Algorithm and Data Structure Analysis (ADSA)

Turing Machines

Hopcroft, Motwani, Ullmann: Automata Theory, Languages and Computation (Chapter 8)

Overview

- Hello, world
- Decidable and undecidable problems
- Turing machines

Hello, world

What is this program printing?

```
main()
{
    printf("hello, world\n");
}
```

Given a program P, is it easy to determine whether it prints “hello, world”?

Figures 8.1 and 8.2 in Hopcroft, Motwani, Ullmann

What about this one?

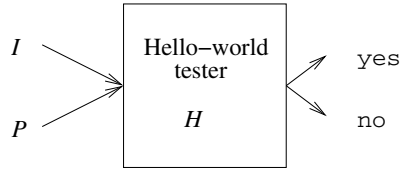
```
int exp(int i, n)
/* computes i to the power n */
{
    int ans, j;
    ans = 1;
    for (j=1; j<=n; j++) ans *= i;
    return(ans);
}

main ()
{
    int n, total, x, y, z;
    scanf("%d", &n);
    total = 3;
    while (1) {
        for (x=1; x<=total-2; x++)
            for (y=1; y<=total-x-1; y++) {
                z = total - x - y;
                if (exp(x,n) + exp(y,n) == exp(z,n))
                    printf("hello, world\n");
            }
        total++;
    }
}
```

For any given n>2, the second program will never print “hello, world”.

Fermat’s last theorem (conjecture): Given integer n>2, there is no triplet of positive integers such that $x^n + y^n = z^n$ holds (took mathematicians over 300 years to prove this).

Hello, world tester

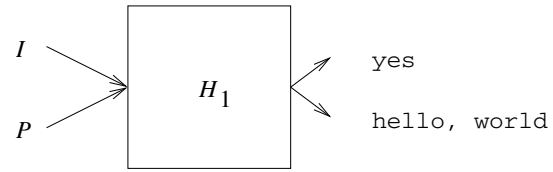
We want to have a Hello-world tester H, that for a given input I and program P, decides whether program P with input I prints “hello, world” as the first 12 characters.



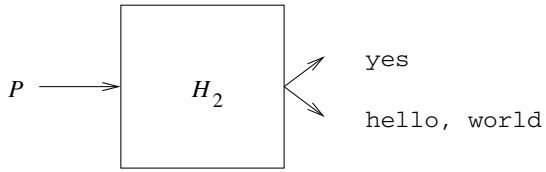
If a problem P always has an algorithm like H, then we say that P is decidable (otherwise undecidable).

We show that such a program H does not exist for P, i.e. the problem whether a given program prints “hello, world” as its first 12 characters is undecidable.

Theorem: The problem of testing whether a given program prints ”hello, world” is undecidable.
Proof:
Modify program H to obtain program H1 that prints “hello, world” when H prints “no”.



Modify program H1 to obtain program H2 that only takes P as an input.

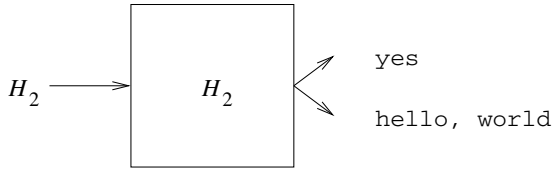


- H2 reads input P and stores it in an array A. H2 takes P as program and input I.
- H2 simulates H1, but always reads from array A if H1 would read from P or I. Mark positions in A until which part has been read.

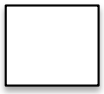
Show that H2 doesn’t exist => H1 doesn’t exist => H doesn’t exist.

- Given any input program P to H2, H2 gives output “yes” if P prints “hello, world” as its first output given itself as input.
- Also H2, prints “hello world” if P given itself as input doesn’t print “hello, world” as it’s first output.

