Polynomial-Time Reductions

Andreas Klappenecker
[partially based on slides by Professor Welch]

Formal Languages and Decision Problems

Languages and Decision Problems

Language: A set of strings over some alphabet

Decision problem: A decision problem can be viewed as the formal language consisting of exactly those strings that encode YES instances of the problem.

Yes instance:

	1		
4		2	
	4		2
1			4

No instance:

	1	3	
4			
1	4	?	2
			4

The Language Prime

Let us encode positive integers in binary representation.

The decision problem “Is x a prime?” has the following representation as a formal language:

$L_{Primes} = \{10,11,101,111,...\}$

where 10 encodes 2, 11 encodes 3, 101 encodes 5, and so on.

Polynomial Reduction

Polynomial Reduction

Let L_1 be a language over an alphabet V_1.

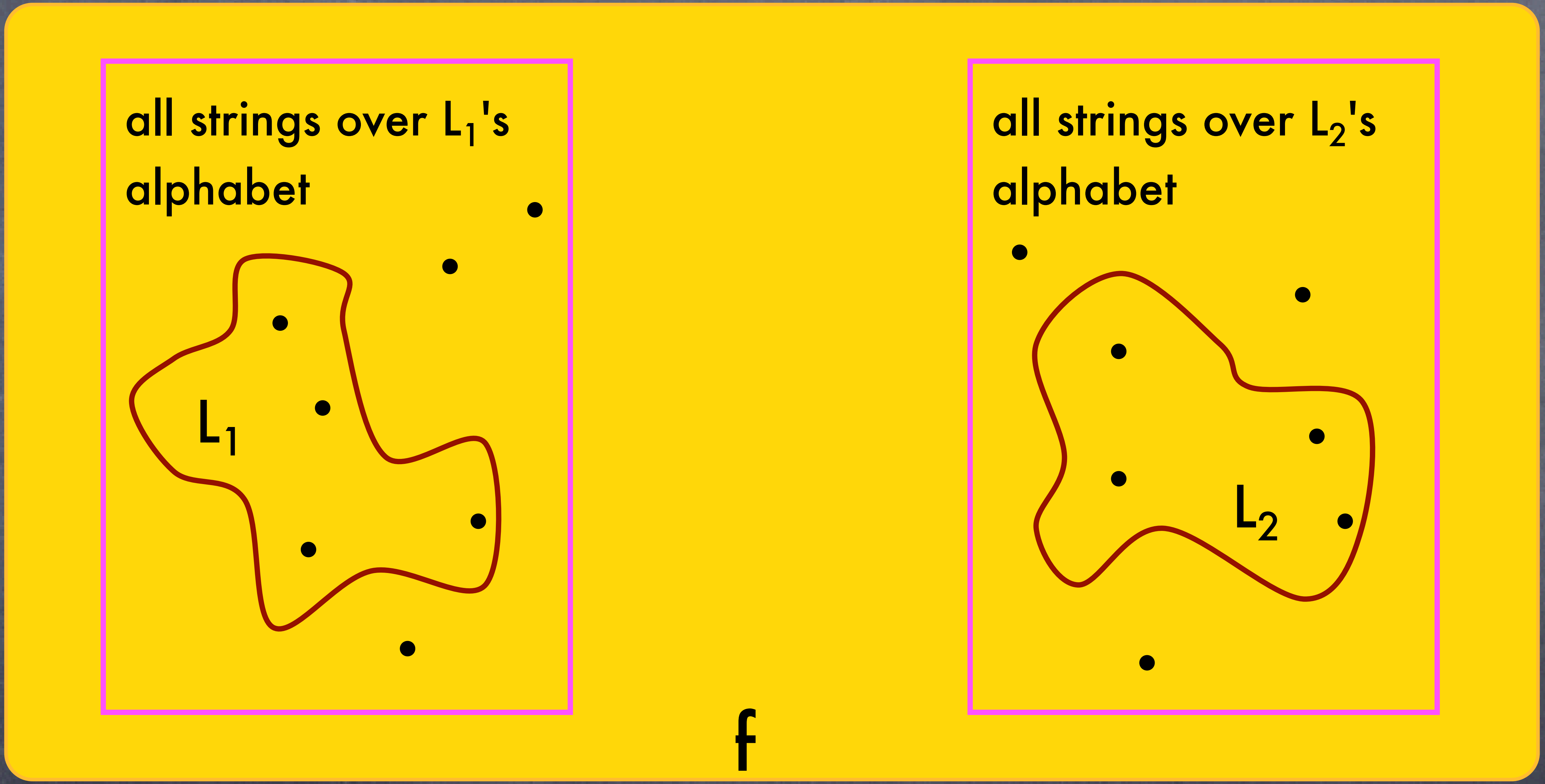
Let L_2 be a language over an alphabet V_2.

