IF5110 Teori Komputasi

4. Undecidabality (Bagian 1)

Oleh: Rinaldi Munir

Program Studi Magister Informatika STEI-ITB

Makna *Undecidabality*

- Persoalan keputusan (*decision problem*): persoalan yang jawabannya hanya salah satu dari dua: "ya" (*yes*) atau "tidak" (*no*), atau ekivalen dengan 1 atau 0 (*true* atau *false*).

- Contoh: Diberikan masukan sebuah bilangan bulat *n*. Apakah *n* bilangan prima? Jawabanya: ya (jika *n* prima) atau tidak (jika *n* bukan prima)

- Persoalan keputusan (*problem*) yang tidak dapat diselesaikan (*solved*) oleh komputer disebut *undecidable*.

  undecidable → tidak dapat diputuskan jawabannya "ya" atau "tidak".

- Sebaliknya persoalan keputusan disebuat *decidable* jika terdapat algoritma yang apabila diberikan instansiasi (*instance*) persoalan tersebut maka akan memberikan jawaban "ya" atau "tidak".

- Contoh: diberikan program *Hello World* yang diperkenalkan oleh Brian Cernighan berikut ini.

```
main()
{
   printf(“Hello, world!\n”);
}
```

- Tetapi ini bukan satu-satunya program *Hello World* . Ada banyak program lain yang juga mencetak “Hello world!” seperti di bawah ini:

```
 main()
{ int X;
  scanf(”%d”, &X);
  while (X != 1)
  { if (X % 2 == 0)    /* X genap */
        X = X / 2
    else
        X = 3*X + 1;
  }
  if (X == 1)
     printf(“Hello, world!\n”);
}
```

Apakah program mencetak “Hello, world!” untuk semua nilai X?

```
int exp(int i, n)
/* computes i to the power n */
{
    int ans, j;
    ans = 1;
    for (j=1; j<=n; j++) ans *= i;
    return(ans);
}

main ()
{
    int n, total, x, y, z;
    scanf("%d", &n);
    total = 3;
    while (1) {
        for (x=1; x<=total-2; x++)
            for (y=1; y<=total-x-1; y++) {
                z = total - x - y;
                if (exp(x,n) + exp(y,n) == exp(z,n))
                    printf("hello, world\n");
            }
        total++;
    }
}
```

-Program menerima input *n*
- Lalu mencari solusi *integer* dari persamaan $x^n + y^n = z^n$.
- Jika ditemukan, maka cetak *"Hello, world!"*

-Jika tidak pernah menemukan *x*, *y*, dan *z* yang memenuhi, ia akan terus *looping* dan tidak pernah mencetak "*Hello, world!*"

- Untuk *n* = 2, solusinya ada yaitu *x* = 3, *y* = 4, dan *z* = 5.
- Jadi untuk *n* = 2 program mencetak "Hello, world!"
- Tetapi untuk *n* > 2 program tidak pernah menemukan *x*, *y*, dan *z* yang memenuhi, sehingga gagal mencetak "*Hello, world!*"

- Apakah ketiga program *Hello world* di atas selalu mencetak string “Hello, world!” untuk setiap *input* yang diberikan?

- **Persoalan *Hello-world* (*Hello-world- problem*)**:

  Diberikan sebuah program *P* beserta masukan *I* (jika ada). Tentukan apakah program *P* mencetak string “Hello, world!” sebagai luarannya?

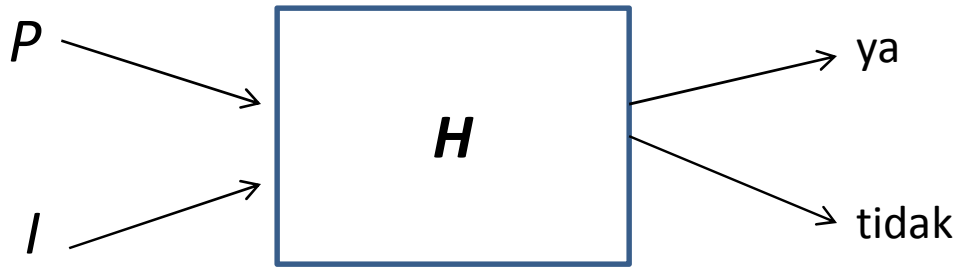
  → *decision problem*

Jawaban :
- ya, jika *P* mencetak “Hello, world!’
- tidak, jika *P* tidak mencetak “Hello, world!”

