COMP0017
Computability and Complexity Theory

Fabio Zanasi
http://www0.cs.ucl.ac.uk/staff/F.Zanasi/

Lecture ten and eleven

Previously on COMP0017

We proved that:

1. the Halting Problem (*HALT*) is undecidable.
2. the Empty Tape Halting Problem (*ETH*) is undecidable.
3. the complement of the Halting Problem ($HALT^{-}$) is unrecognisable.

2 and 3 were proven *as a consequence* of 1.

This lecture

We introduce a simpler, more principled proof method for proving undecidability/unrecognisability of problems: **mapping-reduction**.

Towards mapping-reduction

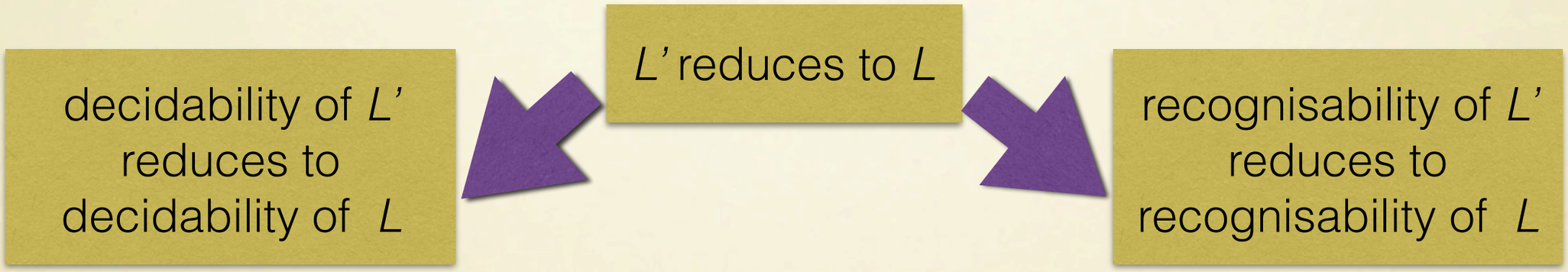
In the mapping-reduction proof method the main insight is understanding how to **reduce** the instances of one problem to the instances of another problem.

For instance, the core of the proof that *ETH* was undecidable was how to turn a *HALT*-question into an *ETH*-question.

In a sense, this is all that matters. **Decidability/ recognisability of the problems involved is not essential to the process**, it is just our ultimate goal.

Towards mapping-reduction

Schematically, we want to define reduction more abstractly, in such a way that:



So we take a more structured approach, were decidability/recognisability issues are shelved and we only focus on the relationship between problems.

This brings to a relation between problems called **mapping-reducibility**.

Mapping-reduction

Definition Given languages *L* and *L'* over an alphabet Σ, we say that *L'* mapping-reduces to *L* and write $L' \leq L$ if there is a TM that computes a (total) function $f : \Sigma^* \rightarrow \Sigma^*$ such that

$$x \in L' \leftrightarrow f(x) \in L$$

In essence, f converts the membership problem for *L'* to the membership problem for *L*.

Intuitive reading of $L' \leq L$: *L* is at least as difficult as *L'*.

Reduction and decidability

Mapping-reducibility does not talk about decidability. However, it is a powerful tool in order to show that a language is (un)decidable.

Theorem 1 If $L' \leq L$ and L is decidable, then L' is decidable.

Exercise

Corollary 1 If $L' \leq L$ and L' is undecidable, then L is undecidable.

Thus, in order to prove L undecidable, we just need to show $HALT \leq L$.

Corollary 2 If L is decidable and L' is not, then $L' \not\leq L$.

Mapping-reduction in action

Halting problem on the empty tape

The empty tape halting (ETH) problem is defined by the following language.

$$ETH = \{ x \in \Sigma^* \mid x = \text{code}(\mathcal{M}) \text{ and } \mathcal{M} \text{ halts on } \varepsilon. \}$$

Theorem *ETH* is undecidable.

Proof outline It is enough to reduce *HALT* to *ETH*, because then the undecidability of *HALT* implies the one of *ETH* by Corollary 1.

Halting problem on the empty tape

Proof We need to construct a computable f such that

f computable by a TM.

$$\langle y,x\rangle \in \mathit{HALT} \leftrightarrow f(\langle y,x\rangle) \in \mathit{ETH}$$

The definition of f is as follows. On argument $\langle y,x\rangle$:

- If $y \neq \text{code}(\mathcal{M})$ for all $\mathcal{M}$, then $f(\langle y,x\rangle) = y \notin \mathit{ETH}$.
- If $y = \text{code}(\mathcal{M})$, then $f(\langle y,x\rangle) = \text{code}(\mathcal{M}_{\mathcal{M},x})$, where $\mathcal{M}_{\mathcal{M},x}$ is constructed as follows:
  1. $\mathcal{M}_{\mathcal{M},x}$ enters a loop on any non-empty string.
  2. On input ε, write x on tape and simulate $\mathcal{M}$ on x.

