LIMITS OF COMPUTABILITY

Decidable problems

- Recall: An algorithmic decision problem is ***decidable***
  if it has a decision algorithm.
- That is, a language $L \subseteq \Sigma^*$ is ***(Turing-) decidable***
  if it is recognized by some ***Turing-decider,*** that is
  a Turing acceptor that ***terminates for every input.***
- A decision problem is *Turing-decidable if*
  its textual representation is.
- *Given the Turing-Church Thesis we*
  identify informal algorithms with Turing machines!

Decidability preserved under set operations

- Let $\mathcal{P}$ and $\mathcal{Q}$ be problems referring to the same instances, decided by algorithms A_P and A_Q respectively.
- The ***complement*** of $\mathcal{P}$ is decidable: to decide $w \in \bar{\mathcal{P}}$ run A_P on input w and flip the answer.

Decidability preserved under set operations

- Let $\mathcal{P}$ and $\mathcal{Q}$ be problems referring to the same instances, decided by algorithms A_P and A_Q respectively.
- The ***complement*** of $\mathcal{P}$ is decidable: to decide $w \in \bar{\mathcal{P}}$ run A_P on input w and flip the answer.
- The ***intersection*** of $\mathcal{P}$ and $\mathcal{Q}$ is decidable: to decide $w \in \mathcal{P} \cap \mathcal{Q}$
  - run A_P on w, if it rejects, reject; if it accepts
  - run A_Q on w, if it rejects, reject; if it accepts accept.
- The ***union*** of $\mathcal{P}$ and $\mathcal{Q}$ is decidable:
  - run A_P on w, if it accepts, accept; if it rejects,
  - run A_Q on w, if it accepts, accept; if it rejects reject.

UNDECIDABILITY

An undecidable problem

- The problem “Self non-accept” (**SNA**):

  Instances: Turing-acceptors M

  Property: M does not accept $M^{\#}$.

An undecidable problem

- The problem “Self non-accept” (**SNA**):

  Instances: Turing-acceptors M

  Property: M does not accept $M^{\#}$.

- We show that **SNA** is not recognized, let alone decidable.

An undecidable problem

- The problem “Self non-accept” (**SNA**):

  Instances: Turing-acceptors M

  Property: M does not accept $M^{\#}$.

- We show that **SNA** is not recognized, let alone decidable.

- Suppose we had an acceptor D recognizing **SNA**, that is:

  D accepts $M^{\#}$ ***iff*** M does not accept $M^{\#}$

An undecidable problem

- The problem “Self non-accept” (**SNA**):

  Instances: Turing-acceptors M

  Property: M does not accept $M^{\#}$.

- We show that **SNA** is not recognized, let alone decidable.

- Suppose we had an acceptor D recognizing **SNA**, that is:

  D accepts $M^{\#}$ ***iff*** M does not accept $M^{\#}$

- Taking for M the particular acceptor D:

  D accepts $D^{\#}$ ***iff*** D does not accept $D^{\#}$

An undecidable problem

- The problem "Self non-accept" (**SNA**):

  Instances: Turing-acceptors M

  Property: M does not accept $M^{\#}$.

- We show that **SNA** is not recognized, let alone decidable.

- Suppose we had an acceptor D recognizing **SNA**, that is:

  D accepts $M^{\#}$ ***iff*** M does not accept $M^{\#}$

- Taking for M the particular acceptor D:

  D accepts $D^{\#}$ ***iff*** D does not accept $D^{\#}$

- **Contradiction!**

  There can be no acceptor D for **SNA** !

Analogy with Russell's Paradox

- Recall Russell's Paradox:

  Define $\mathcal{R} =_{\text{df}} \{x \mid x \text{ a set}, x \notin x\}$

  That is: for any set z $\quad z \in \mathcal{R}$ ***iff*** $z \notin z$.

- In particular taking $\mathcal{R}$ for z: $\quad \mathcal{R} \in \mathcal{R}$ ***iff*** $\mathcal{R} \notin \mathcal{R}$

- $\mathcal{R}$ is a collection of sets, which cannot be admitted as a "set."

  Core of the problem:

  Objects x are both objects and sets.

- **SNA** is a set of acceptors, which cannot be recognized by an acceptor.

  Core of the problem:

  An acceptor M is both a string $M^{\#}$ and a language $\mathcal{L}(M)$.

ACCEPT is undecidable

- **SNA** is a contrived decision problem,

  designed to bootstrap our exploration of undecidability.

- **ACCEPT** is a natural and important problem:

  Instances: Pairs (M, w) , M an acceptor, w a string.

  Property: M accepts w.

- **Theorem:** **ACCEPT** is undecidable.

- **Proof:** If **ACCEPT** were decidable,

  then so would be its complement **NON-ACCEPT**.

- But then **NSA** would also be decidable:

  To determine whether M accepts $M^{\#}$

  just ask whether **NON-ACCEPT** says "yes" for input $(M, M^{\#})$.

SEMI-DECIDABLE PROBLEMS

Semi-decidable problems

- **ACCEPT** is undecidable,

  but it is ***recognized*** by an acceptor: the ***universal interpreter!***

- That's more than we can say about **NSA**,

  which is not recognized even allowing acceptors that are not deciders.

Semi-decidable problems

- **ACCEPT** is undecidable,

  but it is ***recognized*** by an acceptor: the ***universal interpreter!***

- That's more than we can say about **NSA**,

  which is not recognized even allowing acceptors that are not deciders.

- A problem is ***semi-decidable (SD)*** (i.e. half-decidable)
  if it is recognized (as a language) by a Turing-acceptor.
  "Semi-decidable" and "recognized" are synonymous!

Semi-decidable problems

- **ACCEPT** is undecidable,

  but it is ***recognized*** by an acceptor: the ***universal interpreter!***

- That's more than we can say about **NSA**,

  which is not recognized even allowing acceptors that are not deciders.

- A problem is ***semi-decidable (SD)*** (i.e. half-decidable)
  if it is recognized (as a language) by a Turing-acceptor.
  "Semi-decidable" and "recognized" are synonymous!

