a nonempty finite set.
2. The binary predicate $Succ$ is a successor relation over A , i.e., a binary relation corresponding to a linear order, but with maximum out-degree and in-degree equal to one.
3. Let $b \in A$ denote the smallest element with respect to $Succ$. We have $b \notin P^{\mathfrak{A}}$ for each $P \in \sigma$. (This is because we do not want to consider models with the empty domain; the empty word will correspond to the word model with exactly one element.) For each element $a \in A \setminus \{b\}$, there exists exactly one predicate $P \in \sigma$ such that $a \in P^{\mathfrak{A}}$.

Word models canonically encode finite words. For example the word $abbaa$ over the alphabet $\{a, b\}$ is encoded by the word model \mathfrak{M} over the vocabulary $\{Succ, P_a, P_b\}$ defined as follows.

1. $M = \{0, \dots, 5\}$.
2. $Succ^{\mathfrak{M}}$ is the canonical successor relation on M .
3. $P_a^{\mathfrak{M}} = \{1, 4, 5\}$ and $P_b^{\mathfrak{M}} = \{2, 3\}$.

If w is a finite word, we let $\mathcal{M}(w)$ denote its encoding by a word model in the way defined above. If W is a set of finite words, then $\mathcal{M}(W) = \{\mathcal{M}(w) \mid w \in W\}$. If Σ is a finite nonempty alphabet, we let $\mathcal{M}(\Sigma)$ denote the vocabulary $\{Succ\} \cup \{P_a \mid a \in \Sigma\}$.

We define computation of Turing machines in the standard way that involves a possible *tape alphabet* in addition to an *input alphabet*. These two alphabets are disjoint. Let Σ be a finite nonempty alphabet. Then Σ^* is the set of all inputs to a Turing machine TM whose input alphabet is Σ . During computation, TM may employ an additional finite set S of tape symbols. That set S is the tape alphabet of TM. There is a nice loose analogy between tape alphabet symbols of Turing machines and relation variable symbols in VAR_{SO} used in formulas of \mathcal{L} .

Theorem 4.1. *Let Σ be a finite nonempty alphabet. Let TM be a deterministic Turing machine with the input alphabet Σ . Then there exists a sentence $\varphi_{\text{TM}} \in \mathcal{L}(\mathcal{M}(\Sigma))$ such that the following conditions hold.*

1. *Let $W \subseteq \Sigma^*$ be the set of words w such that TM halts in an accepting state with the input w . Then for all $w \in \Sigma^*$, $\mathcal{M}(w) \models^+ \varphi_{\text{TM}}$ iff $w \in W$.*
2. *Let $U \subseteq \Sigma^*$ be the set of words u such that TM halts in a rejecting state with the input u . Then for all $w \in \Sigma^*$, $\mathcal{M}(w) \models^- \varphi_{\text{TM}}$ iff $w \in U$.*

Proof. We shall define a sentence φ_{TM} such that the semantic games involving φ_{TM} simulate the action of TM.

Let Q be the set of states of TM. For each $q \in Q$, reserve a variable symbol x_q . Furthermore, let y_{state} be a variable symbol. Intuitively, the equality $y_{\text{state}} = x_q$ will hold in the semantic game $G(\mathcal{M}(w), \emptyset, +, \varphi_{\text{TM}})$ exactly when TM is in the state q during a run with the input w .

Simulating the action of the head of the Turing machine TM is a bit more complicated, since when defining the new position of the head with a subformula of φ_{TM} , information concerning the old position must be somehow accessible.⁴ Fix two variables x_{head}^1 and x_{head}^2 . These variables will encode the position of the head. Define three further variables y_{head}^1 , y_{head}^2 , and y_{head} . The tape of TM will be encoded by

⁴Note that we assume, w.l.o.g., that TM has a single head.

the (dynamically extendible) successor relation $Succ$, which is a part of the model (or models, to be exact) constructed during the semantic game. The variables x_{head}^1 and x_{head}^2 will denote elements of the successor relation. Intuitively, $y_{head} = y_{head}^1$ will mean that x_{head}^1 indicates the current position of the head of TM, while $y_{head} = y_{head}^2$ will mean that x_{head}^2 indicates the position of the head of TM. The value of x_{head}^1 will always be easily definable based on the value of x_{head}^2 , and vice versa, the value of x_{head}^2 will be definable based on the value of x_{head}^1 .

