Part III

Turing Machines and Effective Computability

PART III
Chapter 3

Undecidable Problems

The Halting Problem

- L : any of your favorite programming languages (C, C++, Java, BASIC, TuringLang, etc.)
  - Assumption: Every L-program P has a source representation(string?) “P” that can be used as an input of L-programs.
  - If P accepts a string as input, we can invoke P with its source “P" to get the result P(“P"). Note This is always true provided L is a *general purpose* language.
- Problem: Design an L-program HALT(String, String) :boolean such that, when invoked with the source "P" of any L-program P(-) and a string X as input,

HALT("P", X), will

  - return true if P(X) halt, and
  - return false if P(X) does not halt.
  - Note: we don’t care about the returned value if the first input is not passed the source of a program which requires a string input.
- The problem of ,given (the source "P" of) an (L-)program P and a data X, determining if P will halt on input X is called the halting problem [for L].
  - It can be shown that the halting problem is undecidable (or unsolvable) [i.e.,*An L-program HALT(.,.) with the above behavior does not exist!!]*

Halt("P",X) does not exist

- Ideas leading to the proof:

Problem1 : What about the truth value of the sentences:

1. L: What "L" describes is false
2. I am lying.

Problem 2 : Let S = {X | X $\notin$ X}. Then does S belong to S or not ?
The analysis: S $\in$ S => S $\notin$ S; S $\notin$ S => S $\in$ S.

Problem 3 : 矛盾說: 1. 我的矛無盾不穿 2. 我的盾可抵擋所有茅
結論: 1. 2. 不可同時為真。

Problem 4 : 萬能上帝: 萬能上帝無所不能 => 可創造一個不服從他的子民
=> 萬能上帝無法使所有子民服從=> 萬能上帝不是萬能 .
結論:萬能上帝不存在。

Conclusion:

- 1. S is a class(,which could not be a member of any class/set,) but not a set!!
- 2. If a language is too powerful, it may produce expressions that is meaningless or can not be realized.

- Question: If HALT(P,X) can be programmed, will it incur any absurd result like the case of S?

Ans: yes!!

The proof

- Assume HALT(String,String) does exist (i.e, can be written as an L-program).
- Now construct a new program Diag(String) with HALT() as a subroutine as follows: For any input string S,

```
Diag(S) {   L1: if HALT(S,S) then goto L1 ; // while(HALT(S,S));
            L2: end.  }                      //
```

- Properties of Diag():
  - 1. Diag(S) halts iff HALT(S,S) returns false.
  - 2. Hence if S= "P" is the source of a program P ==>
  - Diag("P") halts iff HALT("P","P") returns false
  - iff P does not halt on "P" (i.e., P("P") does not halt).
- The absurd result: Will Diag() halt on input "diag" ?

Diag("Diag") halts <=> Diag("Diag") does not halt. --- by(2)

a contradiction!! Hence both Diag and HALT could not be implemented as an L-program.

Analysis of diag("p") and HALT("p","p")

1. Let $p_1, p_2, \ldots$ be the set of all programs accepting one string input.
2. cell(m,n) = 1/0 means m(n) halts/does not halt.
3. The diagonal row diag(.) corresponds to the complement of p("p").
4. if the diagonal row could be decided by the program HALT("p","p") then the diag() program would exist (= p_m for some m).
5. Property of diag("p_j") : p_j("p_j") halts iff diag("p_j") does not halt.
4. There is a logical contradiction in(diag,"diag") as to it is 0 or 1.
6. Hence neither diag() nor HALT() exist. "

	"p_1"	"p_2"	"p_3"	…	"p_k"	"diag"	…	…
p_1	0							
p_2		1						
p_3			1					
…				….				
p_k					1			
diag	1	0	0	…	0	x ~x	0	…
…							1	
…								…

The Halting problem (for Turing machine)

- H $=_{def}$ {"M" "w" |STM M halts on input w }
  $\subseteq \Sigma_0^* = \{a,q,0,1,(,),\ ,\}^*$.
- Notes:

1. By Turing-Church Thesis, any computable function or language can be realized by a [standard] Turing machine; hence *H represents all instances of program-data pairs (P,D) s.t. program P halts on input D*.
2. Hence to say HALT(P,X) does not exist is equivalent to say that there is no (total) Turing machine that can decide the language H (i.e., H is not recursive.)

Theorem: H is r.e. but not recursive.

Pf: (1) H is r.e. since H can be accepted by modifying the universal TM U so that it accepts no matter it would accept or reject in the original UTM.

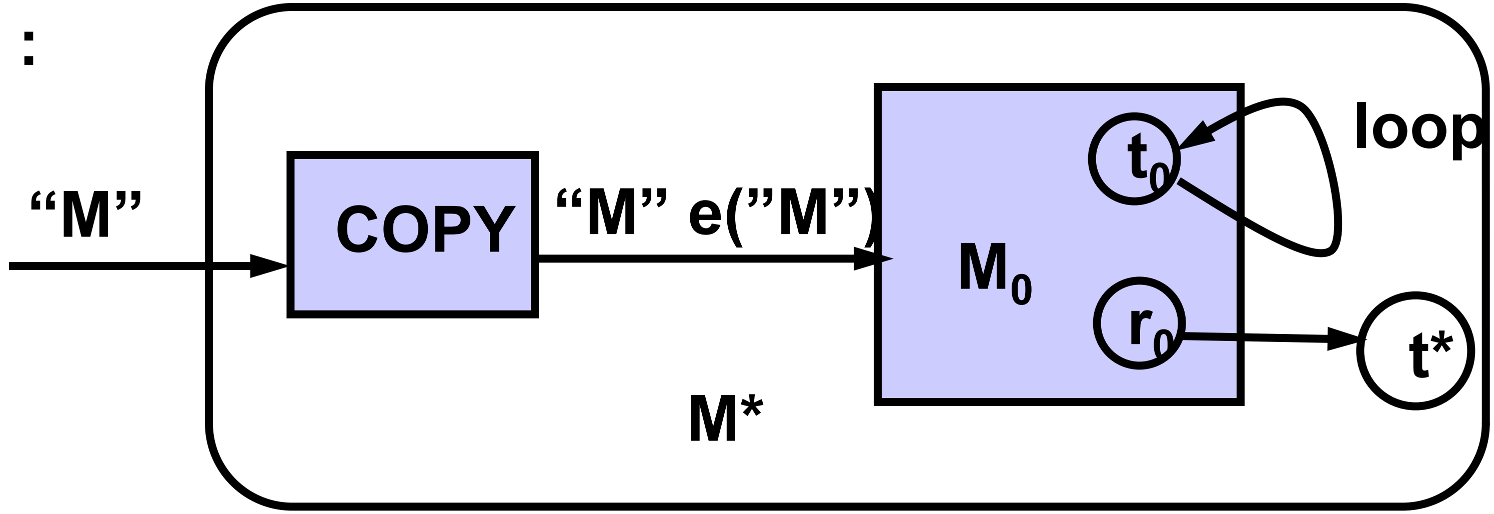
(2) H is not recursive: the proof is analog to previous proof.