- Give H2 itself as input



- Suppose that H2 presented by the box makes the output “yes”. Then H2 in the box is saying about H2 that it prints out “hello, world” as first output. However, we just supposed that the output that H2 makes in this situation is “yes” and not “hello, world”.
- Suppose that H2 prints out “hello, world” as the first output. Then H2 in the box must say “yes” and not “hello, world”.
- Whichever output we assume that H2 makes, we can argue that it makes the other output.
- Situation is paradoxical and we conclude that H2 can not exist and therefore H can not exist.
- We have proved that no program H with input P and I can tell whether P prints “hello, world” as the first output.

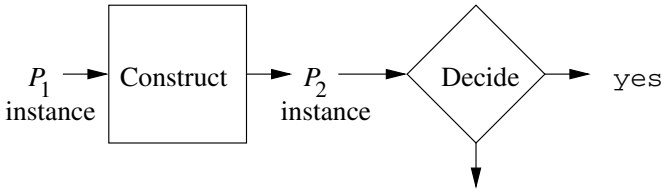


Reductions

- Suppose you are given a new problem P2 and want to show that it's undecidable.
- You can come up with same paradoxical situation, but there is an easier solution.
- Assume that you know that a given problem P1 is undecidable.
- We can try to reduce P1 to P2 to show that P2 is undecidable.

Reduction

- We want to show that a newly given problem P2 is undecidable.
- Assume P1 is undecidable. We show that we can decide P1 if we can decide P2.
- Assume that you can decide P2.
- Take P1 and construct problem P2 from it (direction is very important).



- Answer for P2 should be yes (no) if and only if answer for P1 is yes (no).
- This is called a reduction from P1 to P2.

Example Reduction

"Does program Q, given input y, ever call foo()?" (calls-foo problem) is undecidable.

We know that "hello, world" problem is undecidable and use it as P1 and calls-foo problem as P2.

We reduce the "hello, world" problem to the calls-foo problem.

Example Reduction

Theorem: The calls-foo problem is undecidable.

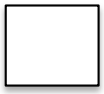
Proof:

Construct from program Q with input y ("hello, world" program), program R with input z (calls-foo program) such that R with input z calls foo if and only if Q with input y prints "hello, world".

1. If Q has a function foo, rename it and all the calls to that function. New program Q1 does exactly what Q does.
2. Add to Q1 a function foo. This function does nothing and is not called. The resulting program is Q2.
3. Modify Q2 to remember the first 12 characters that it prints, storing them in a global array A. Resulting program is Q3.
4. Modify Q3 so that whenever it executes any output statement, it then check in the array A to see if it has written 12 characters or more, and if so, whether "hello, world" are the first 12 characters. In that case, call the new function foo that was added in 2.). The resulting program is R, and the input z is the same as y.

Example Reduction

- Suppose that Q with input y prints “hello, world” as its first output. Then R as constructed will call foo.
- If Q with input y does not print “hello, world” as it’s first output, then R will never call foo.
- IF we can decide whether R with input z calls foo, then we also know whether Q with input y (y=z) prints “hello, world”.



Turing Machines

- Want to have a more formal model independent of programming to show that problems are undecidable or intractable.
- Turing machines are a very simple model that are ”as powerful as “ real computers.
- They provide an abstract way to show what computers are able to compute and whether they can do this efficiently.

Turing machines

- A Turing machine consists of a finite control which can be in any of a finite set of states.
- There is a tape divided into cells, each cell can hold any one of a finite set of symbols.