- Every decidable problem is SD, of course.

- But not conversely: **ACCEPT** is SD but not decidable.

- And **NSA** is not even SD.

Disregarding rejection

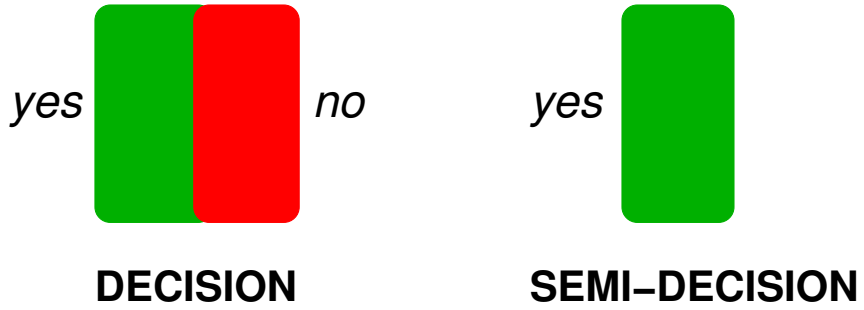
- A ***decision*** algorithm for problem $\mathcal{P}$

  identifies correctly both ***yes*** and ***no*** instances.

- A ***recognition*** (semi-decision) algorithm for $\mathcal{P}$

  identifies correctly the ***yes*** instances,

  but might loop for some (or all) ***no*** instances.



Two levels of "computable" problems

- Since some problems are undecidable but SD (eg. **ACCEPT**)

  we have two levels of problems that are "computationally solvable".

- It make sense to make sure we understand better SD (the "weak" level).

- So we'll consider two characterizations of SD,

  adding to our original definition SD=recognized.

Two levels of "computable" problems

- Since some problems are undecidable but SD (eg. **ACCEPT**)
  we have two levels of problems that are "computationally solvable".
- It make sense to make sure we understand better SD (the "weak" level).
- So we'll consider two characterizations of SD,
  adding to our original definition SD=recognized.
  - The first characterization refers to ***certification***,
    and suggests that a problem is SD iff it becomes decidable
    when we are given a "hint" for each instance.

Two levels of "computable" problems

- Since some problems are undecidable but SD (eg. **ACCEPT**)
  we have two levels of problems that are "computationally solvable".
- It make sense to make sure we understand better SD (the "weak" level).
- So we'll consider two characterizations of SD,
  adding to our original definition SD=recognized.
  - The first characterization refers to ***certification***,
    and suggests that a problem is SD iff it becomes decidable
    when we are given a "hint" for each instance.
  - The other characterization is based on ***enumeration***,
    and states that a problem is SD
    just in case it is generated by a computable process.

CHARACTERIZATIONS OF SD

Certificates

- Many decision problems are of the form

  Given an instance X *is there an object* c *such that ... ?*

Certificates

- Many decision problems are of the form

  Given an instance X is there an object c such that ... ?

  Examples:

  1. Given a graph X,

     is there a cycle c visiting each edge once?

Certificates

- Many decision problems are of the form

  Given an instance X is there an object c such that ... ?

  Examples:

  1. Given a graph X,

     is there a cycle c visiting each edge once?

  2. Given a natural number X, does it have a divisor $c > 1$.

Certificates

- Many decision problems are of the form

  Given an instance X is there an object c such that ... ?

  Examples:

  1. Given a graph X,

     is there a cycle c visiting each edge once?

  2. Given a natural number X, does it have a divisor $c > 1$.

- We say that c is a ***certificate*** for $X \in \mathcal{P}$.

Certificates

- Many decision problems are of the form

  Given an instance X is there an object c such that ... ?

  Examples:

  1. Given a graph X,

     is there a cycle c visiting each edge once?

  2. Given a natural number X, does it have a divisor $c > 1$.

- We say that c is a ***certificate*** for $X \in \mathcal{P}$.

- If the c is provided somehow,

  it only remains to check that it actually works:

  an appropriate cycle for (a),

  a divisor for (b).

Certification defined

- Let $\mathcal{P}$ be a decision-problem.

  A ***certification for $\mathcal{P}$*** is a mapping $\vdash$

  from finite discrete objects to instances of $\mathcal{P}$.

- $c \vdash X$ says that

  c is a ***certificate*** that X satisfies $\mathcal{P}$.

- That is, $X \in \mathcal{P}$ ***iff*** $c \vdash X$ for some c.

Examples of certifications

- **COMPOSITE**:

  A certification is the relation $\vdash$ where

  $c \vdash n$ iff $n > 2$ and c is a divisor of n.

Examples of certifications

- **INT-POLYNOMIALS**:

  A certification is the relation $\vdash$ where

  $c \vdash p[x_1 \ldots x_n]$ iff

  $p[x_1 \ldots x_k]$ is a polynomial with integer coefficients

  and variables among $x_1 \ldots x_k$

  and c is a list $z_1, \ldots, z_k$ of integers

  s.t. $p[z_1, ..., z_k] = 0$.

Examples of certifications

- The **INT-PARTITION** Problem asks whether a finite $S \subset \mathbb{N}$]
  has a subset P s.t. $\Sigma P = (\Sigma S)/2$.

  A certification is the relation $\vdash$ where
  $P \vdash S$ iff $S \subset \mathbb{N}$ is finite, $P \subset S$,
  and $\Sigma P = (\Sigma S)/2$.

Examples of certifications

- **VALIDITY**: Given a first-order formula φ, is it valid,

  i.e. true in all structures.

  A certification is the relation $\vdash$ where

  $\pi \vdash \varphi$ holds iff π is a first-order proof of φ.

  This is a certification of the Validity Problem because

  a formula φ is valid iff it has a proof.

Decidable certifications

- A certification $\vdash$ for a problem $\mathcal{P}$ is **decidable**

  if it is decidable as a set:

  There is an algorithm deciding, given c and instance X,

  whether $c \vdash X$.

