

CSCI 256: Theory of Computing

CHAPTER 8 Introduction to Turing Machines

Dr. Benjoe A. Juliano

Tel. 530 898-4619 (office)
530 898-6442 (dept)
Fax. 530 898-5995

Juliano@ecst.csuchico.edu
http://www.ecst.csuchico.edu/~juliano

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

1

From Figure 8.1 of (ATC, Hopcroft, Motwani, & Ullman, 2001.

Problems Computers Cannot Solve

- **Programs that print “hello, world”**
 - Problem: Determine whether the first thing a program prints is “hello, world” ...
 - Example:


```
main()
{
    printf("hello, world\n");
}
```
 - Note: Programs may take an unimaginably long time before making any output at all.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

2

Problems Computers Cannot Solve

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

main()
{
    int n, total=3, x, y, z;
    cin >> n;
    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))
                    cout <<"hello, world" << endl;
            }
        total++;
    }
}
```

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

3

Problems Computers Cannot Solve

- **Why Undecidable Problems Must Exist**
 - Recall: A “problem” is really a test of membership of a string in a language. The number of different languages over any alphabet of more than one symbol is *not countable*.
 - The set of programs (finite strings over a finite alphabet) is *countable* – order by length and then lexicographically.
 - Therefore, there are infinitely fewer programs than there are problems!

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

4

Problems Computers Cannot Solve

Definitions

- A mapping $f:A \rightarrow B$ is a **bijection** between sets A and B if f is both *one-to-one* and *onto* B .
- Two sets A and B are **equinumerous** if there is a bijection $f:A \rightarrow B$.
- A set is **finite** if it is equinumerous with the set $\{1, 2, \dots, n\}$ for some natural number n .
- A set is **infinite** if it is not finite.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

5

Problems Computers Cannot Solve

Definitions, continued ...

- A set is said to be **countably infinite** if it is equinumerous with \mathbb{U} .
- A set is **countable** if it is finite or countably infinite. A set that is not countable is called **uncountable**.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

6

Problems Computers Cannot Solve

The Hypothetical “Hello, World” Tester, H



where P is a program
 I is some input to P

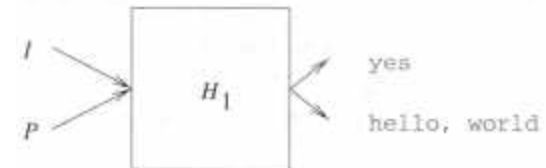
- If a problem has an algorithm like H , then it is said to be **decidable**.
- Show:** H does not exist; *i.e.* the hello-world problem is **undecidable**.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

7

Problems Computers Cannot Solve

Modification #1:



Notes:

- Print “hello, world” instead of printing “no” when program P (on input I) does not print “hello, world”

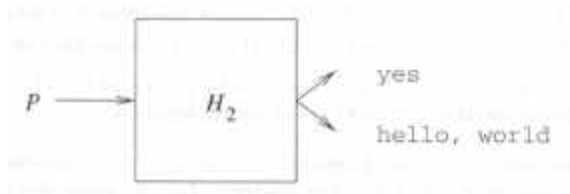
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

8

From Figure 8.5 of (ATLC, Hopcroft, Motwani, & Ullman, 2001).

Problems Computers Cannot Solve

- Modification #2:



- Notes:

- “ Determine what program P would do if its input were its own (source) code ...

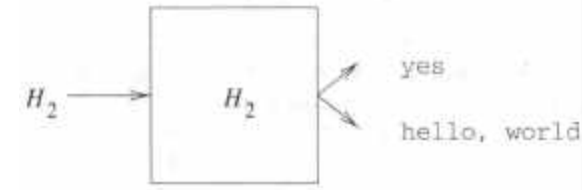
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

9

From Figure 8.6 of (ATLC, Hopcroft, Motwani, & Ullman, 2001).

Problems Computers Cannot Solve

- Finally ...



- Notes:

- “ What does H_2 do when given itself as input?
 - “ In any case, H_2 provides two different outputs, which is a contradiction.

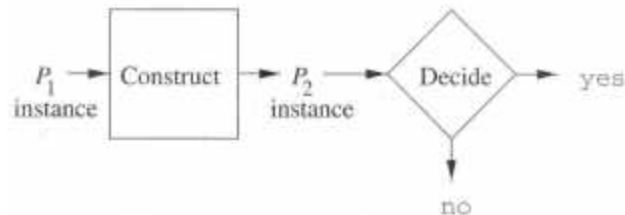
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

10

From Figure 8.7 of (ATLC, Hopcroft, Motwani, & Ullman, 2001).

Problems Computers Cannot Solve

- Reducing One Problem to Another



- Suppose P_1 is known to be *undecidable* and P_2 is a new problem we want to prove undecidable. The above construction is called a *reduction* of P_1 to P_2 .

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

11

The Turing Machine

- Theory of Undecidability

- “ *Undecidability* provides a guidance to programmers about what might or might not be accomplished through programming.
 - “ Some decidable problems are *intractable* – they require large amounts of time to solve.
- We need tools that will allow us to prove everyday questions undecidable or intractable.
 - “ Enter the *Turing Machine* ...