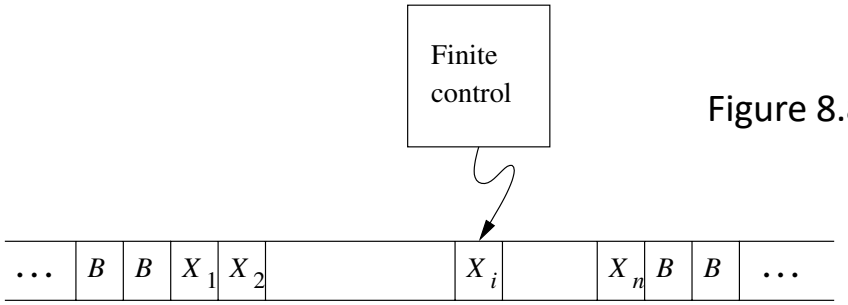


Figure 8.8 in Hopcroft, Motwani, Ullmann

- Input is a finite length string of symbols from a given input alphabet placed on the tape.
- All other tape cells extend infinitely to the left and right and hold initially a special symbol called blank “B” which is a tape symbol but no input symbol.
- There is a tape head always positioned at one of the tape cells. Turing machine is scanning that cell. Initially, the head is at the leftmost symbol of the input.

Turing machine

Move of a Turing machine is a function of the state symbol and the symbol scanned:

1. Change state (may be the same)
2. Write/replace a tape symbol in the cell scanned (may be the same)
3. Move the tape head left or right.

Turing machine

Formal notation for a Turing machine M is given as a 7-tuple M:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

whose components have the following meanings:

- Q: The finite set of *states* of the finite control.
- Σ: The finite set of *input symbols*.
- Γ: The complete set of *tape symbols*; Σ is always a subset of Γ.
- δ: The *transition function*. The arguments of $\delta(q, X)$ are a state q and a tape symbol X. The value of $\delta(q, X)$, if it is defined, is a triple (p, Y, D), where:
  1. p is the next state, in Q.
  2. Y is the symbol, in Γ, written in the cell being scanned, replacing whatever symbol was there.
  3. D is a *direction*, either L or R, standing for "left" or "right," respectively, and telling us the direction in which the head moves.
- q_0: The *start state*, a member of Q, in which the finite control is found initially.
- B: The *blank* symbol. This symbol is in Γ but not in Σ; i.e., it is not an input symbol. The blank appears initially in all but the finite number of initial cells that hold input symbols.
- F: The set of *final* or *accepting* states, a subset of Q.

Section 8.2.2 in Hopcroft, Motwani, Ullmann

Turing machines

- We need a notation for configurations or instantaneous descriptions (IDs) of a Turing machine.
- To represent an ID, we use $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$ where

1. q is the state of the Turing machine.
2. The tape head is scanning the ith symbol from the left.
3. $X_1X_2\cdots X_n$ is the portion of the tape between the leftmost and the rightmost nonblank. As an exception, if the head is to the left of the leftmost nonblank or to the right of the rightmost nonblank, then some prefix or suffix of $X_1X_2\cdots X_n$ will be blank, and i will be 1 or n, respectively.

Section 8.2.2 in Hopcroft, Motwani, Ullmann

We describe moves of the Turing machine M by the symbol $\vdash_M$ (or $\vdash$ if the context is clear)

Turing machines

- Suppose, we have the move left $\delta(q, X_i) = (p, Y, L)$ then we have:

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \underset{M}{\vdash} X_1 X_2 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n$$

The are two exceptions

1. If $i = 1$, then M moves to the blank to the left of X_1. In that case,

$$q X_1 X_2 \cdots X_n \underset{M}{\vdash} pBY X_2 \cdots X_n$$

2. If $i = n$ and $Y = B$, then the symbol B written over X_n joins the infinite sequence of trailing blanks and does not appear in the next ID. Thus,

$$X_1 X_2 \cdots X_{n-1} q X_n \underset{M}{\vdash} X_1 X_2 \cdots X_{n-2} p X_{n\text{-}1}$$

- Suppose that we have the move right $\delta(q, X_i) = (p, Y, R)$ then we have $X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \underset{M}{\vdash} X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$

The are two exceptions

1. If $i = n$, then the $i + 1$st cell holds a blank, and that cell was not part of the previous ID. Thus, we instead have

$$X_1 X_2 \cdots X_{n-1} q X_n \underset{M}{\vdash} X_1 X_2 \cdots X_{n-1} Y p B$$

2. If $i = 1$ and $Y = B$, then the symbol B written over X_1 joins the infinite sequence of leading blanks and does not appear in the next ID. Thus,

$$q X_1 X_2 \cdots X_n \underset{M}{\vdash} p X_2 \cdots X_n$$

Section 8.2.2 in Hopcroft, Motwani, Ullmann

Formal Language

- We want to test whether a given input fullfill a given property and if so have a Turing machine that accepts the input.
- We say that the input is in (part of) a given (formal) language.

