5.7 Additional exercises

Exercise 5.6: Extend the procedure `polarities` (defined on page 271) to the first-order case. Then:

(a) Modify that implementation as specified in the first part of Exercise 4.1, so that every item in the output list consists of a two-element list

[*position polarity*]

comprising the *position* of the relevant occurrence as well as its *polarity*.

(b) Implement the first-order version of the binary procedure `polarities*`, as specified in the second part of Exercise 4.1.

(c) Extend the methods `M+` and `M-` of the third part of that same exercise to handle quantified sentences. □

Exercise 5.7: Similarly to Exercise 5.2, each of the following listings asserts some premises (possibly none) and defines a goal:

Listing 5.7.1

```
define goal :=
  ((forall x . P x <==> Q x) ==> (forall x . P x) <==> (forall x . Q x))
```

Listing 5.7.2

```
declare A: Boolean

assert premise-1 := (exists x . c R x & A)
assert premise-2 := (exists x . Q x & x T x)
assert premise-3 := (forall x . A & Q x ==> ~ S x)

define goal := (exists y . ~ S y & y T y)
```

Listing 5.7.3

```
assert premise-1 := (exists x . P x & forall y . Q y ==> x R y)
assert premise-2 := (forall x . P x ==> forall y . S y ==> ~ x R y)

define goal := (forall  x .  Q x ==> ~ S x)
```

Listing 5.7.4

```
declare f: [Object] -> Object

assert premise-1 := (forall x . x R x)

assert premise-2 := (forall x . f x = f f x)

define goal := (exists y . y R f y)
```

For each of these, clear the assumption base and then derive the `goal` from the corresponding premises. □

Exercise 5.8: Write a proof that derives the sentence

```
(exists x . P x | ~ P x)
```

from the empty assumption base. □

Exercise 5.9: One of the transformations implemented by the quantifier-swapping method of Section 5.4 is this:

$$(\exists x \,.\, \forall y \,.\, p) \longrightarrow (\forall y \,.\, \exists x \,.\, p).$$

Show that the converse transformation is not valid. □

Exercise 5.10: Write a unary method `M` that takes an arbitrary term t (of some sort `Object`) and derives the following theorem:

```
(P t <==> exists x . x = t & P x)
```

□

Exercise 5.11: Define a binary method `pick-all-witnesses` whose first argument is an existentially quantified premise with an arbitrary number $n \geq 0$ of existential quantifiers at the front, followed by a sentence q that is not existentially quantified. The idea here is to use **`pick-witness`** n times on the given premise in order to extract n witnesses (fresh variables) $w_1, \ldots, w_n$ for the n existential quantifiers, along with a corresponding instance q' of the body q. The second argument of `pick-all-witnesses` is a binary method M that will receive (a) the list of witnesses $w_1, \ldots, w_n$, in that order; and (b) the sentence q', which must be in the assumption base when M is called. The result of `pick-all-witnesses` will be whatever result is obtained by applying M to these two arguments.

Note that if n were fixed (statically known), we could implement this method simply by nesting n **`pick-witness`** deductions, followed by a call to M. But the key point here is that the number n of leading existential quantifiers is unknown, hence some form of automatic iteration is needed. □

* ***Exercise 5.12:*** Consider again sentence (5.25):

$$\texttt{(forall } x_1 \cdots x_n \texttt{ . } p \texttt{ <==> } q\texttt{)}$$

and the conjunction of (5.30) and (5.31):

$$\texttt{((forall } x_1 \cdots x_n \texttt{ . } p \texttt{ ==> } q\texttt{) \& (forall } x_1 \cdots x_n \texttt{ . } q \texttt{ ==> } p\texttt{))}.$$

Prove that the two are equivalent. More specifically, write a unary method that can accept either of these two sentences as a premise and derive the other. (Hint: Because n is not statically fixed, you will need some form of iteration akin to what was used in Exercise 5.11, involving a similar proof continuation as an argument.) □

Exercise 5.13: Define a unary method `move-quant` implementing the following inference rules:

(a)
$$\frac{\texttt{(forall } x \texttt{ . } p \texttt{ \& } q\texttt{)}}{\texttt{((forall } x \texttt{ . } p\texttt{) \& } q\texttt{)}}$$
provided that x does not occur free in q