A polynomial-time reduction from L_1 to L_2 is a function $f: V_1^* \text{->} V_2^*$ such that

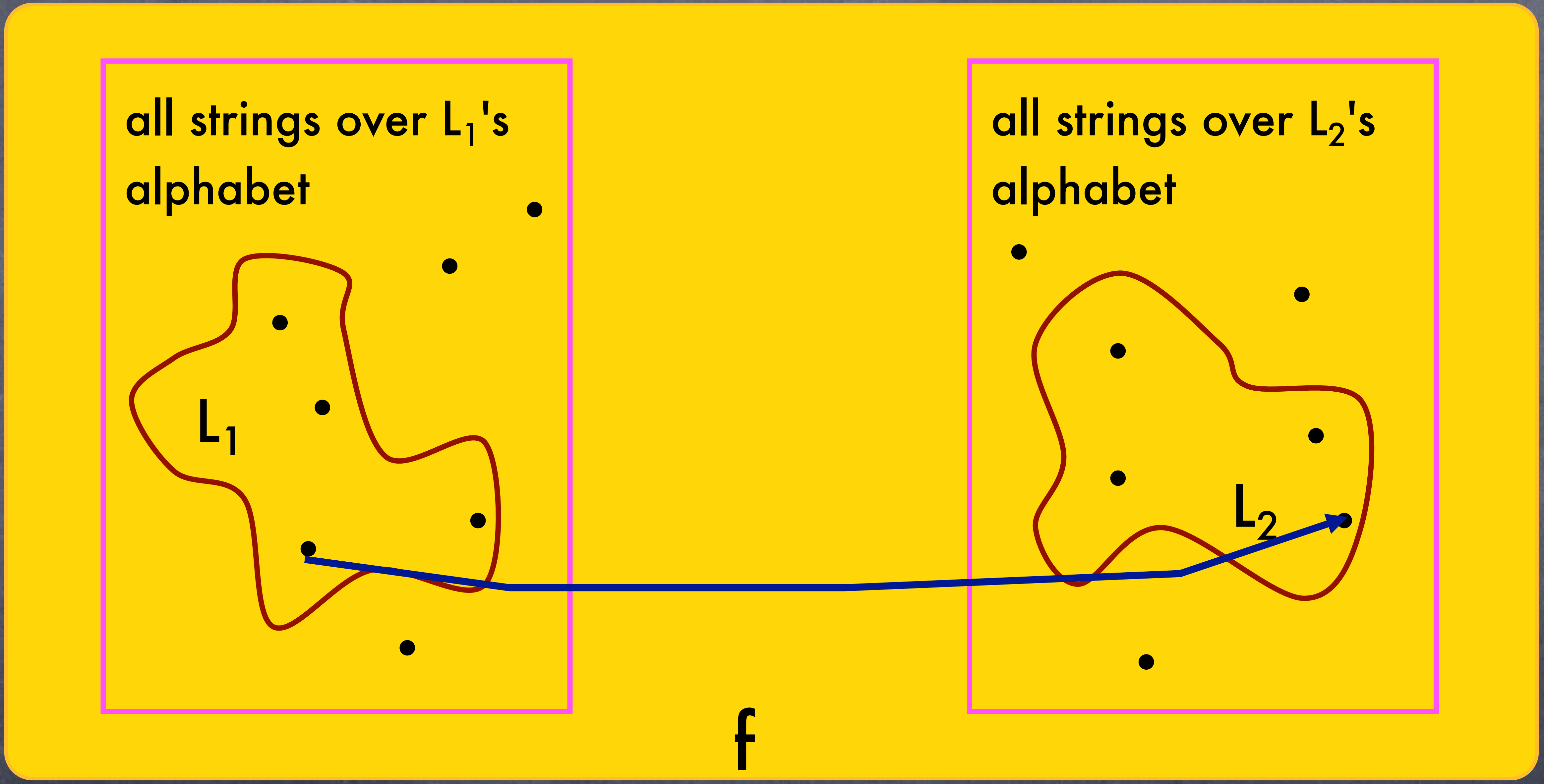
(1) f is computable in polynomial time

(2) for all x in V_1^*, x is in L_1 if and only if f(x) is in L_2

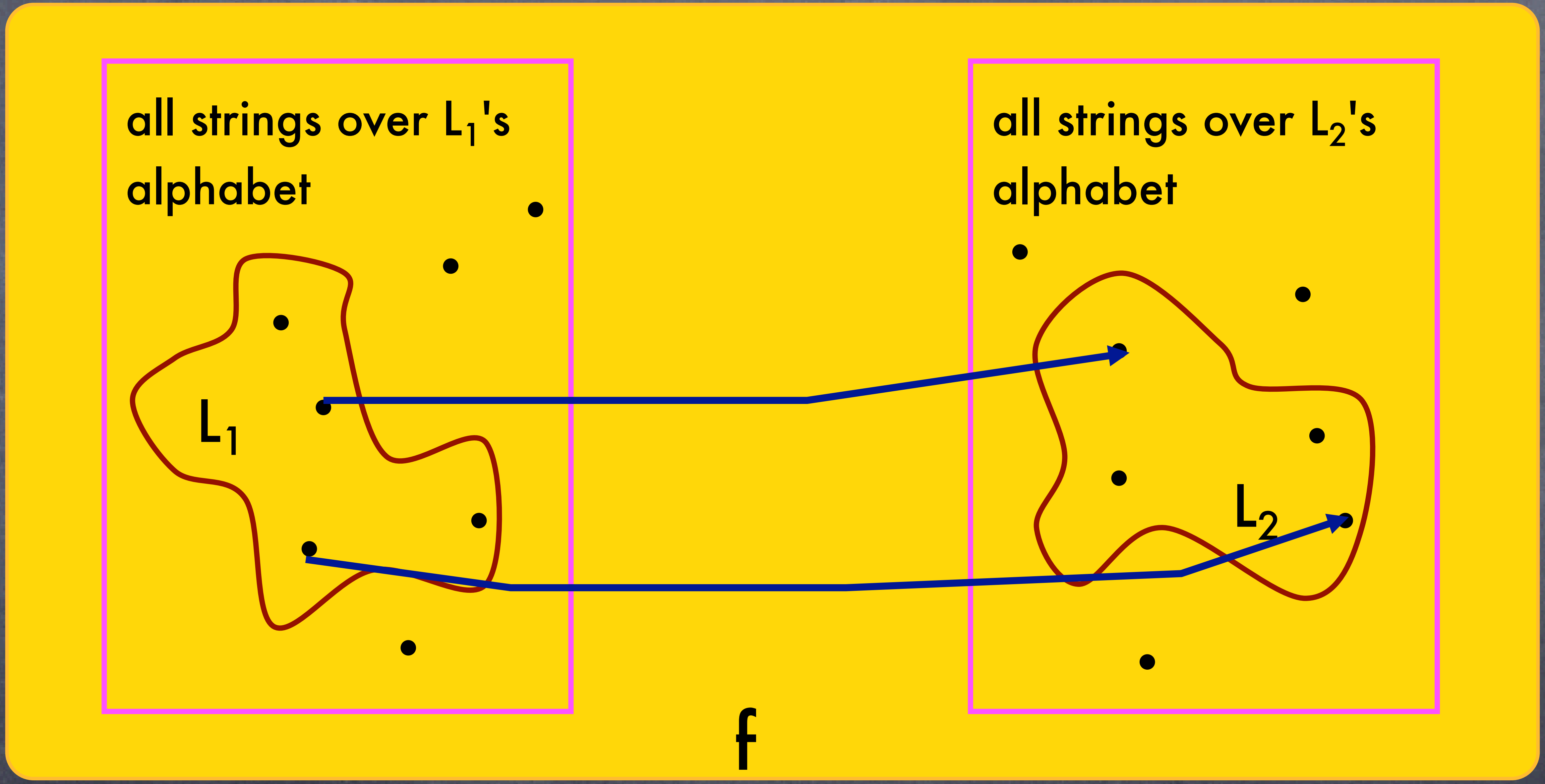
Polynomial Reduction



