Chapter udf

Undecidability

und.1 Introduction

tur:und:int:sec

It might seem obvious that not every function, even every arithmetical function, can be computable. There are just too many, whose behavior is too complicated. Functions defined from the decay of radioactive particles, for instance, or other chaotic or random behavior. Suppose we start counting 1-second intervals from a given time, and define the function $f(n)$ as the number of particles in the universe that decay in the n-th 1-second interval after that initial moment. This seems like a candidate for a function we cannot ever hope to compute.

But it is one thing to not be able to imagine how one would compute such functions, and quite another to actually prove that they are uncomputable. In fact, even functions that seem hopelessly complicated may, in an abstract sense, be computable. For instance, suppose the universe is finite in time—some day, in the very distant future the universe will contract into a single point, as some cosmological theories predict. Then there is only a finite (but incredibly large) number of seconds from that initial moment for which $f(n)$ is defined. And any function which is defined for only finitely many inputs is computable: we could list the outputs in one big table, or code it in one very big Turing machine state transition diagram.

We are often interested in special cases of functions whose values give the answers to yes/no questions. For instance, the question "is n a prime number?" is associated with the function

$$\text{isprime}(n) = \begin{cases} 1 & \text{if } n \text{ is prime} \\ 0 & \text{otherwise.} \end{cases}$$

We say that a yes/no question can be *effectively decided*, if the associated 1/0-valued function is effectively computable.

To prove mathematically that there are functions which cannot be effectively computed, or problems that cannot effectively decided, it is essential to fix a specific model of computation, and show that there are functions it cannot

compute or problems it cannot decide. We can show, for instance, that not every function can be computed by Turing machines, and not every problem can be decided by Turing machines. We can then appeal to the Church-Turing thesis to conclude that not only are Turing machines not powerful enough to compute every function, but no effective procedure can.

The key to proving such negative results is the fact that we can assign numbers to Turing machines themselves. The easiest way to do this is to enumerate them, perhaps by fixing a specific way to write down Turing machines and their programs, and then listing them in a systematic fashion. Once we see that this can be done, then the existence of Turing-uncomputable functions follows by simple cardinality considerations: the set of functions from $\mathbb{N}$ to $\mathbb{N}$ (in fact, even just from $\mathbb{N}$ to $\{0, 1\}$) are non-enumerable, but since we can enumerate all the Turing machines, the set of Turing-computable functions is only denumerable.

We can also define *specific* functions and problems which we can prove to be uncomputable and undecidable, respectively. One such problem is the so-called *Halting Problem*. Turing machines can be finitely described by listing their instructions. Such a description of a Turing machine, i.e., a Turing machine program, can of course be used as input to another Turing machine. So we can consider Turing machines that decide questions about other Turing machines. One particularly interesting question is this: "Does the given Turing machine eventually halt when started on input n?" It would be nice if there were a Turing machine that could decide this question: think of it as a quality-control Turing machine which ensures that Turing machines don't get caught in infinite loops and such. The interesting fact, which Turing proved, is that there cannot be such a Turing machine. There cannot be a single Turing machine which, when started on input consisting of a description of a Turing machine M and some number n, will always halt with either output 1 or 0 according to whether M machine would have halted when started on input n or not.

Once we have examples of specific undecidable problems we can use them to show that other problems are undecidable, too. For instance, one celebrated undecidable problem is the question, "Is the first-order formula φ valid?". There is no Turing machine which, given as input a first-order formula φ, is guaranteed to halt with output 1 or 0 according to whether φ is valid or not. Historically, the question of finding a procedure to effectively solve this problem was called simply "the" decision problem; and so we say that the decision problem is unsolvable. Turing and Church proved this result independently at around the same time, so it is also called the Church-Turing Theorem.

und.2 Enumerating Turing Machines

explanation

We can show that the set of all Turing machines is enumerable. This follows from the fact that each Turing machine can be finitely described. The set of states and the tape vocabulary are finite sets. The transition function is a partial function from $Q \times \Sigma$ to $Q \times \Sigma \times \{L, R, N\}$, and so likewise can be

tur:und:enu:sec

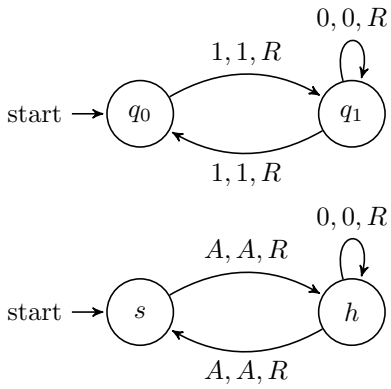

Figure und.1: Variants of the *Even* machine

tur:und:enu:
fig:variants

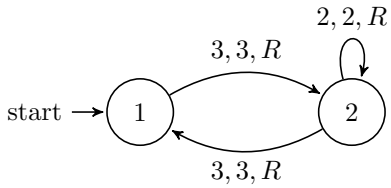

Figure und.2: A standard *Even* machine

tur:und:enu:
fig:standard-even

specified by listing its values for the finitely many argument pairs for which it is defined.

This is true as far as it goes, but there is a subtle difference. The definition of Turing machines made no restriction on what elements the set of states and tape alphabet can have. So, e.g., for every real number, there technically is a Turing machine that uses that number as a state. However, the *behavior* of the Turing machine is independent of which objects serve as states and vocabulary. Consider the two Turing machines in Figure und.1. These two diagrams correspond to two machines, M with the tape alphabet $\Sigma = \{\triangleright, 0, 1\}$ and set of states $\{q_0, q_1\}$, and M' with alphabet $\Sigma' = \{\triangleright, 0, A\}$ and states $\{s, h\}$. But their instructions are otherwise the same: M will halt on a sequence of n 1's iff n is even, and M' will halt on a sequence of n A's iff n is even. All we've done is rename 1 to A, q_0 to s, and q_1 to h. This example generalizes: we can think of Turing machines as the same as long as one results from the other by such a renaming of symbols and states. In fact, we can simply think of the symbols and states of a Turing machine as positive integers: instead of σ_0 think 1, instead of σ_1 think 2, etc.; $\triangleright$ is 1, 0 is 2, etc. In this way, the *Even* machine becomes the machine depicted in Figure und.2. We might call a

Turing machine with states and symbols that are positive integers a *standard* machine, and only consider standard machines from now on.[1]

We wanted to show that the set of Turing machines is enumerable, and with the above considerations in mind, it is enough to show that the set of standard Turing machines is enumerable. Suppose we are given a standard Turing machine $M = \langle Q, \Sigma, q_0, \delta \rangle$. How could we describe it using a finite string of positive integers? We'll first list the number of states, the states themselves, the number of symbols, the symbols themselves, and the starting state. (Remember, all of these are positive integers, since M is a standard machine.) What about δ? The set of possible arguments, i.e., pairs $\langle q, \sigma \rangle$, is finite, since Q and Σ are finite. So the information in δ is simply the finite list of all 5-tuples $\langle q, \sigma, q', \sigma', d \rangle$ where $\delta(q, \sigma) = \langle q', \sigma', D \rangle$, and d is a number that codes the direction D (say, 1 for L, 2 for R, and 3 for N).