Formally:

Given a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Then the language L(M) is the set of all strings w in Σ^* such that $q_0 w \overset{*}{\vdash} \alpha p \beta$ for some p in F. (set of all accepted strings).

Example:

- We want to construct a Turing machine M that test with a given input x is of the form 0^n1^n. A Turing machine that accepts the language $\{0^n1^n \mid n \geq 1\}$.
- Input alphabet of the Turing machine M is {0,1}.

Example Turing machine

Idea:

- Initially the tape contains a finite sequence of 0's and 1's preceded and followed by blanks.
- Alternately, the Turing machine changes a 0 to X and then a 1 to Y until all 0's and 1's have been matched.

Details:

- Starting at the left of the input, it enters a loop in which it changes a 0 to X.
- Afterwards, it moves to the right over whatever 0's or Y's it sees until it comes to a 1.
- It changes the 1 to Y, and moves left over Y's and 0's until it finds an X.
- At that point it looks for a 0 immediately to the right and if it finds one, changes it to X and repeat the process, changing a matching 1 to Y.
- If the nonblank input is not in 0*1*, then the TM will fail to have a next move and will die.
- However, if it finishes changing the last 1 to Y in the same round all 0's have been turned into X's, then it has found that the input is of the form 0^n1^n and it accepts.

Example Turing machine

- Consider the Turing machine M given as

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

where the transition function is given as

	Symbol				
State	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

Figure 8.9 in Hopcroft, Motwani, Ullmann

Example Turing machine

- The portion of the tape, where M's tape head has visited will always be a sequence of symbols described by the regular expression X*0*Y*1* (some 0's changed to X, followed by 0's not changed yet, followed by some 1's changed to Y, followed by some 1's)
- There may or may not be some 0's and 1's following.
- State q_0 is the initial state and M enters q_0 every time it returns to the leftmost remaining 0.
- If M is in state q_0 and scans a 0, it goes into state q_1, changes the 0 to X, and moves right.
- Once in state q_1, it moves over 0's and Y's remaining in state q_1.
- If it sees an X or a B, it dies.
- If it sees a 1 in state q_1, then it change that 1 to Y, enters state q_2 and moves left.
- In state q_2, M moves over 0's and X's. When it reaches the rightmost X (marks right end of block of 0s already changed to X), M returns to state q_0 and moves right.

Example Turing Machine

Two cases:

- If M sees a 0, then it repeats the matching cycle just described.
- If M sees a Y, then it has changed all 0's to X's.
  - If all 1's have been changed to Y, then the input is of the form 0^n1^n and M should accept. M changes to state q_3 and starts moving right over Y's. If it sees a B then it goes into state q_4 and accepts.
  - If M encounters another 1, then there are two many 1's and M dies without accepting.
  - If it encounters a 0, then the input was of the wrong form and it also dies.

Example Turing machine

State	Symbol 0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

Figure 8.9 in Hopcroft, Motwani, Ullmann

Consider input 0011 (in the language).

- Then we have the following sequence of moves:

$$q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \vdash$$
$$X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash$$
$$X X q_0 Y Y \vdash X X Y q_3 Y \vdash X X Y Y q_3 B \vdash X X Y Y B q_4 B$$

Consider input 0010 (not in the language).