If TM employs tape alphabet symbols $s \notin \Sigma$, these can be encoded by unary relation variables X_s . Intuitively, if u is an element of the domain of the model under investigation, then $X_s(u)$ will mean that the point of the tape of TM corresponding to u contains the symbol s . Similarly, for an input alphabet symbol $t \in \Sigma$, $P_t(u)$ will mean that the point of the tape of TM corresponding to u contains the symbol t .

The sentence ϕ_{TM} will contain subformulae which are *essentially* (but not exactly, as we shall see) of the type

$$(\Psi_{state} \wedge \Psi_{tape_position}) \rightarrow (\Psi_{new_state} \wedge \Psi_{new_tape_position} \wedge loop),$$

where $loop$ is simply the atomic formula 1, which indicates that the semantic game ought to be continued from some subformula 1 Ψ of ϕ_{TM} . The sentence ϕ_{TM} will also contain subformulae which are *essentially* of the type

$$(\Psi_{state} \wedge \Psi_{tape_position}) \rightarrow (\Psi_{new_final_state} \wedge \Psi_{new_tape_position} \wedge \top)$$

and

$$(\Psi_{state} \wedge \Psi_{tape_position}) \rightarrow (\Psi_{new_final_state} \wedge \Psi_{new_tape_position} \wedge \perp)$$

where in the first case the final state is an accepting state, and in the second case a rejecting state. Here \top (\perp) is the formula $\forall x x = x$ ($\neg \forall x x = x$).

Let $s, t \in \Sigma$ be input alphabet symbols of TM. Consider a transition instruction of TM of the type $T(q_i, s) = (q_j, t, right)$, which states that if the state is q_i and the symbol scanned is s , then write t to the current cell, change state to q_j , and move right. Let us call this instruction $instr$. The instruction $instr$ defines a formula ψ_{instr} . Assume q_j is *not* a final state. Let us see how ψ_{instr} is constructed.

Define the formula $\psi_{state}^{q_i} := y_{state} = x_{q_i}$. Define the formula ψ_{symbol}^s to be the conjunction of the following formulae.

1. $y_{head} = y_{head}^1 \rightarrow P_s(x_{head}^1)$,
2. $y_{head} = y_{head}^2 \rightarrow P_s(x_{head}^2)$.

Define χ'_1 to be the formula

$$D_{P_s, x} I_{P_t, y} \exists x_{head}^2 \exists y_{head} \exists y_{state} (x = x_{head}^1 \wedge y = x_{head}^1 \wedge y_{head} = y_{head}^2 \wedge y_{state} = x_{q_j} \wedge \chi' \wedge 1),$$

where χ' is a formula that forces x_{head}^2 to be interpreted as the successor of x_{head}^1 with respect to $Succ$. It is possible that no successor of x_{head}^1 exists in the current model. In that case a successor can be constructed by appropriately using the operators Iz and I_{Succuv} . To cover this case, define χ''_1 to be the formula

$$D_{P_s, x} I_{P_t, y} Iz I_{Succuv} \exists x_{head}^2 \exists y_{head} \exists y_{state} (x = x_{head}^1 \wedge y = x_{head}^1 \wedge y_{head} = y_{head}^2 \wedge y_{state} = x_{q_j} \wedge \chi' \wedge \chi'' \wedge 1),$$

where χ'' forces the fresh point z to be the successor of x_{head}^1 with respect to $Succ$, and χ' forces x_{head}^2 to be the successor of x_{head}^1 . Let α be a formula that states that x_{head}^1 has a successor with respect to $Succ$ in the current model. Define χ_1 to be the conjunction $(\alpha \rightarrow \chi'_1) \wedge (\neg \alpha \rightarrow \chi''_1)$.

