Brian Chen
chen187
1002297034
University of Toronto, Scarborough Winter 2017 CSCC63

CSCC63 – Computability and Computational Compl. Finals Notes

Part I - Computability

Computability

Problems that cannot be solved vs problems that can be solved

Turing Machines

Turing Machine – an infinite tape containing squares holding one symbol, and a head-reader. This head can move left, right, read or write a symbol.

Q: Set of states
Σ: Finite alphabet
Γ: Tape alphabet
q_0: Start state
q_{accept}: Accept state
q_{reject}: Reject state
δ: $Q \times \Gamma_\varepsilon \rightarrow Q \times \Gamma_\varepsilon \times \{L, R\}$: Transition function

We have transitions as $(qi, A) \rightarrow (qj, B, L/R)$ (upon A in state q_i we go to q_j and write B and move Left or Right)

Recognizable – A language is recognizable if there exists a TM M that on any input w, TM M either accepts, rejects, or loops

Co-recognizable – A language is co-Turing recognizable if for L-bar there exists TM M that recognizes it

Decidable – A language is decidable if there exists a TM M that on any input w, TM M either accepts or rejects

Non-deterministic Turing machines – Instead of straight line progression, each state has a set of possible ways to continue. Instead of just seeing straight, we need to breadth first search and see if there exists any accepting state. Every NDTM has an equiv. DTM

- Non-deterministic TM "accepts" if there is at least one path in computation tree that leads to q accept.
- Non-deterministic TM "rejects" if every path leads to q reject.
- Non-deterministic TM "loops" if no path accepts and at least one path never halts.

Church Turing Thesis – A function on the natural numbers that is computable by a human following an algorithm is therefore computable by a Turing machine as well. Algorithms are equivalent to TMs.

Theorem – A language is decidable if it is Turing recognizable and co-Turing recognizable.

Enumerators:

Enumerate all the possible strings in $\Sigma*$, where Σ is our input alphabet – sometimes with repeats and infinitely

(B) We loop running E for successively more steps and check whether it prints anything on the last step.

```
M = "On input w:
 1  for s = 1 to ∞
 2      run E for s steps
 3      if E prints a string x on step s then
 4          ▷ code that uses x
```

Dovetailing:

When we have infinite going both ways (e.g. M may take inf steps to accept w_i i.e. loop, we only do s operations on M<w_i>)

We give a corrected TM R.

```
R = "On input ⟨M⟩:
 1  for s = 1 to ∞
 2      for i = 1 to s
 3          run M on input w_i for s steps
 4          if M accepts w_i within s steps then
 5              accept"
```

Proving a language L, recognizable:

Let $w_1, w_2, w_3, \cdots$ be an effective enumeration of Σ^*, where Σ is our input alphabet.
We give a TM D that recognizes L.

```
D = "On input ⟨M⟩:
 1  for s = 1 to ∞
 2      for i = 1 to s
 3          for j = i + 1 to s
 4              run M on w_i for s steps
 5              if M accepts w_i within s steps then
 6                  run M on w_j for s steps
 7                  if M accepts w_j within s steps then
 8                      accept"
```

Part II – Reduction

Unrecognizable Languages

A_{TM} = { M<w> | M is a TM that accepts input w }

Brian Chen
chen187
1002297034
HALT = {<M, w> | M is a TM that halts on input string w}
E_{TM} = = {<M>| M is a TM and L(M) = ∅}
EQ_{TM} = {<M1, M2> | M1, M2 are TMs and L(M1) = L(M2)}

Mapping Reductions (Turing Machines)

Useful for proving a language not **recognizable** or not Turing-recognizable → we simply prove that if a Turing machine can recognize a language L, then it would be able to recognize unrecognizable languages that are given to us (e.g. HALT)
We say that **A <=$_m$ B** as the reduction of A to B. If B is decidable, then A is decidable. If B is recognizable then A is too. The opposite holds when we use A->B, if A is **undecidable,** then so too is B. If A is **unrecognizable** then so too is B.
For example, if we wish to prove a language L unrecognizable, we seek to have HALT <=$_B$ L or L^C

Rice's Theorem

If *P* is a nontrivial property of recognizable languages, then *P* is undecidable. That is, there exists one language that satisfies the property and one language that does not. A property of recognizable languages means a property of L(M) for TMs M that depends only on L(M) and on no other aspect of M.
For example, consider the following languages that can be found decidable or undecidable using Rice's Theorem
E_{TM} = {⟨M⟩ | L(M) = {}} depends only on L(M) and is non-trivial: L(M) = {} for some M, and L(M) ≠ {} for others
L = {⟨M⟩ | M accepts 00101}
INF_{TM} = {⟨M⟩ | L(M) contains infinitely many strings }

