NP-Completeness

Jennya

2013/10/31

為什麼Polynomial time就是”容易解”, ”可解的”?

- (1)
  $\Theta(n^{100})$雖然是polynomial time, 但實務上這麼高次的多項式並不常見
  通常如果找到一個polynomial-time algorithm, 比較快的方法很快也會被找到
- (2)
  通常使用不同的computation model(之後自動機會教到, 現在可以想像是單CPU v.s. 多CPU的機器), 某model可用polynomial-time解的問題在另外一個model也可用polynomial-time解
- (3)
  Polynomials are closed under addition, multiplication, and composition.

Abstract problem

I
problem instances → S
problem solutions

Q: Abstract problem
(binary relation)

Example: PATH

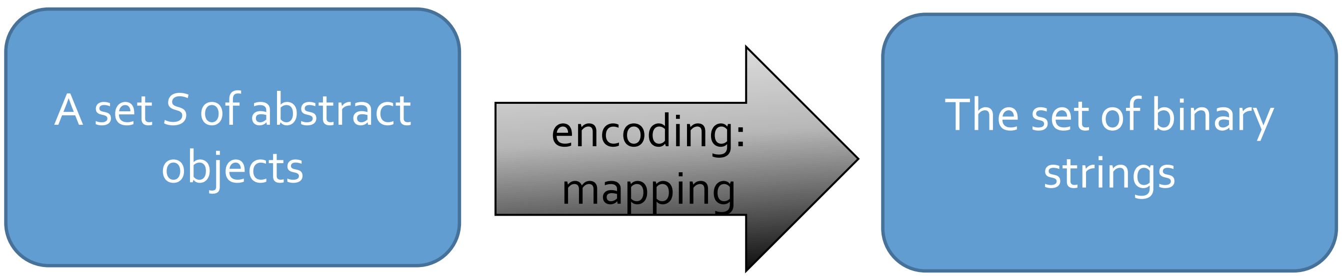
$i = \langle G, u, v, k \rangle$ →

$PATH(i) = 1$ if a shortest path from u to v has at most k edges

$PATH(i) = 0$ otherwise

Decision problem:
S={0,1}

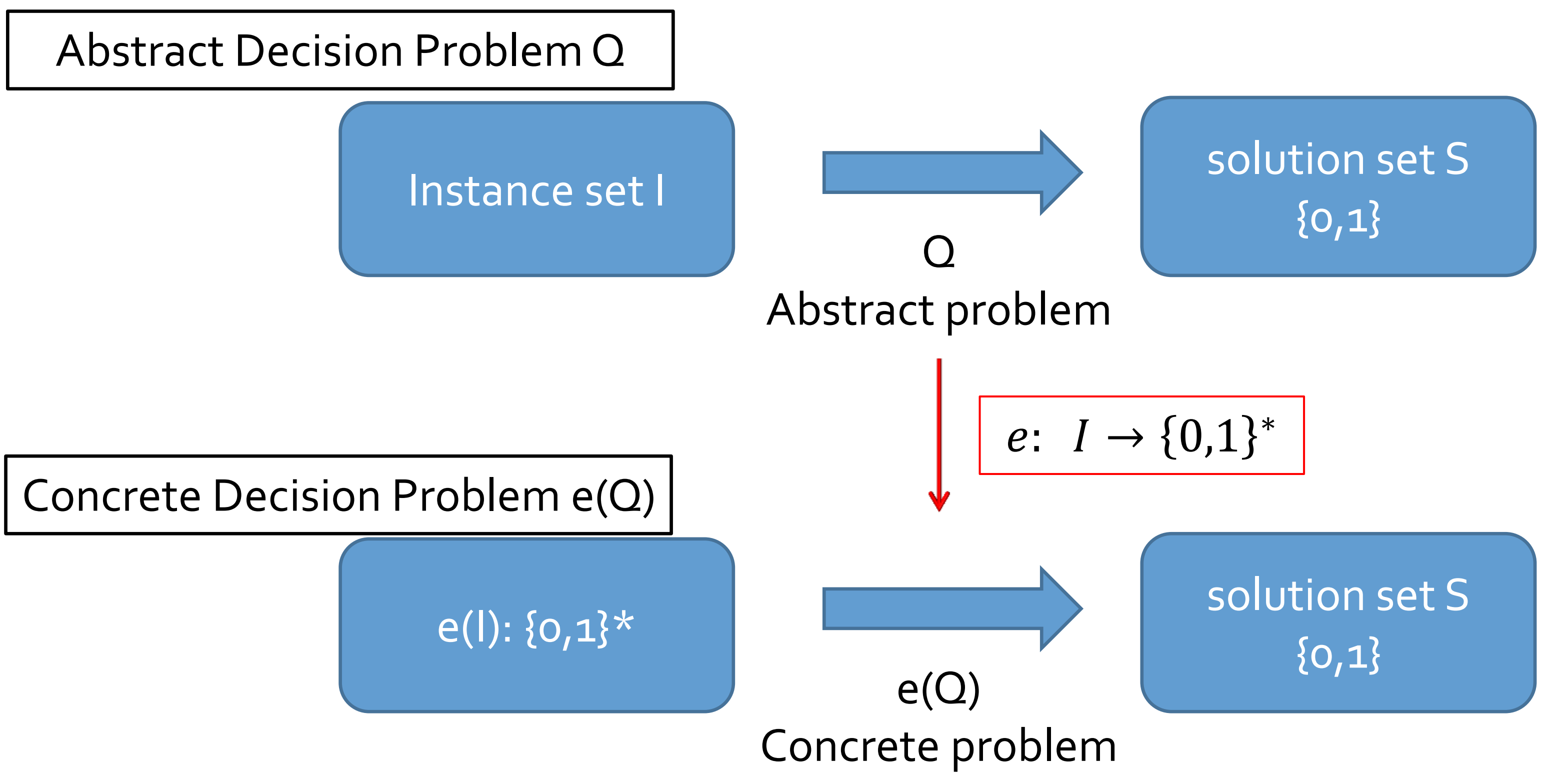
Encoding



Polygons, numbers, graphs, functions, ordered pairs, programs, …

Abstract problem轉換成 concrete problem

