Decidable and Recursive Enumerable (Semi-Decidable) Sets

Why do we need to consider sets? So far, we have dealt with algorithms and programs, objects with which we are very familiar and for which we have good intuition.

To solve some problems, however, this intuition is not well suited, so we need to reformulate the problem in precise terms – i.e., in terms of mathematics. We have already done this for for-loops and for while-loops: to analyze what can be computed by using the corresponding loops. It is reasonable to expect that similar reformulation may help in other cases as well. Let us think of it in the most general terms: how can we reformulate computational concepts in mathematical terms?

In modern mathematics, the most basic notion is the notion of a *set*. All other notions are formulated in terms of sets. Sets is what all the students study in high school – as well as natural set operations such as union, intersection, and complement. Sets is what computer science students study again in a Discrete Math (or Discrete Structures) course. So, let us see how we can reformulate the notions related to computability in terms of sets – so that, in addition to computational intuition, we will be able to use our intuition about sets, about their unions, intersections, and complements.

First notion – of a decidable set. One of the main questions in studying computability is whether there is an algorithm that, given an object n, checks whether this object satisfies the given property $P(n)$. For example:

- we have a natural number n, and we want to check whether this number is prime or not;
- we have a pair $n = (p, d)$ consisting of a program p and data d, and we want to check whether the program p halts of data d;
- we have a program n, and we want to check whether this program always returns 0.

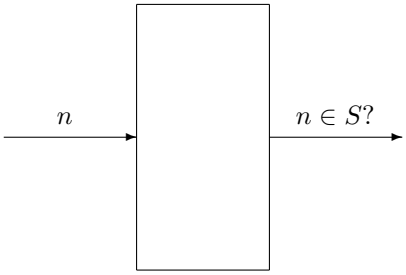
In the first case, there is an algorithm for checking primality. In the second and third cases, as we have shown, checking algorithms are not possible. How can we describe all this in terms of sets?

To do this, let us recall that while in general, we can have objects of different types, in the computer, everything can be represented as a sequence of 0s and

1s, and each such sequence can be interpreted as a natural number. From this viewpoint, it is sufficient to assume that n is a natural number.

Each property $P(n)$ can be naturally described in terms of a set – i.e., as the set of all the natural numbers that have this property. This set is denoted by $\{n : P(n)\}$. For example, the property "to be prime" is described by the set of all prime numbers $\{2, 3, 5, 7, 11, 13, \ldots\}$. So, we arrive at the following definition.

Definition 1. *We say that a set S of natural numbers is* decidable *if there is an algorithm that, given a natural number n, checks whether this number belongs to the set S.*



Comment. We want an algorithm that, given a natural number n:

- returns true if n is an element of the set S (this is denoted by $n \in S$) and
- returns false if n is not an element of the set S, i.e., if $n \notin S$.

As we have mentioned, in a computer, "true" is represented as 1, and "false" as 0. Thus, what we want is to have an algorithm $f(n)$ that, given a natural number n:

- returns $f(n) = 1$ if $n \in S$, and
- returns $f(n) = 0$ is $n \notin S$.

In mathematics, such a function $f(n)$ is called a *characteristic function* of the set S; it is denoted by $\chi_S(n)$. In these terms, we can say that a set S is decidable if and only if its characteristic function is computable – i.e., if and only if there exists an algorithm that computes this characteristic function.

Simplest examples of decidable sets. Let us check whether different sets are decidable. Let us start with the simplest possible sets – empty set $\emptyset$ that has no elements at all and the set N of all natural numbers. Are they decidable?

Theorem 1. *Empty set $\emptyset$ is decidable.*

Brainstorming. Can we prove this? By definition, we want to return 1 if n belongs to this set, and 0 if n does not belong to the given set. But no element

belongs to the empty set, so we should always return 0. Of course, it is easy to come up with an algorithm that ignores the input n and always returns 0.

Proof. By definition, to prove that the empty set $\emptyset$ is decidable, we need to produce an algorithm that, given a natural number n, always returns 0. Such an algorithm is easy to produce:

```
public static int a(int n)
  {return 0;}
```

Thus, the empty set is decidable.
The theorem is proven.

Theorem 2. *The set N of all natural numbers is decidable.*

Proof. All natural numbers belong to the set S, so the corresponding algorithm should always return 1. Such an algorithm is easy to construct:

```
public static int a(int n)
  {return 1;}
```

Thus, the set of all natural numbers is decidable.
The theorem is proven.

Comment. Next in complexity are finite sets, i.e., set that have finitely many elements.