Assume H is recursive => H = L(M) for some total TM M_0.

H is not recursive

Now design a new TM M* with Σ_0 as input alphabet as follows:

- On input “M” :



1. Assume “M” $\in \Sigma_0^*$ is a string over input alphabet of M.
2. Append a further encoding e(“M”) of the TL program “M” to the input program ”M”.
   - Note: ”M” $\in \Sigma_0^*$ = {a,q, 0,1,(,), , }*. Hence if “M” = a q 0 1 ( ) then e("M") = a10 a11 a000 a001 a010 a011 …
3. Call and execute M_0(“M” e(”M”)) // M_0 uses alphabet Σ_0
   2.1 if M_0 accepts (t_0) => M* loop and does not halt.
   2.2 if M_0 rejects (r_0) => goto accept state t* of M* and halt.

H is not recursive

Properties of M*:

0. M* and M_0 use input alphabet Σ_0.
1. M_0 is to HALT what M* is to Diag.
2. If M uses input alphabet Σ_0 (ex: M* and M_0), then M* (“M”) halts iff M_0(“M” e(”M”)) rejects iff M does not halt on input ”M” .

Absurd result: Will M*(“M*”) halt ?

By (2), M*(“M*”) halts iff M* does not halt on input “M*”, a contradiction!!

Hence M* and M_0 does not exist.

Corollary: The class of recursive languages is a proper subclass of the class of r.e. languages.

pf: H is r.e. but not recursive.

More undecidable problems about TMs

- **Theorem: The following problems about TMs are undecidable.**

1. **[General Halting Problem; GHP] Given a TM M and input w, does M halt on input w.**
2. **[Empty input Halting Problem; EHP] Given a TM M, does M halt on the empty input string ε?**
3. **Given a TM M, does M halt on any input ?**
4. **Given a TM M, does M halt on all inputs ?**
5. **Given two TMs M1 and M2, do they halt on the same inputs ?**
6. **Given a TM M, Is the set h(M) = {x | M halts on x} recursive ?**

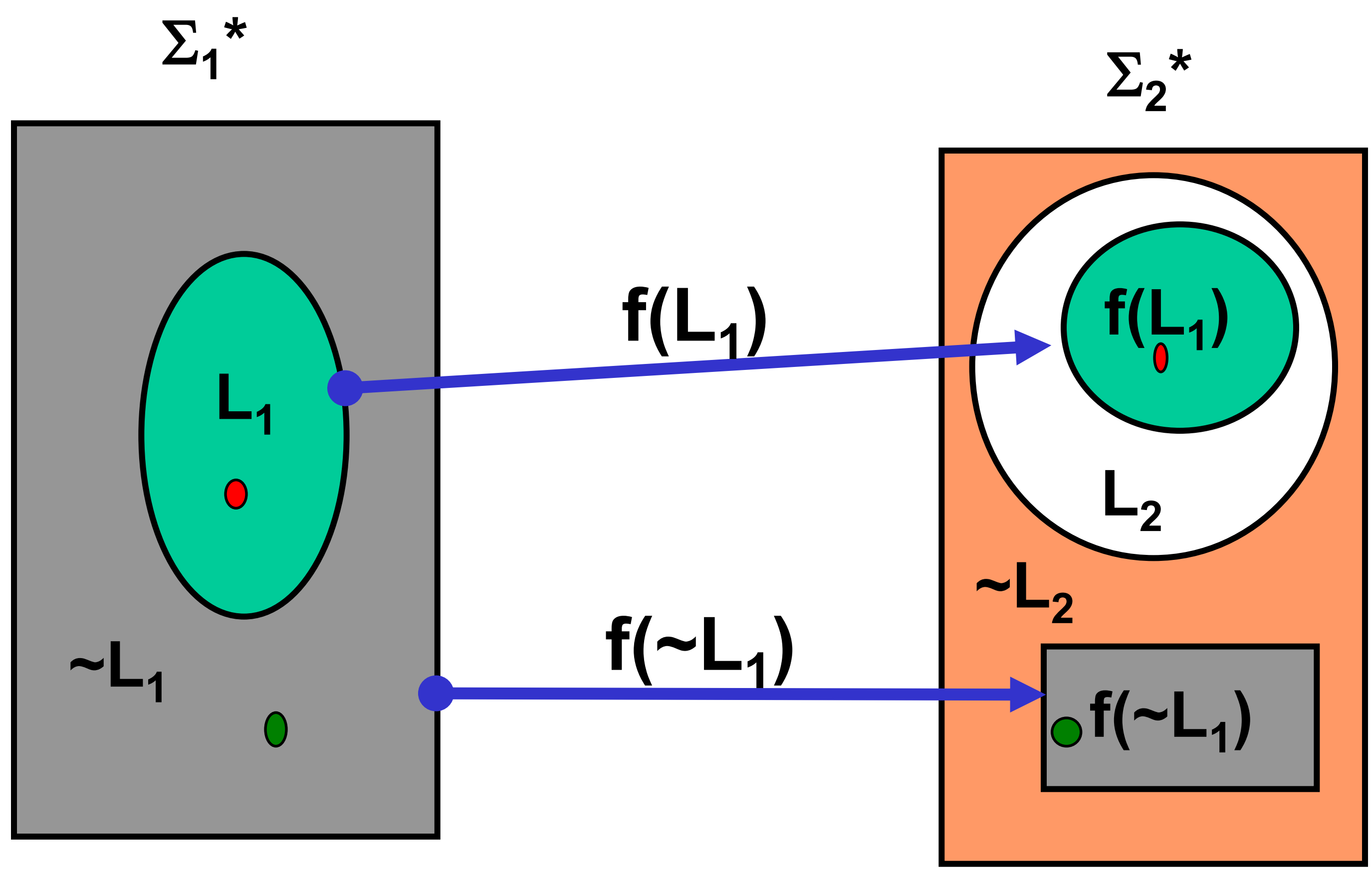
   Ex: h(U') = {x | U'(x) halts, where U' is like UTM U but loops if x is not of form "M" "w"} = { "M" "w" | TM M halts on input w } = H is not recursive. ➔ on input U', the answer is false.
7. **[Halting problem for some fixed TM] Is there a TM M, the halting problem for which is undecidable (i.e., h(M) is not recursive) ?**

pf: (1) has been shown at slide 7; for (7) the UTM U is just one of such machines. (2) ~(6) can be proved by the technique of problem reduction and (1).

Problem Reduction

- L_1, L_2 : two languages over Σ_1 and Σ_2 respectively.
- A reduction from L_1 to L_2 is a computable function $f : \Sigma_1^* \to \Sigma_2^*$ s.t. $f(L_1) \subseteq L_2$ and $f(\sim L_2) \subseteq \sim L_2$.

I.e., for all string $x \in \Sigma_1^*$, $x \in L_1$ iff $f(x) \in L_2$.



Problem reduction.