- Then we have the following sequence of moves:

$$q_0 0010 \vdash X q_1 010 \vdash X 0 q_1 10 \vdash X q_2 0 Y 0 \vdash q_2 X 0 Y 0 \vdash$$
$$X q_0 0 Y 0 \vdash X X q_1 Y 0 \vdash X X Y q_1 0 \vdash X X Y 0 q_1 B$$

Transition diagram for Turing machine

- We can represent the transitions of a Turing machine by a transition diagram.
- Transition diagram consists of a set of nodes corresponding to the states.
- An edge from state q to state p is labeled by one or more items of the form X/Y D where X and Y are tape symbols and D is the direction (corresponds to transition $\delta(q, X) = (p, Y, D)$

Example Transition Diagram

Transition diagram for last Turing machine.

State	Symbol 0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

Figure 8.9 in Hopcroft, Motwani, Ullmann

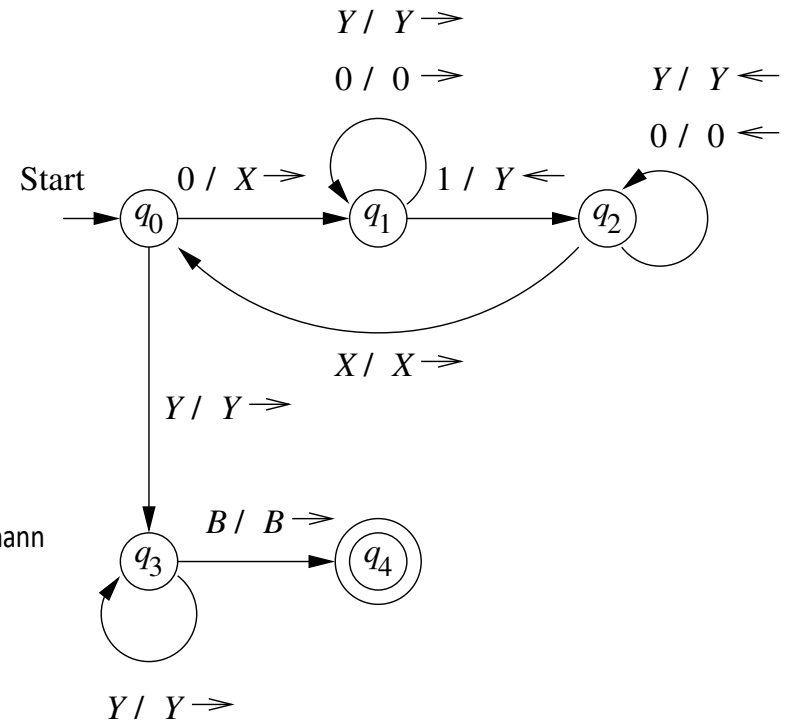


Figure 8.10 in Hopcroft, Motwani, Ullmann

Turing machine for Proper Substraction

- We now consider a Turing machine for proper substraction, i.e. for given m and n we want to compute r= max{m-n,0}.
- The input is given as 0^m10^n.
- We specify the Turing machine M as

$$M = (\{q_0, q_1, \ldots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B)$$

- Note that there is no accepting state as we will write the result of the computation on the tape.
- M halts with 0^r on its tape surrounded by blanks.

Turing machine for Proper Substraction

- M repeatedly finds the leftmost 0 and replaces it with a blank.
- It then searches right to find a 1.
- After finding a 1 it searches right to find a 0 which it replaces by a 1.
- M then turns left, seeking the leftmost 0 which it identifies when it first meets a blank and then moves one cell to the right.

Turing machine for Proper Substraction

Repetition ends:

1. Searching right for a 0, M encounters a blank. Then the n 0's in 0^m10^n have all been changed to 1's, and $n+1$ of the m 0's have been changed to B. M replaces the $n+1$ 1's by one 0 and n B's, leaving $m-n$ 0's on the tape. Since $m \geq n$ in this case, $m - n = m \dot{-} n$.

2. Beginning the cycle, M cannot find a 0 to change to a blank, because the first m 0's already have been changed to B. Then $n \geq m$, so $m \dot{-} n = 0$. M replaces all remaining 1's and 0's by B and ends with a completely blank tape.