- Akan dibuktikan bahwa tidak ada algoritma untuk memecahkan persoalan *Hello-world.*

Hello-world Tester

- Untuk membuktikan bahwa tidak ada algoritma untuk memecahkan persoalan *Hello-world*, kita mengasumsikan terdapat program *H* sebagai *Hello-world tester*.
- *Hello-world tester* memeriksa apakah program *P* beserta data masukan *I* mencetak string "Hello, world!".
- Misalkan program *tester* tersebut adalah *H*. Program *H* menerima masukan berupa *P* dan *I*, dan memberikan luaran berupa string "ya" atau "tidak".



Gambar 1. Program hipotetik H sebagai *Hello-world tester*

- Jika sebuah persoalan memiliki algoritma penyelesaian seperti *H*, maka algoritma tersebut selalu dengan tepat memberikan jawaban "ya" atau "tidak" apapun instansiasi persoalan yang diberikan. Jika demikian maka kita katakan persoalan tersebut *decidable*.

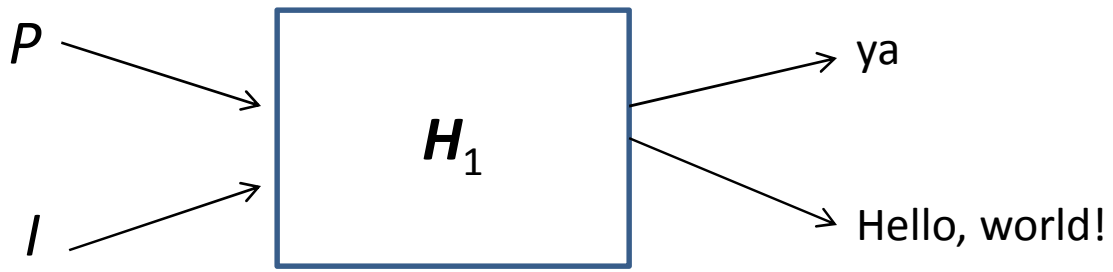
  Sebaliknya disebut *undecidable*.

- Akan dibuktikan bahwa *H* untuk persoalan *Hello-world* tidak pernah ada, yaitu persoalan *Hello-world* adalah *undecidable*.

- Pembuktian bahwa *H* tidak ada dilakukan dengan cara kontradiksi, yaitu mengasumsikan persoalan *Hello-world* adalah *decidable*.

- Kontradiksi dilakukan dengan membuat modifikasi pada *H*. Modifikasi itu mudah dilakukan, karena jika *H* ditulis dalam Bahasa *C* maka modifikasinya hanya dengan mengubah *statement* di dalam program *H*.

- Asumsikan bahwa *H* ada.
- Sekarang, modifikasi *H* menjadi H_1 yang memiliki perilaku seperti *H*, tetapi H_1 tidak mencetak *string* "tidak" sebagai luarannya, melainkan mencetak "Hello, world!".



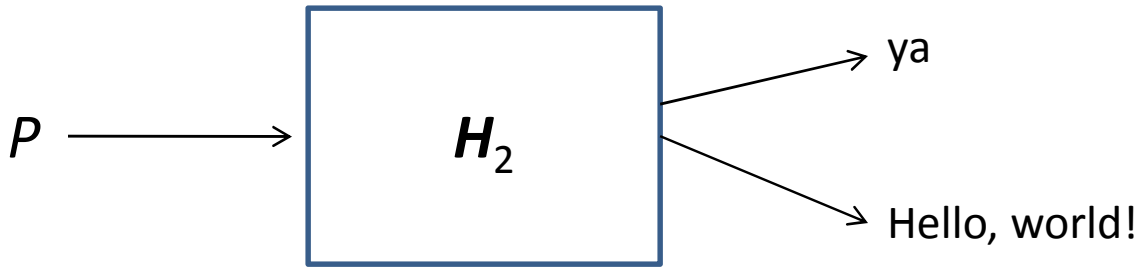
Gambar 2. H_1 berlaku seperti *H*, tetapi H_1 mencetak "Hello, world!" ketika *H* mencetak "tidak"

- Selanjutnya, modifikasi H_1 menjadi H_2.

  - H_2 hanya menerima satu masukan yaitu P.

  - H_2 memiliki perilaku seperti H_1 tetapi baik program dan data masukannya keduanya sama dengan P.

