CSE 2001:
Introduction to Theory of Computation

Fall 2012

Suprakash Datta

datta@cse.yorku.ca

Office: CSEB 3043

Phone: 416-736-2100 ext 77875

Course page: http://www.cs.yorku.ca/course/2001

Turing machine variants

- Multiple tapes
- 2-way infinite tapes
- Non-deterministic TMs

Multitape Turing Machines

A k-tape Turing machine M has k different tapes and read/write heads. It is thus defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{ \_ \} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

$$\delta: Q\backslash\{q_{accept},q_{reject}\} \times \Gamma^k \to Q \times \Gamma^k \times \{L,R\}^k$$

k-tape TMs versus 1-tape TMs

Theorem 3.13: For every multi-tape TM M, there is a single-tape TM M' such that L(M)=L(M').
Or, for every multi-tape TM M, there is an equivalent single-tape TM M'.

Proving and understanding these kinds of robustness results, is essential for appreciating the power of the Turing machine model.

From this theorem Corollary 3.9 follows:
A language L is TM-recognizable if and only if some multi-tape TM recognizes L.

Outline Proof Thm. 3.13

Let $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$ be a k-tape TM.
Construct 1-tape M' with expanded $\Gamma' = \Gamma \cup \underline{\Gamma} \cup \{\#\}$

Represent M-configuration

$$u_1 q_j a_1 v_1, \quad u_2 q_j a_2 v_2, \quad \ldots, \quad u_k q_j a_k v_k$$

by M' configuration,

$$q_j \# u_1 \underline{a}_1 v_1 \# u_2 \underline{a}_2 v_2 \# \ldots \# u_k \underline{a}_k v_k$$

(The tapes are seperated by #, the head positions are marked by underlined letters.)

Proof Thm. 3.13 (cont.)

On input $w=w_1\ldots w_n$, the TM M' does the following:

- Prepare initial string: $\#\underline{w}_1\ldots w_n\#\underline{\_}\#\cdots\#\underline{\_}\#\_\ \cdots$
- Read the underlined input letters $\in \Gamma^k$
- Simulate M by updating the input and the underlining of the head-positions.
- Repeat 2-3 until M has reached a halting state
- Halt accordingly.

PS: If the update requires overwriting a # symbol, then shift the part $\# \cdots \_$ one position to the right.

Non-deterministic TMs

A <u>nondeterministic Turing machine</u> M can have several options at every step. It is defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{ \_ \} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

$$\delta: Q \setminus \{q_{accept},q_{reject}\} \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L,R\})$$

Robustness

Just like k-tape TMs, nondeterministic Turing machines are not more powerful than simple TMs:

Every nondeterministic TM has an equivalent 3-tape Turing machine, which –in turn– has an equivalent 1-tape Turing machine.

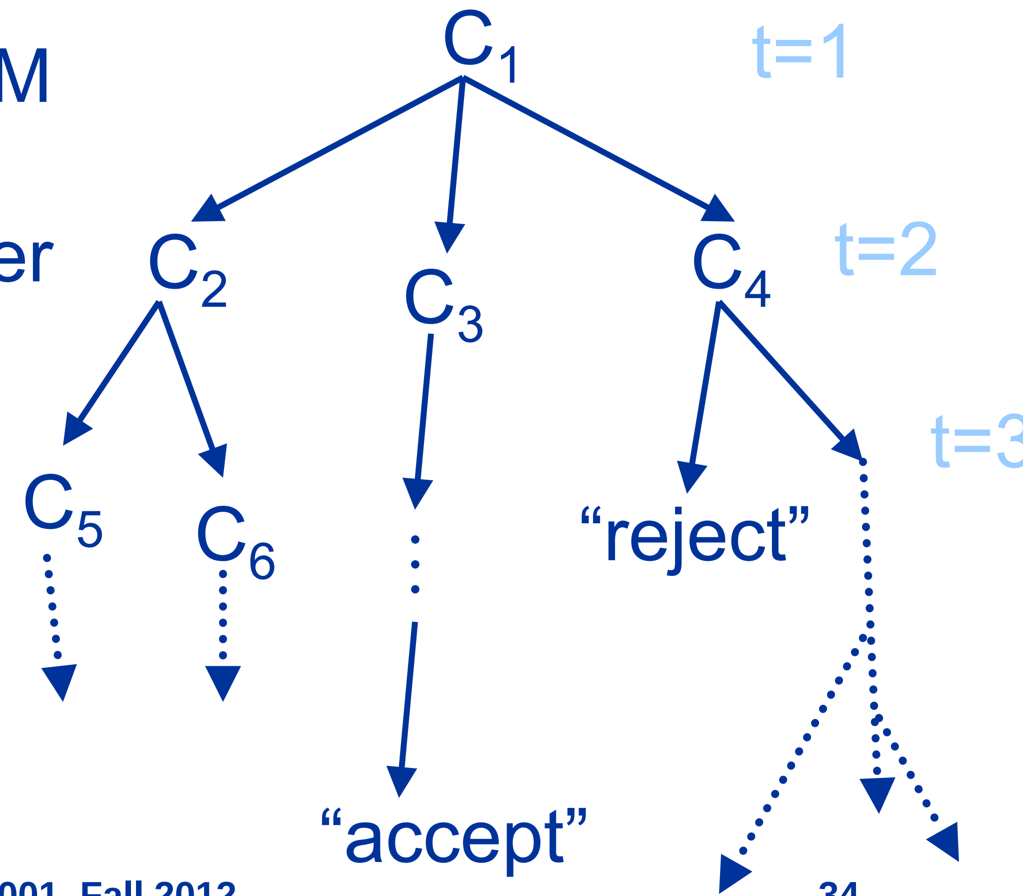
Hence: “A language L is recognizable if and only if some nondeterministic TM recognizes it.”