In this way, every standard Turing machine can be described by a finite list of positive integers, i.e., as a sequence $s_M \in (\mathbb{Z}^+)^*$. For instance, the standard *Even* machine is coded by the sequence

$$2, \underbrace{1,2}_{Q}, 3, \overbrace{1,2,3}^{\Sigma}, 1, \underbrace{1,3,2,3,2}_{\delta(1,3)=\langle 2,3,R\rangle}, \overbrace{2,2,2,2,2}^{\delta(2,2)=\langle 2,2,R\rangle}, \underbrace{2,3,1,3,2}_{\delta(2,3)=\langle 1,3,R\rangle}.$$

Theorem und.1. *There are functions from $\mathbb{N}$ to $\mathbb{N}$ which are not Turing computable.*

Proof. We know that the set of finite sequences of positive integers $(\mathbb{Z}^+)^*$ is enumerable (**??**). This gives us that the set of descriptions of standard Turing machines, as a subset of $(\mathbb{Z}^+)^*$, is itself enumerable. Every Turing computable function $\mathbb{N}$ to $\mathbb{N}$ is computed by some (in fact, many) Turing machines. By renaming its states and symbols to positive integers (in particular, $\triangleright$ as 1, 0 as 2, and 1 as 3) we can see that every Turing computable function is computed by a standard Turing machine. This means that the set of all Turing computable functions from $\mathbb{N}$ to $\mathbb{N}$ is also enumerable.

On the other hand, the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not enumerable (**??**). If all functions were computable by some Turing machine, we could enumerate the set of all functions by listing all the descriptions of Turing machines that compute them. So there are some functions that are not Turing computable. □

Problem und.1. Can you think of a way to describe Turing machines that does not require that the states and alphabet symbols are explicitly listed? You may define your own notion of "standard" machine, but say something about why every Turing machine can be computed by a "standard" machine in your new sense.

[1] The terminology "standard machine" is not standard.

und.3 Universal Turing Machines

tur:und:uni:sec

In section und.2 we discussed how every Turing machine can be described by a finite sequence of integers. This sequence encodes the states, alphabet, start state, and instructions of the Turing machine. We also pointed out that the set of all of these descriptions is enumerable. Since the set of such descriptions is denumerable, this means that there is a surjective function from $\mathbb{N}$ to these descriptions. Such a surjective function can be obtained, for instance, using Cantor's zig-zag method. It gives us a way of enumerating all (descriptions) of Turing machines. If we fix one such enumeration, it now makes sense to talk of the 1st, 2nd, ..., eth Turing machine. These numbers are called *indices*.

Definition und.2. If M is the eth Turing machine (in our fixed enumeration), we say that e is an *index* of M. We write M_e for the eth Turing machine.

A machine may have more than one index, e.g., two descriptions of M may differ in the order in which we list its instructions, and these different descriptions will have different indices.

Importantly, it is possible to give the enumeration of Turing machine descriptions in such a way that we can effectively compute the description of M from its index, and to effectively compute an index of a machine M from its description. By the Church-Turing thesis, it is then possible to find a Turing machine which recovers the description of the Turing machine with index e and writes the corresponding description on its tape as output. The description would be a sequence of blocks of 1's (representing the positive integers in the sequence describing M_e).

Given this, it now becomes natural to ask: what functions of Turing machine indices are themselves computable by Turing machines? What properties of Turing machine indices can be decided by Turing machines? An example: the function that maps an index e to the number of states the Turing machine with index e has, is computable by a Turing machine. Here's what such a Turing machine would do: started on a tape containing a single block of e 1's, it would first decode e into its description. The description is now represented by a sequence of blocks of 1's on the tape. Since the first element in this sequence is the number of states. So all that has to be done now is to erase everything but the first block of 1's and then halt.

A remarkable result is the following:

tur:und:uni:thm:universal-tm

Theorem und.3. *There is a* universal Turing machine U *which, when started on input* $\langle e, n\rangle$

1. *halts iff* M_e *halts on input* n*, and*
2. *if* M_e *halts with output* m*, so does* U*.*

U thus computes the function $f : \mathbb{N} \times \mathbb{N} \nrightarrow \mathbb{N}$ given by $f(e, n) = m$ if M_e started on input n halts with output m, and undefined otherwise.

Proof. To actually produce U is basically impossible, since it is an extremely complicated machine. But we can describe in outline how it works, and then invoke the Church-Turing thesis. When it starts, U's tape contains a block of e 1's followed by a block of n 1's. It first "decodes" the index e to the right of the input n. This produces a list of numbers (i.e., blocks of 1's separated by 0's) that describes the instructions of machine M_e. U then writes the number of the start state of M_e and the number 1 on the tape to the right of the description of M_e. (Again, these are represented in unary, as blocks of 1's.) Next, it copies the input (block of n 1's) to the right—but it replaces each 1 by a block of three 1's (remember, the number of the 1 symbol is 3, 1 being the number of ▷ and 2 being the number of 0). At the left end of this sequence of blocks (separated by 0 symbols on the tape of U), it writes a single 1, the code for ▷.

U now has on its tape: the index e, the number n, the code number of the start state (the "current state"), the number of the initial head position 1 (the "current head position"), and the initial contents of the "tape" (a sequence of blocks of 1's representing the code numbers of the symbols of M_e—the "symbols"—separated by 0's).

It now simulates what M_e would do if started on input n, by doing the following:

1. Find the number k of the "current head position" (at the beginning, that's 1),
2. Move to the kth block in the "tape" to see what the "symbol" there is,
3. Find the instruction matching the current "state" and "symbol," tur:und:uni: find-inst
4. Move back to the kth block on the "tape" and replace the "symbol" there with the code number of the symbol M_e would write,
5. Move the head to where it records the current "state" and replace the number there with the number of the new state,
6. Move to the place where it records the "tape position" and erase a 1 or add a 1 (if the instruction says to move left or right, respectively).
7. Repeat.[2]

If M_e started on input n never halts, then U also never halts, so its output is undefined.

If in step (3) it turns out that the description of M_e contains no instruction for the current "state"/"symbol" pair, then M_e would halt. If this happens, U erases the part of its tape to the left of the "tape." For each block of three 1's (representing a 1 on M_e's tape), it writes a 1 on the left end of its own tape,

[2] We're glossing over some subtle difficulties here. E.g., U may need some extra space when it increases the counter where it keeps track of the "current head position"—in that case it will have to move the entire "tape" to the right.

and successively erases the "tape." When this is done, U's tape contains a single block of 1's of length m.