Theorem 3. *Every finite set of natural numbers is decidable.*

Proof. Indeed, let us assume that $S = \{n_1, \ldots, n_k\}$ is a finite set. Then, for a given natural number n, we can check where n belongs to this set by comparing the number n with all elements of the final set:

```
public static int a(int n)
  {if(n == n1 || ... || n == nk) {return 1;}
     else {return 0;}}
```

So, every finite set is decidable.
The theorem is proven.

Comment. Is the set of all extraterrestrials who, at present, pretend to be humans, decidable? Yes, since this set is finite. We do not know the algorithm that decides whether a person is from another planet, but the definition is not about what we know – it is about whether such an algorithm exists, and it does.

Operations on sets. Let us now consider operations on sets: union, intersection, and complement.

Theorem 4. *If A and B are decidable sets, then their union $A \cup B$ is decidable.*

Proof. Indeed, a natural number n belongs to the union if and only if belongs to A or to B. Since A and B are decidable set, we have:

- an algorithm $a(n)$ that checks whether a number n belongs to the set A, and
- an algorithm $b(n)$ that checks whether a number n belongs to the set B.

To check whether n belongs to the union, we can simply use an easily computable function $or(a(n), b(n))$.

The theorem is proven.

Theorem 5. *If A and B are decidable sets, then their intersection $A \cap B$ is decidable.*

Proof. Indeed, a natural number n belongs to the intersection if and only if belongs to A and to B. Since the sets A and B are decidable, we have:

- an algorithm $a(n)$ that checks whether a number n belongs to the set A, and
- an algorithm $b(n)$ that checks whether a number n belongs to the set B,

To check whether n belongs to the intersection, we can simply use an easily computable function $and(a(n), b(n))$. We know that "and" is simply multiplication, so we can write:

```
public static int intersection(int n)
  {return a(n) * b(n);}
```

The theorem is proven.

Theorem 6. *If A is a decidable set, then its complement $-A$ is also decidable.*

Proof. Indeed, a natural number n belongs to the complement $-A$ if and only this number does not belongs to set A. Since the set A is decidable, we have an algorithm $a(n)$ that checks whether a number n belongs to the set A. So, to check whether n belongs to the complement, we can simply use an easily computable function $not(a(n))$. We know that "not a" is simply $1 - a$, so we can write:

```
public static int complement(int n)
  {return 1 - a(n);}
```

The theorem is proven.

Not all sets are decidable. Let us prove the following result.

Theorem 7. *There exists a set which is not decidable.*

Proof. We know that no algorithm is possible that, given a pair (p, d), checks whether p halts on d. What we need to prove our result is to transform pairs of natural numbers into natural numbers.

This can be done, e.g., as follows.

- First, we list all the pairs in which the sum is 0, and we list them in lexicographic order. There is only one such pair: $(0, 0)$. This will be pair No. 1.
- Then, we list all the pairs in which the sum is 1, and we list them in lexicographic order. So, pair No. 2 is $(0, 1)$, and pair No. 3 is $(1, 0)$.
- After that, we list all the pairs in which the sum is 2, and we list them in lexicographic order. So, pair No. 4 is $(0, 2)$, pair No. 5 is $(1, 1)$, and pair No. 6 is $(2, 0)$, etc.

Let us denote the ordinal number of the pair (p, d) in this ordering by $\pi(p, d)$, and let us denote, for each number n, the two compoents of the n-th pair by $\pi_1(n)$ and $\pi_2(n)$. For example, since pair No. 4 is $(0, 2)$, we have $\pi(0, 2) = 4$, $\pi_1(4) = 0$, and $\pi_2(4) = 2$.

Then, the set

$$\{n : \pi_1(n) \text{ halts on } \pi_2(n)\}$$

of all the pairs for which p halts on d is not decidable – since, as we have proven, we cannot algorithmically decide whether a given program halts on given data.

The theorem is proven.

Comment. We will call the set described in this proof the *halting set* and denote it by H.

Another notion. Sometimes, we simply want to list all the elements of a set. This leads us to a different notion.

Definition 2. *We say that a set S of natural numbers is* recursively enumerable *(r.e., for short) if there exists an algorithm that eventually prints all the elements of this set – and only these elements.*

Comment. R.e. sets are also called *semi-decidable.*

Theorem 8. *The empty set $\emptyset$ is r.e.*

Proof. Indeed, by definition, this means that there exists an algorithm that does not print anything. Such an algorithm is easy to construct.

```
public static void()
  {}
```

The theorem is proven.