  Dengan kata lain, $H_2(P) = H_1(P, P)$



Gambar 3. H_2 berlaku seperti H_1, tetapi masukannya adalah P sebagai P sekaligus I

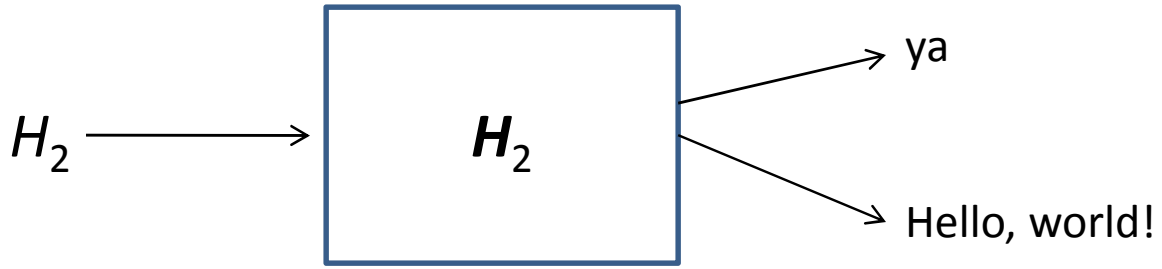
- Untuk memodifikasi H_1 menjadi H_2, caranya adalah sebagai berikut:

  1. H_2 mula-mula membaca seluruh kode program *P* dan menyimpannya di dalam larik (*array*) *A*.
  2. H_2 kemudian mensimulasikan H_1, tetapi ketika H_1 akan membaca *input* dari *P* (yaitu *I*), maka langkahnya diganti dengan membaca string yang sebelumnya disimpan di dalam larik *A*.

- Perhatikan Gambar 3, bahwa H_2, bila diberikan program *P* sebagai masukannya, akan menghasilkan luaran "ya" jika *P* mencetak "Hello, world!" bila masukannya adalah *string* dirinya sendiri.

- Selain itu, H_2 akan mencetak "Hello, world" jika *P* (yang menerima masukan dirinya sendiri) tidak mencetak "Hello, world!" sebagai luarannya.

- Akan ditunjukkan bahwa H_2 tidak ada, akibatnya H_1 tidak ada, dan kesimpulannya H juga tidak ada.
- Kunci pembuktiannya adalah dengan membayangkan apa yang dilakukan H_2 jika masukan yang diberikan adalah dirinya sendiri (Gambar 4).
- Dengan kata lain, apa yang dilakukan oleh $H_2(H_2)$?



Gambar 4. Apa yang dilakukan H_2 bila menerima masukan adalah dirinya sendiri?

1) Jika $H_2(H_2)$ = "ya" maka apabila H_2 diberikan H_2 sebagai data masukannya maka ia tidak mencetak "Hello, world!".

Namun, $H_2(H_2) = H_1(H_2, H_2) = H(H_2, H_2)$, dan H mencetak "ya" jika dan hanya jika masukan pertamanya, yaitu H_2 (yang menerima masukan kedua sebagai data), mencetak "Hello, world!".

Jadi, $H_2(H_2)$ = "ya" mengimplikasikan $H_2(H_2)$ = "Hello, world!"

2) Tetapi, jika $H_2(H_2)$ = "Hello, world!" , maka $H_1(H_2, H_2)$ = "Hello, world!" dan $H(H_2, H_2)$ = "tidak". (maksudnya, H mencetak "tidak" jika dan hanya jika masukan pertamanya, yaitu H_2 (yang menerima masukan kedua sebagai data), tidak mencetak "Hello, word!" (berarti mencetak "ya")

Jadi, $H_2(H_2)$ = "Hello, world!" mengimplikasikan $H_2(H_2)$ = "ya"

- Kedua situasi ini paradoks, sehingga disimpulkan H_2 tidak ada, akibatnya H_1 tidak ada, dan kesimpulannya H juga tidak ada.

- Dengan kata lain, tidak ada algoritma untuk menentukan apakah program *P* dengan masukan *I* mencetak "Hello, world!" sebagai luarannya. Persoalan *Hello-world* adalah *undecidable*.

Bahasa vs. Persoalan

- Ingatlah kembali bahwa bahasa adalah himpunan *string*

  ***Bahasa** (pada alfabet A) adalah himpunan bagian dari A^*.*

- Sebuah instans persoalan (*problem*) dapat diekspresikan sebagai himpunan string **<input,output>.**

- Misalnya, persoalan penjumlahan (a+b) ekivalen dengan himpunan string yang berbentuk {a1b11a+b}. Integer a dan b dinyatakan sebagai 0^a dan 0^b, sedangkan a+b dinyatakan sebagai 0^{a+b}

  Contoh: {0001001100000, 00000100011000000}

  Penjelasan: 0001001100000" artinya apakah 3 + 2 = 5?