If U encounters something other than a block of three 1's on the "tape," it immediately halts. Since U's tape in this case does not contain a single block of 1's, its output is not a natural number, i.e., $f(e, n)$ is undefined in this case.□

und.4 The Halting Problem

tur:und:halsec

Assume we have fixed some enumeration of Turing machine descriptions. Each Turing machine thus receives an *index*: its place in the enumeration M_1, M_2, M_3, ... of Turing machine descriptions. explanation

We know that there must be non-Turing-computable functions: the set of Turing machine descriptions—and hence the set of Turing machines—is enumerable, but the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not. But we can find specific examples of non-computable functions as well. One such function is the halting function.

Definition und.4 (Halting function). The *halting function* h is defined as

$$h(e,n) = \begin{cases} 0 & \text{if machine } M_e \text{ does not halt for input } n \\ 1 & \text{if machine } M_e \text{ halts for input } n \end{cases}$$

Definition und.5 (Halting problem). The *Halting Problem* is the problem of determining (for any e, n) whether the Turing machine M_e halts for an input of n strokes.

We show that h is not Turing-computable by showing that a related function, s, is not Turing-computable. This proof relies on the fact that anything that can be computed by a Turing machine can be computed by a disciplined Turing machine (**??**), and the fact that two Turing machines can be hooked together to create a single machine (**??**). explanation

Definition und.6. The function s is defined as

$$s(e) = \begin{cases} 0 & \text{if machine } M_e \text{ does not halt for input } e \\ 1 & \text{if machine } M_e \text{ halts for input } e \end{cases}$$

Lemma und.7. *The function s is not Turing computable.*

Proof. We suppose, for contradiction, that the function s is Turing computable. Then there would be a Turing machine S that computes s. We may assume, without loss of generality, that when S halts, it does so while scanning the first square (i.e., that it is disciplined). This machine can be "hooked up" to another machine J, which halts if it is started on input 0 (i.e., if it reads 0 in the initial state while scanning the square to the right of the end-of-tape symbol), and otherwise wanders off to the right, never halting. $S \frown J$, the

machine created by hooking S to J, is a Turing machine, so it is M_e for some e (i.e., it appears somewhere in the enumeration). Start M_e on an input of e 1s. There are two possibilities: either M_e halts or it does not halt.

1. Suppose M_e halts for an input of e 1s. Then $s(e) = 1$. So S, when started on e, halts with a single 1 as output on the tape. Then J starts with a 1 on the tape. In that case J does not halt. But M_e is the machine $S \frown J$, so it should do exactly what S followed by J would do (i.e., in this case, wander off to the right and never halt). So M_e cannot halt for an input of e 1's.

2. Now suppose M_e does not halt for an input of e 1s. Then $s(e) = 0$, and S, when started on input e, halts with a blank tape. J, when started on a blank tape, immediately halts. Again, M_e does what S followed by J would do, so M_e must halt for an input of e 1's.

In each case we arrive at a contradiction with our assumption. This shows there cannot be a Turing machine S: s is not Turing computable. □

Theorem und.8 (Unsolvability of the Halting Problem). *The halting problem is unsolvable, i.e., the function h is not Turing computable.* *tur:und:hal: thm:halting-problem*

Proof. Suppose h were Turing computable, say, by a Turing machine H. We could use H to build a Turing machine that computes s: First, make a copy of the input (separated by a 0 symbol). Then move back to the beginning, and run H. We can clearly make a machine that does the former (see **??**), and if H existed, we would be able to "hook it up" to such a copier machine to get a new machine which would determine if M_e halts on input e, i.e., computes s. But we've already shown that no such machine can exist. Hence, h is also not Turing computable. □

Problem und.2. The Three Halting (3-Halt) problem is the problem of giving a decision procedure to determine whether or not an arbitrarily chosen Turing Machine halts for an input of three 1's on an otherwise blank tape. Prove that the 3-Halt problem is unsolvable.

Problem und.3. Show that if the halting problem is solvable for Turing machine and input pairs M_e and n where $e \neq n$, then it is also solvable for the cases where $e = n$.

Problem und.4. We proved that the halting problem is unsolvable if the input is a number e, which identifies a Turing machine M_e via an enumeration of all Turing machines. What if we allow the description of Turing machines from section und.2 directly as input? Can there be a Turing machine which decides the halting problem but takes as input descriptions of Turing machines rather than indices? Explain why or why not.

Problem und.5. Show that the *partial* function s' is defined as

$$s'(e) = \begin{cases} 1 & \text{if machine } M_e \text{ halts for input } e \\ \text{undefined} & \text{if machine } M_e \text{ does not halt for input } e \end{cases}$$

is Turing computable.

und.5 The Decision Problem

tur:und:dec:sec

We say that first-order logic is *decidable* iff there is an effective method for determining whether or not a given sentence is valid. As it turns out, there is no such method: the problem of deciding validity of first-order sentences is unsolvable.

In order to establish this important negative result, we prove that the decision problem cannot be solved by a Turing machine. That is, we show that there is no Turing machine which, whenever it is started on a tape that contains a first-order sentence, eventually halts and outputs either 1 or 0 depending on whether the sentence is valid or not. By the Church-Turing thesis, every function which is computable is Turing computable. So if this "validity function" were effectively computable at all, it would be Turing computable. If it isn't Turing computable, then, it also cannot be effectively computable.

Our strategy for proving that the decision problem is unsolvable is to reduce the halting problem to it. This means the following: We have proved that the function $h(e, w)$ that halts with output 1 if the Turing machine described by e halts on input w and outputs 0 otherwise, is not Turing computable. We will show that if there were a Turing machine that decides validity of first-order sentences, then there is also Turing machine that computes h. Since h cannot be computed by a Turing machine, there cannot be a Turing machine that decides validity either.

The first step in this strategy is to show that for every input w and a Turing machine M, we can effectively describe a sentence $\tau(M, w)$ representing the instruction set of M and the input w and a sentence $\alpha(M, w)$ expressing "M eventually halts" such that:

$$\vDash \tau(M, w) \to \alpha(M, w) \text{ iff } M \text{ halts for input } w.$$

The bulk of our proof will consist in describing these sentences $\tau(M, w)$ and $\alpha(M, w)$ and in verifying that $\tau(M, w) \to \alpha(M, w)$ is valid iff M halts on input w.

und.6 Representing Turing Machines

tur:und:rep:sec

In order to represent Turing machines and their behavior by a sentence of first-order logic, we have to define a suitable language. The language consists of two parts: predicate symbols for describing configurations of the machine, and expressions for numbering execution steps ("moments") and positions on the tape.

explanation

We introduce two kinds of predicate symbols, both of them 2-place: For each state q, a predicate symbol Q_q, and for each tape symbol σ, a predicate symbol S_σ. The former allow us to describe the state of M and the position of its tape head, the latter allow us to describe the contents of the tape.