Theorem 9. *The set N of all natural numbers is r.e.*

Proof. Indeed, it is easy to design an algorithm that print all natural numbers 0, 1, 2, ..., one by one:

```
public static void()
  {int m = 0;
   while(true)
     {System.out.print(m);
      m++;}}
```

The theorem is proven.

Theorem 10. *Every finite set of natural numbers is r.e.*

Proof. Indeed, let us assume that $S = \{n_1, \ldots, n_k\}$ is a finite set. Then, we can print all its elements one by one:

```
public static void()
  {System.out.print(n1);
   ...
   System.out.print(nk);}
```

The theorem is proven.

Comment. These three results make us think that every decidable set is r.e. This is indeed true:

Theorem 11. *Every decidable set is r.e.*

Proof. Let S be a decidable set. This means that there exists an algorithm $a(n)$ that, given a natural number n, checks whether n belongs to this set. Then, to print all the elements of this set, we can simply try all the natural numbers 0, 1, 2, ..., check if they belong to the set S and, if yes, print them:

```
public static void()
  {int m = 0;
   while(true)
     {if(a(m) == 1)
        {System.out.print(m);}
      m++;}}
```

The theorem is proven.

What about union etc.? Let us analyze what happens if we take the union, intersection, or complement to a r.e. set.

Theorem 12. *The union $A \cup B$ of two r.e. sets A and B is r.e.*

Comment. If both sets A and B were finite, we could simply first print all the elements of A, and then print all the elements of B. But what if they are infinite – e.g., A is the set of all even numbers and B is the set of all odd numbers? Then, if we start printing elements of A (i.e., even numbers), we will never finish and never print a single odd number. So, we need a more complex idea.

Proof. The fact that the set A is r.e. means that there exists an algorithm a that eventually prints all the elements of the set A. Similarly, the fact that the set B is r.e. means that there exists an algorithm b that eventually prints all the elements of the set B. Let us describe a new algorithm that prints all the elements of both sets – i.e., all the elements of the union. This new algorithm is as follows:

- first, we run algorithm a for 1 hour;

- then, we run algorithm b for 1 hour;
- then, we resume the run of the algorithm a and run it for 1 more hour;
- then, we resume the run of the algorithm b and run it for 1 more hour;
- etc.

Here:

- the 1st hour of the algorithm a is performed by the new algorithm during its 1st hour,
- the 2nd hour of the algorithm a is performed by the new algorithm during its 3rd hour, and,
- in general, the k-th hour of the algorithm a is performed by the new algorithm in its $(2k-1)$-st hour.

Similarly:

- the 1st hour of the algorithm b is performed by the new algorithm during its 2nd hour,
- the 2nd hour of the algorithm b is performed by the new algorithm during its 4th hour, and,
- in general, the k-th hour of the algorithm b is performed by the new algorithm in its $(2k)$-th hour.

So, the new algorithm prints all the elements of the set A and all the elements of the set B – and only these elements.

The theorem is proven.

Theorem 13. *The intersection $A \cap B$ of two r.e. sets A and B is r.e.*

Comment. If both sets A and B were finite, we could simply first print all the elements of A, then print all the elements of B, and compare these two printouts to find common elements. But what if they are infinite – e.g., A is the set of all even numbers and B is the set of all odd numbers? Then, if we start printing elements of A (i.e., even numbers), we will never finish and never print a single odd number. So, we need a more complex idea.

Proof. The fact that the set A is r.e. means that there exists an algorithm a that eventually prints all the elements of the set A. Similarly, the fact that the set B is r.e. means that there exists an algorithm b that eventually prints all the elements of the set B. Let us describe a new algorithm that prints all the elements of both sets – i.e., all the elements of the union. This algorithm is as follows:

- first, we run algorithm a for 1 hour;
- then, we run algorithm b for 1 hour;

- then, we compare the two lists printed-so-far by these two algorithms, and print all common elements of these two lists;
- then, we resume the run of the algorithm a and run it for 1 more hour;
- then, we resume the run of the algorithm b and run it for 1 more hour;
- then, we compare the two lists printed-so-far by these two algorithms, and print all common elements of these two lists that have not been printed before,
- etc.

Clearly, all printed elements belong to both sets. Let us show that every natural number n that belongs to both sets will thus be printed. Indeed, since this number belongs to the set A, it will be printed by the algorithm a at some time k. Similarly, since the number n belongs to the set A, it will be printed by the algorithm a at some time k.

Here, the k-th hour of the algorithm a is performed by the new algorithm in its $(2k-1)$-st hour, and the ℓ-th hour of the algorithm b is performed by the new algorithm in its (2ℓ)-th hour. So, by the time $\max(2k-1, 2\ell)$, the number n will be generated by both algorithms, and thus, will be printed by the new algorithm.

