# Impossibility Results for Logic
note for Computability & Logic

Jesper Buus Nielsen

Aarhus University

**Abstract.** We prove that any sound logic for arithmetic must be incomplete. From this it follows that Hoare logic is incomplete too. We also see that the validity problem and the provability problem for first-order predicate logic are undecidable problems. We see that any sound logic for second-order predicate logic must be incomplete. The crucial step in proving these results is establishing the expressive power of the logics. We see that propositional logic can express exactly the recursive languages. We see that first-order predicate logic can express exactly the recursively enumerable languages, and we see that the logic for arithmetic and second-order predicate logic can express languages which are not recursively enumerable.

## 1 Logics

We start by formalizing the minimal requirements for calling something a logic. For our purpose, a logic is a triple $(\Phi, \models, \vdash)$, where $\Phi$ is a set of formulæ and $\models \subseteq \Phi$ and $\vdash \subseteq \Phi$. We write $\models \phi$ rather than $\phi \in \models$ and we write $\vdash \phi$ rather than $\phi \in \vdash$. For $\phi \in \Phi$ we read $\models \phi$ as "$\phi$ is *valid*" and we read $\vdash \phi$ as "$\phi$ *is provable*". The predicate $\models$ provides the semantics of the logic and $\vdash$ captures the proof system.

We assume that $\Phi \subseteq \Sigma^*$ for some alphabet $\Sigma^*$ such that a formula is something we can write down. We assume that $\Phi$ is recursive. Typically $\Phi$ is defined via a context-free grammar, such that $\Phi$ is even a context-free language, but all we need is that $\Phi$ is recursive. It would not be reasonable if the set of formulæ was not recursive, as one could then not even recognize if something is a formula of the logic or not.

We assume that $\vdash$ is a recursively enumerable. I.e., we assume that there exist a Turing machine $M$ such that $M(\phi)$ accepts iff $\vdash \phi$. If $\nvdash \phi$, then $M$ might loop forever. The justification for this assumption is that any proof system based on a notion of verifiable proofs will give a set of provable formulæ which is recursively enumerable — one can check if $\vdash \phi$ by going over all possible proofs and see if one of them is a proof of $\phi$. See Exercise 1.

**Exercise 1** *Assume that the proof system $\vdash$ is based on a notion of verifiable proofs, i.e.,*

1. *There exists a Turing computable function* verifyProof *which takes as input $\phi$ and a purported proof $\Pi \in \Sigma^*$ and which outputs $T$ or $F$.*
2. *If there exists $\Pi \in \Sigma^*$ such that* $\mathrm{verifyProof}(\phi, \Pi) = T$*, then $\vdash \phi$.*
3. *If $\vdash \phi$, then there exists $\Pi \in \Sigma^*$ such that* $\mathrm{verifyProof}(\phi, \Pi) = T$.

*Show that in that case $\vdash$ is recursively enumerable.*

We call a logic $(\Phi, \models, \vdash)$ *sound* if $\vdash \phi$ implies that $\models \phi$. I.e., $(\Phi, \models, \vdash)$ is sound iff $\vdash \subseteq \models$.

We call a logic $(\Phi, \models, \vdash)$ *complete* if $\models \phi$ implies that $\vdash \phi$. I.e., $(\Phi, \models, \vdash)$ is complete iff $\models \subseteq \vdash$.

**Lemma 1.** *If a logic $(\Phi, \models, \vdash)$ is sound and complete, then $\vdash = \models$.*

*Proof.* $\vdash \subseteq \models$ and $\models \subseteq \vdash$ implies that $\models = \vdash$.

## 1.1 Propositional Logic

In propositional logic the set $\Phi_{\mathsf{prop}}$ of formulas over variables $\mathcal{V}$ is given by the context-free grammar

$$\phi ::= \top \mid \bot \mid x \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid (\phi) \ ,$$

with start symbol $\phi$, where $x \in \mathcal{V}$.

For $\phi \in \Phi$ we say that $\phi$ is *valid* if $\phi$ evaluates to $\mathsf{T}$ for all possible truth assignments of its variables, which we will write as $\models_{\mathsf{prop}} \phi$.

For $\phi \in \Phi$ we say that $\phi$ is *provable* if $\phi$ can be derived given the natural deduction rules for propositional logic given in Chapter 1.2.1 in [HR04]; We write $\vdash_{\mathsf{prop}} \phi$.

## 1.2 First-Order Predicate Logic

In first-order predicate logic over function symbols $\mathcal{F}$ and predicate symbols $\mathcal{P}$, the set $\Phi_{\mathsf{1pred}}$ of formulæ is given by the context-free grammar

$$t ::= x \mid c \mid f(t, \ldots, t)$$

$$\phi ::= P(t, \ldots, t) \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid (\phi) \mid \forall x \phi \mid \exists x \phi \ ,$$

with start symbol $\phi$, where $x \in \mathcal{V}$, $f \in \mathcal{F}$ and $P \in \mathcal{P}$. There are some extra requirements, concerning arity of function symbols and predicate symbols, discussed in Chapter 2.2 in [HR04]. The constants, $c$, range over the function symbols with arity 0.

For $\phi \in \Phi$ we say that $\phi$ is *valid* if $\phi$ is true in all models of $(\mathcal{F}, \mathcal{P})$ and in all lookup tables. We write $\models_{\mathsf{1pred}} \phi$.

For $\phi \in \Phi$ we say that $\phi$ is *provable* if $\phi$ can be derived given the natural deduction rules for predicate logic given in Chapter 2.3.1 in [HR04]. We write $\vdash_{\mathsf{1pred}} \phi$.

## 1.3 Second-Order Predicate Logic

Second-order predicate logic is an extension of first-order predicate logic, where one can quantify over higher order objects like functions, predicates and subsets. This is somewhat more tricky to give a well-defined semantics than when you can only quantify over elements of the universe as in first-order predicate logic, and we shall not try to give a semantics for full-fledged second-order logic.

We will look at a limited form of second-order logic, where we do not quantify over functions or predicates. We are only allowed to quantify over subsets of the universe.

Syntactically we allow to write $\forall S\phi$ and $\exists S\phi$, where $S$ is a set variable. We also allow to write $t \in S$, where $t$ is a term and $S$ is a set variable.

$$\phi ::= \forall S\phi \mid \exists S\phi \mid t \in S \ .$$

The semantics of $\forall S\phi$ is that $\phi$ should be true no matter which subset $B \subseteq A$ of the universe $S$ is assigned in the lookup table. The semantics of $\exists S\phi$ is that there is some subset $B \subseteq A$ of the universe which $S$ can be assigned in the lookup table which makes $\phi$ true. The semantics of $t \in S$ is that the element which $t$ evaluates to in the lookup table is in the set that $S$ is assigned in the lookup table.

It might seem as a sever restriction that we only introduced the operator $\in$ on sets. Would we not like to say something like $S$ is a subset of $T$, i.e., write $S \subseteq T$? The clue is that we can express $S \subseteq T$ as a macro via $\in$:

$$S \subseteq T \overset{\text{DEF}}{=} \forall x(x \in S \rightarrow x \in T) \ .$$

So, we can just introduce the convention that when we write $S \subseteq T$ in an expression, then it is shorthand for $\forall x(x \in S \rightarrow x \in T)$.

## 1.4 Logic for Arithmetic

In our logic for arithmetic over the natural numbers, the set $\Phi_{\mathsf{AR}}$ of formulas is the set of formulas of predicate logic with predicate symbol $=$, constants 0 and 1 and binary function symbols $+$ and $*$.

This might seem as a very small language, but it is rich enough to express all we need for our impossibility results by using a few macros. Say we would like to express that

$$\forall x((x < 3) \rightarrow (x \leq 2)) \ .$$

We can express a constant like 3 as 1+1+1. We can express $x \leq y$ as $\exists z(x+z = y)$, as $z \geq 0$ when we work in the natural numbers. And, we can express $x < y$ as $(x \leq y) \wedge (x \neq y)$ and in turn express $x \neq y$ as $\neg(x = y)$. So, we can express our statement as

$$\forall x((\exists z(x + z = 1 + 1 + 1)) \wedge \neg(x = 1 + 1 + 1) \rightarrow (\exists z(x + z = 1 + 1))) \ .$$

We will allow ourselves to write $\forall x((x < 3) \rightarrow (x \leq 2))$ and just remember that 2, 3 and $<$ are macros for larger expressions. We require that macros are Turing computable, such that they can be expanded in a systematic manner.