  Jawaban untuk persoalan ini adalah YA atau TIDAK

  Himpunan string-string tersebut membentuk bahasa pada alfabet biner

- **Jadi, setiap persoalan berkoresponden dengan bahasa.**

- Contoh lain: persoalan *hello-world* (yang dibahas pada bagian awal kuliah) dapat direpresentasikan sebagai *string* yang terdiri dari kode program *hello-world* (dalam Bahasa C) diikuti dengan satu atau lebih masukan (jika ada).

  Himpunan string tersebut membentuk bahasa pada alfabet yang tediri dari karakter-karakter ASCII.

  {main(){printf(“Hello, world!\n”);}##’Hello,world!}

  Instans persoalan di atas dibaca: apakah kode-kode program C tersebut memberikan luaran “Hello,world!” sebagai luaran pertamanya?

  Jawaban untuk persoalan ini adalah YA atau TIDAK.

Dua Tipe Mesin Turing

- Terdapat dua tipe Mesin Turing (MT) didasarkan pada apakah ia berhenti atau tidak setelah diberikan input string:
  1. Mesin Turing yang selalu berhenti untuk setiap *input* string yang diberikan, tanpa memperhatikan apakah ia menerima (*accept*) atau tidak menerima *string* tersebut.

     → Persoalan yang diselesaikan oleh mesin Turing tipe ini disebut *decidable problems.*

     → Dengan kata lain, sebuah persoalan disebut *dedicable* jika mesin Turing dapat memecahkan (*solve* atau *decide*)) persoalan tersebut

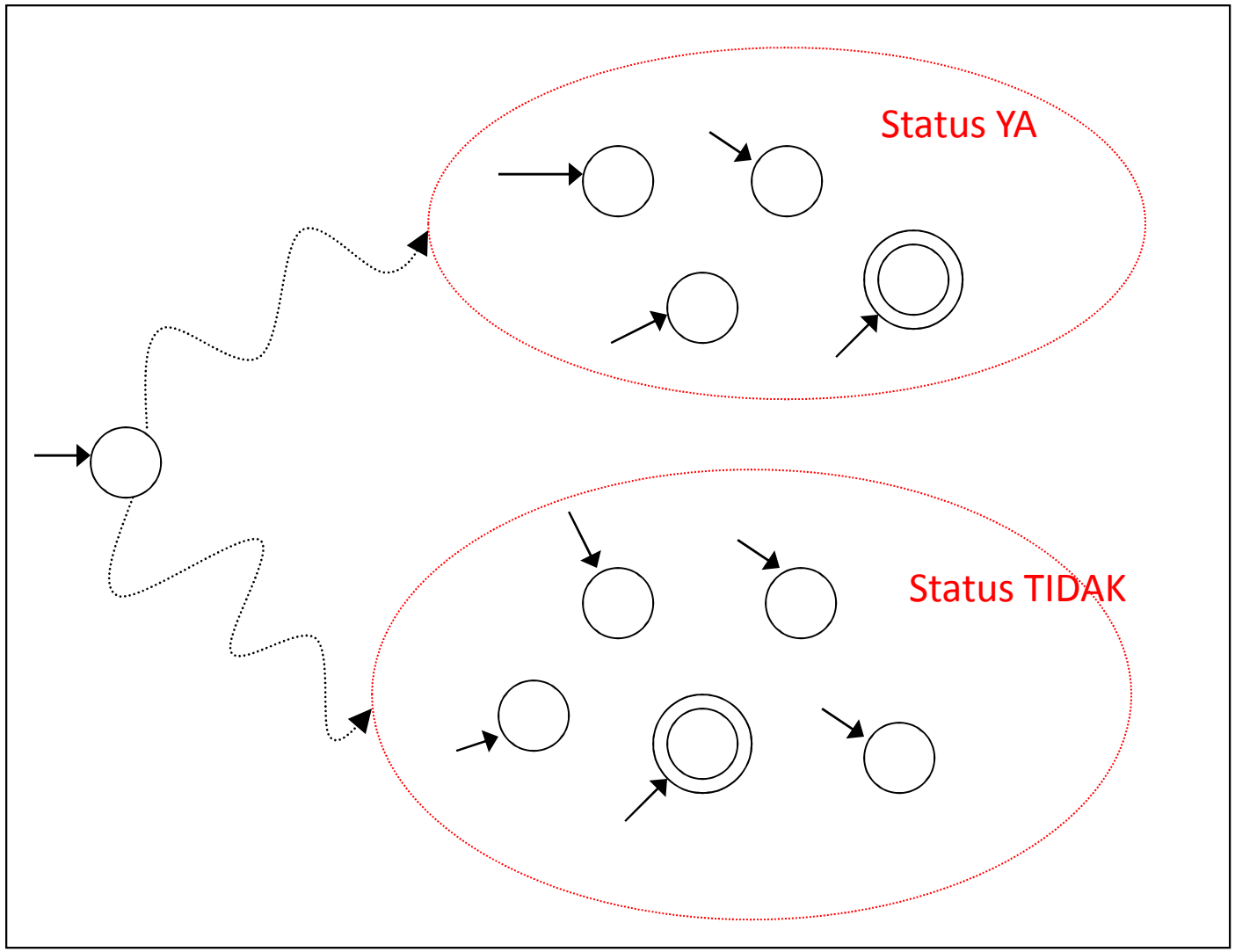
     → Mesin Turing dalam hal dapat disebut sebagai sebuah **algoritma.**