The formula χ_1 simulates the instruction $instr$ when the current position of the head of TM is encoded by x_{head}^1 . The formula determines a new position for x_{head}^2 based on the current position of x_{head}^1 . A similar

formula χ_2 can be defined analogously to deal with the situation where the current position of the head is encoded by x_{head}^2 .

Define β to be the conjunction of the formulae

1. $y_{head} = y_{head}^1 \rightarrow \chi_1$,
2. $y_{head} = y_{head}^2 \rightarrow \chi_2$.

Define ψ_{instr} to be the formula $(\psi_{state}^{q_i} \wedge \psi_{symbol}^s) \rightarrow \beta$. Formulae $\psi_{instr'}$, where $instr'$ tells TM to move to a final state, are defined similarly, but do not have the atom 1. Instead, accepting states have the atom \top and rejecting states the atom \perp . We shall not explicitly discuss for example instructions where the head is to move left, since all possible instructions can be easily specified by formulae analogous to the ones above.

Recall that Q is the set of states of TM. Let q_1, \dots, q_n enumerate the elements of Q . Define

$$I\bar{x} := Iy_{head}^1 Iy_{head}^2 Ix_{q_1} \dots Ix_{q_n}.$$

Let \mathbb{I} be the set of instructions of TM. The sentence φ_{TM} is the formula

$$I\bar{x} \exists y_{head} \exists x_{head}^1 \exists x_{head}^2 \exists y_{state} \left(\psi_{initial} \wedge 1 \left(\bigwedge_{instr \in \mathbb{I}} \psi_{instr} \right) \right),$$

where $\psi_{initial}$ states that the following conditions hold.

1. y_{state} is equal to x_q , where q is the beginning state of TM.
2. y_{head} is equal to y_{head}^1 .
3. x_{head}^1 is interpreted as the point corresponding to the beginning position of the head of TM.

It is not difficult to see that φ_{TM} corresponds to TM in the desired way. \square

We then prove that every sentence of \mathcal{L} specifying a property of word models can be simulated by a Turing machine. For this purpose, we use König's Lemma.

Lemma 4.2 (König). *Let T be a finitely branching tree with infinitely many nodes. Then T contains an infinite branch.*

In the following, *accepting* means halting in an accepting state, and *rejecting* means halting in a rejecting (i.e., non-accepting) state.

Theorem 4.3. *Let Σ be a finite nonempty alphabet. Let φ be a sentence of $\mathcal{L}(\mathcal{M}(\Sigma))$. Then there exists a deterministic Turing machine TM such that the following conditions hold.*

1. *Let $W \subseteq \Sigma^*$ be the set of words w such that $\mathcal{M}(w) \models^+ \varphi$. Then for all $w \in \Sigma^*$, TM accepts w iff $w \in W$.*
2. *Let $U \subseteq \Sigma^*$ be the set of words w such that $\mathcal{M}(w) \models^- \varphi$. Then for all $w \in \Sigma^*$, TM rejects w iff $w \in U$.*

Proof. Fix some positive integer k . Given an input word w , the Turing machine TM first enumerates all plays of $G(\mathcal{M}(w), \emptyset, +, \varphi)$ with k rounds or less. If \exists wins such a play, TM checks whether there is a winning strategy for \exists that always leads to a win in k or fewer rounds, meaning that no play where \exists follows the strategy lasts for $k+1$ rounds or more, and \exists wins all plays where she follows her strategy. Similarly, if \forall wins a play with k or fewer rounds, TM checks whether there is a winning strategy for \forall

that always leads to a win in at most k rounds. If there is such a strategy for $\exists (\forall)$, then TM halts in an accepting (rejecting) state.

If no winning strategy is found, the machine TM checks all plays with $k + 1$ rounds. Again, if \exists wins such a play, TM checks whether there is a winning strategy for \exists that always leads to a win in at most $k + 1$ rounds, and similarly for \forall . Again, if a winning strategy for $\exists (\forall)$ is found, then TM halts in an accepting (rejecting) state.

If no winning strategy is found, the machine scans all plays of the length $k + 2$, and so on. This process of scanning increasingly long plays is carried on potentially infinitely long.