Decidable certifications

- A certification $\vdash$ for a problem $\mathcal{P}$ is ***decidable***

  if it is decidable as a set:

  There is an algorithm deciding, given c and instance X,

  whether $c \vdash X$.

- Example: **ACCEPT** has the certification $\vdash$

  where $c \vdash (M, w)$ iff c is an accepting trace of M for input w.

Decidable certifications

- A certification $\vdash$ for a problem $\mathcal{P}$ is ***decidable***

  if it is decidable as a set:

  There is an algorithm deciding, given c and instance X,

  whether $c \vdash X$.

- Example: **ACCEPT** has the certification $\vdash$

  where $c \vdash (M, w)$ iff c is an accepting trace of M for input w.

- This certification is decidable:

  Given string c and instance $(M^{\#}, w)$ of **ACCEPT**

  it is tedious but easy to check that c is

  an accepting trace of M for input w.

Decidable certification = semi-decidable

- **Theorem.** L *is recognized by an acceptor*
  iff it has a decidable certification.

Decidable certification = semi-decidable

- **Theorem.** L *is recognized by an acceptor*
  iff it has a decidable certification.

  $\Longrightarrow$: Suppose $L = \mathcal{L}(M)$.

  Let $c \vdash w$ iff c is a trace of M that accepts w.

Decidable certification = semi-decidable

- **Theorem.** L *is recognized by an acceptor*
  iff it has a decidable certification.

  $\Longrightarrow$: Suppose $L = \mathcal{L}(M)$.

  Let $c \vdash w$ iff c is a trace of M that accepts w.

  - $\vdash$ is a certification for L, since M recognizes L.

Decidable certification = semi-decidable

- **Theorem.** L *is recognized by an acceptor*
  iff it has a decidable certification.

  $\Longrightarrow$: Suppose $L = \mathcal{L}(M)$.

  Let $c \vdash w$ iff c is a trace of M that accepts w.

  - $\vdash$ is a certification for L, since M recognizes L.
  - $\vdash$ is decidable:

    Check c's first cfg is M's initial cfg for input w.

    Check that successive transitions in c are correct for M.

    Check that c's last cfg is accepting for M.

Decidable certification = semi-decidable

- **Theorem.** L *is recognized by an acceptor*
  iff it has a decidable certification.

  $\Longleftarrow$:

  Suppose $\vdash$ is a decidable certification for L.

  Here is an algorithm that recognizes L:

Decidable certification = semi-decidable

- **Theorem.** L *is recognized by an acceptor iff it has a decidable certification.*

  $\Longleftarrow$:

  Suppose $\vdash$ is a decidable certification for L.

  Here is an algorithm that recognizes L:

  - Given $w \in L$ check successive strings c (in size+lexicographic order) whether $c \vdash w$.

Decidable certification = semi-decidable

- **Theorem.** L *is recognized by an acceptor*
  iff it has a decidable certification.

  $\Longleftarrow$:

  Suppose $\vdash$ is a decidable certification for L.

  Here is an algorithm that recognizes L:

  - Given $w \in L$ check successive strings c
    (in size+lexicographic order) whether $c \vdash w$.
  - Accept w if and when such a c is found.

Computably enumerated problems

- A problem $L \subseteq \Sigma^*$ is **computably-enumerated (CE)**
  if there is a computable function $f : \mathbb{N} \to \Sigma^*$ with image L

Computably enumerated problems

- A problem $L \subseteq \Sigma^*$ is **computably-enumerated (CE)**
  if there is a computable function $f : \mathbb{N} \to \Sigma^*$ with image L
- That is, $L = \{f(0), f(1), \ldots\}$ is a listing of L.
  We say that f ***enumerates*** L.

SD $\Longleftrightarrow$ *computably enumerated*

Theorem.

A non-empty language is ***SD***

iff it is ***computably enumerated****.*

SD $\Longrightarrow$ *computably enumerated*

- Given a non-empty SD language $L \subseteq \Sigma^*$,

  let $w_0 \in L$

  and let $\vdash$ be a decidable certification for L.

SD $\Longrightarrow$ computably enumerated

- Given a non-empty SD language $L \subseteq \Sigma^*$,

  let $w_0 \in L$

  and let $\vdash$ be a decidable certification for L.

- Consider a listing $(c_1, w_1),\ (c_2, w_2), \ldots$ of all pairs

  (c, w) where c is a potential-certificate and $w \in \Sigma^*$.

  (Say the listing is by size-lexicographic order.)

- Now define $f(n) = w_n$ if $c_n \vdash w_n$

  but $f(n) = w_0$ otherwise.

SD $\Longrightarrow$ computably enumerated

- Given a non-empty SD language $L \subseteq \Sigma^*$,

  let $w_0 \in L$

  and let $\vdash$ be a decidable certification for L.

- Consider a listing $(c_1, w_1),\ (c_2, w_2), \ldots$ of all pairs

  (c, w) where c is a potential-certificate and $w \in \Sigma^*$.

  (Say the listing is by size-lexicographic order.)

- Now define $f(n) = w_n$ if $c_n \vdash w_n$

  but $f(n) = w_0$ otherwise.

- Since $\vdash$ is decidable, f is computable.

- And since the enumeration above includes all pairs in $\vdash$,

  the image of f is L.

SD $\Longleftarrow$ computably enumerated

- Suppose L is enumerated by a computable $f : \mathbb{N} \to \Sigma^*$.
- $L = \mathcal{L}(M)$ where M is the acceptor that
  on input w calculates $f(0), f(1), f(2) \ldots$,
  and accepts w if and when it is obtained as output of f.

Orderly-enumerations

- A problem $L \subseteq \Sigma^*$ is ***orderly-enumerated***
  if it is computably-enumerated by some f which is
  - ***Injective***
  - ***Non-contracting***: $|f(n)| \leqslant |f(n+1)|$ for all $n \geqslant 0$.

Orderly-enumerations

- A problem $L \subseteq \Sigma^*$ is ***orderly-enumerated***
  if it is computably-enumerated by some f which is
  - ***Injective***
  - ***Non-contracting***: $|f(n)| \leqslant |f(n+1)|$ for all $n \geqslant 0$.
- That is, $L = \{f(0), f(1), \ldots\}$ is a listing of L
  without repetition and in non-contracting order.