The theorem is proven.

Comment. For complement the situation is different. To prove it, we need to consider several auxiliary results.

Theorem 14. *If both a set S and its complement $-S$ are r.e., then S is decidable.*

Proof. Since both S and $-S$ are r.e., there exists:

- an algorithm s that eventually prints all the elements of the set S, and
- an algorithm c that eventually prints all the elements of the complement $-S$.

Let us show how we can check whether a given natural number n belongs to the set S. For this, we:

- first, we run the algorithm s for 1 hour;
- then, we run the algorithm c for 1 hour;
- then, we resume the run of the algorithm s and run it for 1 more hour;
- then, we resume the run of the algorithm c and run it for 1 more hour;
- etc. until one of these algorithms prints the number n.

If $n \in S$, then the number n will be printed by the algorithm s. If $n \notin S$, then n will be printed by the algorithm c. In both cases, the above procedure will stop, and by checking in which of the two lists the value n appears, we can detect whether n is in the set S or not.

The theorem is proven.

Theorem 15. *The halting set H is r.e.*

Proof. To enumerate all the pairs (p, d) for which p halts on d, we can do the following:

- First, we run all the programs $p \leq 1$ on all data $d \leq 1$ for 1 hour. This means that we run the following pairs $(p, d) = (0, 0)$, $(p, d) = (0, 1)$, $(p, d) = (1, 0)$, and $(p, d) = (1, 1)$. If any of these pairs halt during this time, we print the corresponding pair (p, d).
- Then, we run all the programs $p \leq 2$ on all data $d \leq 2$ for 2 hours. If any of these pairs halt during this time, we print the corresponding pair (p, d).
- etc.

If this algorithm prints a pair (p, d), this means that p halted on d. Vice versa, if p halted on d at time t, then the pair (p, d) will be printed at the cycle $\max(p, d, t)$ of the new algorithm. So, indeed, this algorithm eventually prints all the elements of the halting set – and only them.

The theorem is proven.

Theorem 16. *There exists a r.e. set which is not decidable.*

Proof. According to Theorem 15, the halting set H is r.e., but we have proven that it is not decidable.

The theorem is proven.

Theorem 17. *There exists a r.e. set S whose complement is not r.e.*

Proof. As such an example, we can take $S = H$. Indeed, if its complement $-H$ was r.e., then by Theorem 14, the set H would be decidable, but we know that the halting set H is not decidable.

The theorem is proven.

What about unions and intersections of decidable and r.e. sets? We have shown that the union of two decidable sets is decidable, and the union of two r.e. sets is r.e. But what is we have a union of a r.e. and decidable sets? It turns out that the result is always r.e. but not always decidable.

Theorem 18. *The union of a r.e. set A and a decidable set B is always r.e.*

Proof. Indeed, every decidable set is r.e., and the union of two r.e. sets is always r.e.

The theorem is proven.

Theorem 19. *There exist a r.e. set A and a decidable set B for which the union $A \cup B$ is not decidable.*

Brainstorming. If A was decidable, then the union $A \cup B$ would be decidable. So, to find the desired example, we need to find a set A which is r.e. but not decidable. We know only one such set – the halting set $A = H$. So, it is sufficient to find a decidable set B for which $A \cup B$ is also not decidable. Since we know only one such set, we need to find a set B for which $A \cup B = H \cup B = H$. There is one such set: $B = \emptyset$.

Proof. Take $A = H$ and $B = \emptyset$, then A is r.e., B is decidable, but $A \cup B = H \cup \emptyset = H$ is not decidable.

The theorem is proven.

Theorem 20. *The intersection of a r.e. set A and a decidable set B is always r.e.*

Proof. Indeed, every decidable set is r.e., and the intersection of two r.e. sets is always r.e. The theorem is proven.

Theorem 21. *There exist a r.e. set A and a decidable set B for which the intersection $A \cap B$ is not decidable.*

Brainstorming. If A was decidable, then the intersection $A \cap B$ would be decidable. So, to find the desired example, we need to find a set A which is r.e. but not decidable. We know only one such set – the halting set H. So, it is sufficient to find a decidable set B for which $A \cap B$ is also not decidable. Since we know only one such set, we need to find a set B for which $A \cap B = H \cap B = H$. There is one such set: $B = N$.

Proof. Take $A = H$ and $B = N$, then A is r.e., B is decidable, but $A \cap B = H \cap N = H$ is not decidable. The theorem is proven.