Now assume, for the sake of contradiction, that $\exists (\forall)$ has a winning strategy with arbitrarily long plays resulting from following the strategy. Then the game tree restricted to paths where $\exists (\forall)$ follows the strategy has infinitely many nodes. Let T denote the restriction of the game tree to paths where the strategy is followed. Since each game position can have only finitely many successor positions, and since T is infinite, we conclude by König's lemma that T has an infinite branch. Thus the strategy of $\exists (\forall)$ cannot be a winning strategy. This is a contradiction. Hence each winning strategy has a finite bound n such that each play where the strategy is followed, goes on for at most n rounds.

Thus TM has the desired properties. The crucial issue here is that there exist a *finite* number of possible moves at every position of the game. This finiteness is due to the underlying models always being finite and properties of the operators of the logic \mathcal{L} . \square

Note that our translations of Turing machines to formulae of \mathcal{L} and formulae of \mathcal{L} to Turing machines are both effective.

5 Arbitrary Structures

Above we limited attention to word models. This is not necessary, as Theorems 4.1 and 4.3 can easily be generalized to the context of arbitrary finite structures. In this section we show how this generalization can be done.

When investigating computations on structure classes (rather than strings), Turing machines of course operate on *encodings* of structures. We will use the encoding scheme of [16]. Let τ be a finite relational vocabulary and \mathfrak{A} a finite τ -structure. In order to encode the structure \mathfrak{A} by a binary string, we first need to define a linear ordering of the domain A of \mathfrak{A} . Let $<^{\mathfrak{A}}$ denote such an ordering.

Let $R \in \tau$ be a k -ary relation symbol. The encoding $enc(R^{\mathfrak{A}})$ of $R^{\mathfrak{A}}$ is the $|A|^k$ -bit string defined as follows. Consider an enumeration of all k -tuples over A in the *lexicographic order* defined with respect to $<^{\mathfrak{A}}$. In the lexicographic order, (a_1, \dots, a_k) is smaller than (a'_1, \dots, a'_k) iff there exists $i \in \{1, \dots, k\}$ such that $a_i < a'_i$ and $a_j = a'_j$ for all $j < i$. There are $|A|^k$ tuples in A^k . The string $enc(R^{\mathfrak{A}})$ is the string $t \in \{0, 1\}^*$ of the length $|A|^k$ such that the bit t_i of $t = t_1 \dots t_{|A|^k}$ is 1 if and only if the i -th tuple $(a_1, \dots, a_k) \in A^k$ in the lexicographic order is in the relation $R^{\mathfrak{A}}$.

The encoding $enc(\mathfrak{A})$ is defined as follows. We first order the relations in τ . Let p be the number of relations in τ , and let R_1, \dots, R_p enumerate the symbols in τ according to the order. We define

$$enc(\mathfrak{A}) := 0^{|A|} \cdot 1 \cdot enc(R_1^{\mathfrak{A}}) \cdot \dots \cdot enc(R_p^{\mathfrak{A}}).$$

Notice that the encoding of \mathfrak{A} depends on the order $<^{\mathfrak{A}}$ and the ordering of the relation symbols in τ .

Let \mathcal{C} be the class of exactly all finite τ -models. Let \mathcal{C}_+ , \mathcal{C}_- and \mathcal{C}_0 be subclasses of \mathcal{C} such that the following conditions hold.

1. Each of the three classes \mathcal{C}_+ , \mathcal{C}_- and \mathcal{C}_0 is closed under isomorphism.

2. The classes are disjoint, i.e., the intersection of any two of the three classes is empty.
3. $\mathcal{C}_+ \cup \mathcal{C}_- \cup \mathcal{C}_0 = \mathcal{C}$.

We say that $(\mathcal{C}_+, \mathcal{C}_-, \mathcal{C}_0)$ is a *Turing classification* of finite τ -models if there exists a Turing machine TM such that the following conditions hold.