In order to express the positions of the tape head and the number of steps executed, we need a way to express numbers. This is done using a constant symbol o, and a 1-place function $'$, the successor function. By convention it is written *after* its argument (and we leave out the parentheses). So o names the leftmost position on the tape as well as the time before the first execution step (the initial configuration), o' names the square to the right of the leftmost square, and the time after the first execution step, and so on. We also introduce a predicate symbol $<$ to express both the ordering of tape positions (when it means "to the left of") and execution steps (then it means "before").

Once we have the language in place, we list the "axioms" of $\tau(M, w)$, i.e., the sentences which, taken together, describe the behavior of M when run on input w. There will be sentences which lay down conditions on o, $'$, and $<$, sentences that describes the input configuration, and sentences that describe what the configuration of M is after it executes a particular instruction.

Definition und.9. Given a Turing machine $M = \langle Q, \Sigma, q_0, \delta \rangle$, the language $\mathcal{L}_M$ consists of: tur:und:rep: defn:tm-descr

1. A two-place predicate symbol $Q_q(x, y)$ for every state $q \in Q$. Intuitively, $Q_q(\overline{m}, \overline{n})$ expresses "after n steps, M is in state q scanning the mth square."

2. A two-place predicate symbol $S_\sigma(x, y)$ for every symbol $\sigma \in \Sigma$. Intuitively, $S_\sigma(\overline{m}, \overline{n})$ expresses "after n steps, the mth square contains symbol σ."

3. A constant symbol o

4. A one-place function symbol $'$

5. A two-place predicate symbol $<$

For each number n there is a canonical term $\overline{n}$, the *numeral* for n, which represents it in $\mathcal{L}_M$. $\overline{0}$ is o, $\overline{1}$ is o', $\overline{2}$ is o'', and so on. More formally:

$$\overline{0} = \mathsf{o}$$
$$\overline{n+1} = \overline{n}'$$

The sentences describing the operation of the Turing machine M on input $w = \sigma_{i_1} \dots \sigma_{i_k}$ are the following:

1. Axioms describing numbers and $<$:

   a) A sentence that says that every number is less than its successor:

   $$\forall x\, x < x'$$

b) A sentence that ensures that $<$ is transitive:

$$\forall x \, \forall y \, \forall z \, ((x < y \wedge y < z) \to x < z)$$

2. Axioms describing the input configuration:

   a) After 0 steps—before the machine starts—M is in the initial state q_0, scanning square 1:

   $$Q_{q_0}(\overline{1}, \overline{0})$$

   b) The first $k+1$ squares contain the symbols $\triangleright$, σ_{i_1}, …, σ_{i_k}:

   $$S_{\triangleright}(\overline{0}, \overline{0}) \wedge S_{\sigma_{i_1}}(\overline{1}, \overline{0}) \wedge \cdots \wedge S_{\sigma_{i_k}}(\overline{k}, \overline{0})$$

   c) Otherwise, the tape is empty:

   $$\forall x \, (\overline{k} < x \to S_0(x, \overline{0}))$$

3. Axioms describing the transition from one configuration to the next:

   For the following, let $\varphi(x, y)$ be the conjunction of all sentences of the form

   $$\forall z \, (((z < x \vee x < z) \wedge S_\sigma(z, y)) \to S_\sigma(z, y'))$$

   where $\sigma \in \Sigma$. We use $\varphi(\overline{m}, \overline{n})$ to express "other than at square m, the tape after $n+1$ steps is the same as after n steps."

tur:und:rep: rep-right

   a) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', R \rangle$, the sentence:

   $$\begin{aligned} &\forall x \, \forall y \, ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \to \\ &\qquad (Q_{q_j}(x', y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y))) \end{aligned}$$

   This says that if, after y steps, the machine is in state q_i scanning square x which contains symbol σ, then after $y+1$ steps it is scanning square $x+1$, is in state q_j, square x now contains σ', and every square other than x contains the same symbol as it did after y steps.

tur:und:rep: rep-left

   b) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', L \rangle$, the sentence:

   $$\begin{aligned} &\forall x \, \forall y \, ((Q_{q_i}(x', y) \wedge S_\sigma(x', y)) \to \\ &\qquad (Q_{q_j}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\ &\forall y \, ((Q_{q_i}(\mathsf{o}, y) \wedge S_\sigma(\mathsf{o}, y)) \to \\ &\qquad (Q_{q_j}(\mathsf{o}, y') \wedge S_{\sigma'}(\mathsf{o}, y') \wedge \varphi(\mathsf{o}, y))) \end{aligned}$$

   Take a moment to think about how this works: now we don't start with "if scanning square x …" but: "if scanning square $x+1$ …" A move to the left means that in the next step the machine is scanning square x. But the square that is written on is $x+1$. We do it this way since we don't have subtraction or a predecessor function.

Note that numbers of the form $x+1$ are 1, 2, ..., i.e., this doesn't cover the case where the machine is scanning square 0 and is supposed to move left (which of course it can't—it just stays put). That special case is covered by the second conjunction: it says that if, after y steps, the machine is scanning square 0 in state q_i and square 0 contains symbol σ, then after $y+1$ steps it's still scanning square 0, is now in state q_j, the symbol on square 0 is σ', and the squares other than square 0 contain the same symbols they contained ofter y steps.

c) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', N\rangle$, the sentence: tur:und:rep: rep-stay

$$\forall x\, \forall y\, ((Q_{q_i}(x,y) \wedge S_\sigma(x,y)) \rightarrow (Q_{q_j}(x,y') \wedge S_{\sigma'}(x,y') \wedge \varphi(x,y)))$$

Let $\tau(M,w)$ be the conjunction of all the above sentences for Turing machine M and input w.

In order to express that M eventually halts, we have to find a sentence that says "after some number of steps, the transition function will be undefined." Let X be the set of all pairs $\langle q, \sigma\rangle$ such that $\delta(q,\sigma)$ is undefined. Let $\alpha(M,w)$ then be the sentence

$$\exists x\, \exists y\, (\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x,y) \wedge S_\sigma(x,y)))$$

If we use a Turing machine with a designated halting state h, it is even easier: then the sentence $\alpha(M,w)$

$$\exists x\, \exists y\, Q_h(x,y)$$

expresses that the machine eventually halts.

Proposition und.10. *If $m < k$, then $\tau(M,w) \vDash \overline{m} < \overline{k}$* tur:und:rep: prop:mlessk

Proof. Exercise. □

Problem und.6. Prove Proposition und.10. (Hint: use induction on $k-m$).

und.7 Verifying the Representation

explanation In order to verify that our representation works, we have to prove two things. First, we have to show that if M halts on input w, then $\tau(M,w) \rightarrow \alpha(M,w)$ is valid. Then, we have to show the converse, i.e., that if $\tau(M,w) \rightarrow \alpha(M,w)$ is valid, then M does in fact eventually halt when run on input w. tur:und:ver: sec

The strategy for proving these is very different. For the first result, we have to show that a sentence of first-order logic (namely, $\tau(M,w) \rightarrow \alpha(M,w)$) is valid. The easiest way to do this is to give a derivation. Our proof is supposed to work for all M and w, though, so there isn't really a single sentence for which

we have to give a derivation, but infinitely many. So the best we can do is to prove by induction that, whatever M and w look like, and however many steps it takes M to halt on input w, there will be a derivation of $\tau(M,w) \to \alpha(M,w)$.