(b)
$$\frac{\texttt{(forall } x \texttt{ . } p \texttt{ | } q\texttt{)}}{\texttt{((forall } x \texttt{ . } p\texttt{) | } q\texttt{)}}$$
provided that x does not occur free in q

(c)
$$\frac{\texttt{(exists } x \texttt{ . } p \texttt{ \& } q\texttt{)}}{\texttt{((exists } x \texttt{ . } p\texttt{) \& } q\texttt{)}}$$
provided that x does not occur free in q

(d)
$$\frac{\texttt{(exists } x \texttt{ . } p \texttt{ | } q\texttt{)}}{\texttt{((exists } x \texttt{ . } p\texttt{) | } q\texttt{)}}$$
provided that x does not occur free in q

The method should be bidirectional, capable of applying the inference rules in both directions. □

Exercise 5.14 (Quantifier Distribution): Define a method `quant-dist` implementing the rules for quantifier distribution presented in Section 5.4.

** ***Exercise 5.15 (The halting problem):*** In this exercise we formalize the halting problem [27] and prove its undecidability.[26] In doing so, we will need to talk about and quantify over *algorithms*, so we begin by introducing a corresponding domain:

```
domain Algorithm
```

26 This exercise was inspired by Burkholder [16], although our formalization here is different from—and, we believe, simpler than—the one given there.

Algorithms take inputs of various sorts (e.g., integers, strings, booleans, pairs, lists, and, quite importantly, even algorithms), so we introduce a polymorphic datatype representing algorithm inputs:

```
datatype (Input S) := (input S)
```

Thus, for instance, `(input 4)` represents a single integer input (specifically, the number 4), while `(input (pair 4 true))` represents an ordered-pair input, specifically, an ordered pair consisting of the integer 4 and the boolean `true` (see Section 10.1 for more details on ordered pairs). Likewise, algorithms can generate results of various sorts, so we introduce a polymorphic datatype for outputs:

```
datatype (Output S) := (answer S)
```

Next, we introduce a ternary predicate `outputs` that holds between an algorithm A, an input x, and an output y, iff A, when applied to input x, eventually produces output y. Because inputs and outputs are independently polymorphic, the predicate is parameterized over two sort variables:

```
declare outputs: (S, T) [Algorithm (Input S) (Output T)] -> Boolean
```

We also introduce a binary predicate `halts-on` that is intended to obtain between an algorithm A and an input x iff A halts on input x:

```
declare halts-on: (S) [Algorithm (Input S)] -> Boolean
```

We say that an algorithm A *decides* whether or not an algorithm B halts on some input x iff the following holds:

1. If B does halt on x, then when the pair (B,x) is given to A as input, A outputs `true`.
2. If, on the other hand, B does not halt on x, then when (B,x) is given to A as input, A outputs `false`.

In short, A takes (B,x) as input and outputs `true` iff B halts on x. We formalize this relation as follows:

```
declare decides-halting: (S) [Algorithm Algorithm S] -> Boolean

define [A B A' B'] := [?A ?B ?A' ?B']

assert* decides-halting-def :=
 ((decides-halting A B x) <==>
     (B halts-on input x ==>   outputs A (input B @ x) answer true)
   & (~ B halts-on input x ==> outputs A (input B @ x) answer false))
```

To solve the halting problem mechanically, we must have an algorithm A that is capable of deciding whether *any* algorithm B halts on *any* input x. Let us say that such an algorithm A is a *halting decider*:

```
declare halting-decider: (S) [(Algorithm S)] -> Boolean

assert halting-decider-def :=
  (forall A . halting-decider A <==> forall B x .
                                       (decides-halting A B x))
```

Note that so far we have not made any substantive assertions. Both of the above are conservative definitions which essentially hold by stipulation. They are guaranteed to be consistent. In fact, the only real premise we will need to assert for the proof is the following:

```
assert premise :=
  (forall A .
    exists B .
      forall x .
       ((outputs A (input x @ x) (answer true))  ==> ~ B halts-on input x)
      & ((outputs A (input x @ x) (answer false)) ==> B halts-on input x))
```

What does this sentence say, and why is it true? It says that for any algorithm A, there is an algorithm B that takes an input x and behaves as follows. It applies A to the input pair (x,x); if and when that application outputs the answer `true`, B goes into an infinite loop, and thus fails to halt on x (the outcome described on line 6); but if and when the application of A to (x,x) outputs the answer `false`, B halts (with some unspecified output), as described on line 8. The premise does not specify the behavior of B for any other possibility.