Here are some of the macros we will use.

$$\begin{aligned}
x \neq y &\overset{\text{DEF}}{=} \neg(x = y) \\
x \leq y &\overset{\text{DEF}}{=} \exists z(x + z = y) \\
x \geq y &\overset{\text{DEF}}{=} y \leq x \\
x < y &\overset{\text{DEF}}{=} x \leq y \wedge (x \neq y) \\
q = x \div y &\overset{\text{DEF}}{=} \exists r((q * y + r = x) \wedge r < y) \\
r = x \bmod y &\overset{\text{DEF}}{=} \exists q((q * y + r = x) \wedge r < y) \\
x|y &\overset{\text{DEF}}{=} \exists z(x * z = y) \\
\mathrm{Prime}(y) &\overset{\text{DEF}}{=} y > 1 \wedge \forall x\,((x|y) \rightarrow ((x = 1) \vee (x = y))) \\
\mathrm{PrimePower}(x, p) &\overset{\text{DEF}}{=} \mathrm{Prime}(p) \wedge \forall q((\mathrm{Prime}(q) \wedge q|x) \rightarrow (q = p)) \\
\mathrm{PrimePower}(x) &\overset{\text{DEF}}{=} \exists p\,\mathrm{PrimePower}(x, p)
\end{aligned}$$

For $\phi \in \Phi$ we say that $\phi$ is *valid* if it holds in the usual model of arithmetic. We write $\vDash_{\mathsf{AR}} \phi$. I.e., we say that $\vDash_{\mathsf{AR}} \phi$ iff $\mathcal{M}_{\mathsf{AR}} \vDash_{\mathsf{1pred}} \phi$, where $\mathcal{M}_{\mathsf{AR}}$ is the model where we quantify over the natural numbers $\mathbb{N}$, where 0 and 1 denote the natural numbers zero and one, and where $+$ and $*$ are addition respectively multiplication over $\mathbb{N}$.

**Exercise 2** *Let $\ell : \mathcal{V} \rightarrow \mathbb{N}$ be any lookup table. Argue that if $\vDash_{\mathsf{AR}\ell} x|y$, then $\ell(x)$ divides $\ell(y)$. Argue that if $\vDash_{\mathsf{AR}\ell} z = x \bmod y$, then $\ell(z)$ is the remainder after dividing $\ell(x)$ by $\ell(y)$ in the natural numbers, and argue that if $\vDash_{\mathsf{AR}\ell} z = x \div y$, then $\ell(z)$ is the result of dividing $\ell(x)$ by $\ell(y)$ in the natural numbers.*

**Exercise 3** *A prime power is a natural number $x$ which can be written as $x = p^i$ for a prime number $p$ and a natural number $i$. Show that if $\vDash_{\mathsf{AR}\ell} \mathrm{PrimePower}(x)$, then $\ell(x)$ is a prime power. In particular, show that if $\vDash_{\mathsf{AR}\ell} \mathrm{PrimePower}(x, p)$, then $\ell(p)$ is a prime and $\ell(x) = \ell(p)^i$ for a natural number $i$.*

For $\phi \in \Phi$ we say that $\phi$ is *provable* if $\phi$ can be derived given the natural deduction rules for predicate logic, starting from a set $\Gamma$ of axioms about arithmetic. We write $\vdash_{\mathsf{AR}} \psi$. I.e., we say that $\vdash_{\mathsf{AR}} \psi$ iff $\Gamma \vdash_{\mathsf{1pred}} \psi$. This means that there is not *one* proof system for arithmetic. For each set of axioms we get another proof system.

**The Peano Axioms** For the purpose of this note it is not important exactly which set of axioms we choose for arithmetic, as what we have to say is true about them all. To be concrete we will, however, described the axioms proposed by Peano. The first six of his axioms describe how 0, 1, + and $*$ behave together.

$$\begin{aligned}
\mathrm{P}_1 : &\quad \forall n \neg(0 = n+1) \ , \\
\mathrm{P}_2 : &\quad \forall n(n+0 = n) \ , \\
\mathrm{P}_3 : &\quad \forall n(n * 0 = 0) \ , \\
\mathrm{P}_4 : &\quad \forall m \forall n((m+1 = n+1) \rightarrow (m = n)) \ , \\
\mathrm{P}_5 : &\quad \forall m \forall n(m + (n+1) = (m+n)+1) \ , \\
\mathrm{P}_6 : &\quad \forall m \forall n(m * (n+1) = (m * n) + m) \ , \\
\mathrm{P}_\psi : &\quad (\psi(0) \wedge \forall n(\psi(n) \rightarrow \psi(n+1))) \rightarrow \forall n \psi(n) \ ,
\end{aligned}$$

where the last axiom is actually an infinite set of axioms, one for each $\phi \in \Phi_{\mathsf{1pred}}$.

We call the set of the above axioms $\Gamma_{\mathsf{1Peano}}$. In the last axiom we assume that $\psi$ has some free variable $x$ and then $\psi(0) = \psi[0/x]$, $\psi(n) = \psi[n/x]$ and $\psi(n+1) = \psi[(n+1)/x]$. Notice that these seven axioms all are true for the natural numbers. They are an attempt at capturing the natural numbers axiomatically, i.e., give a list of properties which characterizes the natural numbers. Note that the first six axioms clearly do not fully characterize the natural numbers, as also the real numbers and the complex numbers satisfy these six formulæ. The last axiom is an attempt to capture that we are interested in the natural numbers. This is done by postulating that induction is possible. It is actually an infinite number of axioms, a so-called *axiom schema*, as there is an axiom for each formula $\psi$ stating that one can use induction to argue about $\psi$.

As an exercise in using the first-order Peano axioms, let us use them to argue that if you square a natural number it never gets smaller. Let $\phi(x) \overset{\text{DEF}}{=} x * x \geq x$. Then the claim is that $\vdash_{\mathsf{AR}} \forall n \phi(n)$. This would follow from $\mathrm{P}_\phi$ if $\vdash_{\mathsf{AR}} \phi(0)$ and $\vdash_{\mathsf{AR}} \forall n(\phi(n) \rightarrow \phi(n+1))$, using $\wedge$-introduction and $\rightarrow$-elimination. Note that $\phi(0) \overset{\text{DEF}}{=} 0 * 0 \geq 0$. And, $0 * 0 \geq 0$ is a macro for $\exists z(0 * 0 = 0 + z)$, which would follow from $\vdash_{\mathsf{AR}} 0 * 0 = 0 + 0$ using $\exists$-introduction. And, $\vdash_{\mathsf{AR}} 0 * 0 = 0 + 0$ follows from $\mathrm{P}_2$ and $\mathrm{P}_3$ using $=$-elimination twice. Hence, $\vdash_{\mathsf{AR}} \phi(0)$.

**Exercise 4** *Conclude the proof by arguing that $\vdash_{\mathsf{AR}} \forall n(\phi(n) \rightarrow \phi(n+1))$ for the above $\phi(x)$. Allow yourself a slighly loose argumentation as that above.*

**Second-Order Peano** Peano's original formulation of the induction axiom was as follows

$$\mathrm{P}_7 : \quad \forall S\,((0 \in S \wedge \forall n(n \in S \rightarrow (n+1) \in S)) \rightarrow \forall n(n \in S)) \ .$$

This is a second-order formula. It essentially says that there are no other natural numbers other than 0 and those who can be reached from 0 by adding 1 a number of times, i.e., $\mathbb{N} = \{0, 0+1, 0+1+1, 0+1+1+1, \ldots\}$. We let $\Gamma_{\mathsf{2Peano}}$ denote the first six first-order Peano axioms plus the above second-order axiom. It turns out that $\Gamma_{\mathsf{2Peano}}$ fully characterizes the natural numbers.