The Turing machine model is extremely robust.

Computing with non-deterministic TMs

Evolution of the n.d. TM represented by a tree of configurations (rather than a single path).

If there is (at least) one accepting leave, then the TM accepts.



Simulating Non-deterministic TMs with Deterministic Ones

We want to search every path down the tree for accepting configurations.

Bad idea: "depth first". This approach can get lost in never-halting paths.

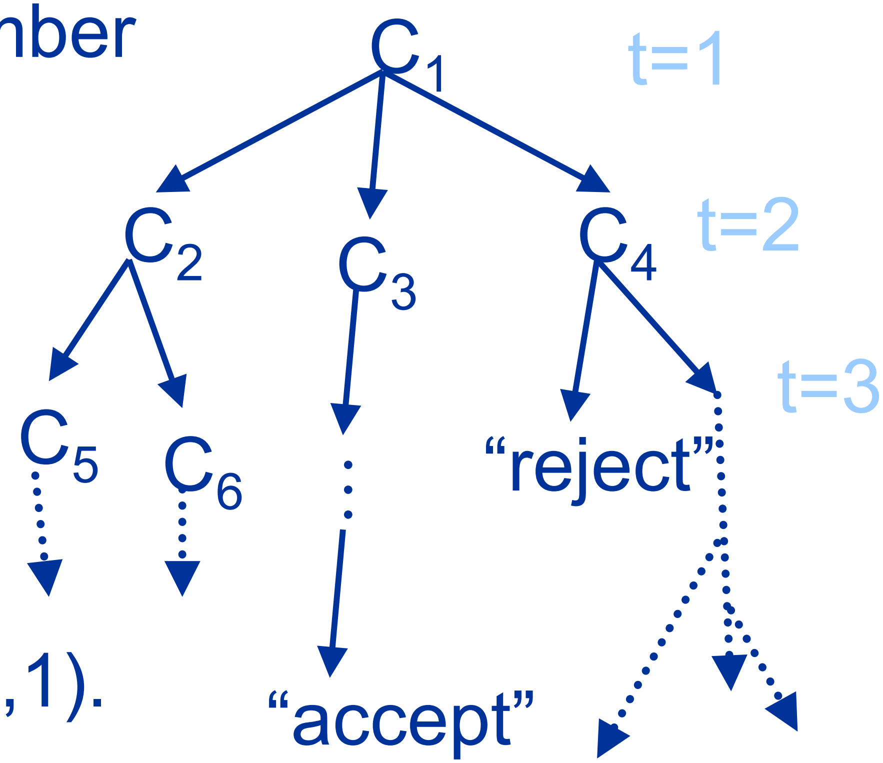
Good idea: "breadth first". For time step 1,2,… we list all possible configurations of the non-deterministic TM. The simulating TM accepts when it lists an accepting configuration.

Breadth First

Let b be the maximum number of children of a node.

Any node in the tree can be uniquely identified by a string $\in \{1,\ldots,b\}^*$.

Example: location of the rejecting configuration is (3,1).



With the lexicographical listing ε, (1), (2),…, (b), (1,1), (1,2),…,(1,b), (2,1),… et cetera, we cover all nodes.

Proof of Theorem 3.10

Let M be the non-deterministic TM on input w.