The premise is true simply because for any algorithm A, the existence of B can be ensured *by construction*. Given A, we can specify an algorithm B that behaves as described: B accepts an input x and runs A on the input pair (x,x). If the execution of A never terminates for that input pair, then of course B will also go on forever. But if A does halt, we examine its output and act as follows. If the output is `true`, we deliberately get into an infinite loop; if the output is `false`, we halt. In fact, if we regard algorithms as unary procedures, we can easily implement this construction in Athena as a higher-order procedure that takes A as input first and then produces the required algorithm B:

```
define loop := lambda () (loop)

define make-B :=
  lambda (A)
    lambda (x)
      match (A (x @ x)) {
        true  => (loop)
      | false => 'ok
      }
```

The problem asks you to prove that there exists no algorithm for deciding the halting problem, or in our notation:

```
define goal := (~ exists A . halting-decider A)
```

That is, give a proof D that derives `goal` from the three asserted sentences. □

Exercise 5.16 (Russell's paradox): Consider:

```
domain D
declare M: [D D] -> Boolean
```

and prove:

```
(~ exists x . forall y . y M x <==> ~ y M y).
```
(5.73)

When we interpret `M` as the membership relation on sets, so that (*x* `M` *y*) iff set *x* is a member of set *y*, then (5.73) states that there is no set whose members are all and only those sets that do not contain themselves. When we interpret `D` as a set of men and (*x* `M` *y*) as the statement that *x* is shaven by *y*, then the sentence says that there is no man who shaves all and only those men who do not shave themselves. (That is the popularized "barber" version of Russell's paradox.) It is a remarkable fact that many truths of set theory, such as the nonexistence of a set that contains all and only those sets that do not contain themselves, can be formulated and proved as tautologies of pure logic. For additional examples refer to the text by Kalish and Montague [56]. □

Exercise 5.17 (Drinker's principle): Prove the following:

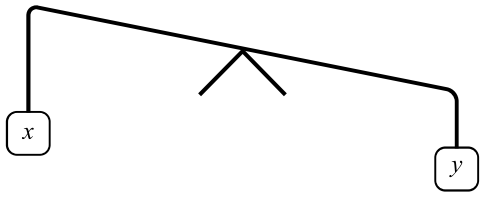
$$(\exists x \,.\, D\,x \Rightarrow \forall y \,.\, D\,y).$$

This is colloquially referred to as *the drinker's principle*: There is always some individual *x* such that, if *x* is drinking, then everyone is drinking. □

Exercise 5.18 (Theory of Measurement): This exercise is based on a paper by Patrick Suppes [99] on the theory of empirical measurement of extensive quantities, and in the present context, specifically on the measurement of mass. The vocabulary (or symbol set, in our terminology) of the theory contains a binary function symbol $*$ and a binary predicate `Q`. Intuitively, the domain of the intended interpretation consists of a large set of physical objects whose masses are to be measured by a scale. We interpret

$$(x \;\texttt{Q}\; y)$$

as the proposition that when we place *x* on the left pan of the scale and *y* on the right, we observe one of two outcomes: either the scale is perfectly balanced, which means that *x* and *y* have identical mass; or else *y* outweighs *x*:



In other words, $(x\ \mathtt{Q}\ y)$ holds iff the observed mass of x is less than or equal to that of y.[27] The operator $*$ can be thought as a way to produce a new object by combining two existing objects. Intuitively, we can think of $(x * y)$ as the operation of placing x followed by y on a pan of the scale. The theory is formalized through the following five axioms, written here in conventional notation:

[A1] $(\forall\, x,y,z \,.\, x\ \mathtt{Q}\ y \wedge y\ \mathtt{Q}\ x \Rightarrow x\ \mathtt{Q}\ z)$

[A2] $(\forall\, x,y,z \,.\, (x * y) * z\ \mathtt{Q}\ x * (y * z))$

[A3] $(\forall\, x,y,z \,.\, x\ \mathtt{Q}\ y \Rightarrow x * y\ \mathtt{Q}\ y * z)$

[A4] $(\forall\, x,y \,.\, \neg\, x\ \mathtt{Q}\ y \Rightarrow \exists\, z \,.\, x\ \mathtt{Q}\ y * z \wedge y * z\ \mathtt{Q}\ x)$