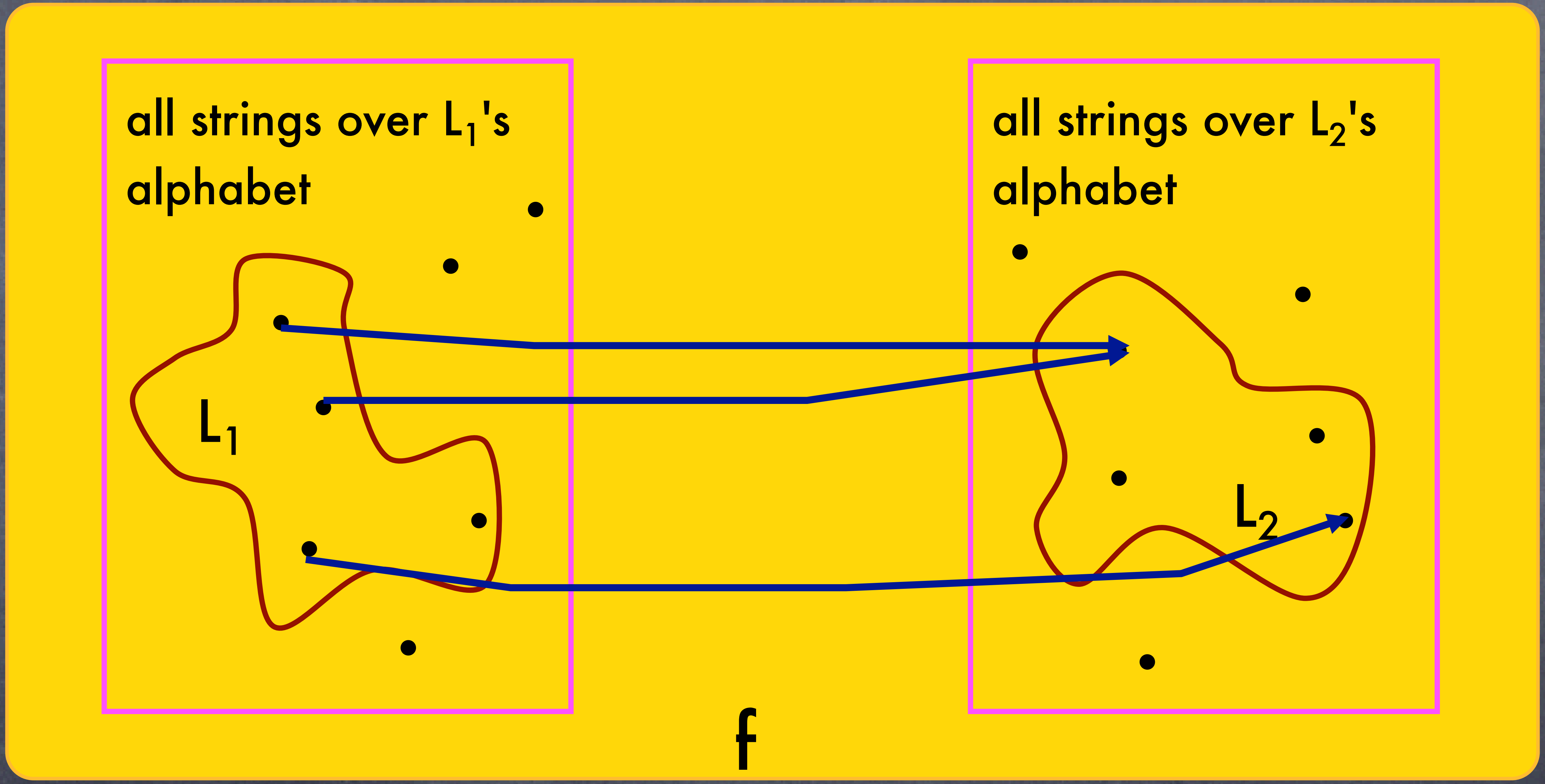
Polynomial Reduction



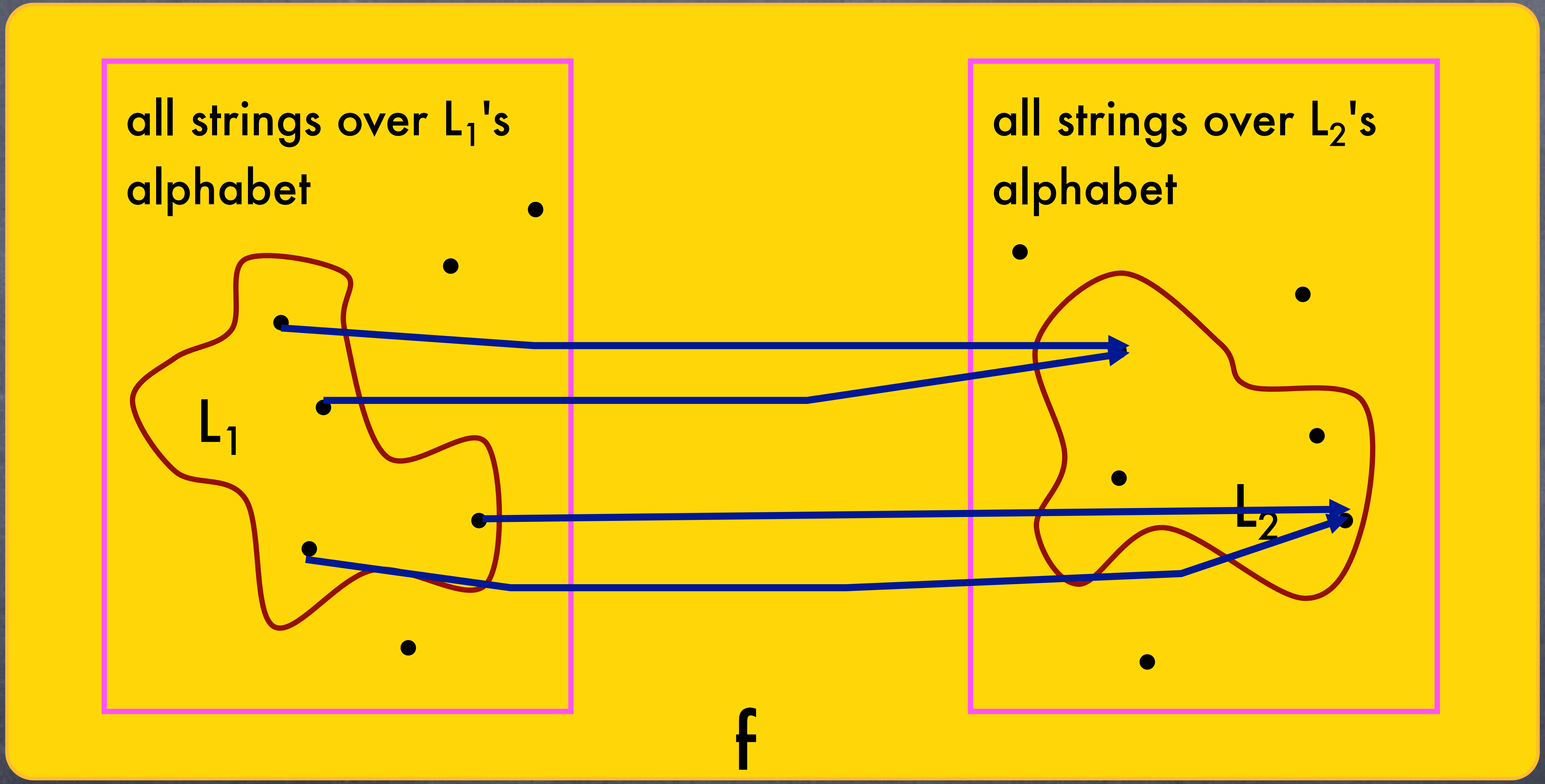
Polynomial Reduction



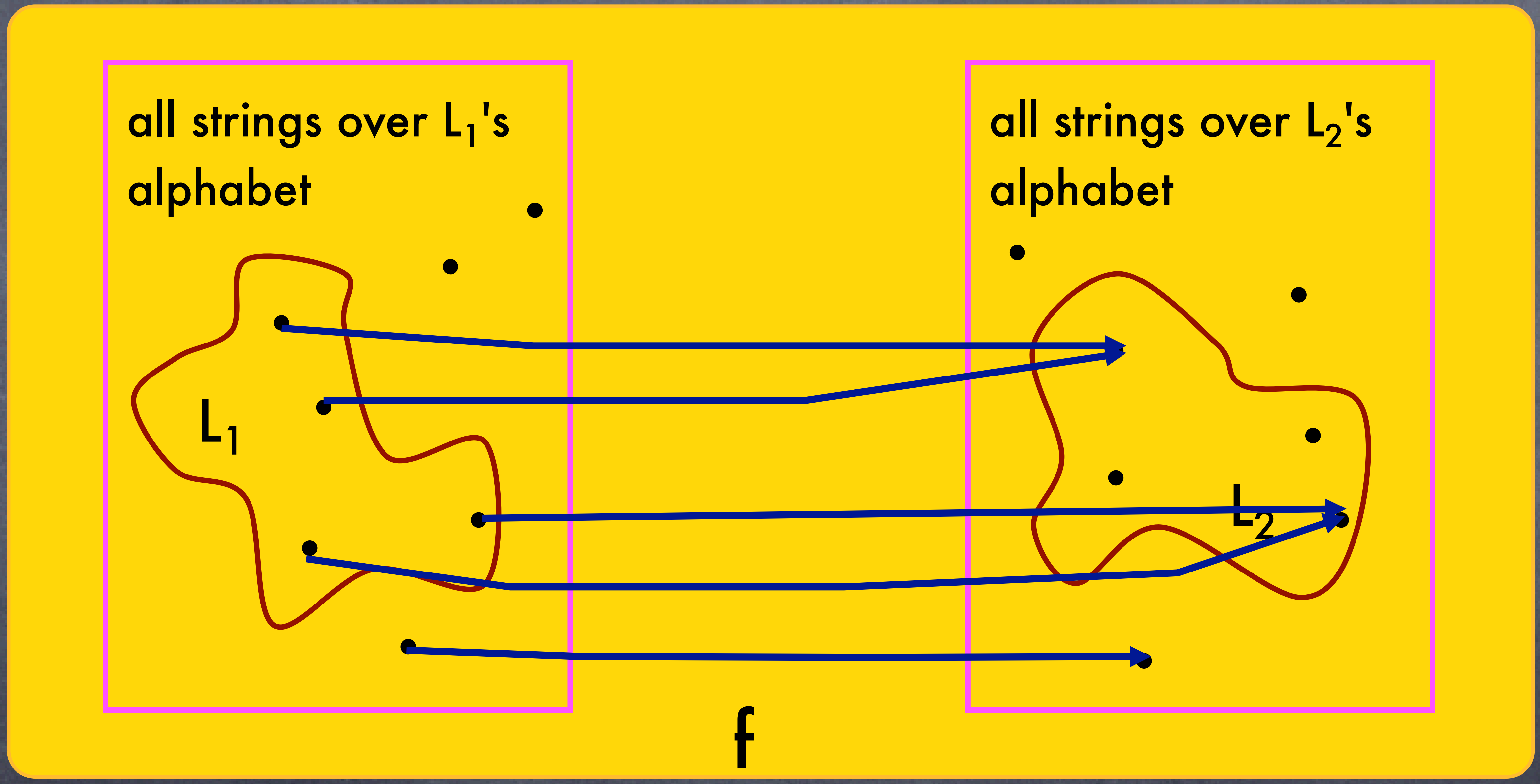
Polynomial Reduction



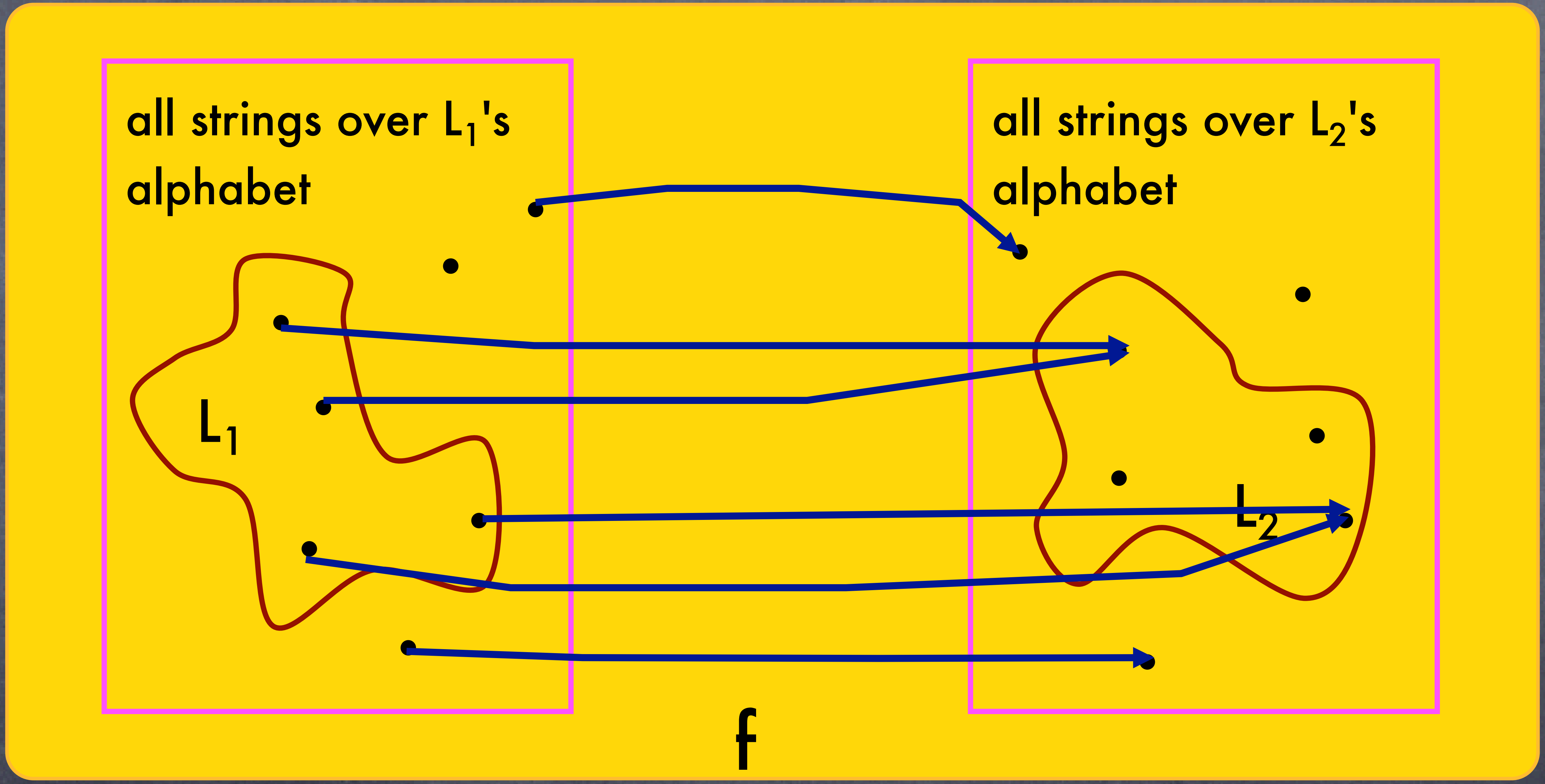
Polynomial Reduction



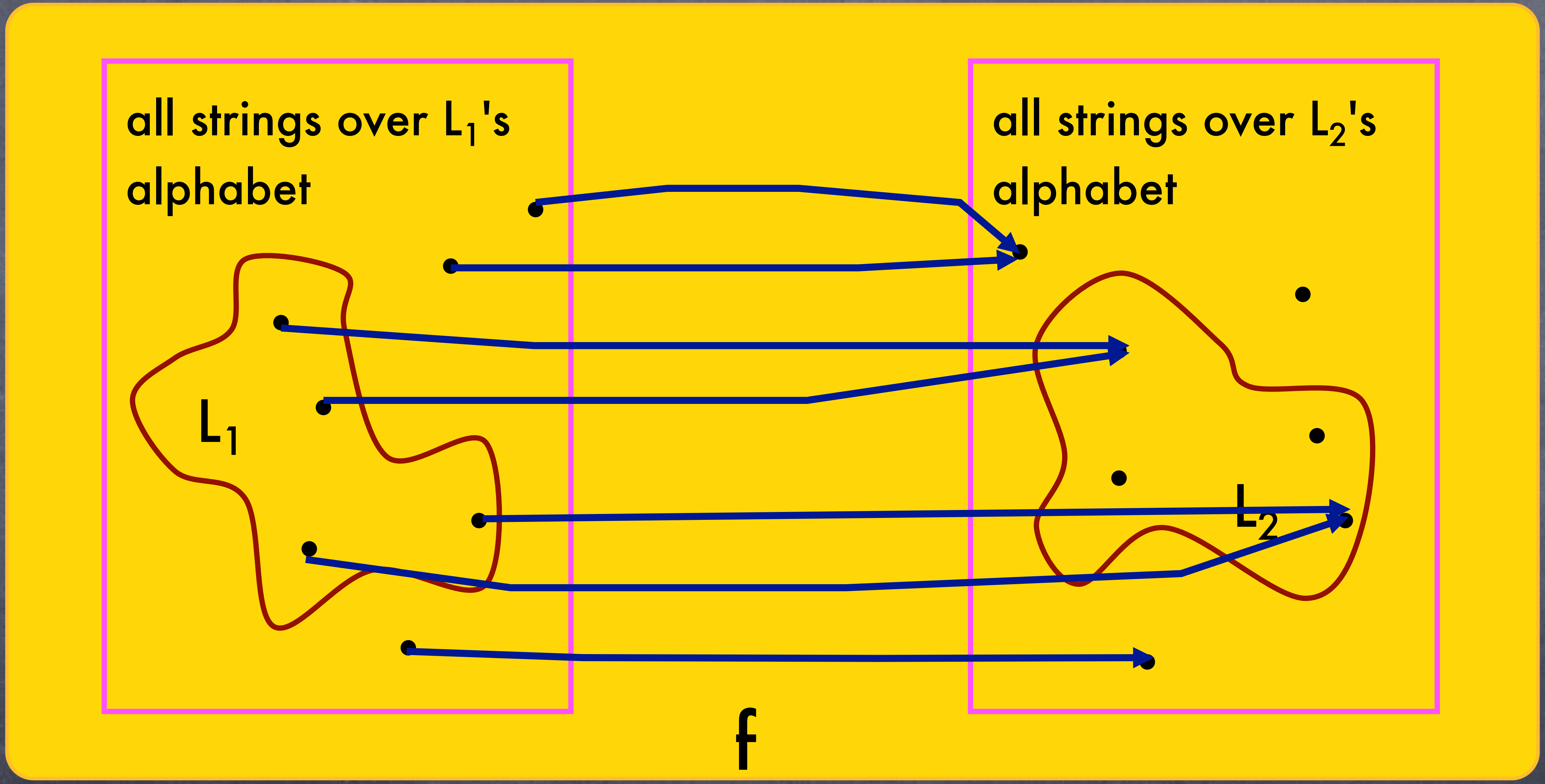
Polynomial Reduction



Polynomial Reduction