We can use encodings to map abstract problems to concrete problems



Concrete problem

- Concrete problem:
  instance set = the set of binary strings

- 「An algorithm solves a concrete problem in O(T(n))」
  一個problem的instance長度為n (i的長度, 即為binary string長度)
  而此algorithm可在O(T(n))時間產生解

- 「A concrete problem is polynomial-time solvable」
  有一個$O(n^k)$ $for\ some\ k$的algorithm可以解此problem

P的正式定義

The complexity class P:

The set of concrete decision problems that are polynomial-time solvable

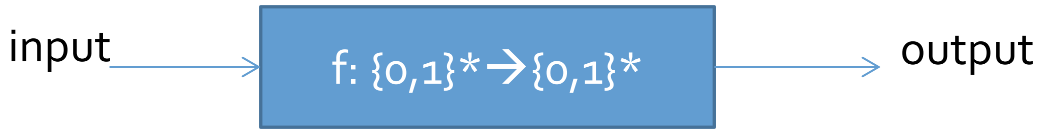
Encoding和花的時間有關嗎？

- 有! 極端的例子: **unary**
- input: integer k
  running time: $\Theta(k)$
- Unary encoding: $\overbrace{11111\ldots1111}^{k個}$
  input length n
  → running time: $\Theta(k) = \Theta(n)$
- binary encoding:
  input length $\mathrm{n} = \lfloor \log k \rfloor + 1$
  → running time: $\Theta(k) = \Theta(2^n)$
- Encoding決定是$\Theta(n)$ or $\Theta(2^n)$!!

Encoding和花的時間有關嗎?

