Number Representation

Thm 1: For every natural number \( n \), there exists a \( p \in \mathbb{N} \) and bits \( b_p, b_{p-1}, \ldots, b_0 \in \{0,1\} \) such that

\[
n = \sum_{i=0}^{p} b_i \cdot 2^i
\]

(for helping with induction, restate)

\[
\forall m \in \mathbb{N}, \quad \forall n \in \mathbb{N}, \quad n \leq m \implies \\
(\exists p \in \mathbb{N}, \exists b_0, \ldots, b_p \in \{0,1\}, \quad n = \sum_{i=0}^{p} b_i \cdot 2^i )
\]

Call this \( P(m) \), where \( m \in \mathbb{N} \)

Form: \( \forall m \in \mathbb{N}, P(m) \)
Proof: by induction.

base case: \( m = 0 \) Prove \( P(0) \).

and

\[
\text{Let } n \in \mathbb{N} \text{ assume } n \leq m \ (= 0)
\]

There is only one number to consider \( n = 0 \).

Let \( p = 0 \) and \( b_0 = 0 \).

Then

\[
\sum_{i=0}^{p} b_i \cdot 2^i = 0 \cdot 2^0 = 0 = n, \text{ as req'd}
\]

inductive step: Let \( k \in \mathbb{N} \) and assume \( P(k) \). That is, every natural number \( \leq k \) has a binary representation. We need to prove \( P(k+1) \).

Let \( n \in \mathbb{N} \) and assume \( n \leq k+1 \).

Case \( n \leq k \)

by the induction hypothesis, we know \( n \) has a binary representation.
case: \( n = k + 1 \)

we need to show this \( n \) has a binary representation.

case 1: Assume \( n = k + 1 \) is even.

i.e. \( \exists q \in \mathbb{Z}, \ n = 2q \).

Let \( q \in \mathbb{Z} \) be such that \( n = 2q \).

we know \( q = n/2 \)

\( 0 \leq q \).

So by the induction hypothesis, \( q \) has a binary representation.

\( q = \sum_{i=0}^{p} b_i \cdot 2^i \quad \text{for some } p \in \mathbb{N} \quad \text{and } b_0, \ldots, b_p \in \{0, 1\} \)

so \( n = 2 \left( \sum_{i=0}^{p} b_i \cdot 2^i \right) \)

\( = \sum_{i=0}^{p} b_i \cdot 2^{i+1} \)

Let \( p' = p + 1 \), \( b_0 = 0 \), and

\( b_i' = b_{i-1} \quad \text{for all } i \in \{1, \ldots, p'\} \)

Then \( n = \sum_{i=0}^{p'} b_i' \cdot 2^i \), as req'd.
case 2: Assume \( n = k+1 \) is odd.

i.e. \( \exists q \in \mathbb{Z} \), \( n = 2q + 1 \)

Let \( q \in \mathbb{Z} \) be such that \( n = 2q + 1 \)

As before, \( q < n \) and \( q \leq k \)

We can write \( q = \sum_{i=0}^{p} b_i 2^i \) for some \( p \in \mathbb{N} \) and \( b_0, \ldots, b_p \in \{0, 1\} \).

Then \( N = 2 \left( \sum_{i=0}^{p} b_i 2^i \right) + 1 \)

\[
= \left( \sum_{i=0}^{p} b_i 2^i \right) + 1 + \underbrace{b_{p+1}}_{=0}
\]

\[
= b_0 2^0 + b_1 2^1 + b_2 2^2 + \ldots + b_p 2^p
\]

Let \( p' = p+1 \), \( b_0' = 1 \), \( b_i' = b_{i-1} + b_i 2^i \).

Then \( N = \sum_{i=0}^{p'} b_i' 2^i \), as required.

\[\square\]

Can conclude every natural number has a binary representation.

\[ n = \sum_{i=0}^{r} c_i 2^i = c_0 2^0 + c_1 2^1 + c_2 2^2 + \ldots + c_r 2^r \]
Ch 5 Analyzing Algorithm Running Time (runtime)

Programming

Focus:
- Write a program that compiles correctly.
- Use an accepted design process and style guidelines.
- Will it finish on time?
- Which algorithm is best for a situation?