[A5] $(\forall\, x,y \,.\, \neg\, x * y\ \mathtt{Q}\ y)$

You are asked to derive the following theorems from these axioms:

[T1] $(\forall\, x \,.\, x\ \mathtt{Q}\ x)$

[T2] $(\forall\, x,y \,.\, x * y\ \mathtt{Q}\ y * x)$

[T3] $(\forall\, x_1,x_2,x_3,x_4 \,.\, x_1\ \mathtt{Q}\ x_2 \wedge x_3\ \mathtt{Q}\ x_4 \Rightarrow x_1 * x_3\ \mathtt{Q}\ x_2 * x_4)$

[T4] $(\forall\, x,y,z \,.\, x * (y * z)\ \mathtt{Q}\ (x * y) * z)$

[T5] $(\forall\, x,y \,.\, x\ \mathtt{Q}\ y \vee y\ \mathtt{Q}\ x)$

[T6] $(\forall\, x,y,z \,.\, x * z\ \mathtt{Q}\ y * z \Rightarrow x\ \mathtt{Q}\ y)$

[T7] $(\forall\, x,y,z \,.\, y * z\ \mathtt{Q}\ u \wedge x\ \mathtt{Q}\ y \Rightarrow x * z\ \mathtt{Q}\ u)$

[T8] $(\forall\, x,y,z \,.\, u\ \mathtt{Q}\ x * z \wedge x\ \mathtt{Q}\ y \Rightarrow u\ \mathtt{Q}\ y * z)$

In Athena notation, derive `theorem-1` through `theorem-8` of module `MT`:

```
module MT {
  domain D
  declare Q: [D D] -> Boolean
  declare *: [D D] -> D

  define [x y z z' u v x1 x2 x3 x4 x5] := [?x:D ?y:D ··· ?x5:D]
```

27 Of course, we could interpret Q directly as the usual numeric less-than relation, provided we also introduce a unary function symbol m from objects to real numbers, intended to represent their mass measurements. However, that would be putting the cart before the horse for the purpose of theories such as that of Suppes, whose task is a conceptual investigation of the fundamental conditions that a physical system must satisfy *in order* for numeric measurement to be possible (and meaningful). For more background on measurement theory, refer to the historical overview of the field by Díez [33].

```
  define [A1 A2 A3 A4 A5] :=
    (map close [(x Q y & y Q z ==> x Q z)
                ((x * y) * z Q x * (y * z))
                (x Q y ==> x * z Q z * y)
                (~ x Q y ==> exists z . x Q y * z & y * z Q x)
                (~ x * y Q x)])

  assert [A1 A2 A3 A4 A5]

  define [theorem-1 theorem-2 theorem-3 theorem-4
          theorem-5 theorem-6 theorem-7 theorem-8] :=
    (map close [(x Q x)
                (x * y Q y * x)
                (x1 Q x2 & x3 Q x4 ==> x1 * x3 Q x2 * x4)
                (x * (y * z) Q (x * y) * z)
                (x Q y | y Q x)
                (x * z Q y * z ==> x Q y)
                (y * z Q u & x Q y ==> x * z Q u)
                (u Q x * z & x Q y ==> u Q y * z)])
}
```

You are encouraged to write your own auxiliary methods to facilitate reasoning in this theory. □

5.8 Chapter notes

Important patterns of inference with quantifiers had already been introduced and studied by Aristotle in his logic of syllogisms [3], but they were limited to sentences with single quantifier occurrences, and therefore to unary predicates.[28] Quantified logic as we understand it today did not emerge until the work of Frege, Peano, and Russell in the late 19th and early 20th centuries. Frege [40], in particular, was the first to explicitly introduce a mathematically precise concept of a logical quantifier.

Multiple quantifier nesting allowed for a huge leap in expressiveness from Aristotle's unary predicates to arbitrary relations. Complicated propositions with intricate logical structure became readily expressible in predicate logic, which has turned out to be capable of capturing all of mathematics and even much of natural language. Frege's work also enabled a much clearer distinction between syntax and semantics, a distinction that was soon sharpened and investigated in depth by Hilbert, Tarski, and others, giving rise to the classic metatheoretic results for first-order logic, such as soundness, completeness, and compactness, as well as the negative results of Gödel. Well-known mathematical logic

28 Although Aristotle viewed quantified statements as binary relations obtaining between unary predicates ("terms").