Decidable $\Longleftrightarrow$ *orderly-enumerated*

- **Theorem.**

  An infinite language L *is* ***decidable*** *iff it is* ***orderly-enumerated***.

- I.e. a language is decidable iff it is finite or orderly-enumerated.

Decidable $\Longrightarrow$ *orderly enumerated*

Suppose L is recognized by a decider M.

Decidable $\Longrightarrow$ *orderly enumerated*

Suppose L is recognized by a decider M.

- Referring to size-lexicographic ordering:

  L is orderly-enumerated by

$$\begin{aligned} f(0) &= \text{first } w \text{ accepted by } M \\ f(n+1) &= \text{first } w \text{ after } f(n) \text{ accepted by } M \end{aligned}$$

Decidable $\Longrightarrow$ orderly enumerated

Suppose L is recognized by a decider M.

- Referring to size-lexicographic ordering:

  L is orderly-enumerated by

$$\begin{aligned} f(0) &= \text{first } w \text{ accepted by } M \\ f(n+1) &= \text{first } w \text{ after } f(n) \text{ accepted by } M \end{aligned}$$

- Since L is infinite, f is a total function.

Decidable $\Longrightarrow$ orderly enumerated

Suppose L is recognized by a decider M.

- Referring to size-lexicographic ordering:

  L is orderly-enumerated by

$$f(0) = \text{first } w \text{ accepted by } M$$
$$f(n+1) = \text{first } w \text{ after } f(n) \text{ accepted by } M$$

- Since L is infinite, f is a total function.
- f is a non-contracting injection by dfn,

  and is computable since M is a decider.

Decidable $\Longleftarrow$ orderly enumerated

- Suppose L is orderly-enumerated by $f : \mathbb{N} \to \Sigma^*$.
- Then $L = \mathcal{L}(M)$, where M implements the following algorithm:

  on input w compute $f(n)$ for successive n's,

  accept if w is reached, stop and reject if $|w|$ is exceeded.
- M is a decider because f is

  total, injective, and non-contracting.

Decidability in terms of semi-decidability

- We characterized SD in terms of decidability:

  L *is SD iff it has a decidable certification.*

- We now characterize decidability in terms of semi-decidability.

Decidability in terms of semi-decidability

- We characterized SD in terms of decidability:

  L *is SD iff it has a decidable certification.*

- We now characterize decidability in terms of semi-decidability.

- **Motivation:**

  A decision algorithm answers `yes/no` correctly.

  A semi-decision algorithm answers just the `yes` cases.

- Decidability of L is like having two semi-decision algorithms:

  one for L and the other for $\bar{L}$.

Decidability in terms of semi-decidability

- We characterized SD in terms of decidability:

  L *is SD iff it has a decidable certification.*

- We now characterize decidability in terms of semi-decidability.

- **Theorem.**

  A language $L \subseteq \Sigma^*$ *is decidable iff*

  both L *and its complement* $\bar{L} = \Sigma^* - L$ *are SD.*

Decidability in terms of semi-decidability

- We characterized SD in terms of decidability:

  L *is SD iff it has a decidable certification.*

- We now characterize decidability in terms of semi-decidability.

- **Theorem.**

  A language $L \subseteq \Sigma^*$ *is decidable iff*

  both L *and its complement* $\bar{L} = \Sigma^* - L$ *are SD.*

- A problem whose complement is SD is said to be ***co-SD.***

  So the Theorem states that

  a problem is decidable iff it is both SD and co-SD.

Decidable $\Longrightarrow$ SD and co-SD

If L is decidable, then so is its complement.

Every decidable language is trivially SD, so both L and $\bar{L}$ are SD.

Decidable $\Longleftarrow$ *SD and co-SD*

- Suppose that L and $\bar{L}$ are both SD.

Decidable $\Longleftarrow$ SD and co-SD

- Suppose that L and $\bar{L}$ are both SD.
- If one of them is empty, then they are both trivially decidable.

Decidable $\Longleftarrow$ *SD and co-SD*

- Suppose that L and $\bar{L}$ are both SD.
- If one of them is empty, then they are both trivially decidable.
- Suppose that neither is empty.

Decidable $\Longleftarrow$ SD and co-SD

- Suppose that L and $\bar{L}$ are both SD.
- If one of them is empty, then they are both trivially decidable.
- Suppose that neither is empty.
  - L is SD, so it is the image of a computable $f^+ : \mathbb{N} \to \Sigma^*$.
  - $\bar{L}$ is also SD, so it too is the image of a computable $f^- : \mathbb{N} \to \Sigma^*$.

Decidable $\Longleftarrow$ SD and co-SD

- Suppose that L and $\bar{L}$ are both SD.
- If one of them is empty, then they are both trivially decidable.
- Suppose that neither is empty.
  - L is SD, so it is the image of a computable $f^+ : \mathbb{N} \to \Sigma^*$.
  - $\bar{L}$ is also SD, so it too is the image of a computable $f^- : \mathbb{N} \to \Sigma^*$.
- To decide $w \in L$ calculate $f^+(0), f^-(0), f^+(1), f^-(1)...$
  until w is obtained as an output.
  If it is an output of f^+ then $w \in L$,
  if of f^- then $w \in \bar{L}$.

Decidable $\Longleftarrow$ SD and co-SD

- Suppose that L and $\bar{L}$ are both SD.
- If one of them is empty, then they are both trivially decidable.
- Suppose that neither is empty.
  - L is SD, so it is the image of a computable $f^+ : \mathbb{N} \to \Sigma^*$.
  - $\bar{L}$ is also SD, so it too is the image of a computable $f^- : \mathbb{N} \to \Sigma^*$.
- To decide $w \in L$ calculate $f^+(0), f^-(0), f^+(1), f^-(1)...$
  until w is obtained as an output.
  If it is an output of f^+ then $w \in L$,
  if of f^- then $w \in \bar{L}$.
- So L is decidable.