How to measure runtime of program/algorithm?
- Stopwatch?

Flaws:
- Can be influenced by many factors.
  - e.g. CPU type, other loads.

Want to be able to answer Q like:
How does runtime change as you double # of observations?

Other factors:
- Insertion sort:
  - Almost sorted list of n items
    - Runtime \( \sim n \)
list almost in reverse order
runtime \sim n^2

- Q: worst case situation upper bound
   best case situation lower bound
   average case situation

take a formal approach to getting answers.

Describe the runtime of:

```python
def print_items(lst: List[str]) -> None:
    for item in lst:
        print(item)
```

- could count # of "basic operations"

1. could count # of prints.
   - assume list length is \( n \) E/N.

   \[ n \] prints.

   could describe runtime as \( n \) basic operations

2. what about cost of loop?
   assign values to item?
n prints + n assignments to item

could describe runtime as 2n basic
n + n operators

3. printing more time than extracting
from list. (suppose 10:1)

could describe runtime as 10n + n
= 11n basic operations

4. act of calling + returning from a func
has a fixed cost

say ~ 1.5 basic operations

could describe runtime as 11n + 1.5

basic operations

In summary

1. n basic operations
2. 2n
3. 11n
4. 11n + 1.5

Q: Which is correct?
A: hard to answer, depends on too many
unknown factors.

But all 4 are descr. of str. lines.
Change Q to:

"How does runtime change as the size of the list changes?"

Describe using formal big-Oh notation.

\[ 2g = b_0 \cdot 2^1 + b_1 \cdot 2^2 + \ldots \]

\[ b_0 \cdot 2^0 + b_1 \cdot 2^1 + b_2 \cdot 2^2 + \ldots + b_i \cdot 2^i + \ldots + b_p \cdot 2^p \]

\[ b_{i-1} \]

\[ \begin{array}{c}
(2)_{10} \\
\frac{(10)}{2} \end{array} \quad \frac{\text{\shortmid} \quad 2}{} = \begin{array}{c}
(1)_{2} \\
\end{array} \]

\[ \begin{array}{c}
(3)_{10} \\
\frac{(11)}{2} \end{array} \quad \frac{\text{\shortmid} \quad 2}{} = \begin{array}{c}
(1)_{2} \\
\end{array} \]

\[ (9)_{10} \quad \frac{(10001)}{2} \quad 2 = \begin{array}{c}
(100)_{2} \\
\end{array} \]

\[ (12)_{10} \quad \frac{(11000)}{2} \quad 2 = \begin{array}{c}
(110)_{2} \\
\end{array} \]
\[ \forall n \quad n \leq k \Rightarrow \sum_{i=0}^{\infty} b_i 2^i \]

\[ \forall n \quad n \leq k+1 \Rightarrow \ldots \]

\[ \text{even } 2(\sum b_i) = \sum b_i 2^i \]

\[ \text{odd } \]

\[ k+1 = 2q + 1 \]

\[ = 2 \left( \sum_{i=0}^{p} b_i 2^i \right) + 1 \]

\[ = \sum_{i=0}^{p} b_i 2^{i+1} + 1 \]

\[ = b_0 \cdot 2^1 + b_1 \cdot 2^2 + b_2 \cdot 2^3 + \ldots + b_p 2^{p+1} \]

\[ = c_0 2^0 + c_1 2^1 + c_2 2^2 + \ldots + c_r 2^r \]

\[ = \sum_{i=0}^{r} c_i 2^i \quad \text{goal} \]

Take \( n = p+1 \), \( c_i = b_i \), \( i \in \{1, \ldots, r\} \), \( c_0 = 0 \).

\[ (101)_2 \times 2^2 = 10100 \]