Polynomial Reduction



Polynomial Reduction

- YES instances map to YES instances
- NO instances map to NO instances
- computable in polynomial time
- Notation: $L_1 \leq_p L_2$
- [Think: L_2 is at least as hard as L_1]

Polynomial Reduction Theorem

Theorem If $L_1 \leq_p L_2$ and L_2 is in P, then L_1 is in P.

Proof. Let A_2 be a polynomial time algorithm for L_2. Here is a polynomial time algorithm A_1 for L_1.

- input: x
- compute f(x)
- run A_2 on input f(x)
- return whatever A_2 returns

Polynomial Reduction Theorem

Theorem If $L_1 \leq_p L_2$ and L_2 is in P, then L_1 is in P.

Proof. Let A_2 be a polynomial time algorithm for L_2. Here is a polynomial time algorithm A_1 for L_1.

- input: x
- compute f(x)
- run A_2 on input f(x)
- return whatever A_2 returns

$|x| = n$

Polynomial Reduction Theorem

Theorem If $L_1 \leq_p L_2$ and L_2 is in P, then L_1 is in P.

Proof. Let A_2 be a polynomial time algorithm for L_2. Here is a polynomial time algorithm A_1 for L_1.

- input: x
- compute f(x)
- run A_2 on input f(x)
- return whatever A_2 returns

$|x| = n$

takes p(n) time

Polynomial Reduction Theorem

Theorem If $L_1 \leq_p L_2$ and L_2 is in P, then L_1 is in P.

Proof. Let A_2 be a polynomial time algorithm for L_2. Here is a polynomial time algorithm A_1 for L_1.