- We use $L_1 \angle_f L_2$ to mean f is a reduction from L_1 to L_2.
  We use $L_1 \angle L_2$ to mean there is a reduction from L_1 to L_2.
  If $L_1 \angle L_2$ we say L_1 is reducible to L_2.
- $L_1 \angle L_2$ implies that L_1 is simpler than L_2 in the sense that we can use any program deciding L_2 (as a subroutine) to decide L_1. Hence

1. if L_2 is decidable then so is L_1 and
2. If L_1 is undecidable, then so is L_2.

Theorem: If $L_1 \angle L_2$ and L_2 is decidable(or semidecidable, respectively), then so is L_1.

Pf: Let $L_2 = L(M)$ and T is the TM computing the reduction f from L_1 to L_2.
Now let M* be the machine: on input x

1. call T, save the result f(x) on input tape
2. Call M // let M* accept (or reject) if M accepts (or rejects).

==> M* can decide (or semidecide) L_1. QED

Proof of other undecidable problems

pf of [EHP]: // Note: M halts on 'abc' iff (W_{abc} · M) halts on "".

- Let f be the function s.t.

f(x) = "W_wM" if x = "M" "w"; o/w let f(x) = "N".

where W_w is a TM which always writes the string w on the input tape , go back to left-end and then exits. And let N is a specific constant machine which never halts.

Lemma: f is a reduction from H to EHP. (namely H $\angle_f$ EHP)

pf: 1. f is computable. (OK!)

2. for any input x if x $\in$ H => x = "M""w" for some TM M and input w and M halts on input w

=> W_w M halts on empty input => f(x) = "W_wM" $\in$ EHP.

3. for any input x if x $\notin$ H =>f(x) = "N" or x = "M""w" for someTM M and input w and M(w) does not halt

=> N or W_w M does not halt on empty input => f(x) = "N" or "W_wM" $\notin$ EHP.

Corollary: H $\angle$ EHP and EHP is undecidable.

Example for why f is computable

- input: x = “M” “abcd”

=> f(x) = “R aR bR cR dL LLL M”

=> = “$(n_0,[,n_1,R)$,

$(n_1, \square, n_2, a)$, (n_2, a, n_3, R)

$(n_3, \square, n_4, b)$, (n_4, b, n_5, R)

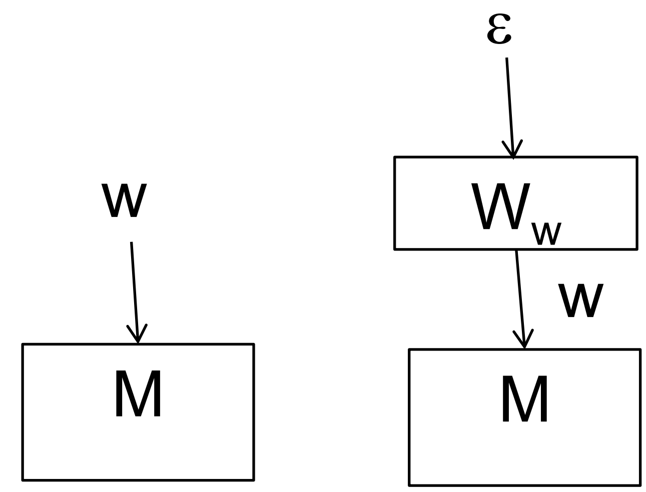
$(n_5, \square, n_6, b)$, (n_6, b, n_7, R)

$(n_7, \square, n_8, d)$, (n_8, d, n_9, L),

(n_9, x, n_9, L), where x = c or b or a

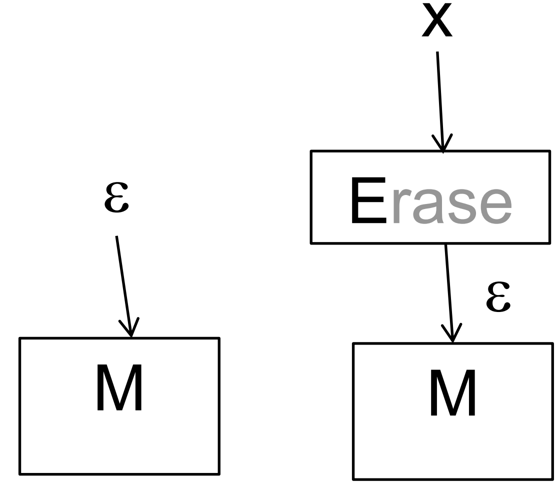
$(n_9, [, s_0, R)$, where s_0 is the initial state of “M”

” + “M“



f("M""w") = "W_w M"

M(w) halts iff W_wM(ε) halts



- EHP $\angle$ (3)={“M” | M halt on some input},
- EHP $\angle$ (4) ={“M” | M halt on all inputs}.
- Let f be the function s.t.for any TM M,

f(“M”) = “ERASE M”, where ERASE is the machine which would erase all its input, and go back to the left end.

f("M") = "E M"
- M(ε) halts iff EM(x) halts for all/some x.

Lemma: f is a reduction from EHP to (3) and(4).

pf: 1. f is computable.

2. for any TM M, “M” $\in$ EHP <=> M halts on empty input <=> (ERASE M) halts on some/all inputs <=> “ERASE M” $\in$ (3),(4). QED

Corollary: (3) and(4) are undecidable.

(5): (4) is reducible to (5). hint: M halts on all inputs iff M and T halt on same inputs, where T is any TM that always halts.

Proof of more undecidable problems

6.1. Given a TM M, Is the set h(M) $=_{def}$ {x | M halts on x} recursive ?

6.2. Given a TM M, Is the set h(M) context free ?

6.3. Given a TM M, Is the set h(M) regular ?

pf: we show that ~EHP (non-halting problem on empty input) ∠ (6.1, 6.2 and 6.3). note: EHP is not recursive implies ~EPH is neither recursive nor r.e..

For any input machine M, construct a new 2-tape machine :

M'(y) = C(y) · M(ε) · U(y). On input y, M'(y) behaves as follows

1. C(y): Move input y from 1st tape to 2nd tape.
2. M(ε): run M as a subroutine with empty input on 1st tape.
3. U(y) : if M halts, then run (1-tape UTM) U with 2nd tape as input.

analysis:

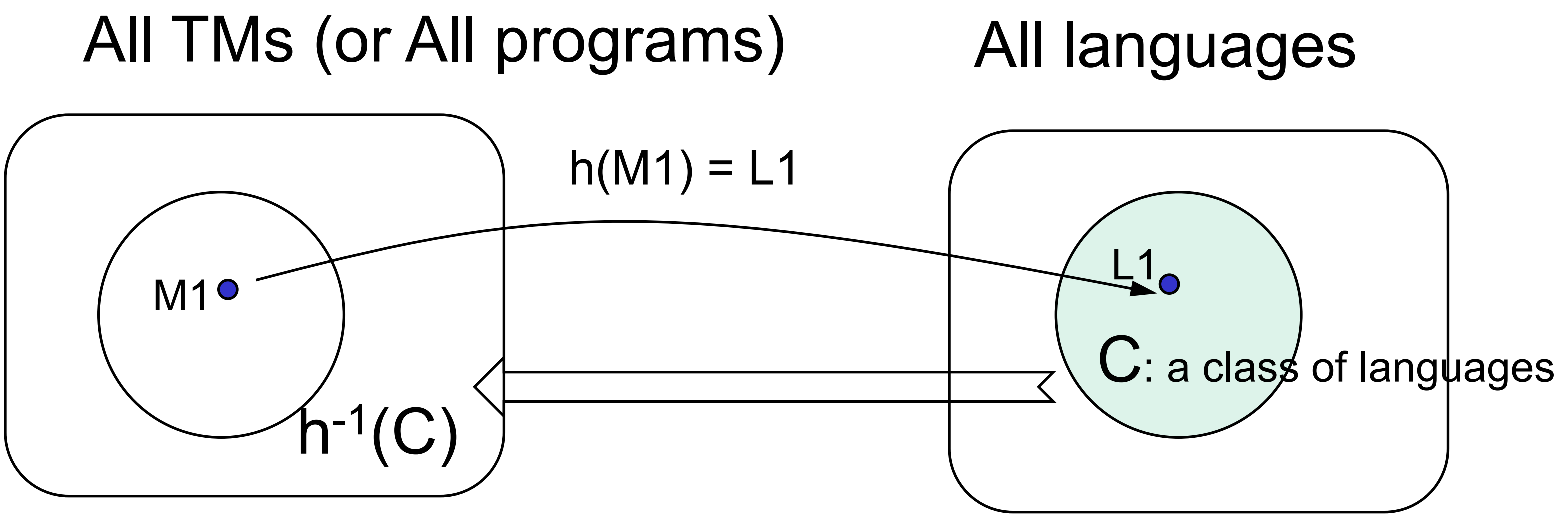
M halts on empty input => M' behaves the same as U => h(M') = h(U) =H is not recursive (and neither context free nor regular).

M loops on empty input => M' loops on all inputs => h(M') = {} is regular (and context-free and recursive as well).

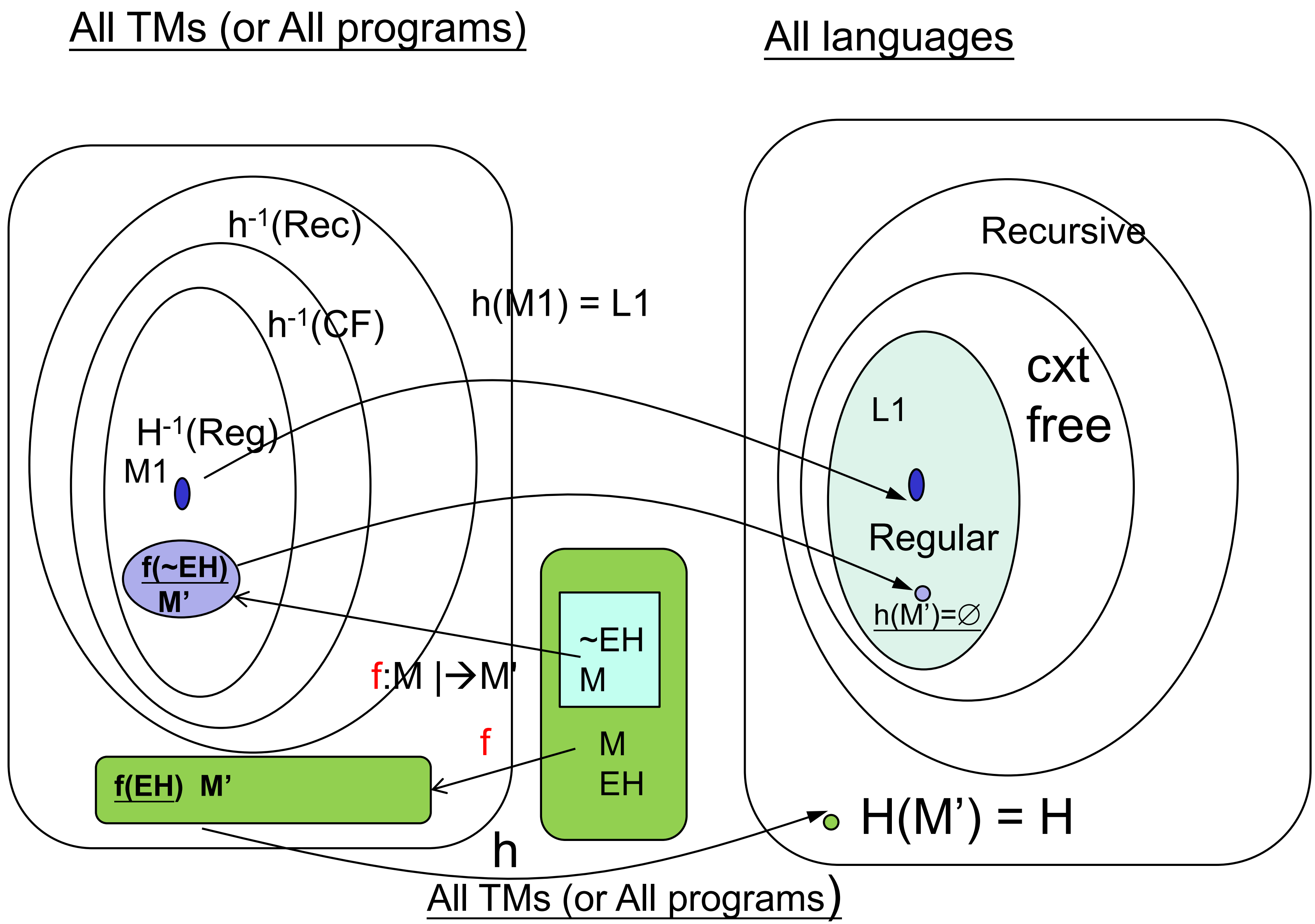
Obviously M' can be computed given M, Hence ~EHP ∠ (6.1, 6.2 and 6.3).

Note: This means all 3 problems are even not semidecidable.

Interesting Problem: Given a class of languages C, is the set $h^{-1}(C)$ decidable ?



The reduction f of NonEmptyHaltingProblem to target Problems



More undecidable problems

- A *semi-Thue system* is a pair $T = (\Sigma, P)$ where
  - Σ is an alphabet and
  - P is a finite set of rules of the form:
  - $\alpha \rightarrow \beta$ where $\alpha \in \Sigma^+$ and $\beta \in \Sigma^*$.
- The derivation relation induced by a semi-Thue system T:

$$\rightarrow_T =_{def} \{ (x\alpha y, x\beta y) \mid x, y \in \Sigma^*, \alpha \rightarrow \beta \in P \}$$

let $\rightarrow^*_T$ be the ref. and trans. closure of $\rightarrow_T$.

- The *word-problem* for semi-Thue system:

Given a semi-Thue system T and two words x and y, determine if $x \rightarrow^*_T y$?

Ex: If $P = \{ 110 \rightarrow 01, 10 \rightarrow 011 \}$, Then $10010 \rightarrow^* 0001$?