The simulating TM uses three tapes:
T1 contains the input w
T2 the tape content of M on w at a node
T3 describes a node in the tree of M on w.

1) T1 contains w, T2 and T3 are empty
2) Simulate M on w via the deterministic path to the node of tape 3. If the node accepts, "accept", otherwise go to 3)
3) Increase the node value on T3; go to 2)

Robustness

Just like k-tape TMs, nondeterministic Turing machines are not more powerful than simple TMs:

Every nondeterministic TM has an equivalent 3-tape Turing machine, which –in turn– has an equivalent 1-tape Turing machine.

Hence: “A language L is recognizable if and only if some nondeterministic TM recognizes it.”

Let’s consider other ways of computing a language…

Enumerating Languages

Thus far, the Turing machines were ‘recognizers’.

When a TM E generates the words of a language, E is an enumerator (cf. “recursively enumerable”).

A Turing machine E, enumerates the language L if it prints an (infinite) list of strings on the tape such that all elements of L will appear on the tape, and all strings on the tape are elements of L. (E starts on an empty input tape. The strings can appear in any order; repetition is allowed.)

Enumerating = Recognizing

Theorem 3.13: A language L is TM-recognizable if and only if L is enumerable.

Proof: ("if") Take the enumerator E and input w.
Run E and check the strings it generates.
If w is produced, then "accept" and stop, otherwise let E continue.
("only if") Take the recognizer M. Let $s_1, s_2, \ldots$ be a listing of all strings $\in \Sigma^* \supseteq L$.
For j=1,2,… run M on $s_1, \ldots, s_j$ for j time-steps.
If M accepts an s, print s. Keep increasing j.

Other Computational Models

We can consider many other ‘reasonable’ models of computation: DNA computing, neural networks, quantum computing…

Experience teaches us that every such model can be simulated by a Turing machine.

Church-Turing Thesis:

The intuitive notion of computing and algorithms is captured by the Turing machine model.

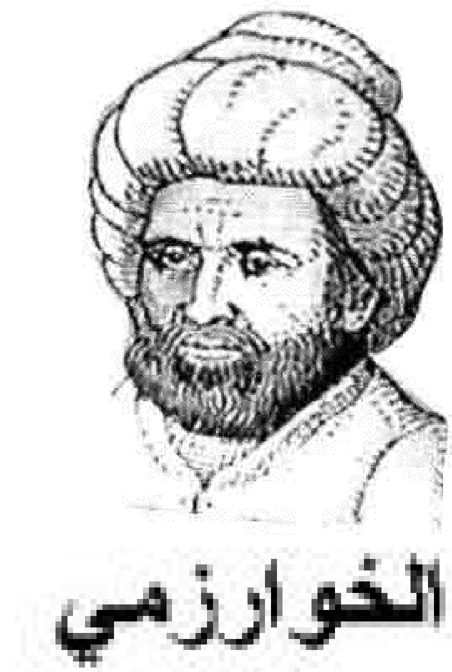
Importance of the Church-Turing Thesis

The Church-Turing thesis marks the end of a long sequence of developments that concern the notions of “way-of-calculating”, “procedure”, “solving”, “algorithm”.

Goes back to Euclid’s GCD algorithm (300 BC).

For a long time, this was an implicit notion that defied proper analysis.

“Algorithm”



After Abū ‘Abd Allāh Muhammed ibn Mūsā al-Khwārizmī (770 – 840)

His “Al-Khwarizmi on the Hindu Art of Reckoning” describes the decimal system (with zero), and gives methods for calculating square roots and other expressions.

“Algebra” is named after an earlier book.

Hilbert's 10th Problem

In 1900, David Hilbert (1862–1943) proposed his *Mathematical Problems* (23 of them).

The Hilbert's 10th problem is: **Determination of the solvability of a Diophantine equation.** Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: *To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.*

Diophantine Equations

Let $P(x_1,\ldots,x_k)$ be a polynomial in k variables with integral coefficients. Does P have an integral root $(x_1,\ldots,x_k) \in Z^k$?

Example: $P(x,y,z) = 6x^3yz + 3xy^2 - x^3 - 10$ has integral root $(x,y,z) = (5,3,0)$.

Other example: $P(x,y) = 21x^2 - 81xy + 1$ does not have an integral root.

(Un)solving Hilbert's 10th

Hilbert's *"…a process according to which it can be determined by a finite number of operations…"* needed to be defined in a proper way.

This was done in 1936 by Church and Turing.

The impossibility of such a process for exponential equations was shown by Davis, Putnam and Robinson.

Matijasevič proved that Hilbert's 10th problem is unsolvable in 1970.