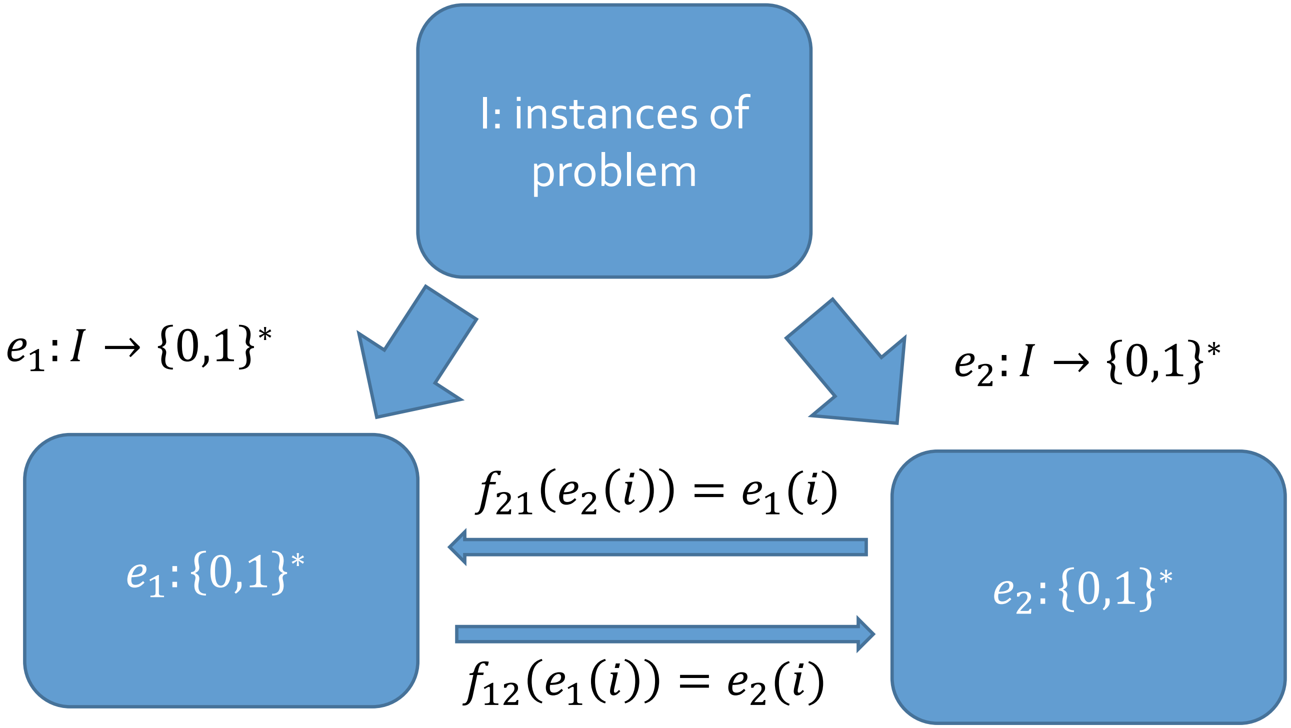
- 然而如果我們不考慮這麼極端的encoding方式(unary), 其他的encoding都不會影響到一個問題是否可以在polynomial time解決.
- 例: 使用三進位數和二進位數是沒有差別的, 因為我們可以在polynomial time裡面將三進位數轉換成二進位數.