1. The input alphabet of TM is $\{0, 1\}$.
2. TM rejects every input string that is not of the type $enc(\mathfrak{A})$ for any finite τ -structure \mathfrak{A} .
3. There exists an ordering $<^\tau$ of τ such that the following conditions hold.
 - (a) Let $\mathfrak{A} \in \mathcal{C}$. Let $enc(\mathfrak{A})$ and $enc'(\mathfrak{A})$ be two encodings of \mathfrak{A} , both using the order $<^\tau$ of τ but possibly a different ordering of A . Then one of the following three conditions holds.
 - i. TM accepts both strings $enc(\mathfrak{A})$ and $enc'(\mathfrak{A})$.
 - ii. TM rejects both strings $enc(\mathfrak{A})$ and $enc'(\mathfrak{A})$.
 - iii. TM diverges on both input strings $enc(\mathfrak{A})$ and $enc'(\mathfrak{A})$.
 - (b) Let $\mathfrak{A} \in \mathcal{C}$. Let $enc(\mathfrak{A})$ be an encoding of \mathfrak{A} according to the order $<^\tau$. The following conditions hold.
 - i. TM accepts $enc(\mathfrak{A})$ iff $\mathfrak{A} \in \mathcal{C}_+$.
 - ii. TM rejects $enc(\mathfrak{A})$ iff $\mathfrak{A} \in \mathcal{C}_-$.
 - iii. TM diverges on the input $enc(\mathfrak{A})$ iff $\mathfrak{A} \in \mathcal{C}_0$.

We say that TM *witnesses* the Turing classification $(\mathcal{C}_+, \mathcal{C}_-, \mathcal{C}_0)$.

The logic \mathcal{L} combines the expressivity of first-order logic with the possibility of building fresh relations over fresh domain elements. The recursive looping capacity enables a flexible way of using such fresh constructions. Therefore it is not difficult to see that the following theorem holds.

Theorem 5.1. *Let τ be a finite relational vocabulary and $(\mathcal{C}_+, \mathcal{C}_-, \mathcal{C}_0)$ a Turing classification of finite τ -models. Let TM be a Turing machine that witnesses the classification $(\mathcal{C}_+, \mathcal{C}_-, \mathcal{C}_0)$. Then there exists a sentence φ_{TM} of $\mathcal{L}(\tau)$ such that the following conditions hold for finite τ -models \mathfrak{A} .*

1. $\mathfrak{A} \models^+ \varphi_{\text{TM}}$ iff $\mathfrak{A} \in \mathcal{C}_+$.
2. $\mathfrak{A} \models^- \varphi_{\text{TM}}$ iff $\mathfrak{A} \in \mathcal{C}_-$.

Proof sketch. The simulation of a machine TM operating on encodings of structures \mathfrak{A} is done by a sentence φ_{TM} of \mathcal{L} as follows.

The “input” to the formula φ_{TM} is a finite τ -structure \mathfrak{A} . The formula φ_{TM} first uses \mathfrak{A} in order to construct a *word model* $\mathfrak{M}_{\mathfrak{A}}$ that corresponds to a string $enc(\mathfrak{A})$ that encodes \mathfrak{A} . The domains of $\mathfrak{M}_{\mathfrak{A}}$ and \mathfrak{A} are disjoint. The relation symbols of $\mathfrak{M}_{\mathfrak{A}}$ are symbols in VAR_{SO} , not symbols in τ . Once $\mathfrak{M}_{\mathfrak{A}}$ has been constructed, the formula φ_{TM} uses $\mathfrak{M}_{\mathfrak{A}}$ in order to simulate the computation of TM on the string $enc(\mathfrak{A})$. The simulation is done in the way described in the proof of Theorem 4.1.

The construction of the word model $\mathfrak{M}_{\mathfrak{A}}$ from \mathfrak{A} is not difficult. First a fresh successor relation $S^{\mathfrak{A}}$ over the domain of \mathfrak{A} is constructed using the operator $I_{S,xy}$. The symbol S is not in τ . Instead, we use a fresh symbol in VAR_{SO} . Also, the successor symbol S will not be part of the vocabulary of the word model $\mathfrak{M}_{\mathfrak{A}}$.