Theorem: the word problem for Semi-Thue system (WST) is undecidable. I.e., $WST =_{def} \{ \text{“}(T,x,y)\text{”} \mid x \rightarrow^*_T y \}$ is undecidable.

The undecidability of the word problem for ST system

- We reduce the halting problem H to WST.
  - TM configuration: (q, xy, n) → 'xqy' where |x| = n-1.
  - TM instruction → SemiThue system rules
- Let M be a STM. we construct a semi-Thue system T(M) with alphabet $\Sigma_M = Q \cup \Gamma \cup \{], q_f, q_g \}$. The rule set P_M of T(M) is defined as follows : Where $a,b \in \Gamma$, $p,q \in Q$,
  - $pa \rightarrow qb$ if $\delta(p,a) = (q, b)$,
  - $pa \rightarrow aq$ if $\delta(p,a) = (q, R)$, + $p] \rightarrow \square q]$ if $a = \square$,
  - $bpa \rightarrow qba$ if $\delta(p,a) = (q,L)$
  - $p \rightarrow q_f$ if $p \in \{t, r\}$ // halt =>enter q_f, ready to eliminate all tape symbols left to current position.
  - $xq_f \rightarrow q_f$ where $x \in \Gamma \setminus \{ [\}$
  - $q_f[\rightarrow [q_f$
  - $[q_f \rightarrow [q_g$, // ready to eliminate all remaining tape symbols
  - $[q_g x \rightarrow [q_g$ where $x \in \Gamma \setminus \{] \}$.

WST is undecidable.

- **Lemma:** $(s, [x, 0)$ $|\!-^*_M (h,y,n)$ iff $s[x] \rightarrow^*_{T(M)} [q_g]$.

Sol: note: h means t or r. Let $(s, [x, 0) = C_0 \,|\!-_M C_1 \,|\!- \ldots |\!-_M C_m = (h,y,n)$ be the derivation.

consider the mapping: $f((p, [z, i)) = [z_{1..i-1}\ p\ z_{i..|z|}]$,

we have $C \,|\!-_M D \Leftrightarrow f(C) \rightarrow_{T(M)} f(D)$ for all configuration C,D (**).

This can be proved by the definition of T(M).

Hence $f(C_0) = s[x] \rightarrow^*_{T(M)} f(C_m) = [y_{1..n-1}\ h\ y_{n..|y|}]$

$\rightarrow^* [y_{1..n-1}\ q_f\ y_{n..|y|}] \rightarrow^* [q_f\ y_{n..|y|}]$

$\rightarrow^* [q_g\ y_{n..|y|}] \qquad \rightarrow^* [q_g]$

Conversely, if $s[x] \rightarrow^* [q_g]$, there must exist intermediate cfgs s.t. $s[x] \rightarrow^* [yhz] \rightarrow^* [yq_fz] \rightarrow^* [q_gz] \rightarrow^* [q_g]$.

Hence $(s, [x, 0)$ $|\!-^*_M (h,yz,|y|)$ and M halts on input x.

Word problem for special SemiThue systems.

Corollary: H is reduciable to WST.

Sol: for any TM M and input x, M halts on x iff (s, [x, 0) $|\!-^*_M$ (h,y ,n) for some y,n iff s[x] $\rightarrow_{T(M)}$ [q_g].

I.e., "(M,x)" ∈ H iff "(T(M), s[x],[q_g])" ∈ WST, for all TM M and input x. Hence H is reducible to WST.

Theorem: WST is undecidable.

[word problem for semi-Thue system T]: Given a semi-Thue system T, the word problem for T is the problem of, given two input strings x, y, determining if x $\rightarrow^*_T$ y.

Define WST(T) = { (x,y) | x $\rightarrow^*_T$ y }

Theorem: Let M be any TM. If the halting problem for M is undecidable, then WST(T(M)) is undecidable.

Pf: since H(M) (Halting Problem for M) is reducible to WST(T(M)).

Corollary: There are semi-Thue systems whose word problem are undeciable. In particular, WST(T(UTM)) is undeciable.

The PCP Problem

- **[Post correspondence system]**

Let $C = [(x_1,y_1),\ldots,(x_k,y_k)]$: a finite list of pairs of strings over Σ

A sequence j1,j2,...jn of numbers in [1,k] (n > 0) is called a solution index (and $x_{j1}\ x_{j2} \ldots x_{jn}$ a solution) of the system C iff $x_{j1}\ x_{j2} \ldots x_{jn} = y_{j1}\ y_{j2} \ldots y_{jn}$.

Ex: Let $\Sigma = \{0,1\}$ and C= [(1,101),(10,00),(011,11)]. Does C has any solution?

Ans: yes. the sequence 1 3 2 3 is a solution index

since $x_1\ x_3\ x_2\ x_3 = 1\ 011\ 10\ \ 011 = 101\ 11\ 00\ 11 = y_1\ y_3\ y_2\ y_3$.

The PCP Problem

- [Post correspondence problem:] Given any correspondence system C, determine if C has a solution ?

I.e. PCP $=_{def}$ {"C" | C = $[(x_1,y_1),\ldots,(x_k,y_k)]$, $k > 0$, is a list of pairs of strings and C has a solution. }

Theorem: PCP is undecidable.

pf: Since the word problem WST(T) of some particular undecidable semi-Thue system T is reducible to PCP.

Undecidability of the PCP Problem

- Let T = (Σ , P) be a semi-Thue system with alphabet {0,1} whose word problem is undecidable.
- For each pair of string $x, y \in \Sigma^*$, we construct a PCS C(x,y) as follows:
  - C(x,y) has alphabet $\Sigma = \{0, 1, *, \underline{0}, \underline{1}, \underline{*}, [,]\}$
  - if $z = a_1a_2\ldots a_k$ is a string over {0,1,*}, then let $\underline{z}$ denote $\underline{a_1a_2\ldots a_k}$.
  - wlog, let $0 \rightarrow 0$, $1 \rightarrow 1 \in P$.
  - $C(x,y) = \{ \ (\alpha,\underline{\beta}),(\underline{\alpha},\beta) \ \mid \alpha \rightarrow \beta \in P\} \ \cup$
  - $\{ \ ([x^*, [), (\underline{*}, *), (*,\underline{*}), (], \underline{*}y]) \ \}$

$\underline{0}$	0	$\underline{1}$	1	$\underline{*}$	*	$\underline{\alpha}$	α	…	[x*	]	
0	$\underline{0}$	1	$\underline{1}$	*	$\underline{*}$	β	$\underline{\beta}$	…	[	$\underline{*}$y]	

Example

Ex: Let T = { 11 → 1, 00→0, 10 → 01 }

Problem x = 1100 $\rightarrow^*_T$ y = 01 ?