Summary of characterizations

Let $L \subseteq \Sigma^*$

- The following are equivalent:
  - (a) L is ***semi-decidable,*** i.e. recognized by an acceptor
  - (b) L is computably-enumerated
  - (c) L has a decidable certification
- The following are equivalent:
  - (a) L is ***decidable,*** i.e. recognized by a terminating acceptor
  - (b) L is orderly-enumerated
  - (c) L is both SD and co-SD
- (a) are characterizations in terms of machine acceptors,
  - (b) in terms of generators,
  - (c) decidability and decidability in terms of each other.

SD is closed under intersection

- Suppose $L, K \subseteq \Sigma^*$ are SD,

  recognized by acceptors A_L and A_k.

- We show that $L \cap K$ is SD.

  The proof is similar to that for closure of decidable languages:

- An acceptor A for $L \cap K$ simulates, on input w,

  A_L, and if and when A_L accepts w it simulates A_k.

- If either of the two processes does not terminate,

  then neither does A.

SD is closed under union

- The union $L \cup K$ is also SD.
- Here we cannot run A_L followed by A_K,

  because A_L may fail to terminate, whereas A_K accepts.

SD is closed under union

- The union $L \cup K$ is also SD.
- Here we cannot run A_L followed by A_K,
  because A_L may fail to terminate, whereas A_K accepts.
- L, K are SD, so they have decidable certifications $\vdash_L$ and $\vdash_K$.

SD is closed under union

- The union $L \cup K$ is also SD.
- Here we cannot run A_L followed by A_K,
  because A_L may fail to terminate, whereas A_K accepts.
- L, K are SD, so they have decidable certifications $\vdash_L$ and $\vdash_K$.
- Let $\vdash_{L \cup K}$ be $\vdash_L \cup \vdash_K$.

SD is closed under union

- The union $L \cup K$ is also SD.
- Here we cannot run A_L followed by A_K,
  because A_L may fail to terminate, whereas A_K accepts.
- L, K are SD, so they have decidable certifications $\vdash_L$ and $\vdash_K$.
- Let $\vdash_{L \cup K}$ be $\vdash_L \cup \vdash_K$.
- Then $\vdash_{L \cup K}$ is a decidable certification for $L \cup K$:

  $w \in L \cup K$ ***iff*** $w \in L$ or $w \in K$

  iff $c \vdash_L w$ or $c \vdash_K w$ for some c since $\vdash_L$ and $\vdash_K$ are certifi

  iff $c \vdash_{L \cup K} w$ for some c by the dfn of $\vdash_{L \cup K}$

SD is not closed under complement!

- We saw that **ACCEPT** is SD but not decidable.
- If L is any undecidable SD language,
  such as **ACCEPT**, then its complement is not SD,
  or else L would be both SD and co-SD,
  and therefore decidable.

REDUCTIONS BETWEEN PROBLEMS

Using other problems' solution

- We often fulfill tasks using tools for other tasks.
  - To match two decks of card, first sort them.
    Matching unsorted card-decks reduces to matching sorted decks.
  - To use biased coins when a fair coin is needed
    use a biased coin in double-rounds:
    take HT as "head," TH as "tail," discard HH and TT.
    Fair-coin is reduced to double-round biased-coin.
  - A calculator with squaring but no multiplication:
    Define multiplication:

$$x \cdot y = (x+y)^2 - (x-y)^2 \quad /2 \; /2$$

Multiplying is reduced to squaring and halving.

Reduction between decision problems

- A ***reduction*** between decision problems means solving problem $\mathcal{P}$ by converting its instances into instances of a problem $\mathcal{Q}$.
- If that mapping is relatively easy, then a solution to $\mathcal{Q}$ yields a solution for $\mathcal{P}$.

Example: EULER-CYCLE reduces to EVEN-DEGREE

- A ***cycle*** in a multi-graph G

  is a list $v_0, \ldots, v_n = v_0$ of vertices

  where every two consecutive ones are adjacent in G.

Example: EULER-CYCLE reduces to EVEN-DEGREE

- A ***cycle*** in a multi-graph G

  is a list $v_0, \ldots, v_n = v_0$ of vertices

  where every two consecutive ones are adjacent in G.

- An ***Euler-cycle*** visits every ***edge*** exactly once.

Example: EULER-CYCLE reduces to EVEN-DEGREE

- A ***cycle*** in a multi-graph G

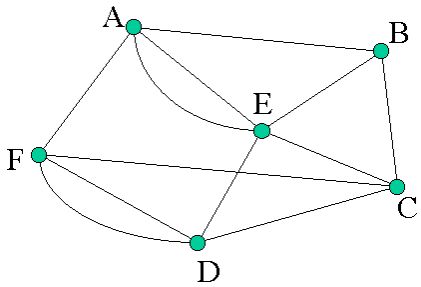
  is a list $v_0, \ldots, v_n = v_0$ of vertices

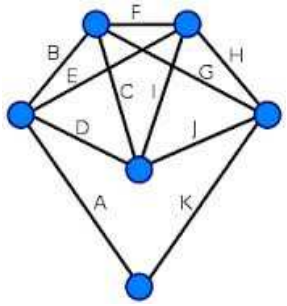
  where every two consecutive ones are adjacent in G.

- An ***Euler-cycle*** visits every ***edge*** exactly once.

EULER-CYCLE:

Given a graph G, does it have an Euler-cycle?





Example: EULER-CYCLE reduces to EVEN-DEGREE

- A ***cycle*** in a multi-graph G

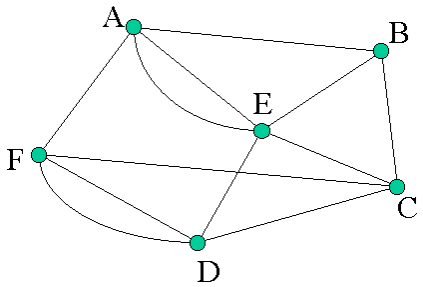
  is a list $v_0, \ldots, v_n = v_0$ of vertices

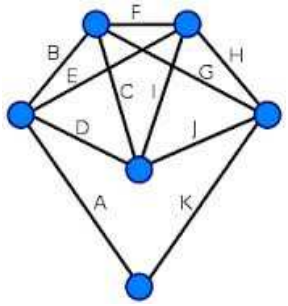
  where every two consecutive ones are adjacent in G.