Let $<^{\mathfrak{A}}$ denote the linear order canonically associated with the successor relation $S^{\mathfrak{A}}$. The order $<^{\mathfrak{A}}$, together with an ordering of τ , define a string $enc(\mathfrak{A})$. The model $\mathfrak{M}_{\mathfrak{A}}$ is the word model corresponding to the string $enc(\mathfrak{A})$.

Due to the *very high expressivity* of the logic \mathcal{L} , is not difficult to build $\mathfrak{M}_{\mathfrak{A}}$ using $S^{\mathfrak{A}}$ and possibly further auxiliary relations. Thus writing the formula φ_{TM} is relatively straightforward. We skip further details. \square

Theorem 5.2. *Let τ be a finite relational vocabulary. Let φ be a τ -sentence of \mathcal{L} . Then there exists a Turing classification $(\mathcal{C}_+, \mathcal{C}_-, \mathcal{C}_0)$ of finite τ -models such that for all finite τ -models \mathfrak{A} , the following conditions hold.*

1. $\mathfrak{A} \in \mathcal{C}_+$ iff $\mathfrak{A} \models^+ \varphi$.
2. $\mathfrak{A} \in \mathcal{C}_-$ iff $\mathfrak{A} \models^- \varphi$.

Proof. The proof is practically identical to the proof of Theorem 4.3. □

6 Generalized Quantifiers and Oracles

The relationship between oracles and Turing machines is analogous to the relationship between generalized quantifiers and logic. Oracles allow arbitrary jumps in computations in a similar way in which generalized quantifiers allow the assertion of arbitrary properties of relations. In this section we briefly discuss extensions of the logic \mathcal{L} with generalized quantifiers. For the sake of simplicity, we only consider *unary quantifiers of the width one*, i.e., quantifiers of the type (1).

A *unary generalized quantifier of the width one* (cf. [17]) is a class \mathcal{C} of structures (A, B) such that the following conditions hold.

1. $A \neq \emptyset$ and $B \subseteq A$.
2. If $(A', B') \in \mathcal{C}$ and if there is an isomorphism $f: A' \rightarrow A''$ from (A', B') to another structure (A'', B'') , then we have $(A'', B'') \in \mathcal{C}$.

Below the word *quantifier* always means a unary generalized quantifier of the width one.

Let Q be a quantifier. Let \mathfrak{A} be a model with the domain A . We define $Q^{\mathfrak{A}} := \{ B \mid (A, B) \in Q \}$. Extend the formula formation rules of first-order logic such that if φ is a formula and x a variable, then $\hat{Q}x\varphi$ is a formula. The operator $\hat{Q}x$ binds the variable x , so the set of free variables of $\hat{Q}x\varphi$ is obtained by removing x from the set of free variables of φ . The standard semantic clause for the formula $\hat{Q}x\varphi$ is as follows.

Let \mathfrak{A} be a model that interprets the non-logical symbols in φ . Let f be an assignment function that interprets the free variables in $\hat{Q}x\varphi$. Then $\mathfrak{A}, f \models \hat{Q}x\varphi$ iff $\{ a \in A \mid \mathfrak{A}, f[x \mapsto a] \models \varphi \} \in Q^{\mathfrak{A}}$.

We then discuss how generalized quantifiers can be incorporated into the logic \mathcal{L} . This simply amounts to extending the game-theoretic semantics such that generalized quantifiers are taken into account. This is accomplished in the canonical way described below.⁵

Assume we have reached a position $(\mathfrak{A}, f, +, \varphi)$ in a semantic game. If $Q^{\mathfrak{A}} = \emptyset$, the player \exists loses the play of the game. Otherwise the player \exists chooses a set $S \in Q^{\mathfrak{A}}$. The player \forall then chooses either a point $s \in S$ of a point $s' \in A \setminus S$. (Here A is of course the domain of \mathfrak{A} .) Suppose first that \forall chooses $s \in S$. Then the game continues from the position $(\mathfrak{A}, f[x \mapsto s], +, \varphi)$. Suppose then that \forall chooses $s' \in A \setminus S$. Then the game continues from the position $(\mathfrak{A}, f[x \mapsto s'], -, \varphi)$. The intuition behind these moves is that \exists first chooses the set S of *exactly all* witnesses for φ , and this set S must be in $Q^{\mathfrak{A}}$. Then \forall either opposes the claim that S contains *only* witnesses of S by choosing a potential counterexample $s \in S$, or alternatively, \forall opposes the claim that S contains *all* witnesses of φ by choosing a potential further witness $s' \in A \setminus S$.