$\langle y, x\rangle \in \mathit{HALT}$ $\Leftrightarrow$ $y = \text{code}(\mathcal{M})$ and $\mathcal{M}$ halts on x. $\Leftrightarrow$ $\mathcal{M}_{\mathcal{M},x}$ halts on ε. $\Leftrightarrow$ $f(\langle y,x\rangle) = \text{code}(\mathcal{M}_{\mathcal{M},x})$ and $f(\langle y,x\rangle) \in \mathit{ETH}$

The ``full language'' problem

The ``full language'' problem (FL) is defined by the following language.

$FL = \{ x \in \Sigma^* \mid x = \text{code}(\mathcal{M}) \text{ and } \mathcal{M} \text{ halts on all inputs.}\}$

Theorem *FL* is undecidable.

Proof outline We reduce *HALT* to *FL*.

The ``full language'' problem

Proof We need to construct a computable f such that

$$\langle y,x\rangle \in \mathit{HALT} \leftrightarrow f(\langle y,x\rangle) \in \mathit{FL}$$

The definition of f is as follows. On argument ⟨y,x⟩:

- If y ≠ code($\mathcal{M}$) for all $\mathcal{M}$, then f(⟨y,x⟩) = y ∉ *FL*.
- If y = code($\mathcal{M}$), then f(⟨y,x⟩) = code($\mathcal{M}_{\mathcal{M},x}$), where $\mathcal{M}_{\mathcal{M},x}$ is constructed as follows:
  1. $\mathcal{M}_{\mathcal{M},x}$ erases its input.
  2. Write x on tape and simulate $\mathcal{M}$ on *x*.

⟨y, *x*⟩ ∈ *HALT* ⇔ y = code($\mathcal{M}$) and $\mathcal{M}$ halts on x. ⇔ $\mathcal{M}_{\mathcal{M},x}$ halts on an arbitrary input. ⇔ f(⟨y,x⟩) = code($\mathcal{M}_{\mathcal{M},x}$) and f(⟨y,x⟩) ∈ *FL*

The equivalence problem

The equivalence problem (EQ) for Turing machines is defined by the following language.

$EQ = \{ \langle y, x \rangle \in \Sigma^* \times \Sigma^* \mid x = \text{code}(\mathcal{M}), y = \text{code}(\mathcal{M}'), \text{ and } \mathcal{M} \text{ and } \mathcal{M}' \text{ compute the same partial function.} \}$

Theorem *EQ* is undecidable.

Proof outline We reduce *FL* to *EQ*.

Disclaimer: we could have used reduction from *HALT* rather than from *FL*. We just varied to show the flexibility of the approach.

The equivalence problem

Proof We need to construct a computable f such that

$$z \in FL \leftrightarrow f(z) \in EQ$$

The definition of f is as follows. On argument z:

- If $z \neq \text{code}(\mathcal{M})$ for all $\mathcal{M}$, then $f(z) = \langle z,z \rangle \notin EQ$.
- If $z = \text{code}(\mathcal{M})$, then $f(z) = \langle \text{code}(\mathcal{M}_1), \text{code}(\mathcal{M}_2) \rangle$, where $\mathcal{M}_1$ and $\mathcal{M}_2$ are defined as follows.
    - $\mathcal{M}_1$ runs $\mathcal{M}$ on its input and outputs 1 if $\mathcal{M}$ halts, and loops otherwise.
    - $\mathcal{M}_2$ outputs 1 for all the inputs.

The equivalence problem

Reminder

$\mathcal{M}_1$ runs $\mathcal{M}$ on its input and outputs 1 if $\mathcal{M}$ halts, and loops otherwise.

$\mathcal{M}_2$ outputs 1 for all the inputs.

z ∈ *FL*

⇕

z = code($\mathcal{M}$) and $\mathcal{M}$ halts on all the inputs.

⇕

$\mathcal{M}_1$ halts on all the inputs with output 1.

⇕

$\mathcal{M}_1$ and $\mathcal{M}_2$ are equivalent.

⇕

f(z) = ⟨code($\mathcal{M}_1$),code($\mathcal{M}_2$)⟩ and f(z) ∈ *EQ*

Reduction and recognisability

Reduction and recognisability

What it is true for reduction and decidability is also true for reduction and recognisability.

Theorem If $L' \leq L$ and L is recognisable, then L' is recognisable.

Corollary If $L' \leq L$ and L' is not recognisable, then L is not recognisable.