- An ***Euler-cycle*** visits every ***edge*** exactly once.

  EULER-CYCLE:

  Given a graph $\mathcal{G}$, does it have an Euler-cycle?





- **Theorem:**

  $\mathcal{G}$ has an Euler-cycle iff every vertex has even degree.

Example: INTEGER-PARTITION and EXACT-SUM

- **INTEGER-PARTITION**:

  Instances: Finite $S \subseteq \mathbb{N}$

  Property: Exists $P \subset S$ s.t. $\Sigma P = \Sigma S/2$.

- Examples: For $S = \{1, 2, 3\}$ the answer is ***no.***

  For $S = \{1, 3, 4, 6\}$ it's ***yes.***

Example: INTEGER-PARTITION and EXACT-SUM

- **INTEGER-PARTITION**:

  Instances: Finite $S \subseteq \mathbb{N}$

  Property: Exists $P \subset S$ s.t. $\Sigma P = \Sigma S/2$.

- Examples: For $S = \{1, 2, 3\}$ the answer is ***no.***

  For $S = \{1, 3, 4, 6\}$ it's ***yes.***

- **EXACT-SUM**:

  Instances: Finite $S \subset \mathbb{N}$ and a target $t \in \mathbb{N}$

  Property: Exists $P \subset S$ s.t. $\Sigma P = t$

Example: INTEGER-PARTITION and EXACT-SUM

- **INTEGER-PARTITION**:

  Instances: Finite $S \subseteq \mathbb{N}$

  Property: Exists $P \subset S$ s.t. $\Sigma P = \Sigma S/2$.

- Examples: For $S = \{1, 2, 3\}$ the answer is ***no.***

  For $S = \{1, 3, 4, 6\}$ it's ***yes.***

- **EXACT-SUM**:

  Instances: Finite $S \subset \mathbb{N}$ and a target $t \in \mathbb{N}$

  Property: Exists $P \subset S$ s.t. $\Sigma P = t$

- Reduction ρ:

  Map each instance S of **INTEGER-PARTITION** to $(S,\ (\Sigma S)/2)$

Example: CLIQUE and INDEP-SET

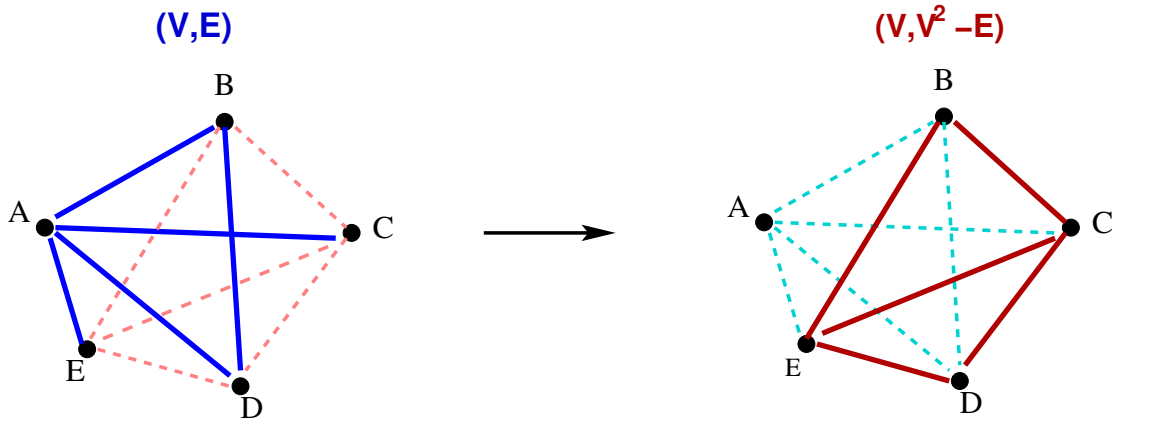
- ***Clique*** in graph $\mathcal{G}$: set of pairwise-adjacent vertices.
- **CLIQUE**: Given $\mathcal{G}$ and $t \in \mathbb{N}$

  does $\mathcal{G}$ have a clique of size $\geqslant t$?
- ***Independent set*** in $\mathcal{G}$: set of pairwise non-adjacent vertices.
- **INDEP-SET**: Given $\mathcal{G}$ and $t \in \mathbb{N}$

  does $\mathcal{G}$ have an independent-set of size $\geqslant t$?

CLIQUE reduces to INDEP-SET

Reduction by a “reverse-video” mapping:



A blue graph
Missing edges are in pink
{A,B,D} a clique of size 3

A red graph
Missing edges are in blue
{A,B,D} an ind set of size 3

Dfn of reductions between problems

- A ***reduction*** of a decision-problem $\mathcal{P}$ to a problem $\mathcal{Q}$ is a function

$$\rho : \{\textit{Instances of } \mathcal{P}\} \rightarrow \{\textit{Instances of } \mathcal{Q}\}$$

such that $X \in \mathcal{P}$ iff $\rho(X) \in \mathcal{Q}$.

That is, if $X \in \mathcal{P}$ then $\rho(X) \in \mathcal{Q}$

and if $X \notin \mathcal{P}$ then $\rho(X) \notin \mathcal{Q}$.

Dfn of reductions between problems

- A ***reduction*** of a decision-problem $\mathcal{P}$ to a problem $\mathcal{Q}$ is a function

$$\rho : \{\textit{Instances of } \mathcal{P}\} \rightarrow \{\textit{Instances of } \mathcal{Q}\}$$

such that $X \in \mathcal{P}$ iff $\rho(X) \in \mathcal{Q}$.

That is, if $X \in \mathcal{P}$ then $\rho(X) \in \mathcal{Q}$

and if $X \notin \mathcal{P}$ then $\rho(X) \notin \mathcal{Q}$.