**Theorem 1.** *If $\mathcal{N} \models_{\mathsf{2pred}} \Gamma_{\mathsf{2Peano}}$, then $\mathcal{N} = \mathcal{M}_{\mathsf{AR}}$.*[1]

To see the relation with the induction axiom, let us for the predicate $\phi$ in the induction axiom, write $\phi(x)$ as $x \in \phi$. I.e., $x \in \phi$ means that $x$ is one of the inputs making $\phi$ true — we often model predicates as the set of inputs making them true. In that case, the induction axiom for $\phi$ becomes

$$(0 \in \psi \land \forall n(n \in \psi \to ((n+1) \in \psi)) \to \forall n(n \in \psi) \ .$$

The infinite collection of induction axioms thus says that the second-order axioms holds for all subsets which can be defined via a first-order formula $\phi$. The second-order axioms talks about *all* subsets of the universe. And, we know by now that this might make a difference:

- There is an uncountable number of subsets of the natural numbers.
- There are only a countably number of formulæ, and therefore only a countable number of induction axioms.

Hence the second-order axiom has the potential to be stronger than the entire collection of induction axioms, and in Exercise 11 you will prove that indeed it is.

### 1.5 Logic for Partial Program Correctness

The set $\Phi_{\mathsf{Hoare}}$ of formulæ in Hoare logic for partial program correctness consists of triples $(\phi, P, \psi)$, where $\phi$ and $\psi$ are formulæ in the logic for arithmetic and $P$ is a program over the natural numbers.

We say that $(\phi, P, \psi)$ is *valid* if whenever $P$ is run in a state satisfying $\phi$, then *if* the program terminates, then it will be in a state satisfying $\psi$. We write $\models_{\mathsf{par}} (\phi, P, \psi)$.

We say that $(\phi, P, \psi)$ is *provable* if it can be derived using the proof calculus in Chapter 4.3 in [HR04]. We write $\vdash_{\mathsf{par}} (\phi, P, \psi)$.

We will not consider total correctness.

## 2 Axiomatization

The way we defined the proof system for arithmetic is called *axiomatization*. We start with the proof system for predicate logic and then adapt it to the natural numbers by adding a set of premises which characterizes the natural numbers.

An axiomatization in predicate logic is a recursively enumerable set $\Gamma$ of formulæ. We say that $\Gamma$ is a *sound* axiomatization of a model $\mathcal{M}$ iff $\mathcal{M} \models_{\mathsf{1pred}} \Gamma$. We say that $\Gamma$ is a *complete* axiomatization of a model $\mathcal{M}$ iff $\mathcal{N} \models_{\mathsf{1pred}} \Gamma$ implies

[1] By $\mathcal{N} = \mathcal{M}_{\mathsf{AR}}$, we mean that $\mathcal{N}$ and $\mathcal{M}_{\mathsf{AR}}$ are the same model up to isomorphism, i.e., up to renaming of the elements in the universe. I.e., if $M$ is the universe of $\mathcal{M}_{\mathsf{AR}}$ and $N$ is the universe of $\mathcal{N}$, then there exists a bijective renaming $r : M \to N$ such that $r(0^{\mathcal{N}}) = 0^{\mathcal{M}_{\mathsf{AR}}}$ and such that for all $a, b \in M$ it holds that $r(a +^{\mathcal{N}} b) = r(a) +^{\mathcal{M}_{\mathsf{AR}}} r(b)$ and $r(a *^{\mathcal{N}} b) = r(a) *^{\mathcal{M}_{\mathsf{AR}}} r(b)$.

that $\mathcal{N} = \mathcal{M}$, up to isomorphism. We require that $\Gamma$ is recursively enumerable, as one cannot check a proof if one cannot recognize an axiom when it appears in the proof. Having sets of axioms which are not recursively enumerable would not give verifiable proof systems.

The following theorem show that a sound and complete axiomatization can play the role of a model in giving semantics. Recall that $\Gamma \vDash \phi$ means that all models which satisfy $\Gamma$ also satisfy $\phi$, whereas $\mathcal{M} \vDash \phi$ says that the specific model $\mathcal{M}$ satisfy $\phi$.

**Theorem 2.** *If $\Gamma$ is a sound and complete axiomatization of $\mathcal{M}$, then $\mathcal{M} \vDash \phi$ iff $\Gamma \vDash \phi$.*

*Proof.* We first show that if $\mathcal{M} \vDash \phi$, then $\Gamma \vDash \phi$. Assume that $\mathcal{M} \vDash \phi$. We have to show that $\Gamma \vDash \phi$, which is the same as showing: if $\mathcal{N} \vDash \Gamma$, then $\mathcal{N} \vDash \phi$. This is equivalent to showing that either $\mathcal{N} \nvDash \Gamma$ or $\mathcal{N} \vDash \phi$. If $\mathcal{N} \neq \mathcal{M}$, then $\mathcal{N} \nvDash \Gamma$ as $\vDash$ is complete. If $\mathcal{N} = \mathcal{M}$, then $\mathcal{N} \vDash \phi$ as $\mathcal{M} \vDash \phi$.

We then show that if $\Gamma \vDash \phi$, then $\mathcal{M} \vDash \phi$. Recall that $\Gamma \vDash \phi$ means that if $\mathcal{N} \vDash \Gamma$, then $\mathcal{N} \vDash \phi$. Since $\mathcal{M} \vDash \Gamma$ (as $\Gamma$ is sound by premise) it follows using Modus Ponens that $\mathcal{M} \vDash \phi$, and we are done.

The above theorem hints at one of the reasons why axiomatization is considered interesting. Until now, when we had to define the model $\mathcal{M}_{\mathsf{AR}}$ we have been rather informal, saying, e.g., that the universe is the set of natural numbers. But this requires that we already agree on what the natural numbers are or define them formally. Defining them formally will seeming lead to a recursion. However, by Thm. 2, if we could agree on a set $\Gamma$ of axioms for the natural numbers which have the property that there is exactly one model $\mathcal{M}$ for which $\mathcal{M} \vDash_{\mathsf{1pred}} \Gamma$, then we can just define the natural numbers to be that one model $\mathcal{M}$.

We call $\Gamma$ a *first-order axiomatization* of $\mathcal{M}$ if it is an axiomatization and all formulæ in $\Gamma$ are first-order predicate logic formulæ. Getting first-order axiomatizations is interesting because first-order predicate logic is sound and complete, which gives us the following theorem.

**Theorem 3.** *If $\Gamma$ is a sound and complete first-order axiomatization of $\mathcal{M}$, then $\mathcal{M} \vDash_{\mathsf{1pred}} \phi$ iff $\Gamma \vdash_{\mathsf{1pred}} \phi$.*

*Proof.* Since $\Gamma$ is sound and complete it follows that $\mathcal{M} \vDash_{\mathsf{1pred}} \phi$ iff $\Gamma \vDash_{\mathsf{1pred}} \phi$. Since first-order predicate logic is sound and complete it follows that $\Gamma \vDash_{\mathsf{1pred}} \phi$ iff $\Gamma \vdash_{\mathsf{1pred}} \phi$.

Axiomatization is not only used to try to capture a specific model, like $\mathcal{M}_{\mathsf{AR}}$. It can also be used to talk about a set of models at the same time, if you want to investigate what follows logically from their common properties. Group theory gives an example of this.