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

12

From Larry Gonick (1983), *The Cartoon Guide to the Computer*. Harper Collins Publishers, New York, page 190.

The Turing Machine

TURING, WHO ENJOYED LONG-DISTANCE RUNNING BACK WHEN THAT WAS CONSIDERED WEIRD, PROBABLY WENT INTO COMPUTERS TO SHRINK THE SIZE OF HIS JOBBING CLOCK.



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

13

From Larry Gonick (1983), *The Cartoon Guide to the Computer*. Harper Collins Publishers, New York, page 191.

The Turing Machine

ROUGHLY SPEAKING, A TURING MACHINE IS AN INPUT-OUTPUT DEVICE: A BLACK BOX THAT READS A SEQUENCE OF 0'S AND 1'S.

THE OUTPUT DEPENDS ONLY ON THE PRESENT INPUT (0 OR 1) AND THE PREVIOUS OUTPUT.

THE NATURE OF THE OUTPUT IS UNIMPORTANT.

THE MAIN THING IS THAT THE CHANGES FROM ONE OUTPUT STATE TO THE NEXT ARE GIVEN BY DEFINITE RULES, CALLED THE TRANSITION RULES.

A model of "any possible computation" ...

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

14

The Turing Machine

Significance

- " D. Hilbert's question: Is it possible to find an algorithm for determining the truth or falsehood of any mathematical proposition?
- " Kurt Gödel (1931) published his famous *incompleteness theorem* – there exists a formula in the predicate calculus applied to integers that could neither be proved nor disproved within the predicate calculus.
- " Alonzo Church (1936) – any way to compute is analogous to TMs (*Church-Turing thesis*).

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

15

From Larry Gonick (1983), *The Cartoon Guide to the Computer*. Harper Collins Publishers, New York, page 191.

The Turing Machine

THE REASON TURING MACHINES ARE IMPORTANT IS THAT THEY ARE A WAY OF THINKING PHYSICALLY ABOUT LOGIC. ANY WELL-DEFINED, STEP-BY-STEP LOGICAL PROCEDURE CAN BE EMBODIED IN SOME TURING MACHINE.

TOT!
THERE'S A TURING MACHINE THAT CAN ADD!

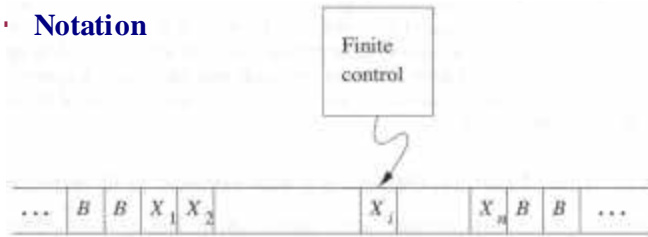
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

16

The Turing Machine

From Figure 8.8 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

Notation



- " Each move by a *Turing Machine* results in
 - B Change of state;
 - B writing a tape symbol in the cell just scanned; and
 - B moving the tape head left or right.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

17

The Turing Machine

- A *Turing Machine (TM)* is formally described by the 7-tuple $M = (Q, S, G, \delta, q_0, B, F)$ where
 - " Q is the finite set of *states* of the finite control
 - " S is the finite set of *input symbols*
 - " G is the finite set of *tape symbols*, and $S \subseteq G$
 - " $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a *transition function*
 - " $q_0 \in Q$ is the *start state*
 - " $B \in \Gamma$ is the *blank symbol*; and
 - " $F \subseteq Q$ is a set of *final states*.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

18

The Turing Machine

- *Instantaneous descriptions (IDs)* are denoted by strings $X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n$ where $q \in Q$, $\alpha = X_1 X_2 \dots X_{i-1}$, $\beta = X_i X_{i+1} \dots X_n \in \Gamma^*$ and
 - " the tape head is scanning the i th symbol from the left;
 - " $X_1 X_2 \dots X_n$ is the portion of the tape between the leftmost and the rightmost nonblank.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

19

The Turing Machine

- *Moves* are described by the \vdash_M notation similar to that used for PDAs (similarly for \vdash_M^*).
- Suppose $\delta(q, X_i) = (p, Y, L)$; i.e. the next move is *leftward*. Then,
 - " If $i=1$, then

$$X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash_M X_1 X_2 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$$
 - " If $i=n$ and $Y=B$, then

$$X_1 X_2 \dots X_{n-1} q X_n \vdash_M X_1 X_2 \dots X_{n-2} p X_{n-1}$$

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

20

The Turing Machine

- Suppose $\delta(q, X_i) = (p, Y, R)$; *i.e.* the next move is **rightward**. Then,

$$X_1 X_2 W X_{i-1} q X_i X_{i+1} W X_n \vdash_M X_1 X_2 W X_{i-1} Y p X_{i+1} W X_n$$

- If $i=n$, then

$$X_1 X_2 W X_{n-1} q X_n \vdash_M X_1 X_2 W X_{n-1} Y p$$

- If $i=1$ and $Y=B$, then

$$q X_1 X_2 W X_n \vdash_M p X_2 W X_n$$

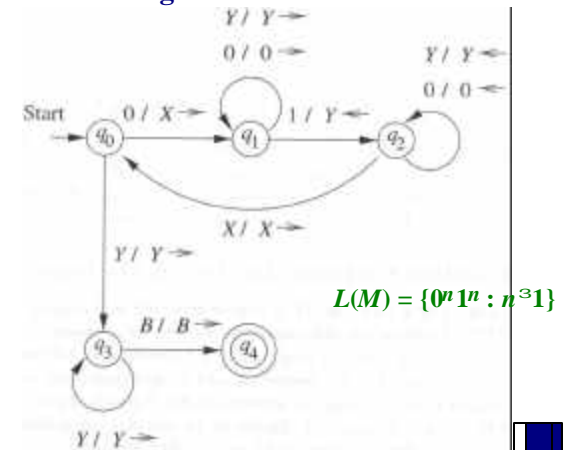
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

21

From Figure 8.10 of (A.T.L.C., Hopcroft, Motwani, & Ullman, 2001.