- We write then $\rho : \mathcal{P} \leqslant \mathcal{Q}$.

Dfn of reductions between problems

- A **reduction** of a decision-problem $\mathcal{P}$ to a problem $\mathcal{Q}$ is a function

$$\rho : \{\textit{Instances of } \mathcal{P}\} \rightarrow \{\textit{Instances of } \mathcal{Q}\}$$

such that $X \in \mathcal{P}$ iff $\rho(X) \in \mathcal{Q}$.

That is, if $X \in \mathcal{P}$ then $\rho(X) \in \mathcal{Q}$

and if $X \notin \mathcal{P}$ then $\rho(X) \notin \mathcal{Q}$.

- We write then $\rho : \mathcal{P} \leqslant \mathcal{Q}$.
- A reduction ρ is helpful when it's easier

to compute the output $\rho(X)$ than to decide whether $X \in \mathcal{P}$.

- When ρ is computable we write $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$

and say that $\mathcal{P}$ **computably-reduces** to $\mathcal{Q}$.

ACCEPT $\leqslant_c$ ε-ACCEPT

- Map instance (M, w) of ACCEPT to instance M_w of ε-ACCEPT
  so that M accepts w iff M_w accepts ε.

ACCEPT $\leqslant_c$ ε-ACCEPT

- Map instance (M, w) of ACCEPT to instance M_w of ε-ACCEPT
  so that M accepts w iff M_w accepts ε.
- Define M_w to be the acceptor that on input x
  runs M on w as input, and accepts x if and when M accepts w.

ACCEPT $\leqslant_c$ ε-ACCEPT

- Map instance (M, w) of **ACCEPT** to instance M_w of **ε-ACCEPT**
  so that M accepts w iff M_w accepts ε.
- Define M_w to be the acceptor that on input x
  runs M on w as input, and accepts x if and when M accepts w.
- If M accepts w then M_w accepts every string.
  Otherwise M_w accepts no string.
- I.e. M accepts w iff $M_w = \rho(M, w)$ accepts ε.

ACCEPT $\leqslant_c$ ε-ACCEPT

- Map instance (M, w) of **ACCEPT** to instance M_w of **ε-ACCEPT**
  so that M accepts w iff M_w accepts ε.
- Define M_w to be the acceptor that on input x
  runs M on w as input, and accepts x if and when M accepts w.
- If M accepts w then M_w accepts every string.
  Otherwise M_w accepts no string.
- I.e. M accepts w iff $M_w = \rho(M, w)$ accepts ε.
- ρ is computable: It is a simple syntactic construction
  of algorithm M_w from algorithm M + string w.

Composing reductions

- *If functions* $f, g : \Sigma^* \to \Sigma^*$ *are computable, the so is* $f \circ g$.
- **Proof.** The output of f is fed to g as input.
- **Theorem**

  If $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$ *and* $\rho' : \mathcal{Q} \leqslant_c \mathcal{R}$ *then* $\rho \circ \rho' : \mathcal{P} \leqslant_c \mathcal{R}$.
- $\rho \circ \rho'$ is computable.

  It is a reduction:

$$x \in \mathcal{P} \quad \textbf{\textit{iff}} \quad \rho(x) \in \mathcal{Q} \qquad \text{(since } \rho \text{ is a reduction)}$$

$$\textbf{\textit{iff}} \quad \rho'(\rho(x)) \in \mathcal{R} \qquad \text{(since } \rho' \text{ is a reduction)}$$

Reductions preserve decidability

- **Theorem.** *Suppose* $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$ *. If* $\mathcal{Q}$ *is decidable then so is* $\mathcal{P}$ *.*
- Proof. To decide whether $X \in \mathcal{P}$

  compute $\rho(X)$ and run the decider for $\mathcal{Q}$ on $\rho(X)$ as input.
- Consequence: Show that a problem $\mathcal{P}$ is ***not decidable***

  by defining $\rho : \mathcal{Q} \leqslant_c \mathcal{P}$ for an undecidable $\mathcal{Q}$.

Proving decidability via computable reductions

- Consider the **PDA-ACCEPT** Problem:

  Given a PDA P over Σ and a string $w \in \Sigma^*$,

  does P accept w?

- We developed an algorithm that converts a PDA P

  to a CFG G_P equivalent to P.

- So the **PDA-ACCEPT** problem computably reduces to the problem:

  - **CFG-GENERATE:**

    Given a CFG G over Σ, and a string $w \in \Sigma^*$,

    does G generate w?

- We have a decision algorithm (CYK) deciding **CFG-GENERATE**,

  so we have a decision algorithm for **PDA-ACCEPT**.

Proving SD via computable reductions

- We know that the problem **accept**, referring to Turing acceptors, is SD.
- There is an algorithm for transforming Turing acceptors M to equivalent general grammars G, that is such that $(\mathcal{G}) = \mathcal{L}(\mathcal{M})$.

So the following problem is also SD.

CFG-GENERATE:

Given a grammar G and a string w does G generate w.

SCOPE PROPERTIES OF COMPUTING DEVICES

Decision problems about Turing machines

- Properties of Turing acceptors may be decidable:

  Runs more than 4 steps on input `001`

  Has more than 4 states

  The accept state is the only terminal state

- These refer to the inner workings of the Turing machine

  not to the language it recognizes.

- The ε-**accept** problem is different:

  It is about the language L recognized, not the recognizing device.

- The answer yes/no would be the same for any acceptor for L.

Scope-properties of machines

- Many important properties of computing devices M

  are ***scope-properties***, in that they are about ***what*** M does, and not about ***how*** it does it.

- So a scope-property of ***acceptors*** M

  is a property of the language that M recognizes, i.e. $\mathcal{L}(M)$.

- If two acceptors recognize the same language

  then they share every scope-property.