Then

$C(x,y) = \{ (11, \underline{1}), (\underline{11}, 1), (\underline{00},0), (00, \underline{0}), (\underline{10},01)\ (10,\underline{01})$

$(\underline{1},1), (1,\underline{1}),(0,\underline{0}),(\underline{0},0), (*,\underline{*}), (\underline{*},*) \} \quad U$

$([1100*, [), (], \underline{*}01] \ \}$

Derivation vs solution

The derivation : 1100 $\rightarrow$ 100 $\rightarrow$ 10 $\rightarrow$ 01 $\rightarrow$ 01 can be used to get a solution and vice versa.

[1100*	1 0 0 *	1 0 *	01 *	01	]						
[	11 0 0 *	1 00 *	10 *	01	*01]						

Def: u,v : two strings. We say u matches v if there are common index sequence j1,j2,…,jm(m > 0) s.t. $u = y_{j1}y_{j2}\ldots y_{jm}$ and $v = x_{j1}x_{j2}\ldots x_{jm}$

Facts :Let x and y be any bit strings, then

1. $x \rightarrow_T y$ implies x* matches y* and x* matches y*,
2. x* matches y* (or x* matches y*) implies $x \rightarrow^*_T y$.

Theorem: $x \rightarrow^*_T y$ iff C(x,y) has a solution

Corollary: PCP is undecidable.

Theorem: $x \rightarrow^*_T y$ iff C(x,y) has a solution

- Only-if part: a direct result of the following lemma:
- Lemma1: If $x = x_0 \rightarrow x_1 \rightarrow x_2 \ldots \rightarrow x_{2k} = y$ is a derivation of y from x, then

$$\alpha = [x^* \underline{x_1}\underline{*}\ x_2 * \underline{x_3}\underline{*} \ldots \underline{x_{2k-1}} * x_{2k}]$$

is a solution of C(x,y).

pf: The arrange of α as a sequence of strings from the 1st (and 2nd) components of C(x,y) is given as follows:

$[x_0{}^*$	$\underline{x_1}\ \underline{*}$	$x_2\ *$	$\underline{x_3}\ \underline{*}$	....	$\underline{x_{2k-1}}\underline{*}$	x_{2k}	]
[	copy $x_0\ *$	$\underline{x_1}\ \underline{*}$	$x_2\ *$	......	$x_{2k-2}{}^*$	$\underline{x_{2k-1}}$	$\underline{*}\ x_{2k}\underline{\ }]$

- Note since $x_j \rightarrow x_{j+1}$, by previous facts, $(\underline{x_{j+1}}\underline{*}, x_j{}^*)$ and $(x_{j+1}{}^*, \underline{x_j}\underline{*})$ match (corresponding to the same index).
- It is thus easy to verify that both sequences correspond to the same solution index, and α hence is a solution.

Theorem: $x \rightarrow^*_T y$ iff C(x,y) has a solution

- if-part: Let a solution U of C(x,y) be arranged as follows:

U	$[x_0*$ derive	$\underline{x}_1\ \underline{*}$ copy	$x_2\ *$	$\underline{x}_3\ \underline{*}$	…….	$\underline{x}_{2k-1}\underline{*}$	x_{2k}	]
V	[	$x_0\ *$ match	$\underline{x}_1\ \underline{*}$	$x_2\ *$	……	$x_{2k-2}*$	$\underline{x}_{2k-1}$	$\underline{*}\ x_{2k}]$

- Then (U,V) must begin with:
  ▯ ([x* , [ ) and must end with ( ] , $\underline{*}$y])
- => the solution must be of the form: [x* w $\underline{*}$y]

if w contains] => can be rewritten as [x*z$\underline{*}$y] v$\underline{*}$y] ==> U = [x*z$\underline{*}$y] = V is also a solu.

⇒ To equal [x* , V must begin with [x*

⇒ To match x* , U must proceeds with [x* $\underline{x}_1\underline{*}$ ➔ $x=x_0 \rightarrow x_1$ (since x match $\underline{x}_1$)

To equal [x* $\underline{x}_1\underline{*}$,V must proceeds with [x* $\underline{x}_1\underline{*}$

To match [x* $\underline{x}_1\underline{*}$,U must proceeds with [x* $\underline{x}_1\underline{*}$ x_2* ➔ $x_1 \rightarrow X_2$

⇒ … ➔ $x=x_0 \rightarrow x_1 \rightarrow \ldots \rightarrow x_{k-1}$ with U = [x*… $x_{2k-2}*$ $x_{2k-1}*$ and V = [x*… $x_{2k-2}*$

To equal U = [x*… $x_{2k-2}*$ $x_{2k-1}*$,V proceeds with [x*… $x_{2k-2}*$ x_{2k-1}

To match V = [x*… $x_{2k-2}*$ x_{2k-1},U must proceeds with U = [x*… $x_{2k-2}*$ x_{2k-1} $*x_{2k}$

⇒ $x \rightarrow^* y$ finally to close the game V proceeds with [x*… $x_{2k-2}*$ x_{2k-1}*x2k] and U proceeds with [x*… $x_{2k-2}*$ $x_{2k-1}*x_{2k}$]

Decidable and undecidable problems about CFLs

- The empty CFG problem:

  Input: a CFG $G = (N, \Sigma, P, S)$

  Output: "yes" if L(G) = {}; "no" if L(G) is not empty.
- There exists efficient algorithm to solve this problem.

  Alg empty-CFG(G)

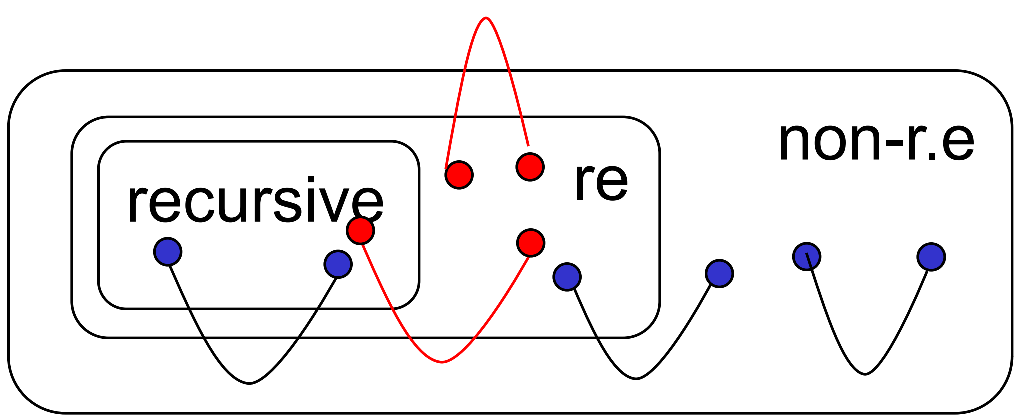
  1. Mark all symbols in Σ.
  2. Repeat until there in no new (nonterminal) symbols marked

     for each rule $A \to X_1 X_2 \dots X_m$ in P do

     if ALL X_i's are marked then mark A
  3. If S is not marked then return("yes") else return("no").
- The alg can be implemented to run in time $O(n^2)$.
- Similar problems existing efficient algorithms:
  - 1. L(G) is infinite   2. L(G) is finite
  - 3. L(G) contains ε (or any specific string)
  - 4. Membership (if a given input string x in L(G))

Undecidable problems about CFL

- But how about the problem:
  - Whether $L(G) = \Sigma^*$, the universal language ?
- Relations between L, ~L and their recursiveness
  - If L is recursive, ~L is recursive.
  - If L and ~L are r.e., then both are recursive.
  - If L is r.e. but not recursive, then ~L is not r.e.
  - If L is not r.e., then ~L is not recursive(but may be r.e.).