Part III - Computability

Theorem – Every t(n) time single tape non deterministic TM has an equivalent $2^{O(t(n))}$ time single-tape deterministic TM
P – The class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine
- ➔ Describe stages of computation that are obviously computable in polytime
- ➔ At most have polynomially many stages

NP – A language belongs to NP iff it is decided by some non-deterministic polynomial time TM

P – The class of languages for which membership can be decided quickly
NP – The class of languages for which membership can be verified quickly

---

Verifiers – A verifier for a language A is algorithm V where A = {w | V accepts <w, c> for some string c}
We cannot prove the existence of a single language in NP that is not in P, therefore we do not know if P = NP or not.
There is no natural certificate for non-membership (as for $HAMPATH^C$ etc.) therefore we cannot conclude that coNP == NP

Part IV – Polytime Reducibility

NP-Complete Problems

HAMPATH - {⟨G⟩ | G is an undirected graph that contains a Hamiltonian path}
CLIQUE = {⟨G, k⟩ | G is an undirected graph with a k-clique.}
SUBSET-SUM = {⟨S, t⟩ | S = {x1, … , xk} and for some {y1, …, y2} ⊆ S, ∑ yi = t.}
3-COL = {<G> | G is 3-colourable? (I.e., does G have a 3-colouring? Or put another way, is there a way to colour the nodes of G with at most 3 colours so that no adjacent nodes have the same colour?)}
VERTEX-COVER = {<G, k> | G contains a vertex cover of size k (set of nodes such that every edge is covered by some node in set}}
3-SAT = {<F> | Is F satisfiable}
SAT = {<F> | Is F satisfiable} – NPC by **Cook Levin Theorem**

Polytime Reductions

A ≤$_p$ B and B ∈ P ➔A ∈ P
A ≤$_p$ B and B ∈ NP ➔A ∈ NP
If we know that we can verify B in polytime, then we know we can verify A in polytime.

A language is NP-Complete if

A ∈ NP
B <=$_p$ A, that is, A is NP-hard

Self-Reducibility

Decision problems – Answer is yes/no given input x
Search problems – Answer is y given input x
Search problems are only polynomially more difficult than corresponding decision problems – efficient solutions for decision problems can be used to solve search problem efficiently – this is called **self reducibility**
Optimization Problems – Use binary search to find the optimal value (e.g. Max-Clique, Min-Vertex-Cover, Max-Independent-Set) e.g. for graphs of n nodes, query binary search on Clique(k//2) etc. until you find largest/smallest clique.
NP-Intermediate – Problems in NP that are not NP-complete e.g. Graph isomorphism, if theres a one-to-one and onto function for nodes in G to nodes in F.

Part V– Space Complexity

SPACE(f(n)) = L that can be decided by a TM running in worst-case space O(f(n))
NSPACE(f(n)) = L that can be decided by a NTM running in worst-case space O(f(n)).
PSPACE = {all languages decided in polyspace}
Space is defined as the length of tape used.
Savitch's Theorem – NSPACE(f(n)) ⊂ SPACE($f(n)^2$)

NPSPACE = PSPACE
P ⊂ **PSPACE**
NP ⊂ N**PSPACE**

Brian Chen
chen187
1002297034

P ⊂ NP ⊂ NPSPACE = PSPACE

PSPACE-Completeness

A language A is PSPACE Complete if

A ∈ PSPACE

∀ B <=$_p$ A, that is, A is PSPACE-hard

PSPACE-Complete Problems

TQBF = { φ | φ is a true, fully quantified boolean formula over domain = {0, 1} }

- ➔ If φ has no quantifiers, then evaluate φ and accept iff it is true.
- ➔ Ifφ=∃xψ, recursively evaluate ψ[x=0] and ψ[x= 1] and accept iff either computation accepts.
- ➔ If φ = ∀xψ, then recursively evaluate ψ[x = 0] and ψ[x = 1] and accept iff both computations accept.

Games – We can create two-player games that we can ask – given a configuration, does there exist a winning strategy that P1 can win

Part VI– Approximate Algorithms

L = language decided by a TM in SPACE(Log n)

NL = language decided by a NTM in SPACE(Log n)

We measure "work" space independently of input space by using a 2-tape TM – the second tape is a working tape.

e.g. instead of crossing off 1s and 0s, we increment binary for 1s and decrement for 0s. This takes log n space!

Other spaces –

EXP = {languages decided by TMs in time of O(2^{n^k})}

NEXP= {languages decided by NTMs in time of O(2^{n^k})}

EXPSPACE = {languages decided by TMs in space O(2^{n^k})}

By Savitch's theorem,

L⊆NL⊆P ⊆NP ⊆PSPACE⊆EXP ⊆ NEXP ⊆ EXPSPACE