Corollary If L is recognisable and L' is not, then $L' \not\leq L$.

The equivalence problem is unrecognisable

Recall the equivalence problem (*EQ*) for TMs:

$EQ = \{ \langle y, x \rangle \in \Sigma^* \times \Sigma^* \mid x = \text{code}(\mathcal{M}), y = \text{code}(\mathcal{M}'),$ and $\mathcal{M}$ and $\mathcal{M}'$ compute the same partial function.$\}$

We proved *EQ* to be undecidable. Now we show that *EQ* is even ``harder'': it is not recognisable.

Theorem *EQ* is not recognisable.

Proof outline We reduce $\overline{HALT}$ (which we know is a non-recognisable problem) to *EQ*.

The equivalence problem is unrecognisable

Proof Recall the definition of $HALT^-$:

$$HALT^- = \{ \langle y, x \rangle \in \Sigma^* \times \Sigma^* \mid y \neq \text{code}(\mathcal{M}) \text{ for any } \mathcal{M} \text{ or } y = \text{code}(\mathcal{M}) \text{ and } \mathcal{M} \text{ does not halt on } x. \}$$

For the reduction, we have to come up with a function f

$$\langle y,x \rangle \in HALT^- \iff f(\langle y,x \rangle) \in EQ$$

which is the same as saying

$$\langle y,x \rangle \in HALT \iff f(\langle y,x \rangle) \in EQ^-$$

so we seek the latter equivalence.

The equivalence problem is unrecognisable

Proof On ⟨y,x⟩ define f as follows:

- If y ≠ code($\mathcal{M}$) for all $\mathcal{M}$, then we pick any $\mathcal{M}'$ and let f(⟨y,x⟩) = ⟨code($\mathcal{M}'$), code($\mathcal{M}'$)⟩. So f(⟨y,x⟩) ∈ *EQ* and thus f(⟨y,x⟩) ∉ EQ^-.

- Otherwise y = code($\mathcal{M}$) and we let f(⟨y,x⟩) = ⟨code($\mathcal{M}_1$),code($\mathcal{M}_2$)⟩, where $\mathcal{M}_1$ and $\mathcal{M}_2$ are defined as
  - $\mathcal{M}_1$ runs $\mathcal{M}$ on x and halts if $\mathcal{M}$ halts, and loops otherwise.
  - $\mathcal{M}_2$ loops on any input.

The equivalence problem is unrecognisable

Reminder

$\mathcal{M}_1$ on all the inputs runs $\mathcal{M}$ on x and halts if $\mathcal{M}$ halts, and loops otherwise.

$\mathcal{M}_2$ loops on all the inputs.

⟨y,x⟩ ∈ *HALT*

y = code($\mathcal{M}$) and $\mathcal{M}$ halts on x.

$\mathcal{M}_1$ accepts some string.

$\mathcal{M}_1$ and $\mathcal{M}_2$ are not equivalent.

f(⟨y,x⟩) = ⟨code($\mathcal{M}_1$),code($\mathcal{M}_2$)⟩ and f(⟨y,x⟩) ∈ EQ^-

In fact, the language of strings accepted by $\mathcal{M}_1$ is either Σ^* or ∅, depending on whether $\mathcal{M}$ halts on x.

Further properties of mapping-reduction

Reduction and decidability

We saw that knowing L decidable or L' undecidable is useful.

Theorem 1 If $L' \leq L$ and L is decidable, then L' is decidable.

Corollary 1 If $L' \leq L$ and L' is undecidable, then L is undecidable.

Corollary 2 If L is decidable and L' is not, then $L' \not\leq L$.

However, if L' is decidable, reduction to some L is of little use.

Theorem 2 If L is any non-trivial language (so $L \neq \Sigma^*$ and $L \neq \varnothing$) then $L' \leq L$ for any L' decidable.

Reduction and decidability

Theorem 2 If L is any non-trivial language (so $L \neq \Sigma^*$ and $L \neq \varnothing$) then $L' \leq L$ for any L' decidable.

Proof

As L is non-trivial we can pick $x \in L$ and $y \notin L$. Define the function $f : \Sigma^* \rightarrow \Sigma^*$ as follows:

$f(z) = x$ if $z \in L'$

$f(z) = y$ otherwise, i.e. if $z \notin L'$

As L' is decidable, f can be implemented by a TM which internally simulates the TM for L' and outputs x or y accordingly. Moreover, $z \in L' \leftrightarrow f(z) \in L$ by definition of f.

Reduction as a relation

Reduction is a relation between languages.

It is <u>reflexive</u>: for all L, $L \leq L$.

It is <u>transitive</u>: $L' \leq L$ and $L \leq L''$ implies $L' \leq L''$.

However, it is not <u>symmetric</u>. Symmetry means that $L' \leq L$ implies $L \leq L'$.