Naturally, our induction will proceed on the number of steps M takes before it reaches a halting configuration. In our inductive proof, we'll establish that for each step n of the run of M on input w, $\tau(M,w) \vDash \chi(M,w,n)$, where $\chi(M,w,n)$ correctly describes the configuration of M run on w after n steps. Now if M halts on input w after, say, n steps, $\chi(M,w,n)$ will describe a halting configuration. We'll also show that $\chi(M,w,n) \vDash \alpha(M,w)$, whenever $\chi(M,w,n)$ describes a halting configuration. So, if M halts on input w, then for some n, M will be in a halting configuration after n steps. Hence, $\tau(M,w) \vDash \chi(M,w,n)$ where $\chi(M,w,n)$ describes a halting configuration, and since in that case $\chi(M,w,n) \vDash \alpha(M,w)$, we get that $T(M,w) \vDash \alpha(M,w)$, i.e., that $\vDash \tau(M,w) \to \alpha(M,w)$.

The strategy for the converse is very different. Here we assume that $\vDash \tau(M,w) \to \alpha(M,w)$ and have to prove that M halts on input w. From the hypothesis we get that $\tau(M,w) \vDash \alpha(M,w)$, i.e., $\alpha(M,w)$ is true in every structure in which $\tau(M,w)$ is true. So we'll describe a structure $\mathfrak{M}$ in which $\tau(M,w)$ is true: its domain will be $\mathbb{N}$, and the interpretation of all the Q_q and S_σ will be given by the configurations of M during a run on input w. So, e.g., $\mathfrak{M} \vDash Q_q(\overline{m},\overline{n})$ iff T, when run on input w for n steps, is in state q and scanning square m. Now since $\tau(M,w) \vDash \alpha(M,w)$ by hypothesis, and since $\mathfrak{M} \vDash \tau(M,w)$ by construction, $\mathfrak{M} \vDash \alpha(M,w)$. But $\mathfrak{M} \vDash \alpha(M,w)$ iff there is some $n \in |\mathfrak{M}| = \mathbb{N}$ so that M, run on input w, is in a halting configuration after n steps.

Definition und.11. Let $\chi(M,w,n)$ be the sentence

$$Q_q(\overline{m},\overline{n}) \wedge S_{\sigma_0}(\overline{0},\overline{n}) \wedge \cdots \wedge S_{\sigma_k}(\overline{k},\overline{n}) \wedge \forall x\,(\overline{k} < x \to S_0(x,\overline{n}))$$

where q is the state of M at time n, M is scanning square m at time n, square i contains symbol σ_i at time n for $0 \leq i \leq k$ and k is the right-most non-blank square of the tape at time 0, or the right-most square the tape head has visited after n steps, whichever is greater.

tur:und:ver: lem:halt-config-implies-halt

Lemma und.12. *If M run on input w is in a halting configuration after n steps, then $\chi(M,w,n) \vDash \alpha(M,w)$.*

Proof. Suppose that M halts for input w after n steps. There is some state q, square m, and symbol σ such that:

1. After n steps, M is in state q scanning square m on which σ appears.
2. The transition function $\delta(q,\sigma)$ is undefined.

$\chi(M,w,n)$ is the description of this configuration and will include the clauses $Q_q(\overline{m},\overline{n})$ and $S_\sigma(\overline{m},\overline{n})$. These clauses together imply $\alpha(M,w)$:

$$\exists x\,\exists y\,(\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x,y) \wedge S_\sigma(x,y)))$$

since $Q_{q'}(\overline{m},\overline{n}) \wedge S_{\sigma'}(\overline{m},\overline{n}) \vDash \bigvee_{\langle q,\sigma\rangle \in X}(Q_q(\overline{m},\overline{n}) \wedge S_\sigma(\overline{m},\overline{n}))$, as $\langle q',\sigma'\rangle \in X$.□

explanation

So if M halts for input w, then there is some n such that $\chi(M,w,n) \vDash \alpha(M,w)$. We will now show that for any time n, $\tau(M,w) \vDash \chi(M,w,n)$.

Lemma und.13. *For each n, if M has not halted after n steps, $\tau(M,w) \vDash \chi(M,w,n)$.* *tur:und:ver: lem:config*

Proof. Induction basis: If $n = 0$, then the conjuncts of $\chi(M,w,0)$ are also conjuncts of $\tau(M,w)$, so entailed by it.

Inductive hypothesis: If M has not halted before the nth step, then $\tau(M,w) \vDash \chi(M,w,n)$. We have to show that (unless $\chi(M,w,n)$ describes a halting configuration), $\tau(M,w) \vDash \chi(M,w,n+1)$.

Suppose $n > 0$ and after n steps, M started on w is in state q scanning square m. Since M does not halt after n steps, there must be an instruction of one of the following three forms in the program of M:

1. $\delta(q,\sigma) = \langle q',\sigma',R\rangle$ *tur:und:ver: right*
2. $\delta(q,\sigma) = \langle q',\sigma',L\rangle$ *tur:und:ver: left*
3. $\delta(q,\sigma) = \langle q',\sigma',N\rangle$ *tur:und:ver: stay*

We will consider each of these three cases in turn.

1. Suppose there is an instruction of the form (1). By Definition und.9(3a), this means that

$$\forall x\, \forall y\, ((Q_q(x,y) \wedge S_\sigma(x,y)) \rightarrow (Q_{q'}(x',y') \wedge S_{\sigma'}(x,y') \wedge \varphi(x,y)))$$

is a conjunct of $\tau(M,w)$. This entails the following sentence (universal instantiation, $\overline{m}$ for x and $\overline{n}$ for y):

$$(Q_q(\overline{m},\overline{n}) \wedge S_\sigma(\overline{m},\overline{n})) \rightarrow (Q_{q'}(\overline{m}',\overline{n}') \wedge S_{\sigma'}(\overline{m},\overline{n}') \wedge \varphi(\overline{m},\overline{n})).$$

By induction hypothesis, $\tau(M,w) \vDash \chi(M,w,n)$, i.e.,

$$Q_q(\overline{m},\overline{n}) \wedge S_{\sigma_0}(\overline{0},\overline{n}) \wedge \cdots \wedge S_{\sigma_k}(\overline{k},\overline{n}) \wedge \forall x\, (\overline{k} < x \rightarrow S_0(x,\overline{n}))$$

Since after n steps, tape square m contains σ, the corresponding conjunct is $S_\sigma(\overline{m},\overline{n})$, so this entails:

$$Q_q(\overline{m},\overline{n}) \wedge S_\sigma(\overline{m},\overline{n})$$

We now get

$$\begin{aligned}&Q_{q'}(\overline{m}', \overline{n}') \wedge S_{\sigma'}(\overline{m}, \overline{n}') \wedge \\ &\quad S_{\sigma_0}(\overline{0}, \overline{n}') \wedge \cdots \wedge S_{\sigma_k}(\overline{k}, \overline{n}') \wedge \\ &\quad \forall x\, (\overline{k} < x \to S_0(x, \overline{n}'))\end{aligned}$$

as follows: The first line comes directly from the consequent of the preceding conditional, by modus ponens. Each conjunct in the middle line—which excludes $S_{\sigma_m}(\overline{m}, \overline{n}')$—follows from the corresponding conjunct in $\chi(M, w, n)$ together with $\varphi(\overline{m}, \overline{n})$.

If $m < k$, $\tau(M, w) \vdash \overline{m} < \overline{k}$ (Proposition und.10) and by transitivity of $<$, we have $\forall x\, (\overline{k} < x \to \overline{m} < x)$. If $m = k$, then $\forall x\, (\overline{k} < x \to \overline{m} < x)$ by logic alone. The last line then follows from the corresponding conjunct in $\chi(M, w, n)$, $\forall x\, (\overline{k} < x \to \overline{m} < x)$, and $\varphi(\overline{m}, \overline{n})$. If $m < k$, this already is $\chi(M, w, n+1)$.

Now suppose $m = k$. In that case, after $n+1$ steps, the tape head has also visited square $k+1$, which now is the right-most square visited. So $\chi(M, w, n+1)$ has a new conjunct, $S_0(\overline{k}', \overline{n}')$, and the last conjunct is $\forall x\, (\overline{k}' < x \to S_0(x, \overline{n}'))$. We have to verify that these two sentences are also implied.

We already have $\forall x\, (\overline{k} < x \to S_0(x, \overline{n}'))$. In particular, this gives us $\overline{k} < \overline{k}' \to S_0(\overline{k}', \overline{n}')$. From the axiom $\forall x\, x < x'$ we get $\overline{k} < \overline{k}'$. By modus ponens, $S_0(\overline{k}', \overline{n}')$ follows.

Also, since $\tau(M, w) \vdash \overline{k} < \overline{k}'$, the axiom for transitivity of $<$ gives us $\forall x\, (\overline{k}' < x \to S_0(x, \overline{n}'))$. (We leave the verification of this as an exercise.)

2. Suppose there is an instruction of the form (2). Then, by Definition und.9(3b),

$$\begin{aligned}&\forall x\, \forall y\, ((Q_q(x', y) \wedge S_\sigma(x', y)) \to \\ &\quad (Q_{q'}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\ &\forall y\, ((Q_{q_i}(\mathsf{o}, y) \wedge S_\sigma(\mathsf{o}, y)) \to \\ &\quad (Q_{q_j}(\mathsf{o}, y') \wedge S_{\sigma'}(\mathsf{o}, y') \wedge \varphi(\mathsf{o}, y)))\end{aligned}$$

is a conjunct of $\tau(M, w)$. If $m > 0$, then let $l = m - 1$ (i.e., $m = l + 1$). The first conjunct of the above sentence entails the following:

$$\begin{aligned}&(Q_q(\overline{l}', \overline{n}) \wedge S_\sigma(\overline{l}', \overline{n})) \to \\ &\quad (Q_{q'}(\overline{l}, \overline{n}') \wedge S_{\sigma'}(\overline{l}', \overline{n}') \wedge \varphi(\overline{l}, \overline{n}))\end{aligned}$$

Otherwise, let $l = m = 0$ and consider the following sentence entailed by the second conjunct:

$$\begin{aligned}&((Q_{q_i}(\mathsf{o}, \overline{n}) \wedge S_\sigma(\mathsf{o}, \overline{n})) \to \\ &\quad (Q_{q_j}(\mathsf{o}, \overline{n}') \wedge S_{\sigma'}(\mathsf{o}, \overline{n}') \wedge \varphi(\mathsf{o}, \overline{n})))\end{aligned}$$

Either sentence implies

$$Q_{q'}(\overline{l}, \overline{n}') \wedge S_{\sigma'}(\overline{m}, \overline{n}') \wedge$$
$$S_{\sigma_0}(\overline{0}, \overline{n}') \wedge \cdots \wedge S_{\sigma_k}(\overline{k}, \overline{n}') \wedge$$
$$\forall x\, (\overline{k} < x \to S_0(x, \overline{n}'))$$

as before. (Note that in the first case, $\overline{l}' \equiv \overline{l+1} \equiv \overline{m}$ and in the second case $\overline{l} \equiv \mathsf{0}$.) But this just is $\chi(M, w, n+1)$.

3. Case (3) is left as an exercise.

We have shown that for any n, $\tau(M, w) \vDash \chi(M, w, n)$. □

Problem und.7. Complete case (3) of the proof of Lemma und.13.

Problem und.8. Give a derivation of $S_{\sigma_i}(\overline{i}, \overline{n}')$ from $S_{\sigma_i}(\overline{i}, \overline{n})$ and $\varphi(m, n)$ (assuming $i \neq m$, i.e., either $i < m$ or $m < i$).

Problem und.9. Give a derivation of $\forall x\, (\overline{k}' < x \to S_0(x, \overline{n}'))$ from $\forall x\, (\overline{k} < x \to S_0(x, \overline{n}'))$, $\forall x\, x < x'$, and $\forall x \forall y \forall z\, ((x < y \wedge y < z) \to x < z)$.)

Lemma und.14. *If M halts on input w, then $\tau(M, w) \to \alpha(M, w)$ is valid.* *tur:und:ver: lem:valid-if-halt*

Proof. By Lemma und.13, we know that, for any time n, the description $\chi(M, w, n)$ of the configuration of M at time n is entailed by $\tau(M, w)$. Suppose M halts after k steps. At that point, it will be scanning square m, for some $m \in \mathbb{N}$. Then $\chi(M, w, k)$ describes a halting configuration of M, i.e., it contains as conjuncts both $Q_q(\overline{m}, \overline{k})$ and $S_\sigma(\overline{m}, \overline{k})$ with $\delta(q, \sigma)$ undefined. Thus, by Lemma und.12, $\chi(M, w, k) \vDash \alpha(M, w)$. But since $\tau(M, w) \vDash \chi(M, w, k)$, we have $\tau(M, w) \vDash \alpha(M, w)$ and therefore $\tau(M, w) \to \alpha(M, w)$ is valid. □

explanation

To complete the verification of our claim, we also have to establish the reverse direction: if $\tau(M, w) \to \alpha(M, w)$ is valid, then M does in fact halt when started on input w.

