Basic definitions of RAMs

- ➤ Data structure: an array of registers, each capable of containing an arbitrarily large integer, possibly negative.
- A RAM program Π = (π₁, π₂,..., π_m) is a finite sequence of *instructions* (of assembler language type).
- ► Register 0 serves as an accumulator
- ▶ Three modes of addressing: j' / j' / j' = j'
- ▶ Input is contained in a finite array of input registers $I = (i_1, ..., i_n)$.

© 2007 TKK, Laboratory for Theoretical Computer Science

T-79.5103	/ Autun	in 2007	More about Turir	ng Mach	Ines
Instruction	ı set				
Instruction	Ор	Semantics	Instruction	Ор	Semantics
READ	j	$r_0 := i_j$	JUMP	j	$\kappa := j$
READ	$\uparrow j$	$r_0 := i_{r_j}$	JPOS	j	if $r_0 > 0$, $\kappa := .$
STORE	j	$r_j := r_0$	JZERO	j	if $r_0 = 0$, $\kappa := .$
STORE	$\uparrow j$	$r_{r_j} := r_0$	JNEG	j	if $r_0 < 0$, $\kappa := .$
LOAD	x	$r_0 := x'$	HALT		$\kappa := 0$
ADD	x	$r_0 := r_0 + x'$	where x (res	sp. x')	is one of
SUB	x	$r_0 := r_0 - x'$	'j' / '↑ j' /	'=j'	
HALF		$r_0 := \lfloor \frac{r_0}{2} \rfloor$	(resp. r_j / r_j	r _{rj} / j)
				<i>,,,</i> ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	, ,

MORE ABOUT TURING MACHINES

- ► Random access machines
- ► Nondeterministic machines
- ► Universal Turing machine
- ► Halting problem
- ► Undecidability
- (C. Papadimitriou: Computational complexity, Chapters 2.6-3.3)

© 2007 TKK, Laboratory for Theoretical Computer Science

T-79.5103 / Autumn 2007

More about Turing Machines

1. Random Access Machines

- ► Can Turing machines implement arbitrary algorithms?
- ► Conjecture (a strengthening of Church's thesis):
 - "Any reasonable attempt to model mathematically computer algorithms and their time performance ends up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial."
- Further evidence: Turing machines can simulate random access machines (RAMs) which idealize a computer capable of handling arbitrarily large integers.

Operational semantics

T-79.5103 / Autumn 2007

- A configuration is a pair C = (κ, R) where κ is the number of the instruction to be executed and R = {(j₁, r_{j₁}), (j₂, r_{j₂}),..., (j_k, r_{j_k})} is a finite set of register-value pairs. The initial configuration: (1,0).
- For a RAM program Π and an input $I = (i_1, \ldots, i_n)$, the relation $(\kappa, R) \stackrel{\Pi, I}{\rightarrow} (\kappa', R')$ (yields in one step) is defined as follows:
 - $-\kappa'$ is the new value of κ after executing the κ th instruction of Π , -R' is R with possibly some pair (j,x) deleted and (j',x') added
 - according to the κ th instruction of Π .
- ► The relation $\xrightarrow{\Pi,I}$ induces $\xrightarrow{\Pi,I^k}$ and $\xrightarrow{\Pi,I^*}$ as previously.

Definition. Let *D* be a set of finite sequences of integers. A RAM Π computes $\phi: D \to \mathbf{Z}$ iff $\forall I \in D$, $(1, \emptyset) \stackrel{\Pi, I^*}{\to} (0, R)$ so that $(0, \phi(I)) \in R$.

 \odot 2007 TKK, Laboratory for Theoretical Computer Science

More about Turing Machines

Example. A RAM programming computing $\phi(x, y) = |x - y|$

Input:	Program:	Configurations:
I = (6, 10)	1. READ 2	(1, {})
	2. STORE 2	(2, {(0,10)})
$\phi(I) = 4$	3. READ 1	(3, {(0,10), (2,10)})
	4. STORE 1	(4, {(0,6), (2,10)})
	5. SUB 2	(5, {(0,6), (2,10), (1,6)})
	6. JNEG 8	(6, {(0,-4), (2,10), (1,6)})
	7. HALT	(8, {(0,-4), (2,10), (1,6)})
	8. LOAD 2	(9, {(0,10), (2,10), (1,6)})
	9. SUB 1	$(10, \{(0,4), (2,10), (1,6)\})$
	10. HALT	(0, {(0,4), (2,10), (1,6)})

Counting time and space

- ► The execution of each RAM instruction counts as one time step.
 - Addition of large integers takes place in one step.
 - Multiplication is not included in the instruction set.
- The size of the input is computed in terms of logarithms:
 For an integer *i*, b(*i*) is its binary representation with no redundant leading 0s and with a minus sign in front if negative.
 - The length of integer *i*, l(i) = |b(i)|.
 - For a sequence of integers $I = (i_1, \dots, i_n)$, $l(I) = \sum_{i=1}^n l(i_i)$.

\bigodot 2007 TKK, Laboratory for Theoretical Computer Science

More about Turing Machines

8

Time bounds for RAMs

Definition. Suppose that a RAM program Π computes a function $\phi: D \rightarrow \mathbf{Z}$ and let $f: \mathbf{N}^+ \rightarrow \mathbf{N}^+$.

The program Π computes ϕ *in time* f(n) iff

2007

for any
$$I \in D$$
, $(1, \emptyset) \xrightarrow{\Pi, I^k} (0, R)$ so that $k \leq f(\mathbf{I}(I))$.

Example. The multiplication of arbitrarily large integers is accomplished by a RAM in linear number of steps (i.e., the number of steps is propositional to the logarithm of the input integers).

RAM programs are powerful.

Example. A RAM for solving REACHABILITY can be found in the textbook.

Simulating TMs with RAMs

- The simulation of a Turing machine having an alphabet $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ is possible with a linear loss of efficiency.
- ► The domain of inputs for the simulating RAM is $D_{\Sigma} = \{(i_1, \dots, i_n, 0) \mid n \ge 0, 1 \le i_j \le k, j = 1, \dots, n\}.$
- ► For a language $L \subset (\Sigma \{\sqcup\})^*$, define $\phi_L : D_\Sigma \mapsto \{0, 1\}$ so that $\phi_L(i_1, \dots, i_n, 0) = 1$ iff $\sigma_{i_1} \cdots \sigma_{i_n} \in L$.
 - \bigcirc Deciding *L* is equivalent to computing ϕ_L .

Theorem. Let $L \in TIME(f(n))$. Then there is a RAM program which computes the function ϕ_L in time O(f(n)).

Proof sketch. Given a Turing machine M deciding L construct for each state of M a subroutine (a RAM program) which simulates the state.

 \odot 2007 TKK, Laboratory for Theoretical Computer Science

T-79.5103 / Autumn 2007	More about Turing Machines
-	
Simulating RAMs with	TMs
 Any RAM can be simu polynomial loss of effic 	lated by a Turing machine with only a iency.
➤ The binary representat denoted by b(I), is the	ion of a sequence $I = (i_1, \dots, i_n)$ of integers, string $\mathbf{b}(i_1); \dots; \mathbf{b}(i_n)$.
	of finite sequences of integers and ne M computes ϕ iff for any $I \in D$,
Λ	$A(\mathbf{b}(I)) = \mathbf{b}(\mathbf{\phi}(I)).$

Simulating RAMs with TMs

Theorem. If a RAM program computes ϕ in time f(n), then there is a 7-string Turing machine *M* which computes ϕ in time $O(f(n)^3)$. Proof sketch

roof skelch.

The strings of the machine serve the following purposes:

1. Input

- Representation of register contents ...; b(i) : b(r_i);...⊲
 (update: erase old value by ⊔s and add new value to the right)
- 3. Program counter
- 4. Register address currently sought
- 5.-7. Extra space reserved for the execution of instructions

 \bigodot 2007 TKK, Laboratory for Theoretical Computer Science

I-79.5103 / Autumn 2007 More about Turing Machines
Proof skecth—cont'd.
► Each instruction of the RAM program is implemented by a group of states of <i>M</i> .
 Simulating an instruction of Π on M takes O(f(n)l) steps where l is the size of the largest integer in the registers
(as there are $O(f(n))$ pairs on string 2).
Simulating Π on M takes $O(f(n)^2 l)$ steps.
► It remains to establish that $l = O(f(n))$.
Claim: After the <i>t</i> th step of a RAM program Π computation on input <i>I</i> , the contents of any register have length at most $t + l(I) + l(B)$ where <i>B</i> is the largest integer referred to in an instruction of Π .

12

Inductive proof of the claim

- > Base case: the claim is true when t = 0.
- ▶ Induction hypothesis: the claim is true after the (t-1)th step.
- Case analysis over instruction types of the *t*th instruction: Most of the instruction do no create new values (jumps, HALT, LOAD, STORE, READ). For these the claim continues to hold after the execution of the instruction.

Consider arithmetic, say ADD, involving two integers *i* and *j*. The length of the result is one plus the length of the longest operand which is by induction hypothesis at most t - 1 + l(I) + l(B).

Hence, the result has length at most t + l(I) + l(B).

 \odot 2007 TKK, Laboratory for Theoretical Computer Science

T-79.5103 / Autumn 2007 More about Turing Machines
2. Nondeterministic Machines
 Nondeterministic machines are an unrealistic model of computation.
 Nondeterministic TMs can be simulated by deterministic TMs with an exponential loss of efficiency.
An open question: is a polynomial simulation possible? (i.e. P = NP?)

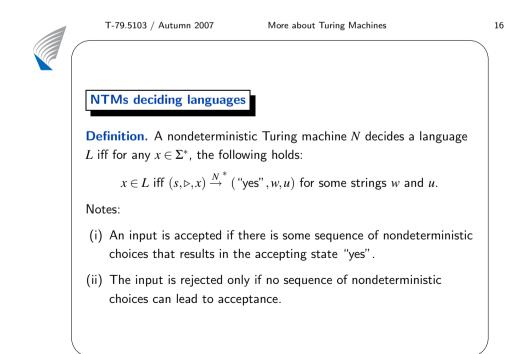
Transition relation

Definition. A nondeterministic Turing machine (NTM) is a quadruple $N = (K, \Sigma, \Delta, s)$ like the ordinary Turing machine except that Δ is a *transition relation* (rather than a transition function):

 $\Delta \subset (K \times \Sigma) \times [(K \cup \{h, \text{``yes''}, \text{``no''}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\}]$

- ➤ Configurations are defined as before but "yields" is a relation (rather than a function) for a NTM N: $(q, w, u) \xrightarrow{N} (q', w', u')$ iff there is a tuple in Δ that makes this a legal transition.
- The power of nondeterminism boils down to the weak input-output behavior demanded of NTMs.

C 2007 TKK, Laboratory for Theoretical Computer Science



Time complexity classes

Definition. A nondeterministic Turing machine N decides a language L in time f(n) iff N decides L and for any $x \in \Sigma^*$,

if $(s, \triangleright, x) \xrightarrow{N^k} (q, w, u)$, then $k \le f(|x|)$.

 \bigcirc All computation paths should have length at most f(|x|).

Definition. A *time complexity class* NTIME(f(n)) is a set of *languages* L such that L is decided by a nondeterministic Turing machine in time f(n).

 \odot 2007 TKK, Laboratory for Theoretical Computer Science

T-79.5103 / Autumn 2007

More about Turing Machines

Time complexity classes—cont'd

Definition. The set \mathbf{NP} of all languages decidable in polynomial time by a nondeterministic Turing machine:

$$\mathbf{NP} = \bigcup_{k>0} \mathbf{NTIME}(n^k)$$

Theorem. Suppose that a language *L* is decided by a NTM *N* in time f(n). Then it is decided by a 3-string deterministic TM *M* in time $O(c^{f(n)})$, where c > 1 is some constant depending on *N*. Thus

$$\mathbf{NTIME}(f(n)) \subseteq \bigcup_{c>1} \mathbf{TIME}(c^{f(n)}).$$

Proof sketch

- ► Let $N = (K, \Sigma, \Delta, s)$ be a NTM.
- Let d be the degree on nondeterminism (maximal number of possible moves for any state-symbol pair in Δ) for N.
- ➤ Any computation of N is a sequence of nondeterministic choices. A sequence of t choices is a sequence of t integers from 0,1,...,d-1.
- ➤ The simulating machine *M* considers all such sequences of choices in order of increasing length and simulates *N* on each.
- ➤ While considering a sequence (c₁, c₂,...,c_t), M maintains the sequence on its second string.
- ➤ With sequence (c₁, c₂,..., c_t) M simulates the actions that N would have taken if N had taken choice c_i at step i for its first t steps.

 \bigodot 2007 TKK, Laboratory for Theoretical Computer Science

T-79.5103 / Autumn 2007

More about Turing Machines

20

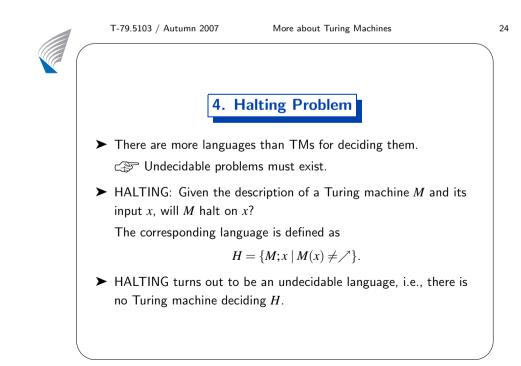
Proof sketch—cont'd

- If a sequence leads N to halting with "yes", then M does, too.
 Otherwise it considers the next sequence.
- Generating the next sequence (on the second string) is like calculating the next integer in base d.
- When should M reject?
 (Note that the bound f(n) is not available to M!)
- Machine *M* rejects its input whenever it has simulated all sequences of choices of length *t* and finds among them *no continuing computation* (i.e. all computations of length *t* end with halting).
- ► The time bound $O(c^{f(n)})$ is then established as the product of - the number of sequences $\sum_{t=1}^{f(n)} d^t = O(d^{f(n)+1})$ and
 - the cost of each sequence which is $O(2^{f(n)})$.

Encoding TMs using integers

- An entire TM M = (K,Σ,δ,s) is encoded as b(|Σ|);b(|K|);e(δ) where all integers i are represented as b(i) with exactly [log(|Σ| + |K| + 6)] bits and e(δ) is a sequence of pairs ((q,σ), (p,ρ,D)) describing the transition function δ.
 (The symbol M is also used to denote this description of M).
- Then U simulates M using a string S₁ for the description of M and a string S₂ for the current configuration (q,w,u) of M. Simulation of a step of M is performed as follows:
 - (i) Scan S_2 to find an integer corresponding to a state.
 - (ii) Search S_1 for a rule of δ matching the current state.
 - (iii) Implement the rule. (When M halts, so does U.)

© 2007 TKK, Laboratory for Theoretical Computer Science



Space complexity

- For space considerations, a nondeterministic Turing machine with input and output is needed.
- ► Given a k-string NTM N with input and output, we say that N decides language L within space f(n) if N decides L and for any $x \in (\Sigma \{\sqcup\})^*$, if $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon,) \xrightarrow{N^*} (q, w_1, u_1, \ldots, w_k, u_k)$, then $\sum_{i=2}^{k-1} |w_i u_i| \le f(|x|)$.

Example. REACHABILITY is nondeterministically solvable within space $O(\log n)$.

© 2007 TKK, Laboratory for Theoretical Computer Science

T-79.5103 / Autumn 2007

More about Turing Machines

3. Universal Turing Machine

- ► A TM has a fixed program which solves a single problem.
- ➤ A universal Turing machine U takes as input a description of another Turing machine M and an input x for M, and then simulates M on x so that U(M;x) = M(x).
- ► Encoding a Turing machine $M = (K, \Sigma, \delta, s)$ using integers:
 - $-\Sigma = \{1, 2, \ldots, |\Sigma|\}$

$$-K = \{ |\Sigma| + 1, |\Sigma| + 2, \dots, |\Sigma| + |K| \}$$

- $-s = |\Sigma| + 1$
- $|\Sigma| + |K| + 1, |\Sigma| + |K| + 2, \dots, |\Sigma| + |K| + 6 \text{ encode}$ $\leftarrow, \rightarrow, -, h, \text{"yes"}, \text{"no", respectively.}$

Properties of HALTING

- ► HALTING is recursively enumerable (r.e. for short).
 Proof: A slight variant U' of the universal Turing machine U accepts H: all halting states of U are forced to be "yes" states.
 1. If M; x ∈ H, then M(x) ≠ ↗, U(M,x) ≠ ↗, U'(M,x) = "yes".
 2. If M; x ∉ H, then M(x) = U(M,x) = U'(M,x) = ↗.
- ➤ HALTING is *complete* for r.e. languages, i.e. any r.e. language L can be reduced to it.