A *group* consists of a set $A$ and a function $\circ : A \times A \to A$ which fulfills certain restriction, namely that it is associative, has a neutral element and allows taking inverses. These restrictions are conveniently formulated in first-order predicate logic. We look at the language containing a binary function symbol $\circ$ and a constant $e$. Let $\Gamma_{\mathsf{group}}$ be the set consisting of these formulæ:

$$\begin{aligned} \mathrm{G}_1 : &\quad \forall x \forall y \forall z ((x \circ y) \circ z = x \circ (y \circ z)) \ , \\ \mathrm{G}_2 : &\quad \forall x (x \circ e = x) \ , \\ \mathrm{G}_3 : &\quad \forall x \exists y (x \circ y = e) \ . \end{aligned}$$

A group is a model satisfying these *group axioms*. I.e., $\mathcal{M}$ is a *group* iff $\mathcal{M} \models_{\mathsf{1pred}} \Gamma_{\mathsf{group}}$. A valid group theoretic formula is one which is true for all groups. I.e., $\models_{\mathsf{group}} \phi$ iff $\Gamma_{\mathsf{group}} \models_{\mathsf{1pred}} \phi$. A formula $\phi$ is said to be provable in group theory, written $\vdash_{\mathsf{group}} \phi$, if it can be proved from the group axioms. I.e., $\vdash_{\mathsf{group}} \phi$ iff $\Gamma_{\mathsf{group}} \vdash_{\mathsf{1pred}} \phi$.

Since first-order predicate logic is sound and complete, we have that $\Gamma_{\mathsf{group}} \models_{\mathsf{1pred}} \phi$ iff $\Gamma_{\mathsf{group}} \vdash_{\mathsf{1pred}} \phi$. This implies that $\models_{\mathsf{group}} \phi$ iff $\vdash_{\mathsf{group}} \phi$. So, our logic for groups is sound and complete. Note that this means that we could create a program which takes a formula about groups and then, if the formula is valid, find the proof for us.[2] Such a program is called an automated theorem prover. This would be a great help in reasoning about groups. There is nothing special about groups here. Any theory which can be captured by first-order axiomatization will lean itself towards automated theorem proving. The interested reader can find links to automated theorem provers at `http://en.wikipedia.org/wiki/Automated_theorem_proving#First-order_theorem_proving`.

## 3 Language Expressability of a Logic

A central notion in proving our impossibility results for logics is the language expressability of a logic. As an example, consider any natural number constant $n$. The formula

$$n > 1 \wedge \forall x \left( (\exists y (x * y = n) ) \rightarrow ( (x = n) \vee (x = 1) ) \right) \ ,$$

when given its semantics in $\mathcal{M}_{\mathsf{AR}}$, says that $n$ is a prime number. This shows that our logic for arithmetic can express the language consisting of all the primes. We need a formal definition.

**Definition 1.** *Let $L \subseteq \Sigma^*$ be a formal language over alphabet $\Sigma$. We say that $(\Phi, \models)$ can* express *$L$ if there exist a Turing-computable function, called the* encoding,

$$e : \Sigma^* \to \Phi$$

*such that $x \in L$ iff $\models e(x)$.*

[2] By soundness and completeness, the list of true formulæ is the same as the list of provable formulæ (Lemma 1), and the list of provable formulæ is recursively enumerable.

We later use the following two simple observations.

**Lemma 2.** *If $(\Phi, \models)$ can express a language which is not recursively enumerable, then the set $\models$ is not recursively enumerable.*

*Proof.* If $(\Phi, \models)$ can express a language which is not recursively enumerable, then there exist a formal language $L$ such that $L$ is not recursively enumerable and there exists a Turing-computable encoding $e$ such that $\models e(x)$ iff $x \in L$. This is the same as saying that $e(x) \in \models$ iff $x \in L$. Hence, $e$ shows that $L$ is reducible to $\models$. Since $L$ is not recursively enumerable it follows that $\models$ is not recursively enumerable.

Almost the same proof can be used to show the following lemma.

**Lemma 3.** *If $(\Phi, \models)$ can express a language which is not recursive, then $\models$ is not recursive.*

**Exercise 5** *Show that if $(\Phi, \models)$ can express $L_1$ and $L_2 \leq L_1$, then $(\Phi, \models)$ can express $L_2$.*

It will, in particular, be important if a logic can express a language which is not recursive or maybe even one which is not recursively enumerable. In showing that some of them can, we will use variants of Post's correspondence system (PCP).

In Chapter 11.5 in [Mar02] it is shown that PCP is undecidable by showing that Accepts $\leq$ PCP. Here is the binary version of the problem — we call it *binary* PCP because it is restricted to an alphabet of size 2.

**Definition 2 (binary PCP).** *An instance of the BPCP problem is a list of pairs, $x = ((\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n))$, where $\alpha_i, \beta_i \in \{1,2\}^*$. An instance $x$ is a Yes-instance if there exists $k \in \mathbb{N}$, $k > 0$, and $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_k} = \beta_{i_1}\beta_{i_2} \cdots \beta_{i_k}$.*

When we look at the logic for arithmetic it will be convenient to have an arithmetic version of the PCP problem. Recall that a function $f : \mathbb{N} \to \mathbb{N}$ is called *affine* if it can be written as $f(x) = ax+b$ for $a, b \in \mathbb{N}$. Examples of affine functions are $f_1(x) = 10x + 1$ and $f_{22}(x) = 100x + 22$ and $f_{1211}(x) = 10000x + 1211$. Note that $f_1(3333) = 33331$, $f_{22}(3333) = 333322$ and $f_{1211}(3333) = 33331211$. So, in our usual decimal notation for writing natural numbers, our example functions $f_\sigma(x)$ concatenate $\sigma$ to the end of the input $x$. This shows that concatenation with an element from $\{1,2\}^*$ can be represented as an affine function.

**Definition 3 (affine PCP).** *An instance of the APCP problem is a list of pairs, $x = ((f_1, g_1), \ldots, (f_n, g_n))$, where all $f_i$ and $g_i$ are affine functions over $\mathbb{N}$. An instance $x$ is a Yes-instance if there exists $k \in \mathbb{N}$, $k > 0$, and $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $f_{i_k}(\cdots f_{i_2}(f_{i_1}(0)) \cdots) = g_{i_k}(\cdots g_{i_2}(g_{i_1}(0)) \cdots)$.*

**Exercise 6** *Show that Accepts $\leq$ APCP, i.e., that Accepts reduces to APCP. You can do it in three steps. First recall that Accepts $\leq$ PCP. Then show that PCP $\leq$ BPCP, using a suitable encoding of any alphabet into the alphabet $\{1,2\}$. Then show that BPCP $\leq$ APCP by representing concatenations as affine functions.*

## 4 Impossibility Results

We are now ready to prove our impossibility results.

### 4.1 Decidability of Validity

**Definition 4.** *The problem* Validity *for $(\Phi, \models)$ is the problem of determining whether a formula is valid. I.e., the instances are $\Phi$, the Yes-instances are $Y = \models$ and the No-instances are $N = \Phi \setminus \models$. In other words, the Yes-instances are those $\phi \in \Phi$ for which $\models \phi$ and the No-instances are those $\phi \in \Phi$ for which $\not\models \phi$.*

**Theorem 4.** *No logic $(\Phi, \models, \vdash)$ can have the following two properties at the same time:*

1. *Validity is decidable for $(\Phi, \models)$.*
2. *$(\Phi, \models)$ can express a language which is not recursive.*

*Proof.* Assume that there exists $(\Phi, \models, \vdash)$ which have both properties. If Validity is decidable, then almost by definition $\models$ is recursive. On the other hand, if $(\Phi, \models)$ can express a language which is not recursive, then (by Lemma 3) $\vdash$ is *not* recursive, a contradiction.

### 4.2 Soundness and Completeness

We prove an abstract variant of Gödel's incompleteness theorem, which says that if a logic is expressive enough and is sound, then it cannot be complete. Here *expressive enough* means that the logic can express some language which is not recursively enumerable. Proving it, as we do, by going via recursive enumerability is an idea of Turing.

**Theorem 5.** *No logic $(\Phi, \models, \vdash)$ can have the following three properties at the same time:*

1. *$(\Phi, \models, \vdash)$ is sound.*
2. *$(\Phi, \models, \vdash)$ is complete.*
3. *$(\Phi, \models)$ can express a language which is not recursively enumerable.*

*Proof.* Assume that $(\Phi, \models, \vdash)$ have all three properties. If $(\Phi, \models, \vdash)$ is complete and sound, then by Lemma 1 we have that $\vdash = \models$. By requirement of being a logic we have that $\vdash$ is recursively enumerable. From $\vdash = \models$ and $\vdash$ being recursively enumerable, it follows that $\models$ is recursively enumerable. On the other hand, if $(\Phi, \models)$ can express a language which is not recursively enumerable, then by Lemma 2 it follows that $\models$ is *not* recursively enumerable. So, if the logic has all three properties, then $\models$ is both recursively enumerable and *not* recursively enumerable, a contradiction.