The halting problem is the counterexample to symmetry. By Theorem 2, for any decidable L', $L' \leq HALT$. If also $HALT \leq L'$, then *HALT* would be decidable by Theorem 1, contradiction.

Therefore, reduction is not an equivalence relation.

Reduction and complement

Theorem $L_1 \leq L_2$ if and only if $L_1^- \leq L_2^-$

Exercise

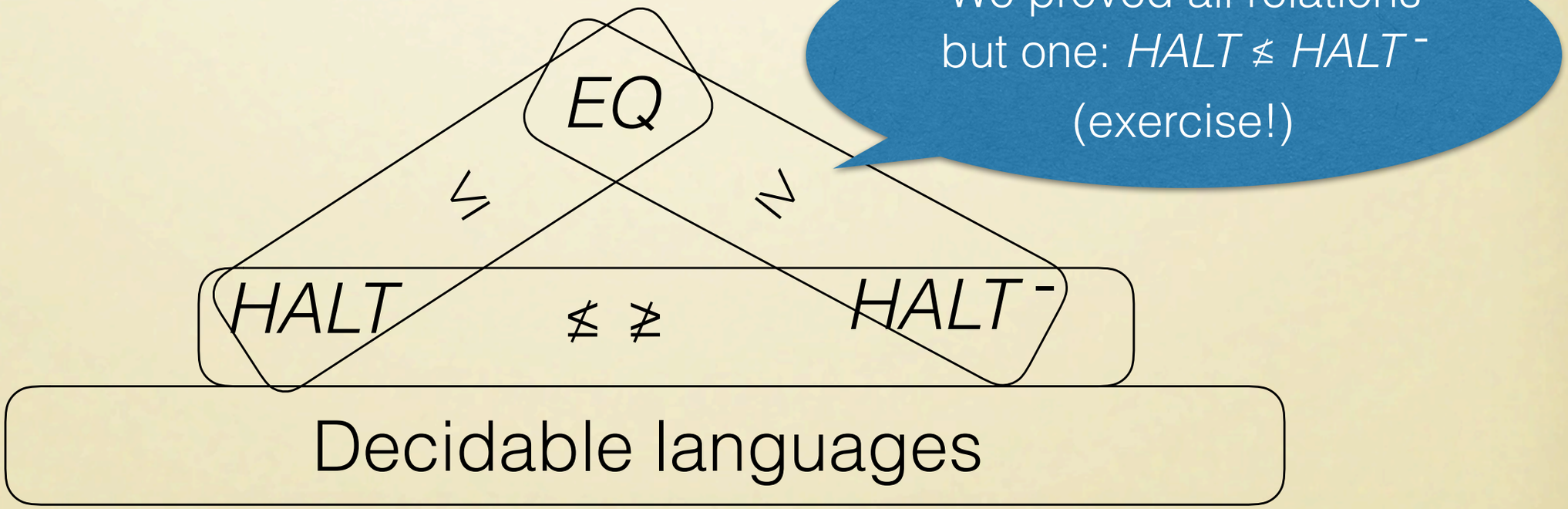
Corollary $L_1^- \leq L_2$ if and only if $L_1 \leq L_2^-$

Implicitly used in the last proof with $L_1 = HALT$ and $L_2 = EQ$.

Putting all together, we have ``for free'' a proof that EQ^- is not recognisable.
Indeed, $HALT \leq FL \leq EQ$, thus $HALT^- \leq EQ^-$ by the theorem. In this situation, non-recognisability *of* $HALT^-$ implies non-recognisability of EQ^-.

A hierarchy of problems

…	Decidable
HALT	Undecidable, but recognisable
$HALT^-$	Not recognisable, but complement recognisable
EQ	Not recognisable, and complement not recognisable



We proved all relations but one: $HALT \not\leq HALT^-$ (exercise!)

Other notions or reducibility

Turing-reducibility

Mapping-reducibility is just one way of defining the concept of a problem being reducible to another one.

Another one is what is usually called **Turing-reducibility.**

An **oracle** for a language *L* is an external device (a ``black box'') that is capable of answering the question ``x ∈ *L* ?'' for all strings x.

A language *L'* is **Turing-reducible** to *L* if we can decide *L'* given an oracle for *L*.

Comparing notions of reducibility

Mapping-reducibility implies Turing-reducibility but not vice-versa.

Example: we know $HALT^- \not\leq HALT$, i.e., $HALT^-$ is not mapping reducible to *HALT.*
But $HALT^-$ is Turing-reducible to *HALT.* Indeed, if we have an Oracle for the question ``$\langle y, x \rangle \in HALT$?", then this obviously can be used to decide whether $\langle y, x \rangle \in HALT^-$.