CS 70 Discrete Mathematics and Probability Theory
Fall 2018 Alistair Sinclair and Yun Song

# DIS 6B

## 1 More Countability

Given:

- $A$ is a countable, non-empty set. For all $i \in A$, $S_i$ is an uncountable set.
- $B$ is an uncountable set. For all $i \in B$, $Q_i$ is a countable set.

For each of the following, decide if the expression is "Always Countable", "Always Uncountable", "Sometimes Countable, Sometimes Uncountable".

For the "Always" cases, prove your claim. For the "Sometimes" case, provide two examples – one where the expression is countable, and one where the expression is uncountable.

(a) $A \cap B$

(b) $A \cup B$

(c) $\bigcup_{i \in A} S_i$

(d) $\bigcap_{i \in A} S_i$

(e) $\bigcup_{i \in B} Q_i$

(f) $\bigcap_{i \in B} Q_i$

**Solution:**

(a) Always countable. $A \cap B \subseteq A$, which is countable.

(b) Always uncountable. $A \cup B$ is a superset of $B$, which is uncountable.

(c) Always uncountable. Let $a$ be any element of $A$. $S_a$ is uncountable. Thus, $\bigcup_{i \in A} S_i$, a superset of $S_a$, is uncountable.

(d) Sometimes countable, sometimes uncountable.

Countable: When the $S_i$ are disjoint, the intersection is empty, and thus countable. Formally, let $A = \mathbb{N}$, let $S_i = \{i\} \times \mathbb{R} = \{(i, x) \mid x \in \mathbb{R}\}$. Then, $\bigcap_{i \in A} S_i = \varnothing$.

Uncountable: When the $S_i$ are identical, the intersection is uncountable. Let $A = \mathbb{N}$, let $S_i = \mathbb{R}$ for all $i$. $\bigcap_{i \in A} S_i = \mathbb{R}$ is uncountable.

(e) Sometimes countable, sometimes uncountable.

Countable: Make all the $Q_i$ identical. Formally, let $B = \mathbb{R}$, and $Q_i = \mathbb{N}$. Then, $\bigcup_{i \in B} Q_i = \mathbb{N}$ is countable.

Uncountable: Let $B = \mathbb{R}$. Let $Q_i = \{i\}$. Then, $\bigcup_{i \in B} Q_i = \mathbb{R}$ is uncountable.

(f) Always countable. Let $b$ be any element of $B$. $Q_b$ is countable. Thus, $\bigcap_{i \in B} Q_i$, a subset of $Q_b$, is also countable.

## 2 Hello World!

Determine the computability of the following tasks. If it's not computable, write a reduction or self-reference proof. If it is, write the program.

(a) You want to determine whether a program $P$ on input $x$ prints "Hello World!". Is there a computer program that can perform this task? Justify your answer.

(b) You want to determine whether a program $P$ prints "Hello World!" before running the $k$th line in the program. Is there a computer program that can perform this task? Justify your answer.

(c) You want to determine whether a program $P$ prints "Hello World!" in the first $k$ steps of its execution. Is there a computer program that can perform this task? Justify your answer.

**Solution:**

(a) Uncomputable. We will reduce `TestHalt` to `PrintsHW`$(P,x)$.

```
TestHalt(P, x):
  P'(x):
    run P(x) while suppressing print statements
    print("Hello World!")
  if PrintsHW(P', x):
    return true
  else:
    return false
```

If `PrintsHW` exists, `TestHalt` must also exist by this reduction. Since `TestHalt` cannot exist, `PrintsHW` cannot exist.

(b) Uncomputable. Reduce `PrintsHW`$(P,x)$ from part (a) to this program `PrintsHWByK`$(P,x,k)$.

```
PrintsHW(P, x):
  # Find all "return" statements in P and put these
  # line numbers in set return_statements
  return_statements = []
  for i in range(len(P)):
```

```
    if isReturnStatement(i):
      return_statements.append(i)

  # For each "return" statement, check if "Hello World!"
  # is printed
  for r in return_statements:
    if PrintsHWByK(P, x, r):
      return true
  return false
```

(c) Computable. You can simply run the program until $k$ steps are executed. If $P$ has printed "Hello World!" by then, return true. Else, return false.

The reason that part (b) is uncomputable while part (c) is computable is that it's not possible to determine if we ever execute a specific line because this depends on the logic of the program, but the number of computer instructions can be counted.

# 3 Computability

Decide whether the following statements are true or false. Please justify your answers.

(a) The problem of determining whether a program halts in time $2^{n^2}$ on an input of size $n$ is undecidable.

(b) There is no computer program `Line` which takes a program $P$, an input $x$, and a line number $L$, and determines whether the $L^{\text{th}}$ line of code is executed when the program $P$ is run on the input $x$.

**Solution:**

(a) False. You can simulate a program for $2^{n^2}$ steps and see if it halts.

Generally, we can always run a program for any fixed *finite* amount of time to see what it does. The problem of undecidability arises when no bounds on time are available.

(b) True.

We implement `Halt` which takes a program $P$, an input $x$ and decides whether $P(x)$ halts, using `Line` as follows. We take the input $P$ and modify it so that each exit or return statement jumps to a particular new line. Call the resulting program $P'$. We then hand that program to `Line` along with the input $x$ and the number of the new line. If the original program halts than `Line` would return true, and if not `Line` would return false.

This contradicts the fact that the program `Halt` does not exist, so `Line` does not exist either.

There are two ways to show undecidability: 1. Use your program as a subroutine to solve a problem we know is undecidable or to do a diagonalization proof like we did for `Halt`. The

former is natural for computer programmers and flows from the fact that you are given $P$ as text! Therefore you can look at it and modify it. This is what the solution above does.