The Turing Machine

Transition Diagrams

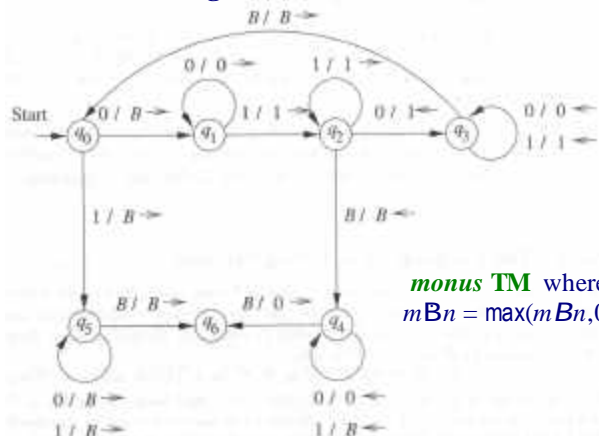


Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

22

The Turing Machine

Transition Diagrams



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

23

The Turing Machine

The Language of a Turing Machine

- Given TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$.
- $L(M)$ is the set of strings $w \in \Sigma^*$ such that $q_0 w \vdash^* \alpha p \beta$ for some state $p \in F$ and any tape strings $\alpha, \beta \in \Gamma^*$.
- The set of languages we can accept using a TM is called the **recursively enumerable languages**, or **RE languages**.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

24

The Turing Machine

- **Turing Machines and Halting**
 - " Another notion of "acceptance" commonly used for TMs is **acceptance by halting**. A TM **halts** if it enters a state $q \in Q$, scanning a tape symbol $X \in \Gamma$, and $\delta(q, X)$ is undefined.
 - " We assume that a TM always halts when it is in an accepting state. (*How can we do this?*)
 - " (Model of "algorithm") Languages with TMs that halt eventually, whether or not they accept, are called **recursive languages**.

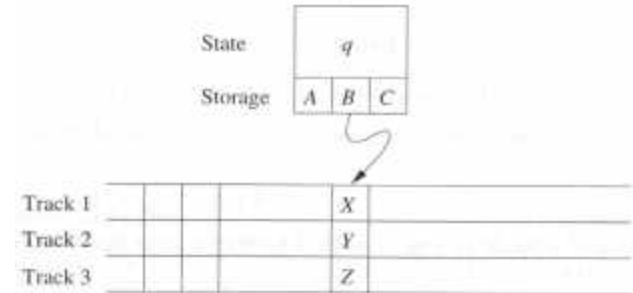
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

25

From Figure 8.13 of (A.T.C., Hopcroft, Motwani, & Ullman, 2001).

TM Programming Techniques

- **Storage in the State**
 - " treat state as a tuple to "store" data values.



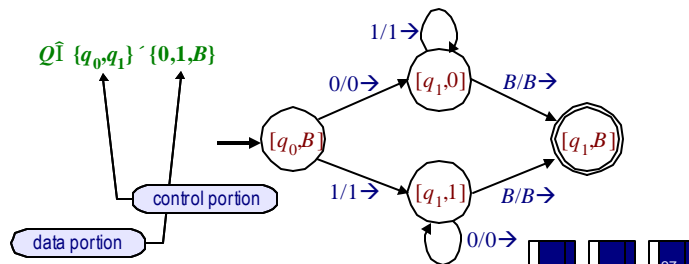
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

26

TM Programming Techniques

- **Example**
 - " Consider TM

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], B, \{[q_1, B]\})$$
 where $L(M) = 01^* + 10^*$



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

27

TM Programming Techniques

- **Multiple Tracks**
 - " each track can hold one symbol and the tape alphabet consists of tuples.
 - " with multiple tracks, one track can be used as a "mark" holder.
 - B TM M_1 where $L(M_1) = \{0^n 1^n : n \geq 1\}$ (see Slide #22)
 - B *monus* TM (see Slide #23)
 - B TM M_2 where $L(M_2) = \{wcw : w \in \{0,1\}^*\}$

$$Q \in \{q_i\} \times \{0,1,B\}$$

$$\Gamma \in \{B, I\} \times \{0,1,c,B\}$$

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

28

TM Programming Techniques

- Subroutines

- Since TMs are used to represent algorithms, TMs can be built by connecting interacting components (other TMs) or subroutines.