polynomial-time computable function



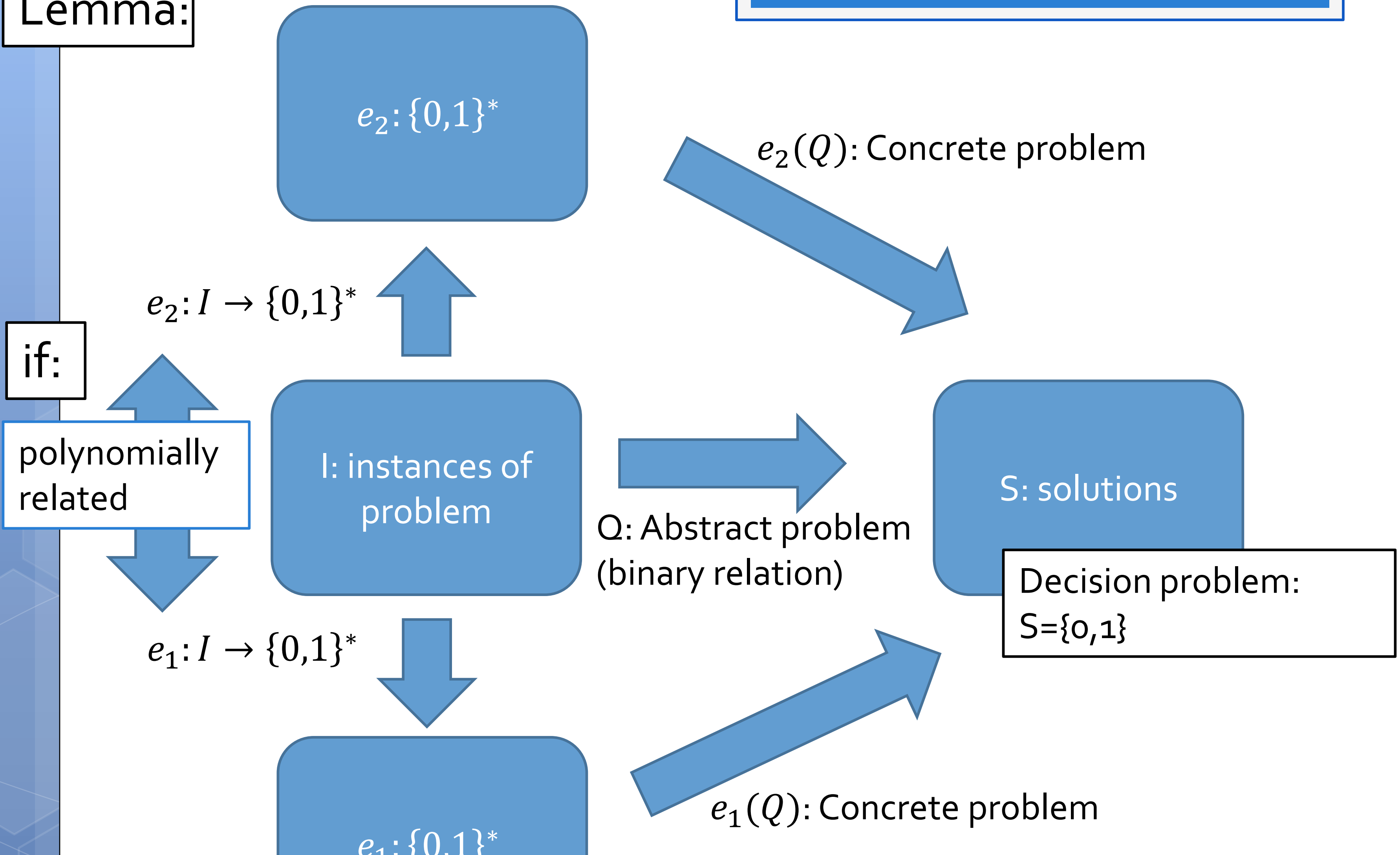
如果f花polynomial time可以把任何input轉成output, 則稱為**polynomial-time computable**

Polynomially related



如果有f_{12}和f_{21}是polynomial-time computable, 則e_1和e_2為 **polynomially related**.

Lemma:



Then: $e_1(Q) \in P$ if and only if $e_2(Q) \in P$

- Proof:
- 假設$e_1(Q)$可以在$O(n^k)$時間內解決(for some constant k)
- 假設 對每個problem instance i, $e_2(i)$轉換成$e_1(i)$需花$O(n^c)$(for some constant c), $n = |e_2(i)|$
- 則解決$e_2(Q)$ (input為$e_2(i)$) 先花$O(n^c)$轉換成$e_1(i)$
- $|e_1(i)| = O(n^c)$
- 再解決$e_1(Q)$ (input為$e_1(i)$), 花$O\left(|e_1(i)|^k\right) = O\left(n^{ck}\right)$
- c, k都是constant, 因此為polynomial time
- 因為是對稱的, 所以只需要證明一個方向.