Assume then that we have reached a position $(\mathfrak{A}, f, -, \varphi)$ in a semantic game. If $Q^{\mathfrak{A}} = \emptyset$, the player \forall loses the play of the game. Otherwise the player \forall chooses a set $S \in Q^{\mathfrak{A}}$. The player \exists then chooses

⁵Somewhat surprisingly, the semantic game moves for generalized quantifiers we are about to define do not seem to have been defined in the exact same way in the literature before. However, the article [15] provides a rather similar but not exactly the same treatment.

either a point $s \in S$ of a point $s' \in A \setminus S$. Suppose that \exists chooses $s \in S$. Then the game continues from the position $(\mathfrak{A}, f[x \mapsto s], -, \varphi)$. Suppose then that \exists chooses $s' \in A \setminus S$. Then the game continues from the position $(\mathfrak{A}, f[x \mapsto s'], +, \varphi)$.

It is straightforward to prove that these rules give a semantics such that in restriction to formulae of first-order logic extended with generalized quantifiers, the standard Tarski style semantics and the game-theoretic semantics are equivalent. For the sake of brevity, we shall not attempt to formulate extensions of Theorems 5.1 and 5.2 that apply to extensions of \mathcal{L} with quantifiers and Turing machines with corresponding oracles. Instead, further investigations in this direction are left for the future.

7 Concluding remarks

It is easy to see that various interesting operators can be added to \mathcal{L} without sacrificing Turing-completeness. For example, second-order quantifiers can easily be added. There are only finitely many ways to interpret a quantified second-order variable in a finite model, and therefore König's lemma can still be applied so that Theorems 4.3 and 5.2 hold. Also, it is possible to add to \mathcal{L} an operator that, say, adds $|\mathcal{P}(W)|$ fresh elements to the domain W , and then extends the interpretations of selected relation symbols and second-order variables non-deterministically to all of the new domain. In the finite, this operator does not add anything to the expressivity of \mathcal{L} , but of course more delicate features of the underlying logic change.

Connections between \mathcal{L} and team semantics ought to be investigated thoroughly. Both P and NP can be characterized nicely by logics based on team semantics; NP is captured by both dependence logic and IF logic, and P is captured on ordered models by *inclusion logic* (see [6]). Further interesting complexity classes will probably be characterized in terms of logics based on team semantics in the near future. We conjecture that by attaching suitable operators to the atoms of \mathcal{L} of the type $k \in \mathbb{N}$, it should be possible to extend \mathcal{L} such that resulting logics accommodate typical logics based on team semantics as fragments in a natural way. The game-theoretic approaches to team semantics developed in [4, 8, 15, 18, 21] provide some starting points for related investigations.

Let R be a binary relation symbol. Let \mathcal{L}_0 denote the fragment of \mathcal{L} that extends first-order logic by operators that enable the manipulation of the relation R (only), the insertion of fresh points to the domain, and recursive looping. We conjecture that on models whose vocabulary contains the binary relation symbol R , already \mathcal{L}_0 is Turing-complete. Indeed, this does not seem to be difficult to prove using suitable gadgets, but we leave it as a conjecture at this stage.

Finally, it would be interesting to classify fragments of \mathcal{L} according to whether their (finite) satisfiability problem is decidable. This would nicely extend the research on decidability of fragments of first-order logic.