- Example: TM M that implements the “multiplication” function by reading input $0^m 10^n$ and computing 0^{mn} .
 - can be implemented by using a “copy” function that makes m copies of 0^n ...

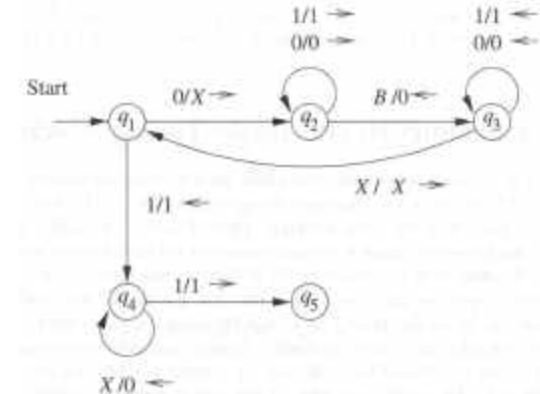
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

29

From Figure 8.14 of (A.T.L.C., Hopcroft, Motwani, & Ullman, 2001).

TM Programming Techniques

- The “copy” TM as a subroutine:

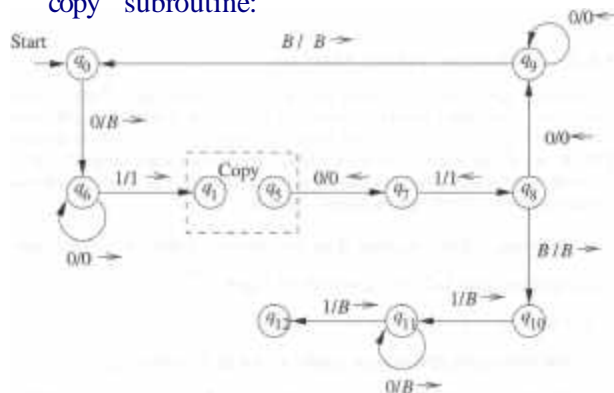


Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

30

TM Programming Techniques

- The complete “multiplication” TM that uses the “copy” subroutine:



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

31

Adapted from Elements of the Theory of Computation H.R. Lewis & C.H. Papadimitriou, 1998.

TM Programming Techniques

- A Hierarchical Notation for TMs
 - complex machines built from simpler ones.
 - consists of *basic machines* and *rules for combining machines*.
- Basic Machines
 - Head-moving machines*
 - M , abbreviated L , is the machine

$$M = (\{s, h\}, \Sigma, \Sigma \setminus \{B\}, \delta, s, B, \{h\})$$
 where $\delta(s, a) = (h, a, L)$
 - Similarly for machine M_{\perp} , abbreviated R .

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

32

Adapted from *Elements of the Theory of Computation* H.R. Lewis & C.H. Papadimitriou, 1998.

TM Programming Techniques

- **Basic Machines, continued ...**
 - **Symbol-writing machines**
 - For each $a \in \Sigma$, define M_a , abbreviated a , is the machine

$$M_a = (\{s,h\}, \Sigma, \Sigma \cup \{B\}, \delta, s, B, \{h\})$$
 where $\delta(s,b) = (h,a,R)$ for all $b \in \Sigma$
- **Rules for Combining Machines**
 - TMs will be combined in a way suggestive of the structure of a finite automaton.
 - Individual machines treated as states and connected by input tape symbols.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

33

Adapted from *Elements of the Theory of Computation* H.R. Lewis & C.H. Papadimitriou, 1998.

TM Programming Techniques

- **Conventions (Notational Conveniences)**
 - Unlabelled arrows assume any symbol $a \in \Sigma$.
 - Arrows labelled a denotes any $b \in \Sigma$ where $b \neq a$.
 - For a machine M , M^n for $n > 0$ denotes n copies of M connected by unlabelled arrows.
- **Note:**
 - This notation works nicely with subroutines!

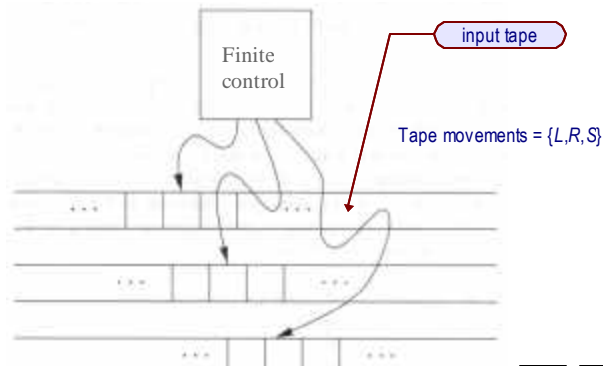
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

34

From Figure 8.16 of *IA TLG*, Hopcroft, Motwani, & Ullman, 2001.

TM Extensions

- **Multitape Turing Machines**



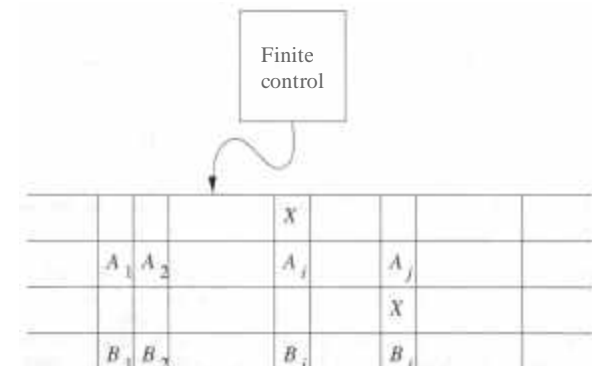
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

35

From Figure 8.17 of *IA TLG*, Hopcroft, Motwani, & Ullman, 2001.