Scope-properties of machines

- Many important properties of computing devices M
  are ***scope-properties***, in that they are about ***what*** M does, and not about ***how*** it does it.
- So a scope-property of ***acceptors*** M
  is a property of the language that M recognizes, i.e. $\mathcal{L}(M)$.
- If two acceptors recognize the same language
  then they share every scope-property.
- Similarly, a scope-property of ***transducers*** M
  is a property of the partial-function that it computes.
- If two transducers compute the same partial-function
  then they share every scope-property.

Examples of scope-properties of Turing-acceptors

- $\mathcal{L}(M)$ is finite.
- $\mathcal{L}(M)$ is infinite.
- Accepts at least two strings, i.e. $\mathcal{L}(M) \geqslant 2$ elements.
- Every string accepted by M has even length.
- $\mathcal{L}(M)$ is a regular language.

  This does ***not*** mean that M is a DFA.
- For some $n > 0$ M accepts every string of length n.
- For every $n \geqslant 0$ M accepts some string of length n.

Examples for scope-properties of Turing-transducers M

- Computes a total function.
- Undefined for input ε.
- Define for all input of even length.
- Undefined for all input of even length.
- Constant (same output for all input
- Increasing: If $|x| < |y|$ then $|f(x)| < |f(y)|$
- Bounded: There is an $n \in \mathbb{N}$ s.t. $|f(x)| \leqslant n$ for all x.
- Unbounded: For every n there is some x s.t. $|f(x)| > n$.
- Inflationary: $|f(x)| \geqslant |x|$ for all x.

Non-scope properties of Turing machines

- Has more than 100 states.
- Reads every input to its end.
- For some input visits every state during computation
- Never runs more than n^2 steps for input of size $\leqslant n$
- Is a decider

  (but “recognizes a deciable language” ***is*** a scope-property!)

Rice's Theorem

- A property is ***trivial*** for a language L

  if it is either true of every $w \in L$ or false for every w.

- Example: The property $\mathcal{L}(M)$ *is SD*

  is always true: it just conveys the definition of SD.

- **Theorem.** (Henry Rice, 1951).

  There is no decidable scope-property of Turing-acceptors,

  other than the trivial ones.

- **Proof idea:**

  If $\mathcal{P}$ is non-trivial, then ε-**ACCEPT** $\leqslant_c$ $\mathcal{P}$.

  So $\mathcal{P}$ is undecidable.

Proof of Rice's Theorem

- Let $\mathcal{P}$ be a non-trivial scope-property of Turing acceptors.

  Fix some acceptor E recognizing $\emptyset$.

  Assume $E \notin \mathcal{P}$ (it won't matter).

  Also, $\mathcal{P}$ is non-trivial, so it is true of some acceptor A.

Note: E and A are on opposite sides of $\mathcal{P}$!

Proof of Rice's Theorem

- Let $\mathcal{P}$ be a non-trivial scope-property of Turing acceptors.

  Fix some acceptor E recognizing $\emptyset$.

  Assume $E \notin \mathcal{P}$ (it won't matter).

  Also, $\mathcal{P}$ is non-trivial, so it is true of some acceptor A.

  Note: E and A are on opposite sides of $\mathcal{P}$!

- Define $\rho : \varepsilon\text{-ACCEPT} \leqslant_c \mathcal{P}$:

- The acceptor $\rho(M)$, call it $M]$,

  initially disregards its input x and runs M on ε.

  If and when M accepts ε, M' fires A on x.

Proof of Rice's Theorem

- Let $\mathcal{P}$ be a non-trivial scope-property of Turing acceptors.

  Fix some acceptor E recognizing $\emptyset$.

  Assume $E \notin \mathcal{P}$ (it won't matter).

  Also, $\mathcal{P}$ is non-trivial, so it is true of some acceptor A.

  Note: E and A are on opposite sides of $\mathcal{P}$!

- Define $\rho : \varepsilon\text{-ACCEPT} \leqslant_c \mathcal{P}$:

- The acceptor $\rho(M)$, call it $M]$,

  initially disregards its input x and runs M on ε.

  If and when M accepts ε, M' fires A on x.

- So $\mathcal{L}(M') =$ if M accepts ε then $\mathcal{L}(A)$

  else $\emptyset$, i.e. $\mathcal{L}(E)$

- I.e. M accepts ε just in case $M' = \rho(M) \in \mathcal{P}$.

Proof of Rice's Theorem

- Let $\mathcal{P}$ be a non-trivial scope-property of Turing acceptors.

  Fix some acceptor E recognizing $\emptyset$.

  Assume $E \notin \mathcal{P}$ (it won't matter).

  Also, $\mathcal{P}$ is non-trivial, so it is true of some acceptor A.

  Note: E and A are on opposite sides of $\mathcal{P}$!

- Define $\rho : \varepsilon\text{-ACCEPT} \leqslant_c \mathcal{P}$:

- The acceptor $\rho(M)$, call it $M]$,

  initially disregards its input x and runs M on ε.

  If and when M accepts ε, M' fires A on x.

- So $\mathcal{L}(M') =$ if M accepts ε then $\mathcal{L}(A)$

  else $\emptyset$, i.e. $\mathcal{L}(E)$

- I.e. M accepts ε just in case $M' = \rho(M) \in \mathcal{P}$.

- The reduction ρ merely tinkers with algorithms' sytax,
  so it is computable.

More examples of scope problems

All problems below are non-trivial scope problems,
and are therefore undecidable, bu Rice's Theorem.

- **FINITE**: $\mathcal{L}(M)$ is finite.
- **INFINITE**: $\mathcal{L}(M)$ is infinite.
- **NOT-SINGLETON** $\mathcal{L}(M)$ has at least two elements.
- **EVEN** Every $w \in \mathcal{L}(M)$ has even length.
- **REGULAR**: $\mathcal{L}(M)$ is a regular language.
  (Note: M here can be any acceptor)
- **FILLED-LENGTH**: For some $n > 0$ M accepts all strings of length n.
  That is $\mathcal{L}(M) \supseteq \Sigma^n$ for some n.
- **NO-EMPTY-LENGTH**: For every $n \geqslant 0$ M accepts some string of length n.
  That is $\mathcal{L}(M) \cap \Sigma^n \neq \emptyset$ for all n.