Mapping Reducability

I’m sorry Dave, I’m afraid I can’t do that.
(Hal in 2001: A Space Odyssey)

We formalize the idea of reducibility by giving a definition of mapping reducibility, often called many-one reducibility.

A function \( f : \Sigma^* \rightarrow \Sigma^* \) is a computable function if some Turing machine \( M \) with any input \( w \) halts with \( f(w) \) on its tape.

Examples of computable functions: arithmetic operations, constructing a DFA from an NFA, constructing a CFG for the union of two CFGs.

A language \( A \) is mapping reducible to language \( B \), written \( A \leq_m B \), if there is a computable function \( f \) such that

\[
    w \in A \leftrightarrow f(w) \in B
\]

\( f \) is called the reduction from \( A \) to \( B \).

\[\begin{array}{c}
    w \\
    \rightarrow \\
    f \\
    f(w) \\
    \rightarrow \\
    A \quad f \quad f(w) \quad B \\
\end{array}\]

\( w \) \rightarrow accept
\( a \)
\( r \) \rightarrow reject

Definition

A language \( A \) is mapping reducible to language \( B \), written \( A \leq_m B \), if there is a computable function \( f \) such that

\[
    w \in A \leftrightarrow f(w) \in B
\]

\( f \) is called the reduction from \( A \) to \( B \).
Examples

\( A = \) set of even numbers
\( B = \) set of odd numbers
\( f(x) = x + 1 \)
\( x \) is an even number iff \( f(x) \) is an odd number.

\( A = \) TMs that always halt
\( B = \) TMs that accept all inputs
\( f(M) = \) replace each \( q_{\text{reject}} \) with \( q_{\text{accept}} \)
\( M \) always halts iff \( f(M) \) accepts all inputs.

Characteristics of Mapping Reducibility

If \( A \leq_m B \) and \( B \) is decidable, then \( A \) is decidable.
Proof Sketch: Let \( M \) be a decider for \( B \).
Let \( f \) be a reduction from \( A \) to \( B \).
\( M(f(w)) \) is a decider for \( A \).

It follows that:
If \( A \leq_m B \) and \( A \) is undecidable, then \( B \) is undecidable.

Characteristic, Part 2

If \( A \leq_m B \) and \( B \) is Turing-recognizable, then \( A \) is Turing-recognizable.
Proof Sketch: Let \( M \) be a recognizer for \( B \).
Let \( f \) be a reduction from \( A \) to \( B \).
\( M(f(w)) \) is a recognizer for \( A \).

It follows that:
If \( A \leq_m B \) and \( A \) is not Turing-recognizable, then \( B \) is not Turing-recognizable.

Example I: \( A_{\text{TM}} \leq_m \text{HALT}_{\text{TM}} \)

Need a computable function \( f \) such that:
\( \langle M, w \rangle \in A_{\text{TM}} \iff f(\langle M, w \rangle) \in \text{HALT}_{\text{TM}} \)

This algorithm computes the reduction:
1. Construct a TM \( M' \) such that
\( M'(x) = \begin{cases} \text{accept} & \text{if } M(x) = \text{accept} \\ \text{loop forever} & \text{otherwise} \end{cases} \)
2. Output \( \langle M', w \rangle \)

This is a reduction because
\( M \) accepts \( w \) iff \( M' \) halts on \( w \).
Example II: $EQ_{DFA} \leq_m E_{DFA}$

Need a computable function $f$ such that:

$\langle B, C \rangle \in EQ_{DFA} \iff f(\langle B, C \rangle) \in E_{DFA}$

Define $f$ to output a DFA $D$ such that:

$L(D) = (L(B) \cap L(C)) \cup (L(B) \cap L(C))$

The closure properties of DFAs ensures that we can do this.

This is a reduction because $L(B) = L(C)$ iff $L(D) = \emptyset$. 