Proof: Let M_L be the machine *accepting* L.

Then $x \in L$ iff $M_L; x \in H$, i.e., deciding L can be reduced to deciding H.

This holds as $x \in L$ iff $M_L(x) =$ "yes" iff $M_L(x) \neq \nearrow$ iff $M_L; x \in H$.

► HALTING is not recursive.

T-79.5103 / Autumn 2007

 \odot 2007 TKK, Laboratory for Theoretical Computer Science

More about Turing Machines

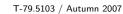
Proof skecth

- > Assume that H is recursive, i.e., some M_H decides H.
- ➤ There is no satisfactory answer to D(D):

If $D(D) = \nearrow$, then $M_H(D;D) =$ "yes", i.e., $D(D) \neq \nearrow$, a contradiction.

Hence, $D(D) \neq \nearrow$. Then $M_H(D,D) \neq$ "yes". But as M_H decides H, $M_H(D,D) =$ "no" and, hence, $D; D \notin H$, i.e. $D(D) = \nearrow$, a contradiction.

 $rac{1}{3}$ H is not recursive.



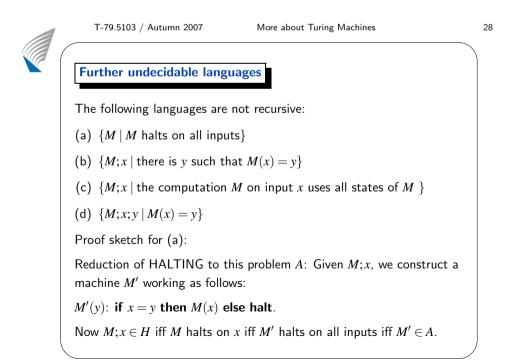
5. Undecidability

- ► HALTING spawns a range of other undecidable problems using a *reduction technique*.
- ➤ To show a problem A undecidable, establish that if there is an algorithm for A, then there is an algorithm for HALTING.
- ➤ This can be shown by devising a *reduction from* HALTING to A, i.e., a transformation t (computable by a Turing machine) of the input M;x of HALTING to the input t(M;x) of A such that

$M; x \in H$ iff $t(M; x) \in A$.

This implies that A is undecidable as follows. Assume A is decided by a Turing machine M_A . This leads to a contradiction as then H is decided by a machine M_H which runs first on input M; x machine M_t computing the transformation t and then M_A on the result: $M_H(M;x): y := M_T(M;x)$; if $M_A(y) = "yes"$ then "yes" else "no"

 \bigodot 2007 TKK, Laboratory for Theoretical Computer Science



Learning Objectives

- ➤ You should be able to justify why Turing machines make a powerful model of algorithms/computation.
- ► Basic understanding of differences between deterministic and nondeterministic Turing machines.
- ► The definitions and background of complexity class NP and the problem whether $\mathbf{P} = \mathbf{NP}$ or not.
- ► The definitions of recursive and recursively enumerable languages (including examples of such languages).

© 2007 TKK, Laboratory for Theoretical Computer Science

Properties of recursive languages

Proposition. If L is recursive, then so is \overline{L} (the complement of L).

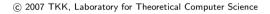
Proposition. A language L is recursive iff both L and \overline{L} are recursively enumerable.

Proof sketch.

 (\Rightarrow) By previous proposition and the fact that every recursive language is also recursively enumerable.

- (\Leftarrow) Simulate M_L and $M_{\overline{L}}$ on input x in an interleaved fashion:
- $\text{ If } M_L \text{ accepts, return "yes" and }$
- if $M_{\overline{T}}$ accepts, return "no".

 \bigcirc The complement \overline{H} of H is not recursively enumerable.



More about Turing Machines

T-79.5103 / Autumn 2007
Recursively enumerat

Proposition. A language *L* is *recursively enumerable* iff there is a machine *M* such that $L = E(M) = \{x \mid (s, \triangleright, \varepsilon) \xrightarrow{M^*} (q, y \sqcup x \sqcup, \varepsilon)\}.$

Any non-trivial property of Turing machines is undecidable:

Theorem. (Rice's Theorem) Let C be a proper non-empty subset of r.e. languages. Then the following problem is undecidable: given a Turing machine *M*, is $L(M) \in C$?

Here L(M) is the language accepted by a Turing machine M.