TM Extensions

- **Equivalence of One-Tape and Multitape TMs**



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

36

TM Extensions

Theorem 8.9

- Every language accepted by a multitape TM is recursively enumerable.

Proof: (*Many-Tapes-to-One Construction*)

- Let language L be accepted by a k -tape TM M .
- Simulate TM M with one-tape TM N whose tape has $2k$ tracks.
 - Only half of the tracks hold contents of tapes of M .
 - The other half contain single marker for current head position at the corresponding tape of M .

TM Extensions

Proof: (*Many-Tapes-to-One Construction*), continued ...

- N simulates M by visiting the k head markers in order to determine the current tape symbols ...
 - A *count* is stored as a component of N 's finite control to keep track of the number of head markers to its left.
 - Additionally, for each head marker, N stores the scanned symbol as another component of its finite control.
- Hence, $Q_N \subseteq Q_M \times \{0, 1, \dots, k\} \times \Gamma^k$.

TM Extensions

Proof: (*Many-Tapes-to-One Construction*), continued ...

- Next, N has to
 - revisit each head marker on its tape;
 - change the symbol in the track representing the corresponding tapes of M ; and
 - move the head markers left or right, if necessary.
- Finally, N changes the state of M as recorded in its own finite control (based on tuples for Q_M).

TM Extensions

Theorem 8.10

- The time taken by the one-tape TM N of Theorem 8.9 to simulate n moves of the k -tape TM M is $O(n^2)$.

Important implication:

- If the multitape TM takes polynomial time, so does the one-tape TM.

TM Extensions

- A *nondeterministic Turing Machine (NTM)* differs from the deterministic variety by

$$\delta(q, X) \subseteq Q \times \Gamma \times \{L, R\}$$

where $q \in Q$ and $X \in \Gamma$

- NTMs can choose, at each step, any of the triples to be the next move.
- Similar definitions for IDs, moves, and language accepted by an NTM M apply as derived from the deterministic variety ...

TM Extensions

Theorem 8.11

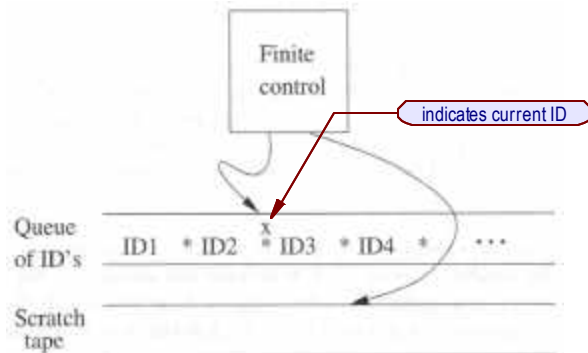
- If M_N is a nondeterministic TM, then there is a deterministic TM M_D such that $L(M_N) = L(M_D)$.

Proof:

- Idea: NTM $\rightarrow k$ -tape DTM $\rightarrow 1$ -tape DTM.
- Construct M_D as a k -tape DTM.
 - Use first (multitrack) tape to hold a *queue* of IDs of M_N , including the state of M_N , separated by inter-ID markers such as “*”.
 - Second tape is merely used as a marker.

TM Extensions

- Nondeterministic Turing Machines**



TM Extensions

To process current ID, M_D does the following:

- M_D examines state $q \in Q$ and scanned symbol $X \in \Gamma$ of current ID. The finite control of M_D knows what choices of moves M_N has for each state and symbol. If $q \in F$, then M_D accepts and simulates M_N no further.
- If $q \notin F$ and M_N has k moves, then M_D uses its second tape to copy the current ID and then make k copies of that ID at the end of the sequence of IDs on tape 1 (thereby adding them to the back of the *queue*).

TM Extensions

M_D processing the current ID, *continued ...*

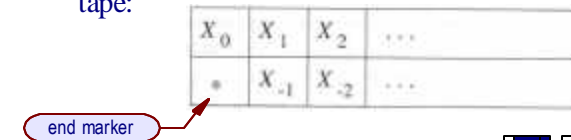
- " M_D modifies each of the k IDs just copied onto tape 1 to a different one of the k choices of moves that M_N has from its current ID.
- " M_D returns to the marked, current ID to erase the mark and to move the mark to the next ID to the right. The cycle repeats on this next ID ...

Note: This is similar to the use of a queue for the breadth-first search algorithm.

Restricted TMs

• Semi-infinite Tapes

- " A TM with a *semi-infinite* tape means that there are no cells to the left of the initial head position.
- " A TM with a semi-infinite tape simulates a TM with an infinite tape by using a two-track tape:



Restricted TMs

Theorem 8.12

- Every language accepted by a TM M_2 is also accepted by a TM M_1 with the following restrictions:
 - " M_1 's head never moves left of its initial position; and
 - " M_1 never writes a blank.