## 4.3 Decidability of Provability, Soundness and Completeness

**Definition 5.** *The problem* Provability *for $(\Phi, \vdash)$ is the problem of determining whether a formula is provable. I.e., the instances are $\Phi$, the Yes-instances are $Y = \vdash$ and the No-instances are $N = \Phi \setminus \vdash$. In other words, the Yes-instances are those $\phi \in \Phi$ for which $\vdash \phi$ and the No-instances are those $\phi \in \Phi$ for which $\nvdash \phi$.*

**Theorem 6.** *No logic $(\Phi, \models, \vdash)$ can have the following four properties at the same time:*

1. *$(\Phi, \models, \vdash)$ is sound.*
2. *$(\Phi, \models, \vdash)$ is complete.*
3. *Provability is decidable for $(\Phi, \vdash)$.*
4. *$(\Phi, \models)$ can express a language which is not recursive.*

*Proof.* Assume that there exists $(\Phi, \models, \vdash)$ which have all four properties. If Provability is decidable, then almost by definition $\vdash$ is recursive. If $(\Phi, \models, \vdash)$ is sound and complete, then $\models = \vdash$, and hence $\models$ is recursive too. This contradicts that $(\Phi, \models)$ can express a language which is not recursive.

## 4.4 Corollaries

In the following sections we are going to show the following lemmata.

**Lemma 4.** *$(\Phi_{\mathsf{prop}}, \models_{\mathsf{prop}})$ can express all recursive languages.*

**Lemma 5.** *$(\Phi_{\mathsf{1pred}}, \models_{\mathsf{1pred}})$ can express the set of Yes-instances of $\mathsf{BPCP}$.*

**Lemma 6.** *$(\Phi_{\mathsf{AR}}, \models_{\mathsf{AR}})$ can express the set of No-instances of $\mathsf{APCP}$.*

**Exercise 7** *Argue that the Yes-instances of $\mathsf{BPCP}$ is not recursive and that the No-instances of $\mathsf{APCP}$ is not recursively enumerable.*

Combining these results with the ones above, it is possible to derive the following results.

**Corollary 1.** *$(\Phi_{\mathsf{prop}}, \models_{\mathsf{prop}})$ can express exactly the recursive languages. I.e., $L$ can be expressed by propositional logic iff $L$ is recursive.*

*Proof.* From Lemma 4 we know that propositional logic can express all recursive languages, so to prove Corollary 1 we only have to show that propositional logic cannot express a language which is not recursive. We know from [HR04] that the validity problem for propositional logic is decidable (simply do the truth table). We know from Thm. 4 that if the validity problem is decidable for a logic, then it cannot express a language which is not recursive. Hence propositional logic cannot express a language which is not recursive.

**Corollary 2.** *The validity problem for first-order predicate logic is undecidable.*

*Proof.* We know from Lemma 5 that first-order predicate logic can express a problem which is not decidable. So, by Thm. 4 we know that the validity problem for predicate logic cannot be decidable.

**Corollary 3.** *The provability problem for first-order predicate logic is not decidable.*

**Exercise 8** *Prove Corollary 3.*

**Corollary 4 (expressability of first-order predicate logic).** *$(\Phi_{\mathsf{1pred}}, \models_{\mathsf{1pred}})$ can express exactly the recursively enumerable languages. I.e., $L$ can be expressed by predicate logic iff $L$ is recursively enumerable.*

*Proof.* From [HR04] we know that $(\Phi_{\mathsf{1pred}}, \models_{\mathsf{1pred}})$ is sound and complete, so by Thm. 5 it cannot express a language which is not recursively enumerable. All there is to show is hence that $(\Phi_{\mathsf{1pred}}, \models_{\mathsf{1pred}})$ can express all languages which are recursively enumerable. We leave that as an exercise.

**Exercise 9** *Show that $(\Phi_{\mathsf{1pred}}, \models_{\mathsf{1pred}})$ can express all languages which are recursively enumerable. Use that if $L$ is recursively enumerable, then there exists a TM $M$ such that $x \in L$ iff $M$ accepts $x$. Then use Lemma 5, Exercise 6 and Exercise 5. You also have to use that $\mathsf{Accepts}$ reduces to $\mathsf{BPCP}$.*

**Corollary 5.** *The validity problem for the logic for arithmetic is not semi-decidable.*

*Proof.* If the validity problem for the logic for arithmetic was semi-decidable, then $\models_{\mathsf{AR}}$ would be recursively enumerable. From Lemma 6 we know that the logic for arithmetic can express a language which is not recursively enumerable, which, by Lemma 2, implies that $\models_{\mathsf{AR}}$ is not recursively enumerable.

**Corollary 6 (Gödel's incompleteness theorem).** *There does not exist a proof system $\vdash_{\mathsf{AR}}$ such that $(\Phi_{\mathsf{AR}}, \models_{\mathsf{AR}}, \vdash_{\mathsf{AR}})$ is sound and complete.*

*Proof.* From Thm. 5 we know that if $(\Phi_{\mathsf{AR}}, \models_{\mathsf{AR}}, \vdash_{\mathsf{AR}})$ is sound and complete, then it cannot express a language which is not recursively enumerable. From Lemma 6 we know that $(\Phi_{\mathsf{AR}}, \models_{\mathsf{AR}})$ can express a language which is not recursively enumerable.

**Corollary 7 (no complete axiomatization of arithmetic).** *For any first-order axiomatization of arithmetic, there exists a formula which is valid for arithmetic but which cannot be proved from the axioms. I.e., for any recursively enumerable set $\Gamma$ of first-order formulæ for which $\mathcal{M}_{\mathsf{AR}} \models_{\mathsf{AR}} \Gamma$, there exists $\phi$ such that $\mathcal{M}_{\mathsf{AR}} \models_{\mathsf{1pred}} \phi$ but $\Gamma \not\vdash_{\mathsf{1pred}} \phi$.*

**Exercise 10** *Prove Corollary 7.*

**Corollary 8 (incompleteness of first-order Peano axioms).** *Let $\Gamma$ be the first-order version of the Peano axioms. Then there exists a model $\mathcal{M}$ which is not the natural numbers, or isomorphic to the natural numbers, and for which $\mathcal{M} \vDash_{\mathsf{1pred}} \Gamma$.*

**Exercise 11** *Prove Corollary 8.*

**Corollary 9 (incompleteness of second-order predicate logic).** *There is no sound proof system for second-order predicate logic which is complete.*

*Proof.* If we had a sound and complete proof system for second-order predicate logic, then because the second-order Peano axioms is an axiomatization of $\mathcal{M}_{\mathsf{AR}}$ (Thm. 1), we would have a sound and complete logic for arithmetic (Thm. 3), contradicting Corollary 6.

## 5 Expressability of Propositional Logic

We prove Lemma 4, i.e., we show that if $L$ is recursive, then propositional logic can express $L$. Let $e : \Sigma^* \to \Phi$ be the function where $e(x) = \top$ if $x \in L$ and $e(x) = \bot$ if $x \notin L$. Since $L$ is recursive, we have that it is decidable whether $x \in L$. Therefore $e$ is Turing-computable. And, since $\top$ always is valid and $\bot$ always is invalid, it follows that $\vDash_{\mathsf{prop}} e(x)$ iff $x \in L$. Hence $e$ is an encoding which shows that propositional logic can express $L$.

It might seem silly to claim that propositional logic can express all recursive languages, when it is really the encoding which does the job. Note, however, that we have required the encoding to be Turing computable, so the encoding cannot help use express anything beyond the recursive languages. So, for a logic to express a language beyond recursive, the logic actually have to step in and give expressive powers extra to what is computable. Predicate logic is the first example of this happening.