References

- [1] Serge Abiteboul, Richard Hull & Victor Vianu (1995): *Foundations of Databases*. Addison-Wesley.
- [2] Andreas Blass, Yuri Gurevich & Saharon Shelah (1999): *Choiceless Polynomial Time*. *Ann. Pure Appl. Logic* 100(1-3), pp. 141–187, doi:10.1016/S0168-0072(99)00005-6.
- [3] Egon Börger & Robert F. Stärk (2003): *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, doi:10.1007/978-1-84882-736-3_3.
- [4] J. C. Bradfield (2013): *Team building in dependence*. In: *CSL*, pp. 116–128, doi:10.4230/LIPIcs.CSL.2013.116.

- [5] Ashok K. Chandra & David Harel (1980): *Computable Queries for Relational Data Bases*. *J. Comput. Syst. Sci.* 21(2), pp. 156–178, doi:10.1016/0022-0000(80)90032-X.
- [6] P. Galliani & Lauri Hella (2013): *Inclusion logic and fixed point logic*. In: *CSL*, pp. 281–295, doi:10.4230/LIPIcs.CSL.2013.281.
- [7] Jean-Yves Girard (1987): *Linear Logic*. *Theor. Comput. Sci.* 50, pp. 1–102, doi:10.1016/0304-3975(87)90045-4.
- [8] E. Grädel (2013): *Model-checking games for logics of imperfect information*. *Theor. Comput. Sci.* 493, pp. 2–14, doi:10.1016/j.tcs.2012.10.033.
- [9] E. Grädel & S. Siebertz (2012): *Dynamic definability*. In: *ICDT*, pp. 236–248, doi:10.1145/2274576.2274601.
- [10] Y. Gurevich (1985): *A new thesis*. *American Mathematical Society Abstracts* 6(4), p. 317.
- [11] J. Hintikka (1973): *Logic, Language-Games and Information: Kantian Themes in the Philosophy of Logic*. Oxford: Clarendon Press.
- [12] J. Hintikka (1996): *The Principles of Mathematics Revisited*. Cambridge University Press, doi:10.1017/cbo9780511624919.
- [13] J. Hintikka & G. Sandu (1989): *Informational independence as a semantical phenomenon*. In: *Proceedings of the Eighth International Congress of Logic Methodology and Philosophy of Science*, pp. 571–589, doi:10.1016/S0049-237X(08)70066-1.
- [14] Giorgi Japaridze (2003): *Introduction to computability logic*. *Ann. Pure Appl. Logic* 123(1-3), pp. 1–99, doi:10.1016/S0168-0072(03)00023-X.
- [15] A. Kuusisto (2013): *A double team semantics for generalized quantifiers*. *CoRR* abs/1310.3032. Available at <http://arxiv.org/abs/1310.3032>.
- [16] L. Libkin (2004): *Elements of Finite Model Theory*. Springer, doi:10.1007/978-3-662-07003-1.
- [17] P. Lindström (1966): *First order predicate logic with generalized quantifiers*. *Theoria* 32, p. 186195, doi:10.1111/j.1755-2567.1966.tb00600.x.
- [18] A. L. Mann, G. Sandu & M. Sevenster (2011): *Independence-Friendly Logic - a Game-Theoretic Approach*. *London Mathematical Society lecture note series* 386, Cambridge University Press, doi:10.1017/CB09780511981418.
- [19] S. Patnaik & N. Immerman (1994): *Dyn-FO: A parallel, dynamic complexity class*. In: *PODS*, pp. 210–221, doi:10.1145/182591.182614.
- [20] J. A. Väänänen (1979): *Abstract logic and set theory. I. Definability*. In: *Logic Colloquium, '78, Studies in Logic Foundations Mathematics*, North Holland, Amsterdam, pp. 391–421, doi:10.1016/s0049-237x(08)71637-9.
- [21] J. A. Väänänen (2007): *Dependence Logic - A New Approach to Independence Friendly Logic*. *London Mathematical Society student texts* 70, Cambridge University Press, doi:10.1017/cbo9780511611193.004.
- [22] J. A. Väänänen (2012): *Second order logic or set theory?* *Bulletin of Symbolic Logic* 18(1), pp. 91–121, doi:10.2178/bsl/1327328440.