Restricted TMs

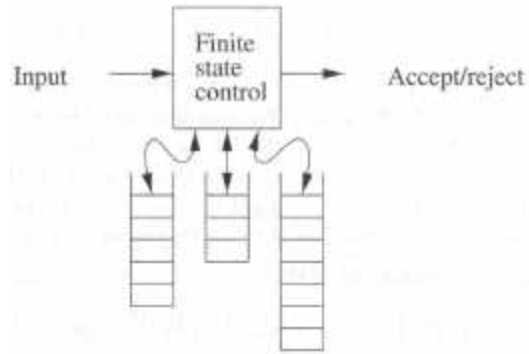
• Multistack Machines

- " Generalizations of the PDAs
- " Recall: TMs can accept languages that are not accepted by any PDA with one stack.
- " Note: PDA with two stacks can accept any language that a TM can accept!
- " In one move, a *multistack machine* can
 - , change to a new state $q \in Q$; and
 - , replace the top symbol of each stack with a string of zero or more stack symbols $X \in \Gamma^*$.

From Figure 8.20 of *ATLC*, Hopcroft, Motwani, & Ullman, 2001.

Restricted TMs

- **Example:** A 3-stack machine ...



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

49

Restricted TMs

Theorem 8.13

- If a language L is accepted by a TM M , then L is accepted by a two-stack machine.

Proof:

- **Idea:** Use one stack to hold what is to the left of the tape, and use the other to hold what is to the right of the tape.
- *Details in the textbook ...*

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

50

Restricted TMs

- **Counter Machines**
 - Two equivalent ways to think of a *counter machine*:
 - A stack with a bottom marker, say Z_0 , and one other symbol, say X , that can be placed on the stack.
 - Thus, the stack always looks like $XXXw XZ_0$, or specifically, X^nZ_0 .
 - A device that holds a non-negative integer with operations *increment-by-1*, *decrement-by-1*, and *test-if-zero*.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

51

Restricted TMs

The Power of Counter Machines

Note:

- **Every language accepted by a counter machine is recursively enumerable.**
 - Counter machines are special cases of stack machines, which are special cases of multitape TMs, which accept only recursively enumerable languages.
- **Every language accepted by a one-counter machine is a CFL.**
 - A one-counter machine is a special case of a one-stack machine; *i.e.*, a PDA.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

52

Restricted TMs

Theorem 8.14

- Every recursively enumerable language is accepted by a three-counter machine.

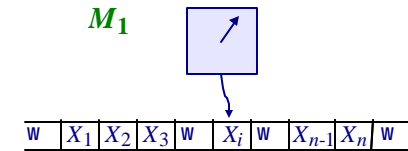
Proof:

- Idea: Use Theorem 8.13 to derive a two-stack machine, then develop a constructive algorithm for a *2-stacks-to-3-counters* conversion.

Restricted TMs

2-stacks-to-3-counters conversion:

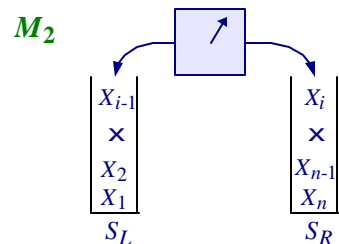
- Consider $L = \{a^n b^n c^n : n \geq 0\}$, TM M_1 where $L(M_1) = L$, and $w \in \{a, b, c\}^*$ where $|w| = n$.



- So, by Theorem 8.13 we can derive an equivalent 2-stack machine, M_2 ...

Restricted TMs

2-stacks-to-3-counters conversion, continued:

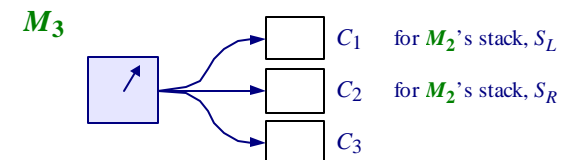


- If a stack has $r-1$ symbols, think of the stack contents as a base- r number with the symbols as the digits 1 through $r-1$.

Restricted TMs

2-stacks-to-3-counters conversion, continued:

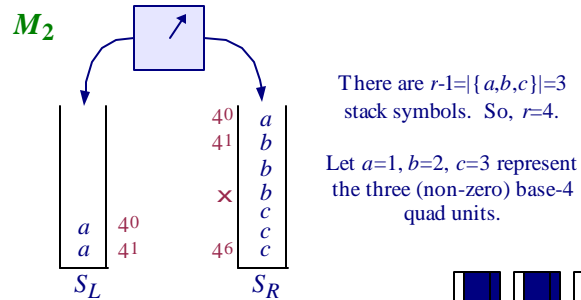
- So, for the 3-counter machine, M_3 , use one counter for each stack plus one “scratch” counter:



Restricted TMs

2-stacks-to-3-counters conversion, continued:

- Consider the string $w=aaabbbccc$ and the ID $aaqabbbccc$ for M_1 where $q \in Q \dots$



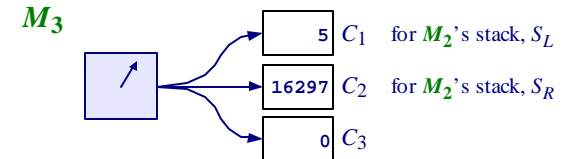
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

57

Restricted TMs

2-stacks-to-3-counters conversion, continued:

- which is represented by M_3 as



- Note: To move M_1 's read-write head one cell to the right requires popping X_i from S_R and pushing X_i into S_L for M_2 .