## 6 Expressability of Predicate Logic

We show Lemma 5, i.e., we show that predicate logic can express $\mathsf{BPCP}$. In doing this, we use a logic with the following components:

- One constant symbol $\epsilon$.
- A unary function $\text{pre}_1$.
- A unary function $\text{pre}_2$.
- A binary predicate Reachable.

One possible model of this language is as follows:

- The universe is $A = \{1, 2\}^*$.
- $\epsilon$ denotes the empty string from $\{1, 2\}^*$.

- $\text{pre}_1(a) = 1\|a$, where $\|$ denotes concatenation. I.e., $\text{pre}_1$ prepends 1 to the input.
- $\text{pre}_2(a) = 2\|a$.
- The predicate Reachable is defined in context of a BPCP instance and we let $\text{Reachable}(a, b)$ be true iff there exists $k \in \mathbb{N}$, $k > 0$, and $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $a = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}$ and $b = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_k}$.

Let us call this model $\mathcal{M}_{\mathsf{str}}$. Notice that the BPCP instance has a solution iff $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}} \exists z\, \text{Reachable}(z, z)$. We now want to cook up a more involved predicate $\phi$ such that the BPCP instance has a solution iff $\models_{\mathsf{1pred}} \phi$. To some extend we are going to find an axiomatization $\Gamma = \{\phi_1, \phi_2\}$ of $\mathcal{M}_{\mathsf{str}}$ and then look at the formula $\models_{\mathsf{1pred}} \phi_1 \wedge \phi_2 \rightarrow \exists z\, \text{Reachable}(z, z)$ as a replacement of $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}} \exists z\, \text{Reachable}(z, z)$.

We use the following macro

$$\text{pre}_{a_1 a_2 \cdots a_m}(x) \overset{\text{DEF}}{=} \text{pre}_{a_1}(\text{pre}_{a_2}(\cdots \text{pre}_{a_m}(x) \cdots)) \ .$$

Now let

$$\phi_1 \overset{\text{DEF}}{=} \bigwedge_{i=1}^{n} \text{Reachable}(\text{pre}_{\alpha_i}(\epsilon), \text{pre}_{\beta_i}(\epsilon)) \ ,$$

$$\phi_2 \overset{\text{DEF}}{=} \forall x \forall y (\text{Reachable}(x, y) \rightarrow \bigwedge_{i=1}^{n} \text{Reachable}(\text{pre}_{\alpha_i}(x), \text{pre}_{\beta_i}(y))) \ ,$$

$$\text{ContainsR} \overset{\text{DEF}}{=} \phi_1 \wedge \phi_2 \ ,$$

$$\phi \overset{\text{DEF}}{=} \text{ContainsR} \rightarrow \exists z\, \text{Reachable}(z, z) \ .$$

We are going to show that $\models_{\mathsf{1pred}} \phi$ iff $\phi$ is an encoding of a Yes-instance.

**Exercise 12** *Show that $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}}$* ContainsR.

## 6.1 BPCP as a Reachability Problem

In analyzing $\phi$, we are going to look at the BPCP problem as a reachability problem in a transition system. A state is of the form $\langle x, y\rangle$ for strings $x$ and $y$. From any state $\langle x, y\rangle$ you are allowed to go to state $\langle x', y'\rangle$ if there exists $i \in \{1, \ldots, n\}$ such that $x' = x\|\alpha_i$ and $y' = y\|\beta_i$. The BPCP problem is then whether there is a non-empty sequence of transitions which allows you to start from $\langle \epsilon, \epsilon\rangle$ and move to a state of the form $\langle z, z\rangle$ for a string $z$.

Let us be a little more precise. We say that $\langle x, y\rangle \triangleright_i \langle x', y'\rangle$ iff $i \in \{1, \ldots, n\}$ and $\langle x', y'\rangle = \langle x\|\alpha_i, y\|\beta_i\rangle$; In words: you can reach $\langle x', y'\rangle$ in one step from $\langle x, y\rangle$ by using the pair of strings $(\alpha_i, \beta_i)$. We say that $\langle x, y\rangle \triangleright \langle x', y'\rangle$ if there exists $i$ such that $(x, y) \triangleright_i (x', y')$; In words: you can reach $\langle x', y'\rangle$ from $\langle x, y\rangle$ in one step using *some* $(\alpha_i, \beta_i)$. We say that $\langle x, y\rangle \triangleright^* \langle x', y'\rangle$ if there exists $\langle x_0, y_0\rangle, \ldots, \langle x_\ell, y_\ell\rangle$ such that $\langle x_0, y_0\rangle = \langle x, y\rangle$ and for $i = 0, \ldots, \ell - 1$, $\langle x_i, y_i\rangle \triangleright \langle x_{i+1}, y_{i+1}\rangle$, and

$\langle x_\ell, y_\ell\rangle = \langle x', y'\rangle$. The operator $^*$, which takes a relation like $\rhd$ and expresses its transitive closure $\rhd^*$ is known as the *Kleene star* operator.

We define $\langle x, y\rangle \rhd^+ \langle x', y'\rangle$ as $\langle x, y\rangle \rhd^* \langle x', y'\rangle$ but additionally require that $\ell > 0$. An equivalent way of saying this is that there exists $\langle x'', y''\rangle$ such that $\langle x, y\rangle \rhd \langle x'', y''\rangle$ and $\langle x'', y''\rangle \rhd^* \langle x', y'\rangle$.

It is easy to verify that we are dealing with a Yes-instance of the BPCP problem iff there exists a string $z$ such that $\langle 0, 0\rangle \rhd^+ \langle z, z\rangle$.

## 6.2 Generalizing to other Models

The above formulation applies only to the string model $\mathcal{M}_{\mathsf{str}}$. One clue to proving our theorem is to formulate a similar transition system for *any* model $\mathcal{M}$ of $\phi$.

Let $\mathcal{M}$ be any model of $\phi$. For the functions $\mathrm{pre}_0^{\mathcal{M}}$ and $\mathrm{pre}_1^{\mathcal{M}}$ specified by $\mathcal{M}$, we let

$$\mathrm{pre}_{a_1 a_2 \cdots a_m}^{\mathcal{M}}(x) \overset{\text{DEF}}{=} \mathrm{pre}_{a_1}^{\mathcal{M}}(\mathrm{pre}_{a_2}^{\mathcal{M}}(\cdots \mathrm{pre}_{a_m}^{\mathcal{M}}(x)\cdots)) \ .$$

We define $\langle a, b\rangle \succ_i \langle a', b'\rangle$ if $a' = \mathrm{pre}_{\alpha_i}^{\mathcal{M}}(a)$ and $b' = \mathrm{pre}_{\beta_i}^{\mathcal{M}}(b)$. We then define $\succ$, $\succ^*$ and $\succ^+$ from $\succ_i$ as we did for $\rhd$.

We let

$$R^{\mathcal{M}} = \{\langle a', b'\rangle \in A^{\mathcal{M}} \times A^{\mathcal{M}} | \langle \epsilon^{\mathcal{M}}, \epsilon^{\mathcal{M}}\rangle \succ^+ \langle a', b'\rangle\} \ ,$$

i.e., $R^{\mathcal{M}}$ is all those state which can be reached from $\langle \epsilon^{\mathcal{M}}, \epsilon^{\mathcal{M}}\rangle$. It turns out that the semantics of ContainsR is that the model of Reachable contains $R^{\mathcal{M}}$.

**Lemma 7.** *If $\mathcal{M} \models_{\mathsf{1pred}}$ ContainsR then $R^{\mathcal{M}} \subseteq \mathrm{Reachable}^{\mathcal{M}}$.*

*Proof.* If $\mathcal{M} \models_{\mathsf{1pred}} \phi_1$, then $\langle \epsilon^{\mathcal{M}}, \epsilon^{\mathcal{M}}\rangle \succ \langle a', b'\rangle$ implies that $\langle a', b'\rangle \in \mathrm{Reachable}^{\mathcal{M}}$. If $\mathcal{M} \models_{\mathsf{1pred}} \phi_2$, then $\langle a, b\rangle \in \mathrm{Reachable}^{\mathcal{M}}$ and $\langle a, b\rangle \succ \langle a', b'\rangle$ implies that $\langle a', b'\rangle \in \mathrm{Reachable}^{\mathcal{M}}$. Hence, if $\mathcal{M} \models_{\mathsf{1pred}} \phi_1 \wedge \phi_2$ then $R^{\mathcal{M}} \subseteq \mathrm{Reachable}^{\mathcal{M}}$. This can be shown using induction in the length of the path needed to reach $\langle a', b'\rangle$ from $\langle \epsilon^{\mathcal{M}}, \epsilon^{\mathcal{M}}\rangle$.