Undecidable problems for CFGs

- **Theorem** : there is no algorithm that can determine whether the languages of two CFGs are not disjoint (overlap).

(i.e., the set NDCFG = { "(G1,G2)" | G1 and G2 are CFGs and $L(G1) \cap L(G2)$ is not empty } is undecidable (but it is r.e (why ?) => its complementation is not r.e.).

Pf: Reduce PCP to NDCFG.

Let $C = (\Sigma_c, \{ (x_1,y_1), \ldots, (x_n,y_n) \})$ be a PCS.

Let $G_z = (N_z, \Sigma_z, S_z, P_z)$, where z = x or y,

- $N_z = \{S_z\}$ $\quad\quad$ $\Sigma_z = \Sigma_c \cup \{1,2,\ldots, n\}$
- $P_z = \{ S_z \rightarrow z_i\ S_z\ i \quad , \quad S_z \rightarrow z_i\ i \mid i = 1..n \}$

Lemma1: $S_z \rightarrow^* w$ iff there is seq $j_k \ldots j_1$ in $[1,n]^*$ with $w = z_{j1}\ z_{j2} \ldots z_{jk}\ j_k\ j_{k-1} \ldots j_1$.

Lemma2: C has a solution iff $L(G_x) \cap L(G_y) \neq \varnothing$

pf: C has a solution w' iff $w' = x_{j1}\ x_{j2} \ldots x_{jk} = y_{j1}\ y_{j2} \ldots y_{jk}$ for some $j_1\ j_2 \ldots j_k$ iff $S_x \rightarrow^* w'\ j_k\ j_{k-1} \ldots j_1$. and $S_y \rightarrow^* w'\ j_k\ j_{k-1} \ldots j_1$ iff $L(G_x) \cap L(G_y) \neq \varnothing$

corollary: NDCFG is undecidable.

pf: since PCP is reducible to NDCFG.

Example

Ex: Let Σ = {a,b} and C= [(a,aba),(ab,bb),(baa,aa)].

$\Rightarrow$ G_x : $S_x \rightarrow$ a1 | ab2 | baa3

$\Rightarrow$ | a S_x 1 | ab S_x 2 | baa S_x 3

$\Rightarrow$ G_y : $S_y \rightarrow$ aba 1 | bb 2 | aa 3

$\Rightarrow$ | aba S_y 1 | bb S_y 2 | aa S_y 3

L(G_x) and L(G_y) has a common member

$\Rightarrow$ a baa ab baa 3231 (Gx).

$\Rightarrow$ aba aa bb aa 3231 (Gy)

Whether $L(G) = \Sigma^*$ is undecidable for CFGs

- The set UCFG = { "G" | G is a CFG and $L(G) = \Sigma^*$ } is undecidable.

Pf: 1. For the previous grammar G_x and G_y, it can be shown that $\sim L(G_x)$ and $\sim L(G_y)$ are both context-free languages.

Hence the set $A =_{def} \sim(L(G_x) \cap L(G_y)) = \sim L(G_x) \cup \sim L(G_y)$ is context-free. Now let G_C be the CFG for A.

By previous lemma :

C has no solution iff $L(G_x) \cap L(G_y) = \varnothing = \sim A$

iff $A = \Sigma^*$

iff $G_C \in$ UCFG.

Hence any program deciding UCFG could be used to decide ~PCP,

but we know ~PCP is undecidable (indeed not r.e.), UCFG thus is undecidable (not r.e.).

~L(Gz) is context-free

- $\alpha \notin L(G_z)$ iff
  1. α is not of the form $\Sigma_z^+ \{1,\ldots,n\}^+$ (I.e., $\alpha \in \sim \Sigma_z^+ \{1,\ldots,n\}^+$) or
  2. α is one of the form: where $k > 0$,

  2.1 $z_{jk} \ldots z_{j1}\ j_1\ j_2 \ldots j_k \{1,\ldots,n\}^+$ or

  2.2 $\Sigma_c^+ z_{jk} \ldots z_{j1}\ j_1\ j_2 \ldots j_k$ or

  2.3 $\sim(\Sigma_c^* z_{jk})\ z_{k-1} \ldots z_{j1}\ j_1\ j_2 \ldots j_{k-1}\ j_k \{1,\ldots,n\}^*$

2.1 : $G_1 : S_1 \rightarrow S_z A$; $A \rightarrow 1 \mid 2 \ldots \mid n \mid 1A \mid 2A \mid \ldots \mid nA$

2.2 : $G_2 : S_2 \rightarrow B S_z$; $B \rightarrow a \mid b \ldots \mid Ba \mid Bb \mid \ldots$

2.3 : $G_3 : S_3 \rightarrow N_k S_z k A' \mid N_k k A'$ for all $k = 1..n$, where

- N_k is the start symbol of the linear grammar for the reg expr $\sim(\Sigma_c^* z_{jk})$,
- $A' \rightarrow A \mid \varepsilon$

Ambiguity of CFGs is undecidable

- The set AMBCFG = {"G" | G is an ambiguous CFG } is undecidable.

Pf: reduce PCP to AMBCFG.

Let G_x, G_y be the two grammars as given previously.

let G be the CFG with

- $N = \{S, S_x, S_y\}$,
- $S_G = S$
- $P = P_x \cup P_y \cup \{S \rightarrow S_x, S \rightarrow S_y\}$

Lemma: C has a solution iff $L(G_x)$ and $L(G_y)$ are not disjoint iff G is ambiguous.

pf: 1. C has a solution w

=> $w = x_{j1}\, x_{j2} \ldots x_{jk} = y_{j1}\, y_{j2} \ldots y_{jk}$ for some $J = j_k\, j_{k-1} \ldots j_1$

=> $S \to S_x \to^* wJ$ and $S \to S_y \to^* wJ$

=> G is ambiguous.

2. G ambiguous

=> there exist two distinct derivations Δ_1 and Δ_2 for a certain string α=wJ => Δ_1 and Δ_2 must have distinct 1st steps (since G_x and G_y are deterministic)

=> C has a solution w with solution index J.