- input: x
- compute f(x)
- run A_2 on input f(x)
- return whatever A_2 returns

$|x| = n$

takes $p(n)$ time

takes $q(p(n))$ time

Polynomial Reduction Theorem

Theorem If $L_1 \leq_p L_2$ and L_2 is in P, then L_1 is in P.

Proof. Let A_2 be a polynomial time algorithm for L_2. Here is a polynomial time algorithm A_1 for L_1.

- input: x — $|x| = n$
- compute $f(x)$ — takes $p(n)$ time
- run A_2 on input $f(x)$ — takes $q(p(n))$ time
- return whatever A_2 returns — takes $O(1)$ time

Implications

- Suppose that $L_1 \leq_p L_2$
- If there is a polynomial time algorithm for L_2, then there is a polynomial time algorithm for L_1.
- If there is no polynomial time algorithm for L_1, then there is no polynomial time algorithm for L_2.

$HC \leq_p TSP$

Traveling Salesman Problem

Suppose that we are given a set of cities, distances between all pairs of cities, and a distance bound B.

Traveling Salesman Problem: Does there exist a route that visits each city exactly once and returns to the origin city with a total travel distance <= B?

TSP is in NP: Given a candidate solution (a tour), add up all the distances and check if total is at most B.

Example of a Reduction

Theorem HC $\leq_p$ TSP.

Proof. Given a graph G, the Hamiltonian circuit decision problem tries to decide whether or not G has a Hamiltonian circuit.

A polynomial reduction from HC to TSP has to transform G into an input for the TSP decision problem. More precisely, the graph G needs to be transformed in polynomial time into a configuration of (cities, distances, and bound B) such that

G has a Hamiltonian circuit iff the resulting TSP input has a tour of cities that has a total distance <= B.

The Reduction

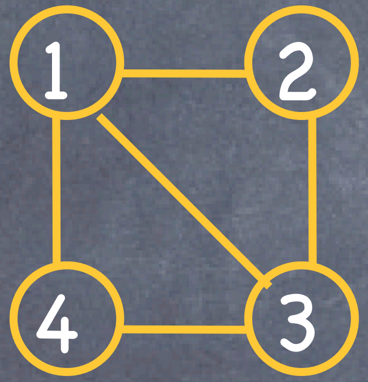
Given undirected graph $G = (V,E)$ with m nodes, construct a TSP input like this:

- set of m cities, labeled with names of nodes in V
- distance between u and v is 1 if (u,v) is in E, and is 2 otherwise
- bound $B = m$

This TSP input be constructed in time polynomial in the size of G.

Figure for Reduction

HC input



Hamiltonian cycle: 1,2,3,4,1

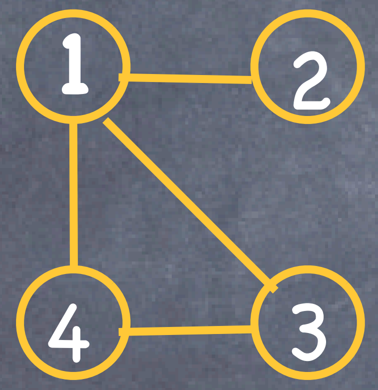
TSP input

dist(1,2) = 1
dist(1,3) = 1
dist(1,4) = 1
dist(2,3) = 1
dist(2,4) = 2
dist(3,4) = 1
bound = 4

tour w/ distance 4: 1,2,3,4,1

Figure for Reduction

HC input



TSP input

dist(1,2) = 1
dist(1,3) = 1
dist(2,4) = 2
dist(2,3) = 2
dist(1,4) = 1
dist(3,4) = 1
bound = 4

no Hamiltonian cycle

no tour w/ distance at most 4

Correctness of the Reduction

- Check that input G is in HC (has a Hamiltonian cycle) if and only if the input constructed is in TSP (has a tour of length at most m).
- => Suppose G has a Hamiltonian cycle $v_1, v_2, \ldots, v_m, v_1$.
  - Then in the TSP input, $v_1, v_2, \ldots, v_m, v_1$ is a tour (visits every city once and returns to the start) and its distance is $1 \cdot m = B$.

Correctness of the Reduction

- <=: Suppose the TSP input constructed has a tour of total length at most m.
  - Since all distances are either 1 or 2, and there are m of them in the tour, all distances in the tour must be 1.
  - Thus each consecutive pair of cities in the tour correspond to an edge in G.
  - Thus the tour corresponds to a Hamiltonian cycle in G.

Implications

- If there is a polynomial time algorithm for TSP, then there is a polynomial time algorithm for HC.
- If there is no polynomial time algorithm for HC, then there is no polynomial time algorithm TSP.

Transitivity of Reductions

Theorem: If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$,

then $L_1 \leq_p L_3$.

Proof:

Transitivity of Reductions

Theorem: If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$,

then $L_1 \leq_p L_3$.

Proof: