Guide to Exam 3

Time and Place

Exam 3 will be held in 5 Kariotis Hall on Monday, April 24 from 8:00-10:00 a.m.

The exam is open book: You may use your notes, homeworks, any handouts and solutions provided to you, the textbook (Sipser), and any other paper-based references.

This is not a cumulative final exam. It will instead focus on material covered in the last third of the course, since Exam 2. In particular, it will not include questions like those appearing on Exams 1 and 2 or in Homeworks 1-6. Instead, expect to see questions pertaining to material appearing in Homeworks 7-9.

General Things to Know

- Decision problems: decidable and undecidable languages
- Turing-recognizable and non-Turing-recognizable languages, co-Turing-recognizability
- Proofs of undecidability or non-Turing-recognizability using mapping reductions
- Time complexity of deterministic and nondeterministic Turing machines; polynomial vs. exponential time
- The language classes P and NP
- Polytime reductions and NP-completeness

Specific Things You Should Know How to Do

- Design a Turing machine (possibly a nondeterministic one), using an informal high-level description, to do a specified task. This TM may include "calls" to other TMs or simulate their operation.
- Provide an asymptotic time-complexity analysis of a Turing machine, especially to distinguish polynomial running time from super-polynomial (e.g., exponential) running time.

The purpose of designing such a TM and/or to provide such time-complexity analysis may be to:

- Prove closure or non-closure of a specified language class (especially the decidable languages, the Turing-recognizable languages, P, or NP) under a certain operation applied to its languages.
- Prove that a given language is: decidable, Turing-recognizable, in P, or in NP.
- Prove that a given language is: undecidable, non-Turing-recognizable, or NP-complete.

In addition, you should have sufficient domain knowledge of Boolean formulas and graphs to be able to construct simple reductions from the standard NP-complete languages like *SAT*, *3SAT*, *HAMPATH*, *VERTEX-COVER*, *CLIQUE*, etc.

Specific Things That Will Definitely Appear on the Exam

- You will be required to do at least one of the following:
  - Prove that a given language is decidable.
  - Prove that a given language is Turing-recognizable.
  - Prove that a given language is in P.
  - Prove that a given language is in NP.
- You will be required to do at least one of the following:
  - Prove that a given language is undecidable.
  - Prove that a given language is non-Turing-recognizable.
- You will be required to:
  - Prove that a given language is NP-complete.

Additional Insights into What to Expect on the Exam

Where a reduction is required:

- You may be given
  - the language to reduce it from; and/or
  - a description of the actual function that performs the reduction; or
  - some other hint to guide you in determining the appropriate reduction.
- The reduction will be fairly simple and it should be straightforward to prove it has the correct properties (assuming you understand what these properties are).

"Design Recipes" for Proofs of Decidability or Turing-Recognizability on the Exam

To prove that a given language is decidable:

- Construct an algorithm that decides the language.
- This algorithm may "call" any other algorithms from the textbook, lectures, class handouts, or homework assignments (but you should cite the appropriate reference).
- How do you decide which existing algorithms it should call, if any? This is a familiar problem for any program designer. You may be expected to figure this out for yourself or a hint may be provided to allow you to proceed under the time constraints of the exam. When proving closure of the class of decidable languages under a given operation the obvious choice is an assumed decider for a given decidable language.
- Prove that the language it recognizes is equal to the given language and that the algorithm halts on all inputs.

To prove that a given language is Turing-recognizable:

- Construct an algorithm that accepts exactly those strings that are in the language. It must either reject or loop on any string not in the language.
- This algorithm may "call" any other algorithms from the textbook, lectures, class handouts, or homework assignments (but you should cite the appropriate reference).
- How do you decide which existing algorithms it should call, if any? This is a familiar problem for any program designer. You may be expected to figure this out for yourself or a hint may be provided to allow you to proceed under the time constraints of the exam. When proving closure of the class of Turing-recognizable languages under a given operation the obvious choice is an assumed recognizer for a given Turing-recognizable language.
- Prove that the language it recognizes is equal to the given language.