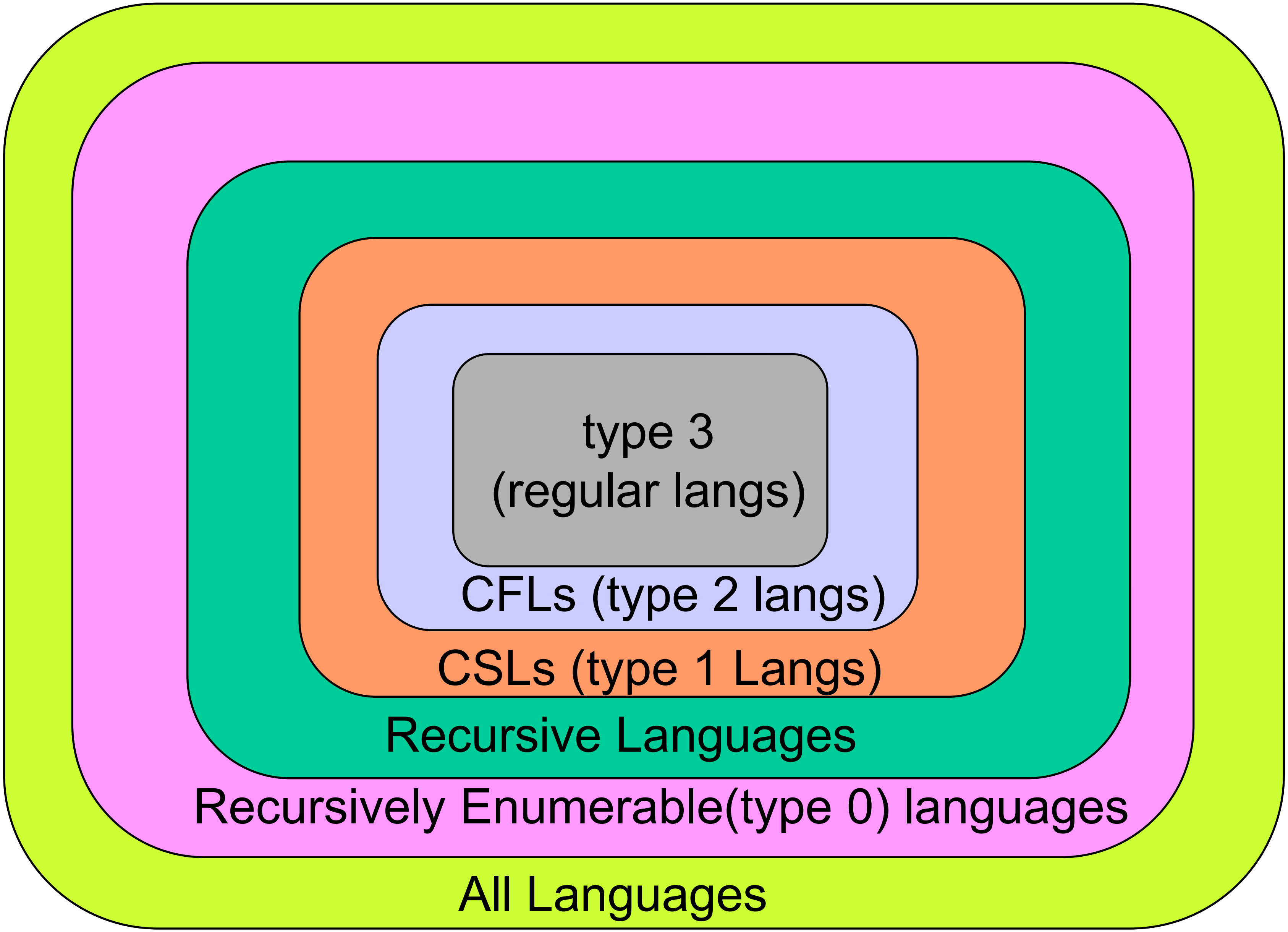
Corollary: AMBCFG is undecidable.

The Chomsky Hierarchy

- Relationship of Languages, Grammars and machines

Language	recognition model	generation model
Regular languages (type 3) languages	Finite automata (DFA, NFA)	regular expressions linear grammars
CFL (type 2, Context Free) languages	Pushdown automata	CFG ; type 2 (context free) grammars
CSL (type 1, Context sensitive) Languages	LBA (Linear Bound Automata)	CSG (Context sensitive, type 1 Grammars)
Recursive Languages	Total Turing machines	-
R.E. (Recursively enumerative, type 0) Languages	Turing machines	GPSG(type 0, general phrase-structure, unrestricted) grammar

The Chomsky Hierarchy



Phrase-structure grammar

Def.: A phrase-structure grammar G is a tuple $G=(N, \Sigma, S, P)$ where

- N, Σ, and S are the same as for CFG, and
- P, a finite subset of $(N\cup\Sigma)^* N (N\cup\Sigma)^* \times (N\cup\Sigma)^*$, is a set of production rules of the form:
- $\alpha \rightarrow \beta$ where
- $\alpha \in (N\cup\Sigma)^* N (N\cup\Sigma)^*$ is a string over $(N\cup\Sigma)^*$ containing at least on nonterminal.
- $\beta \in (N\cup\Sigma)^*$ is a string over $(N\cup\Sigma)^*$.

Def: G is of type

- 2 => $\alpha \in N$.
- 3 (right linear)=> $A \rightarrow a B$ or $A \rightarrow a$ ($a \neq \varepsilon$) or $S \rightarrow \varepsilon$.
- 1 => $S \rightarrow \varepsilon$ or $|\alpha| \leq |\beta|$.

Derivations

- Derivation $\rightarrow_G \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ is the least set of pairs such that :

  $\forall$ x,y $\in (\Sigma \cup N)^*$, $\alpha \rightarrow \beta \in P$, $x\alpha y \rightarrow_G x\beta y$.

- Let $\rightarrow^*_G$ be the ref. and tran. closure of $\rightarrow_G$.
- L(G) : the languages generated by grammar G is the set:

$$L(G) =_{def} \{x \in \Sigma^* \mid S \rightarrow^*_G x\}$$

Example

- Design CSG to generate the language $L=\{0^n1^n2^n \mid n \geq 0\}$, which is known to be not context free.

Sol:

Consider the CSG G_1 with the following productions:

$S \rightarrow \varepsilon$, $S \rightarrow 0SA2$ $2A \rightarrow A2$,

$0A \rightarrow 01$ $1A \rightarrow 11$

For G_1 we have

$S \rightarrow 0SAB \rightarrow \ldots \rightarrow 0^k(A2)^k \rightarrow^* 0^kA^k2^k \rightarrow 0^k1^k2^k \therefore L \subseteq L(G1)$.

Also note that

- if $S \rightarrow^* \alpha \Longrightarrow \#0(\alpha) = \#(A|1)(\alpha) = \#(2)(\alpha)$.
- if $S \rightarrow^* \alpha \in \{0,1,2\}^* \Rightarrow \alpha_k = 0 \Longrightarrow \alpha_j = 0$ for all $j < k$.
- $\alpha_k = 1 \Rightarrow \alpha_j = 1$ or 0 for all $j < k$.
- Hence α must be of the form $0^*1^*2^* \Rightarrow \alpha \in L$. QED

- **Lemma 1 : if $S \rightarrow^* \alpha \in \Sigma^*$, then it must be the case that**

$S \rightarrow^* 0^* S (A + 2)^* \rightarrow 0^* (A+2)^* \rightarrow^* 0^*1 (A+2)^* \rightarrow^* 0^*1^*2^*$.