Lemma und.15. *If $\vDash \tau(M, w) \to \alpha(M, w)$, then M halts on input w.* *tur:und:ver: lem:halt-if-valid*

Proof. Consider the $\mathcal{L}_M$-structure $\mathfrak{M}$ with domain $\mathbb{N}$ which interprets $\mathsf{0}$ as 0, $'$ as the successor function, and $<$ as the less-than relation, and the predicates Q_q and S_σ as follows:

$$Q_q^{\mathfrak{M}} = \{\langle m, n\rangle : \begin{array}{l}\text{started on } w, \text{ after } n \text{ steps,} \\ M \text{ is in state } q \text{ scanning square } m\end{array}\}$$

$$S_\sigma^{\mathfrak{M}} = \{\langle m, n\rangle : \begin{array}{l}\text{started on } w, \text{ after } n \text{ steps,} \\ \text{square } m \text{ of } M \text{ contains symbol } \sigma\end{array}\}$$

In other words, we construct the structure $\mathfrak{M}$ so that it describes what M started on input w actually does, step by step. Clearly, $\mathfrak{M} \vDash \tau(M, w)$. If $\vDash \tau(M, w) \to \alpha(M, w)$, then also $\mathfrak{M} \vDash \alpha(M, w)$, i.e.,

$$\mathfrak{M} \vDash \exists x \, \exists y \, (\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y))).$$

As $|\mathfrak{M}| = \mathbb{N}$, there must be m, $n \in \mathbb{N}$ so that $\mathfrak{M} \vDash Q_q(\overline{m}, \overline{n}) \wedge S_\sigma(\overline{m}, \overline{n})$ for some q and σ such that $\delta(q, \sigma)$ is undefined. By the definition of $\mathfrak{M}$, this means that M started on input w after n steps is in state q and reading symbol σ, and the transition function is undefined, i.e., M has halted. □

und.8 The Decision Problem is Unsolvable

tur:und:uns: sec
tur:und:uns: thm:decision-prob

Theorem und.16. *The decision problem is unsolvable: There is no Turing machine D, which when started on a tape that contains a sentence ψ of first-order logic as input, D eventually halts, and outputs 1 iff ψ is valid and 0 otherwise.*

Proof. Suppose the decision problem were solvable, i.e., suppose there were a Turing machine D. Then we could solve the halting problem as follows. We construct a Turing machine E that, given as input the number e of Turing machine M_e and input w, computes the corresponding sentence $\tau(M_e, w) \to \alpha(M_e, w)$ and halts, scanning the leftmost square on the tape. The machine $E \frown D$ would then, given input e and w, first compute $\tau(M_e, w) \to \alpha(M_e, w)$ and then run the decision problem machine D on that input. D halts with output 1 iff $\tau(M_e, w) \to \alpha(M_e, w)$ is valid and outputs 0 otherwise. By Lemma und.15 and Lemma und.14, $\tau(M_e, w) \to \alpha(M_e, w)$ is valid iff M_e halts on input w. Thus, $E \frown D$, given input e and w halts with output 1 iff M_e halts on input w and halts with output 0 otherwise. In other words, $E \frown D$ would solve the halting problem. But we know, by Theorem und.8, that no such Turing machine can exist. □

tur:und:uns: cor:undecidable-sat

Corollary und.17. *It is undecidable if an arbitrary sentence of first-order logic is satisfiable.*

Proof. Suppose satisfiability were decidable by a Turing machine S. Then we could solve the decision problem as follows: Given a sentence B as input, move ψ to the right one square. Return to square 1 and write the symbol ¬.

Now run the Turing machine S. It eventually halts with output either 1 (if $\neg\psi$ is satisfiable) or 0 (if $\neg\psi$ is unsatisfiable) on the tape. If there is a 1 on square 1, erase it; if square 1 is empty, write a 1, then halt.

This Turing machine always halts, and its output is 1 iff $\neg\psi$ is unsatisfiable and 0 otherwise. Since ψ is valid iff $\neg\psi$ is unsatisfiable, the machine outputs 1 iff ψ is valid, and 0 otherwise, i.e., it would solve the decision problem. □

explanation

So there is no Turing machine which always gives a correct "yes" or "no" answer to the question "Is ψ a valid sentence of first-order logic?" However, there *is* a Turing machine that always gives a correct "yes" answer—but simply does not halt if the answer is "no." This follows from the soundness and completeness theorem of first-order logic, and the fact that derivations can be effectively enumerated.

Theorem und.18. *Validity of first-order sentences is semi-decidable: There is a Turing machine E, which when started on a tape that contains a sentence ψ of first-order logic as input, E eventually halts and outputs 1 iff ψ is valid, but does not halt otherwise.* tur:und:uns: thm:valid-ce

Proof. All possible derivations of first-order logic can be generated, one after another, by an effective algorithm. The machine E does this, and when it finds a derivation that shows that $\vdash \psi$, it halts with output 1. By the soundness theorem, if E halts with output 1, it's because $\vDash \psi$. By the completeness theorem, if $\vDash \psi$ there is a derivation that shows that $\vdash \psi$. Since E systematically generates all possible derivations, it will eventually find one that shows $\vdash \psi$, so will eventually halt with output 1. □

und.9 Trakthenbrot's Theorem

explanation

In section und.6 we defined sentences $\tau(M, w)$ and $\alpha(M, w)$ for a Turing machine M and input string w. Then we showed in Lemma und.14 and Lemma und.15 that $\tau(M, w) \to \alpha(M, w)$ is valid iff M, started on input w, eventually halts. Since the Halting Problem is undecidable, this implies that validity and satisfiability of sentences of first-order logic is undecidable (Theorem und.16 and Corollary und.17). tur:und:tra: sec

But validity and satisfiability of sentences is defined for arbitrary structures, finite or infinite. You might suspect that it is easier to decide if a sentence is satisfiable in a finite structure (or valid in all finite structures). We can adapt the proof of the unsolvability of the decision problem so that it shows this is not the case.

First, if you go back to the proof of Lemma und.15, you'll see that what we did there is produce a model $\mathfrak{M}$ of $\tau(M, w)$ which describes exactly what machine M does when started on input w. The domain of that model was $\mathbb{N}$, i.e., infinite. But if M actually halts on input w, we can build a finite model $\mathfrak{M}'$ in the same way. Suppose M started on input w halts after k steps. Take as domain $|\mathfrak{M}'|$ the set $\{0, \ldots, n\}$, where n is the larger of k and the length of w, and let

$$\prime^{\mathfrak{M}'}(x) = \begin{cases} x+1 & \text{if } x < n \\ n & \text{otherwise,} \end{cases}$$

and $\langle x, y\rangle \in <^{\mathfrak{M}'}$ iff $x < y$ or $x = y = n$. Otherwise $\mathfrak{M}'$ is defined just like $\mathfrak{M}$. By the definition of $\mathfrak{M}'$, just like in the proof of Lemma und.15, $\mathfrak{M}' \vDash \tau(M, w)$.