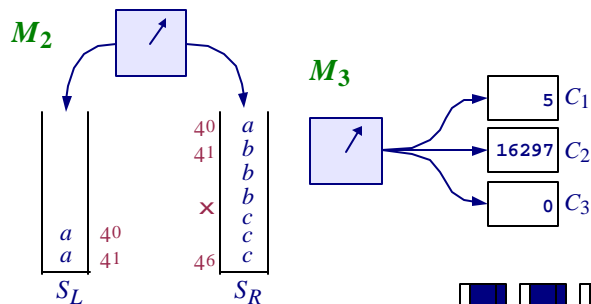
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

58

Restricted TMs

2-stacks-to-3-counters conversion, continued:

- Consider the move from $aaqabbbccc$ to $aaarbbbccc$ in M_1 where $q, r \in Q \dots$



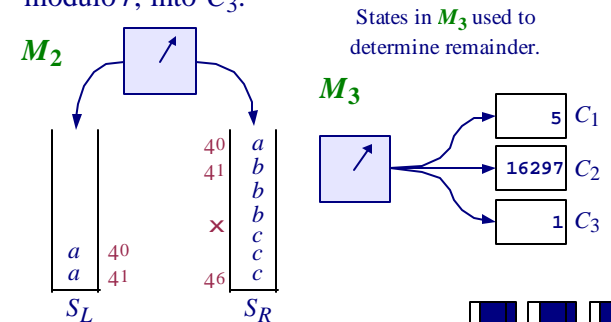
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

59

Restricted TMs

2-stacks-to-3-counters conversion, continued:

- Read top symbol of $S_R =$ store the remainder, C_2 modulo r , into C_3 .



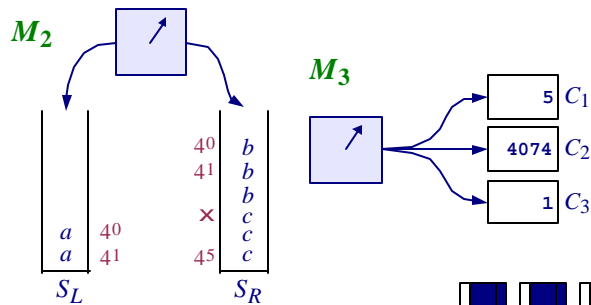
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

60

Restricted TMs

2-stacks-to-3-counters conversion, continued:

- Pop from S_R = divide C_2 by r , discarding the remainder.



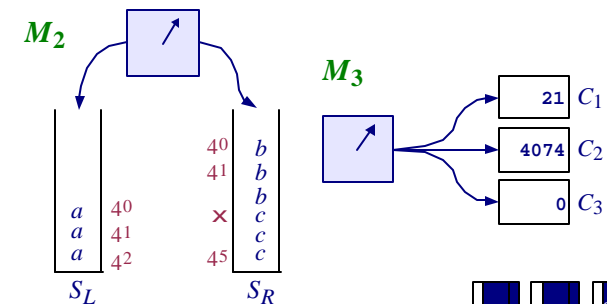
Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

61

Restricted TMs

2-stacks-to-3-counters conversion, continued:

- Push into S_L = multiply C_1 by r , then add the value stored in C_3 .



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

62

Restricted TMs

Theorem 8.15

- Every recursively enumerable language is accepted by a two-counter machine.

Proof:

- Idea: Develop a constructive algorithm for a 3-counters-to-2-counters conversion. Do this by representing the 3 counters i , j , and k , by a single integer $m=2^i3^j5^k$.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

63

Restricted TMs

3-counters-to-2-counters conversion:

- Store the number $m=2^i3^j5^k$ in one counter and use the other one as a "scratch" counter.
- Test if $i=0$ by moving count from one counter to the "scratch," counting modulo 2 in the state.
 - $i=0$ if and only if the number $m=2^i3^j5^k$ is not divisible by 2.
 - tests for $j=0$ and $k=0$ analogous.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

64

Restricted TMs

3-counters-to-2-counters conversion, continued:

- " Incrementing counters i , j , or k is equivalent to multiplications of the count $m=2^i3^j5^k$ by 2, 3, or 5; respectively.
- " Decrementing counters i , j , or k is equivalent to divisions of the count $m=2^i3^j5^k$ by 2, 3, or 5; respectively.

TMs and Computers

Real Computers

- " In one sense, a (real) computer has a finite number of states, and is thus *weaker* than a TM ...
 - We have to postulate an infinite supply of tapes, disks, or some peripheral storage device to simulate an infinite tape TM.
 - Assume a human operator can mount disks, keeping them stacked neatly on the sides of the computer.

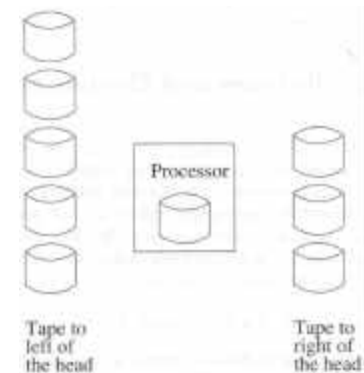
TMs and Computers

Simulating a TM by a Computer

- " A computer can simulate finite control, and mount one disk that holds the region of the tape around the tape head.