"Design Recipes" for Proofs of Undecidability or Non-Turing-Recognizability on the Exam

To prove that a given language is undecidable:

- Construct a (mapping) reduction from another language already known to be undecidable to the given language.
- This known undecidable language can be any language for which undecidability has been proved in the textbook, in lectures, in class handouts, or in homework problems (but you should cite the appropriate reference).
- How do you decide which existing language to use to reduce to the given language? You may be given a hint (or told outright), but if not, it's because there's a fairly obvious choice (like A_{TM}).
- Prove that your reduction has the desired properties (i.e., that it truly is a reduction from that undecidable language to the given language).

To prove that a given language is non-Turing-recognizable:

Either do both of these:

- Prove that its complement is Turing-recognizable.
- Prove that its complement is undecidable.

Or:

- Construct a (mapping) reduction from another language already known to be non-Turing-recognizable to the given language.
- This known non-Turing-recognizable language can be any language for which non-Turing-recognizability has been proved in the textbook, in lectures, in class handouts, or in homework problems (but you should cite the appropriate reference).
- How do you decide which existing language to use to reduce to the given language? You may be given a hint (or told outright), but if not, it's because there's a fairly obvious choice (like $\overline{A_{\text{TM}}}$).
- Prove that your reduction has the desired properties (i.e., that it truly is a reduction from that non-Turing-recognizable language to the given language).

"Design Recipes" for Proofs of Membership in P or NP on the Exam

To prove that a given language is in P:

- Construct an algorithm that decides the language.
- This algorithm may "call" any other algorithms from the textbook, lectures, class handouts, or homework assignments (but you should cite the appropriate reference).
- How do you decide which existing algorithms it should call, if any? This is a familiar problem for any program designer. You may be expected to figure this out for yourself or a hint may be provided to allow you to proceed under the time constraints of the exam. When proving closure of P under a given operation the obvious choice is an assumed polytime decider for a given member of P.
- Prove that the language it recognizes is equal to the given language and that the algorithm runs in polynomial time.

To prove that a given language is in NP:

Either:

- Construct (a high-level description of) an NTM (equivalently, a nondeterministic algorithm) that decides the language.
- This nondeterministic algorithm may "call" any other algorithms from the textbook, lectures, class handouts, or homework assignments (but you should cite the appropriate reference).
- How do you decide which existing algorithms it should call, if any? This is a familiar problem for any program designer. You may be expected to figure this out for yourself or a hint may be provided to allow you to proceed under the time constraints of the exam. When proving closure of NP under a given operation the obvious choice is an assumed nondeterministic polytime decider for a given member of NP.
- Prove that the language it recognizes is equal to the given language and that the algorithm runs in nondeterministic polynomial time.

Or:

- Specify a certificate that can be used with a verifier to decide the language.
- Give a verifier that uses that certificate to verify membership in the given language.
- Prove that the language recognized by the verifier is the given language and that the verifier runs in polynomial time.

In all the cases we've studied, the certificate/verifier approach is so obvious that it generally takes at most one sentence to describe the certificate and at most another sentence to describe how the verifier works. This will almost certainly be the case for any such problems on the exam.

"Design Recipe" for Proofs of NP-Completeness on the Exam

To prove that a given language is NP-complete:

- Prove that the language is in NP.
- Construct a (mapping) reduction from another language already known to be NP-complete to the given language.
- This known NP-complete language can be any language for which NP-completeness has been proved in the textbook, in lectures, in class handouts, or in homework problems (but you should cite the appropriate reference).
- How do you decide which existing language to use to reduce to the given language? You may be given a hint (or told outright), but if not, it's because there's a fairly obvious choice, most likely based on similarity to the given language.
- Prove that your reduction has the desired properties:
  - that it truly is a reduction from that NP-complete language to the given language; and
  - that it runs in polynomial time.