**Lemma 8.** *If $\mathcal{M} \models_{\mathsf{1pred}} \phi$ for all models $\mathcal{M}$, then the BPCP instance has a solution.*

*Proof.* If $\mathcal{M} \models_{\mathsf{1pred}} \phi$ for all models $\mathcal{M}$, then $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}} \phi$. Since $\phi \equiv \mathrm{ContainsR} \rightarrow \exists z\, \mathrm{Reachable}(z, z)$, it follows from $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}} \phi$ that *if* $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}}$ ContainsR *then* $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}} \exists z\, \mathrm{Reachable}(z, z)$. Since $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}}$ ContainsR (Exercise 12), it follows using Modus Ponens that $\mathcal{M}_{\mathsf{str}} \models_{\mathsf{1pred}} \exists z\, \mathrm{Reachable}(z, z)$, which means that there exists a string $z$ such that $\langle z, z\rangle \in \mathrm{Reachable}^{\mathcal{M}_{\mathsf{str}}}$. This means that the BPCP instance has a solution, which is what we had to prove.

**Lemma 9.** *If the BPCP instance has a solution, then $\mathcal{M} \models_{\mathsf{1pred}} \phi$ for all models $\mathcal{M}$.*

*Proof.* Let $\mathcal{M}$ be a arbitrary model. We show that if the BPCP instance has a solution, then $\mathcal{M} \vDash_{\mathsf{1pred}} \phi$. Since $\phi \equiv \text{ContainsR} \to \exists z\,\text{Reachable}(z,z)$, to show that $\mathcal{M} \vDash_{\mathsf{1pred}} \phi$ we have to show that *if* $\mathcal{M} \vDash_{\mathsf{1pred}}$ ContainsR *then* $\mathcal{M} \vDash_{\mathsf{1pred}}$ $\exists z\,\text{Reachable}(z,z)$. This is the same a showing that if the BPCP instance has a solution and $\mathcal{M} \vDash_{\mathsf{1pred}}$ ContainsR, then $\mathcal{M} \vDash_{\mathsf{1pred}} \exists z\,\text{Reachable}(z,z)$. So, assume that the BPCP instance has a solution and that $\mathcal{M} \vDash_{\mathsf{1pred}}$ ContainsR.

Since the BPCP instance has a solution there exists $k \in \mathbb{N}$, $k > 0$, and $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_k}$. Let $z = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}$. Since $\text{pre}_z^{\mathcal{M}} = \text{pre}_z^{\mathcal{M}}$ it follows that

$$\text{pre}^{\mathcal{M}}_{\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}} = \text{pre}^{\mathcal{M}}_{\beta_{i_1}\beta_{i_2}\cdots\beta_{i_k}} \ .$$

Clearly,

$$\text{pre}^{\mathcal{M}}_{\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}}(\epsilon^{\mathcal{M}}) = \text{pre}^{\mathcal{M}}_{\alpha_{i_1}}(\text{pre}^{\mathcal{M}}_{\alpha_{i_2}}(\cdots\text{pre}^{\mathcal{M}}_{\alpha_{i_k}}(\epsilon^{\mathcal{M}})\cdots))$$

and

$$\text{pre}^{\mathcal{M}}_{\beta_{i_1}\beta_{i_2}\cdots\beta_{i_k}}(\epsilon^{\mathcal{M}}) = \text{pre}^{\mathcal{M}}_{\beta_{i_1}}(\text{pre}^{\mathcal{M}}_{\beta_{i_2}}(\cdots\text{pre}^{\mathcal{M}}_{\beta_{i_k}}(\epsilon^{\mathcal{M}})\cdots)) \ .$$

So, if we let $z^{\mathcal{M}} = \text{pre}^{\mathcal{M}}_{\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}}(\epsilon^{\mathcal{M}})$, then $\langle z^{\mathcal{M}}, z^{\mathcal{M}}\rangle \in R^{\mathcal{M}}$. Since $\mathcal{M} \vDash_{\mathsf{1pred}}$ ContainsR implies that $R^{\mathcal{M}} \subseteq \text{Reachable}^{\mathcal{M}}$ it follows that $\langle z^{\mathcal{M}}, z^{\mathcal{M}}\rangle \in \text{Reachable}^{\mathcal{M}}$. This shows that $\mathcal{M} \vDash_{\mathsf{1pred}[z \mapsto z^{\mathcal{M}}]} \text{Reachable}(z,z)$, which implies that $\mathcal{M} \vDash_{\mathsf{1pred}} \exists z\,\text{Reachable}(z,z)$, which was what we had to show.

**Theorem 7.** *$\vDash_{\mathsf{1pred}} \phi$ iff the BPCP instance has a solution.*

*Proof.* $\vDash_{\mathsf{1pred}} \phi$ iff $\mathcal{M} \vDash_{\mathsf{1pred}} \phi$ for all models $\mathcal{M}$, so the theorem follows from Lemma 8 and Lemma 9.

## 7 Expressability of the Logic for Arithmetic

We show Lemma 6, i.e., we show that $(\Phi_{\mathsf{AR}}, \vDash_{\mathsf{AR}})$ can express the No-instances of APCP. We do this in two steps. First we show that $(\Phi_{\mathsf{AR}}, \vDash_{\mathsf{AR}})$ can express the Yes-instances of APCP. Then we show that if $(\Phi_{\mathsf{AR}}, \vDash_{\mathsf{AR}})$ can express the Yes-instances of a problem, then it can also express the No-instances.

In expressing the Yes-instances, we are going to look at the APCP problem as a transition system. A state is of the form $\langle x, y\rangle$ for natural numbers $x$ and $y$. From any state $\langle x, y\rangle$ you are allowed to go to state $\langle x', y'\rangle$ if there exists $i \in \{1, \ldots, n\}$ such that $x' = f_i(x)$ and $y' = g_i(y)$. The APCP problem is then whether there is a non-empty sequence of transitions which allows you to start from $\langle 0, 0\rangle$ and move to a state of the form $\langle z, z\rangle$.

A bit more formal: We say that $\langle x, y\rangle \rhd_i \langle x', y'\rangle$ iff $i \in \{1, \ldots, n\}$ and $\langle x', y'\rangle = \langle f_i(x), g_i(x)\rangle$. We say that $\langle x, y\rangle \rhd \langle x', y'\rangle$ iff there exists $i$ such that $(x, y) \rhd_i (x', y')$. We say that $\langle x, y\rangle \rhd^* \langle x', y'\rangle$ iff there exists $\langle x_0, y_0\rangle, \ldots, \langle x_\ell, y_\ell\rangle$ such that $\langle x_0, y_0\rangle = \langle x, y\rangle$ and for $i = 0, \ldots, \ell - 1$, $\langle x_i, y_i\rangle \rhd \langle x_{i+1}, y_{i+1}\rangle$, and $\langle x_\ell, y_\ell\rangle = \langle x', y'\rangle$. We say that $\langle x, y\rangle \rhd^+ \langle x', y'\rangle$ iff there exists $\langle x'', y''\rangle$ such that $\langle x, y\rangle \rhd \langle x'', y''\rangle$ and $\langle x'', y''\rangle \rhd^* \langle x', y'\rangle$.

It is easy to verify that we are dealing with a Yes-instance of the $\mathsf{APCP}$ problem iff there exists a natural number $z$ such that $\langle 0, 0\rangle \rhd^+ \langle z, z\rangle$.

We now show how to express the above in our first-order predicate logic for arithmetic. The main challenge is going to be that we do not have the Kleene star operator in our syntax, or semantics.