只要encoding都是"合理的" ("簡要的")表示方式, 一個問題的複雜度(能否在polynomial time裡面解掉)不會被encoding影響.

A Formal-language Framework

- An alphabet Σ: a finite set of symbols
- A language L over Σ: 使用Σ裡面的symbol組合而成的字串 (不一定包含全部可能的字串)
- Ex: $\Sigma = \{0,1\}$, L (over Σ)=$\{10,11,101,111, \ldots\}$
- empty string: ϵ
- empty language: $\emptyset$
- Σ^*: the language with all strings over Σ

Operations on languages

- Union
- Intersection
- Complement: $\overline{L} = \Sigma^* - L$
- Concatenation of $L_1 L_2$:
  $L = \{x_1 x_2 \colon x_1 \in L_1 \; and \; x_2 \in L_2\}$
- Closure (Kleene Star):
  $L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots$
  L^k:concatenation 自己k次

應用formal language framework…

- We can view
  「a decision problem Q 」as
  「a language L over $\Sigma = \{0,1\}$」
  $=> L = \{x \in \Sigma^*: Q(x) = 1\}$
- Q的instance set為Σ^*
- Q = 能夠產生答案為1(yes)的這些instances

- i.e. PATH problem的language：

$$\begin{aligned}\text{PATH} = \{\langle G, u, v, k\rangle : \ & G = (V, E) \text{ is an undirected graph,} \\ & u, v \in V, \\ & k \geq 0 \text{ is an integer, and} \\ & \text{there exists a path from } u \text{ to } v \text{ in } G \\ & \text{consisting of at most } k \text{ edges}\}\ .\end{aligned}$$

Accepts and Rejects

- An algorithm A **accepts** a string $x \in \{0,1\}^*$ if, given input x, the algorithm's output A(x) = 1
- An algorithm A **rejects** a string $x \in \{0,1\}^*$ if, given input x, the algorithm's output A(x) = 0
- **The language accepted by an algorithm A** is the set of strings $L = \{x \in \{0,1\}^*: A(x) = 1\}$
- 注意: L is accepted by A, 不一定表示$x \notin L$會被A reject! (ex. 無窮迴圈)
- A language is **decided** by an algorithm A if every binary string in L is accepted by A and every binary string not in L is rejected by A
- A language is **accepted in polynomial time** if it is accepted by A and if A accepts x in time $O(n^k)$ for a constant k and any length-n string $x \in L$.

使用formal-language framework定義complexity class P

- 可以用「a set of languages」定義「complexity class」

  如何決定是不是在這個class(set)中：
  由「決定一個string x是否屬於L」的 algorithm的running time而定

- 使用這個方式, 我們可以重新定義P這個complexity class:

$P = \{L \subseteq \{0,1\}^*:$
there exists an algorithm A that decides L in polynomial time$\}$

- Theorem:
  $P = \{L : L$ is accepted by a polynomial time algorithm$\}$.

- $P = \{L \subseteq \{0,1\}^*:$
  there exists an algorithm A that **decides** L in polynomial time$\}$

  ↓ ↑

  $P = \{L \subseteq \{0,1\}^*:$
  there exists an algorithm A that **accpets** L in polynomial time$\}$

- Proof:
- The class of languages decided by polynomial-time algorithms是the class of languages accepted by polynomial-time algorithms的subset.
- 所以我們只需要證如果L is accepted by a polynomial-time algorithm, 它也可以decided by a polynomial-time algorithm.

- 假設L是被某polynomial-time algorithm A accept.
- 我們要利用A做成一個algorithm A′可以decides L.
- 因為A accepts L in $O(n^k)$ for some constant k, 所以我們也可以說 A accepts L 最多花cn^k個steps for a constant c
- 對任何input x, A′ 利用A, 先執行cn^k個steps. 如果這時候A accept x了, A′就accept x. 如果A還沒accept x, A′就reject x.
- A′使用A的overhead不會超過一個polynomial factor, 所以A′是一個可以decide L的polynomial time algorithm.