Section 8.2.4 in Hopcroft, Motwani, Ullmann

Transition Diagram for Proper Substraction

State	Symbol 0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	–
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	–
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	–	–	–

Figure 8.11 in Hopcroft, Motwani, Ullmann

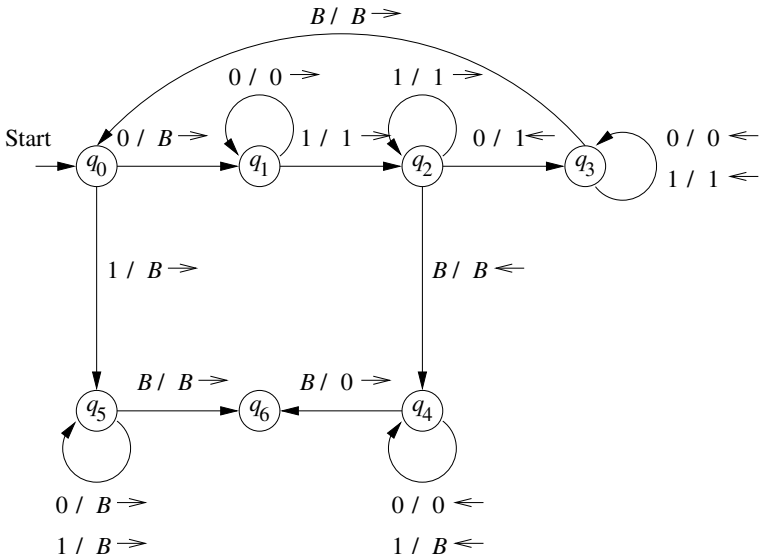


Figure 8.12 in Hopcroft, Motwani, Ullmann

Explanation of states for Proper Substraction

q_0: This state begins the cycle, and also breaks the cycle when appropriate. If M is scanning a 0, the cycle must repeat. The 0 is replaced by B, the head moves right, and state q_1 is entered. On the other hand, if M is scanning 1, then all possible matches between the two groups of 0's on the tape have been made, and M goes to state q_5 to make the tape blank.

q_1: In this state, M searches right, through the initial block of 0's, looking for the leftmost 1. When found, M goes to state q_2.

q_2: M moves right, skipping over 1's, until it finds a 0. It changes that 0 to a 1, turns leftward, and enters state q_3. However, it is also possible that there are no more 0's left after the block of 1's. In that case, M in state q_2 encounters a blank. We have case (1) described above, where n 0's in the second block of 0's have been used to cancel n of the m 0's in the first block, and the subtraction is complete. M enters state q_4, whose purpose is to convert the 1's on the tape to blanks.

q_3: M moves left, skipping over 0's and 1's, until it finds a blank. When it finds B, it moves right and returns to state q_0, beginning the cycle again.

Section 8.2.4 in Hopcroft, Motwani, Ullmann

Explanation of states for Proper Substraction

q_4: Here, the subtraction is complete, but one unmatched 0 in the first block was incorrectly changed to a B. M therefore moves left, changing 1's to B's, until it encounters a B on the tape. It changes that B back to 0, and enters state q_6, wherein M halts.

q_5: State q_5 is entered from q_0 when it is found that all 0's in the first block have been changed to B. In this case, described in (2) above, the result of the proper subtraction is 0. M changes all remaining 0's and 1's to B and enters state q_6.

q_6: The sole purpose of this state is to allow M to halt when it has finished its task. If the subtraction had been a subroutine of some more complex function, then q_6 would initiate the next step of that larger computation.

Section 8.2.4 in Hopcroft, Motwani, Ullmann

Summary

- Turing machines provide a simplified abstraction of a computer.
- They are (up to a polynomial factor) as powerful as a standard computers (register machine).
- Simplified computation model allows to argue in a formal way about the possibility of solving a given problem.