- " When the tape head moves off this region, the computer outputs the following request:
 - # Move the current disk to the top of the left or right pile;
 - # Mount the disk at the top of the other pile.
- " See the figure on the next slide ...

TMs and Computers

Simulating a TM by a Computer, continued ...



TMs and Computers

- **Simulating a Computer by a TM**
 - **Idea:** Simulation is at the level of stored instructions and words in memory ...
 - TM has one tape that holds all the used memory locations and their contents.
 - Other TM tapes hold the instruction counter, memory address, computer input file, and “scratch.”

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

69

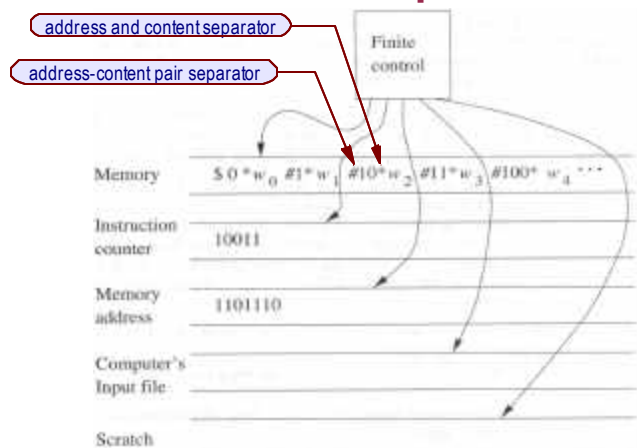
TMs and Computers

- **Simulating a Computer by a TM, *continued* ...**
 - Instruction cycle of computer simulated by:
 - # Find the word indicated by the instruction counter on the memory tape.
 - # Examine the instruction code (a finite set of options), and get the contents of any memory words mentioned in the instruction, using the “scratch” tape.
 - # Perform the instruction, changing word values as needed, and adding new address-value pairs to the memory tape, if needed.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

70

TMs and Computers



Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

71

TMs and Computers

- **Comparing the Running Times of Computers and TMs**
 - Use TM as a tool not just to examine *what can be computed*, but also *what can be computed with enough efficiency* that a problem's computer-based solution can be used in practice.
 - **Tractable** (polynomial; what can be solved efficiently) vs. **intractable** (more than polynomial; can be solved, but not fast enough for the solution to be usable)

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

72

TMs and Computers

- **Comparing the Running Times of Computers and TMs, *continued* ...**
 - " If a problem can be solved in polynomial time on a typical computer, then it can be solved in polynomial time by a TM, and conversely.
 - " Hence, conclusions about what a TM can or cannot do with adequate efficiency apply equally well to a computer.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

73

TMs and Computers

- **Comparing the Running Times of Computers and TMs, *continued* ...**
 - " If a computer can do *multiplication* of words whose length is not limited (*e.g.* to 64 bits, as on most computers), then the length of the longest value can double at each step.
 - # Assume that a computer is about to perform n multiplications requiring $T(n)$ steps ...
 - # It would take at least $O(2^{T(n)})$ steps for a TM to simulate $T(n)$ steps of the computer.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

74

TMs and Computers

- **Comparing the Running Times of Computers and TMs, *continued* ...**
 - " However, if we limit the length of words to, say, 64 bits, or we allow arbitrarily long words but only instructions that add at most 1 to the length in one step (*e.g.* addition) ...
 - # Assume that a computer is about to perform n additions requiring $T(n)$ steps ...
 - # Then $O(T(n)^3)$ steps of the TM suffice to simulate $T(n)$ computer steps.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

75

TMs and Computers

Theorem 8.17

- If a computer
 - " has only instructions that increase the maximum word length by at most 1; and
 - " has only instructions that a multitape TM can perform on words of length k in $O(k^2)$ steps or less;

Then the TM described in Slide #64 can simulate n steps of the computer in $O(n^3)$ of its own steps.

Chapter 8 Developed by B. Juliano ©2002, based on notes by J. Ullman

76

TMs and Computers

Theorem 8.18

- A computer of the type described in Theorem 8.17 can be simulated for n steps by a one-tape TM, using at most $O(n^6)$ steps of the TM.

Proof:

- Recall: A one-tape TM can simulate a multitape TM by squaring the number of steps, at most.

Copyright and Intellectual Property Notice

This document and parts of its contents are the *Intellectual Property* (IP) of Dr. Benjoe A. Juliano of the Department of Computer Science at California State University, Chico (CSUC). Dr. Juliano claims exclusive moral rights of ownership under current *Copyright Laws* (Title 17 of the United States Code and 1998 Digital Millennium Copyright Act) and *IP Policies/Guidelines* (CSUC EM83-08, EM97-07, and Article 39 of the CFA/CSU Contract) including, but not limited to:

- the exclusive right to copy, reproduce, and/or distribute this document;
- the right to be identified as the creator of this work (the right of attribution);
- the right to take action against false attribution; and
- the right to object to derogatory treatment of this work (the right of integrity).