And since we assumed that M halts on input w, $\mathfrak{M}' \vDash \alpha(M, w)$. So, $\mathfrak{M}'$ is a finite model of $\tau(M, w) \wedge \alpha(M, w)$ (note that we've replaced $\rightarrow$ with $\wedge$).

We are halfway to a proof: we've shown that if M halts on input w, then $\tau(M, e) \wedge \alpha(M, w)$ has a finite model. Unfortunately, the "only if" direction does not hold. For instance, if M after n steps is in state q and reads a symbol σ, and $\delta(q, \sigma) = \langle q, \sigma, N\rangle$, then the configuration after $n + 1$ steps is exactly the same as the configuration after n steps (same state, same head position, same tape contents). But the machine never halts; it's in an infinite loop. The corresponding structure $\mathfrak{M}'$ above satisfies $\tau(M, w)$ but not $\alpha(M, w)$. (In it, the values of $\overline{n + l}$ are all the same, so it is finite). But by changing $\tau(M, w)$ in a suitable way we can rule out structures like this.

Consider the sentences describing the operation of the Turing machine M on input $w = \sigma_{i_1} \dots \sigma_{i_k}$:

1. Axioms describing numbers and $<$ (just like in the definition of $\tau(M, w)$ in section und.6).

2. Axioms describing the input configuration: just like in the definition of $\tau(M, w)$.

3. Axioms describing the transition from one configuration to the next:

   For the following, let $\varphi(x, y)$ be as before, and let

   $$\psi(y) \equiv \forall x\, (x < y \rightarrow x \neq y).$$

   tur:und:tra: rep-right

   a) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', R\rangle$, the sentence:

   $$\begin{aligned}&\forall x\, \forall y\, ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \rightarrow \\ &\qquad (Q_{q_j}(x', y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y) \wedge \psi(y')))\end{aligned}$$

   tur:und:tra: rep-left

   b) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', L\rangle$, the sentence

   $$\begin{aligned}&\forall x\, \forall y\, ((Q_{q_i}(x', y) \wedge S_\sigma(x', y)) \rightarrow \\ &\qquad (Q_{q_j}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\ &\forall y\, ((Q_{q_i}(\mathsf{o}, y) \wedge S_\sigma(\mathsf{o}, y)) \rightarrow \\ &\qquad (Q_{q_j}(\mathsf{o}, y') \wedge S_{\sigma'}(\mathsf{o}, y') \wedge \varphi(\mathsf{o}, y) \wedge \psi(y')))\end{aligned}$$

   tur:und:tra: rep-stay

   c) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', N\rangle$, the sentence:

   $$\begin{aligned}&\forall x\, \forall y\, ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \rightarrow \\ &\qquad (Q_{q_j}(x, y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y) \wedge \psi(y')))\end{aligned}$$

   As you can see, the sentences describing the transitions of M are the same as the corresponding sentence in $\tau(M, w)$, except we add $\psi(y')$ at the end. $\psi(y')$ ensures that the number y' of the "next" configuration is different from all previous numbers o, o', ...

Let $\tau'(M,w)$ be the conjunction of all the above sentences for Turing machine M and input w.

Lemma und.19. *If M started on input w halts, then $\tau'(M,w) \wedge \alpha(M,w)$ has a finite model.* tur:und:tra: lem:halts-sat

Proof. Let $\mathfrak{M}'$ be as in the proof of Lemma und.15, except

$$|\mathfrak{M}'| = \{0, \ldots, n\},$$
$$\prime^{\mathfrak{M}'}(x) = \begin{cases} x+1 & \text{if } x < n \\ n & \text{otherwise,} \end{cases}$$
$$\langle x, y\rangle \in <^{\mathfrak{M}'} \text{iff } x < y \text{ or } x = y = n,$$

where $n = \max(k, \text{len}(w))$ and k is the least number such that M started on input w has halted after k steps. We leave the verification that $\mathfrak{M}' \vDash \tau'(M,w) \wedge E(M,w)$ as an exercise. □

Problem und.10. Complete the proof of Lemma und.19 by proving that $\mathfrak{M}' \vDash \tau(M,w) \wedge E(M,w)$.

Lemma und.20. *If $\tau'(M,w) \wedge \alpha(M,w)$ has a finite model, then M started on input w halts.* tur:und:tra: lem:sat-halts

Proof. We show the contrapositive. Suppose that M started on w does not halt. If $\tau'(M,w) \wedge \alpha(M,w)$ has no model at all, we are done. So assume $\mathfrak{M}$ is a model of $\tau(M,w) \wedge \alpha(M,w)$. We have to show that it cannot be finite.

We can prove, just like in Lemma und.13, that if M, started on input w, has not halted after n steps, then $\tau'(M,w) \vDash \chi(M,w,n) \wedge \psi(\overline{n})$. Since M started on input w does not halt, $\tau'(M,w) \vDash \chi(M,w,n) \wedge \psi(\overline{n})$ for all $n \in \mathbb{N}$. Note that by Proposition und.10, $\tau'(M,w) \vDash \overline{k} < \overline{n}$ for all $k < n$. Also $\psi(\overline{n}) \vDash \overline{k} < \overline{n} \rightarrow \overline{k} \neq \overline{n}$. So, $\mathfrak{M} \vDash \overline{k} \neq \overline{n}$ for all $k < n$, i.e., the infinitely many terms $\overline{k}$ must all have different values in $\mathfrak{M}$. But this requires that $|\mathfrak{M}|$ be infinite, so $\mathfrak{M}$ cannot be a finite model of $\tau'(M,w) \wedge \alpha(M,w)$. □

Problem und.11. Complete the proof of Lemma und.20 by proving that if M, started on input w, has not halted after n steps, then $\tau'(M,w) \vDash \psi(\overline{n})$.

Theorem und.21 (Trakthenbrot's Theorem). *It is undecidable if an arbitrary sentence of first-order logic has a finite model (i.e., is finitely satisfiable).* tur:und:tra: thm:trakhtenbrodt

Proof. Suppose there were a Turing machine F that decides the finite satisfiability problem. Then given any Turing machine M and input w, we could compute the sentence $\tau'(M,w) \wedge \alpha(M,w)$, and use F to decide if it has a finite model. By Lemmata und.19 and und.20, it does iff M started on input w halts. So we could use F to solve the halting problem, which we know is unsolvable. □

tur:und:tra:cor:fproof-incomp **Corollary und.22.** *There can be no derivation system that is sound and complete for* finite *validity, i.e., a derivation system which has* $\vdash \psi$ *iff* $\mathfrak{M} \vDash \psi$ *for every finite structure* $\mathfrak{M}$.

Proof. Exercise. □

Problem und.12. Prove Corollary und.22. Observe that ψ is satisfied in every finite structure iff $\neg\psi$ is not finitely satisfiable. Explain why finite satisfiability is semi-decidable in the sense of Theorem und.18. Use this to argue that if there were a derivation system for finite validity, then finite satisfiability would be decidable.

Photo Credits

Bibliography