First we introduce the macro

$$(x, y) \rhd (x', y') \overset{\text{DEF}}{=} \bigvee_{i=1}^{n} (x' = f_i(x) \wedge y' = g_i(y)) \ ,$$

where $\bigvee_{i=1}^{n} \phi_i$ is a macro for $\phi_1 \vee \phi_2 \vee \cdots \vee \phi_n$. This clearly defines the transition relation $\rhd$.

Below we show how to express $(x, y) \rhd^* (x', y')$, but let us for a second believe that we can do this. We can then make the macros

$$\begin{aligned} \langle x, y\rangle \rhd^+ \langle x', y'\rangle &\overset{\text{DEF}}{=} \exists x''\exists y''((\langle x, y\rangle \rhd \langle x'', y''\rangle) \wedge (\langle x'', y''\rangle \rhd^* \langle x', y'\rangle)) \\ Y &\overset{\text{DEF}}{=} \exists z(\langle 0, 0\rangle \rhd^+ \langle z, z\rangle) \ . \end{aligned}$$

From our discussion above, it is clear that if we can define $\rhd^*$ to have the right meaning, then we are dealing with a Yes-instance iff $\models_{\mathsf{AR}} Y$.

## 7.1 Expressing Kleene Star using First-Order Arithmetic

All that remains is hence to show how to express the Kleene star operator. We will first going to solve the problem, assuming that we we are allowed to quantify over vectors of natural numbers. We then see how to express vectors using macros.

We add variable names $\hat{x}$ for vectors of natural numbers. The semantics of a variable $\hat{x}$ is an infinite vector $(x_0, x_1, x_2, \ldots)$ of natural numbers $x_i$. Given a vector $\hat{x}$ and a natural number $i$ we use $\hat{x}_i$ to denote the natural number $x_i$ at position $i$ in the vector. All vectors are only finitely filled and the default value is 0. I.e., there exists some natural number $\ell$ such that $x_i = 0$ for $i \geq \ell$. One can think of $\ell$ as the length — we simply have the convention that $\hat{x}_i = 0$ when $i$ is out of bound. Given the possibility of quantifying over vectors, we can express Kleene star as follows.

$$\begin{aligned} \langle x, y\rangle \rhd^* \langle x', y'\rangle \overset{\text{DEF}}{=} \exists \hat{x}\exists \hat{y}\exists k\ (&(\hat{x}_0 = x \wedge \hat{y}_0 = y)\wedge \\ &(\forall i((i < k) \rightarrow (\langle \hat{x}_i, \hat{y}_i\rangle \rhd \langle \hat{x}_{i+1}, \hat{y}_{i+1}\rangle)))\wedge \\ &(\hat{x}_k = x' \wedge \hat{y}_k = y')\ ) \ . \end{aligned}$$

It is easy to verify that this gives $\rhd^*$ the correct semantics in $\mathcal{M}_{\mathsf{AR}}$ extended to quantifying over vectors.

We then show how to express vectors using arithmetic. We will express a vector $\hat{x}$ of natural numbers as just two natural numbers. Let $p$ be a prime number.[3] Given a vector $\hat{x} = (x_0, x_1, x_2, \ldots)$ where $x_i < p$ for all $i$, we can store $\hat{x}$ as two natural numbers $(p, x)$, where

$$x = \sum_{i=0}^{\infty} x_i p^i = x_0 + x_1 p + x_2 p^2 + \cdots + x_i p^i + x_{i+1} p^{i+1} + \cdots \ ,$$

i.e., we look at $x_i$ as the $i$'th entry in the $p$-ary notation for $x$.

Since each $x_i < p$ it is clear that $x$ contains the information of all the $x_i$. There is even an easy way to retrieve $x_i$ from $x$. First note that

$$x \div p^i = x_i + x_{i+1} p + x_{i+2} p^2 + \cdots \ .$$

And,

$$(x_i + x_{i+1} p + x_{i+2} p^2 + \cdots) \bmod p = x_i \bmod p = x_i \ .$$

So,

$$(x \div p^i) \bmod p = x_i \ .$$

Here we meet the problem that we cannot express $p^i$ in our logic unless $i$ is a constant. To work around this problem, we will quantify over the prime powers $p^i$ directly instead of quantifying over $i$ and then computing $p^i$.

We make the macro

$$x[j, p] \overset{\text{DEF}}{=} (x \div j) \bmod p \ .$$

Note that if $\vDash_{\mathsf{AR}\ell} z = x[p^i, p]$ and $\ell(x) = \sum_{i=0}^{\infty} x_i p^i$, then $\ell(z) = x_i$. Note also that we cheated a little with notation here. We only have a macro for expressing that $y = x \div j$, we do not have a macro for expressing the function $x \div j$ as a term. Hence we cannot use the nested $(x \div j) \bmod p$ construction. There is, however, a simple way to accomplish the same. We can write $z = (x \div j) \bmod p$ as

$$\exists y (y = x \div j \wedge z = y \bmod p) \ .$$

Since the $y$ for which it holds that $y = x \div j$ is unique, this is the same as writing `let` $y = x \div j$ `in` $z = y \bmod p$. The expanded form is too cumbersome, so we use the macro $z = (x \div j) \bmod p$.

Finally, note that given two prime powers $p^i$ and $p^j$ we have that $i < j$ iff $p^i < p^j$. And, if we have some prime power $i = p^I$, then $i * p = p^{I+1}$. So, multiplying by $p$ correspond to adding one in the exponent. And, $p^0 = 1$. All in all, this allows us to express $\langle a, b \rangle \rhd^* \langle a', b' \rangle$ as follows.

$$\begin{aligned}
&\exists p(\mathrm{Prime}(p) \wedge \exists x \exists y \exists k(\mathrm{PrimePower}(k, p) \wedge \\
&\qquad ((x[1, p] = a \wedge y[1, p] = b) \wedge \\
&\qquad\quad \forall i((\mathrm{PrimePower}(i, p) \wedge i < k) \\
&\qquad\qquad \to (\langle x[i, p], y[i, p] \rangle \rhd \langle x[i * p, p], y[i * p, p] \rangle)))) \wedge \\
&\qquad (x[k, p] = a' \wedge y[k, p] = b')\ )) \ .
\end{aligned}$$

[3] Most of what we do now would work even if $p$ is not a prime number, but having $p$ be a prime makes certain things more convenient.

We now have $Y$ fully expressed in our logic, using a lot of macros, and we have that $\vDash_{\mathsf{AR}} Y$ iff we started from a Yes-instance. This shows that $(\Phi_{\mathsf{AR}}, \vDash_{\mathsf{AR}})$ allows to express the Yes-instances of APCP.

We then show how to express the No-instances. The obvious attempt works. Let

$$N \overset{\text{DEF}}{=} \neg Y \ .$$

Then $\vDash_{\mathsf{AR}} N$ iff we started from a No-instance.

This reason for this that $Y$ is a closed formula (no free variables), and if $\nvDash_{\mathsf{AR}} \phi$ for a closed formula, then $\vDash_{\mathsf{AR}} \neg\phi$, as the semantics is given using a fixed model.

**Exercise 13** *Show that if $\phi$ is a closed formula, then $\nvDash_{AR} \phi$ iff $\vDash_{AR} \neg\phi$.*

**Exercise 14** *Show that the languages that $(\Phi_{AR}, \vDash_{AR})$ can express is closed under complement. I.e., if $(\Phi_{AR}, \vDash_{AR})$ can express $L$, then $(\Phi_{AR}, \vDash_{AR})$ can express $L'$, the complement of $L$. Hint: there is extra work to do to handle the case where the encoding function does not output closed formulæ. Then show that the class of languages that $(\Phi_{1pred}, \vDash_{1pred})$ can express is* not *closed under complement.*

**Exercise 15** *Give a formula in our first-order logic for arithmetic which given three terms $x$, $y$ and $z$ expresses that $z = x^y$.*

## References

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: modelling and reasoning about systems (second edition).* Cambridge University Press, 2004.

[Mar02] John C. Martin. *Introduction to Languages and the Theory of Computation.* McGraw-Hill, Inc., New York, NY, USA, 2002.