     → Bahasa yang diterima oleh mesin Turing tipe ini disebut **bahasa rekursif** (*recursive language*)

→ Mesin Turing tipe ini memberikan jawaban "YA" (*accept*) atau "TIDAK" (*reject*) untuk setiap instans persoalan.

if jawaban persoalan adalah YA
maka berhenti pada status ya

if jawaban persoalan adalah TIDAK
maka berhenti pada status TIDAK



Status YA dan status TIDAK adalah status berhenti (*final state*)

→ Contoh *decidable problem*:

Apakah jawaban persoalan di bawah ini, YA atau TIDAK?

- Apakah mesin *M* memiliki tiga status?
- Apakah string *w* adalah bilangan biner?
- Apakah sebuah DFA *M* menerima sembarang masukan string?

2. Mesin Turing yang hanya berhenti jika ia menerima *string* yang diberikan (dinyatakan oleh status akhir). Jika mesin tidak menerima, maka ia tidak akan berhenti (*infinite loop*)

   → Jadi, kita tidak pernah yakin apakah *string* tersebut diterima atau ditolak.

   → Bahasa yang diterima oleh mesin Turing tipe ini disebut **bahasa yang dapat dienumerasi secara rekursif** (*recursively enumerable language*)

   → Mesin Turing tipe ini sering diacu sebagai **prosedur**

Recursive Languages vs. Recursively Enumerable (RE) Languages

- Sebuah bahasa formal disebut bahasa rekursif (*recursive lanauage*) jika terdapat mesin Turing yang selalu berhenti untuk setiap input *string* yang diberikan.

- Sembarang mesin Turing *M* untuk bahasa rekursif akan terlihat seperti ini:

- Sebuah bahasa formal disebut bahasa *recursively enumerable* (RE) jika terdapat mesin Turing yang hanya berhenti jika jika menerima (*accept*) input *string* yang diberikan.

- Sembarang mesin Turing *M* untuk bahasa RE akan terlihat seperti ini:

- Istilah “rekursif” sinonim dengan *decidable*.
- Di dalam pemrograman atau matematika, sebuah fungsi rekursif akan pasti berhenti.
- Jadi, sebuah persoalan disebut rekursif jika kita dapat membuat fungsi rekursif untuk memecahkannya, dan fungsi rekursif tersebut selalu berhenti.

- Istilah “*recursively enumerable*” memiliki makna yang serupa dengan di atas. Sebuah fungsi yang dapat mendaftarkan semua *string* dari sebuah bahasa disebut fungsi yang “mengenumerasi” bahasa tersebut.

- Bahasa yang anggota-anggotanya didaftarkan dalam suatu urutan tertentu sama dengan bahasa yang diterima oleh Mesin Turing, meskipun mesin Turing mungkin akan *infinite loop* pada input string yang tidak ia terima.

Definisi Formal

- **Definisi**. Misalkan $\sum$ adalah alfabet dan $L \subseteq \sum^*$ adalah sebuah bahasa. Kita katakan bahasa L adalah *rekursif* jika $L = L(M)$ untuk mesin Turing M sedemikian sehingga untuk setiap $w \in \sum^*$ berlaku kondisi berikut:

  1. Jika $w \in L$, maka komputasi mesin Turing M terhadap input string w berhenti pada status diterima (*accept status*).
  2. Jika $w \notin L$, maka komputasi mesin Turing M terhadap input string w berhenti pada status ditolak (*reject status*).

Keterangan: $L = L(M)$ artinya bahasa yang diterima oleh mesin Turing M

- Mesin Turing *M* yang bertipe ini dapat dianggap sebagai sebuah "algoritma". Algoritma adalah urutan langkah-langkah yang terdefinisi dengan baik dan selalu berhenti dan menghasilkan jawaban.

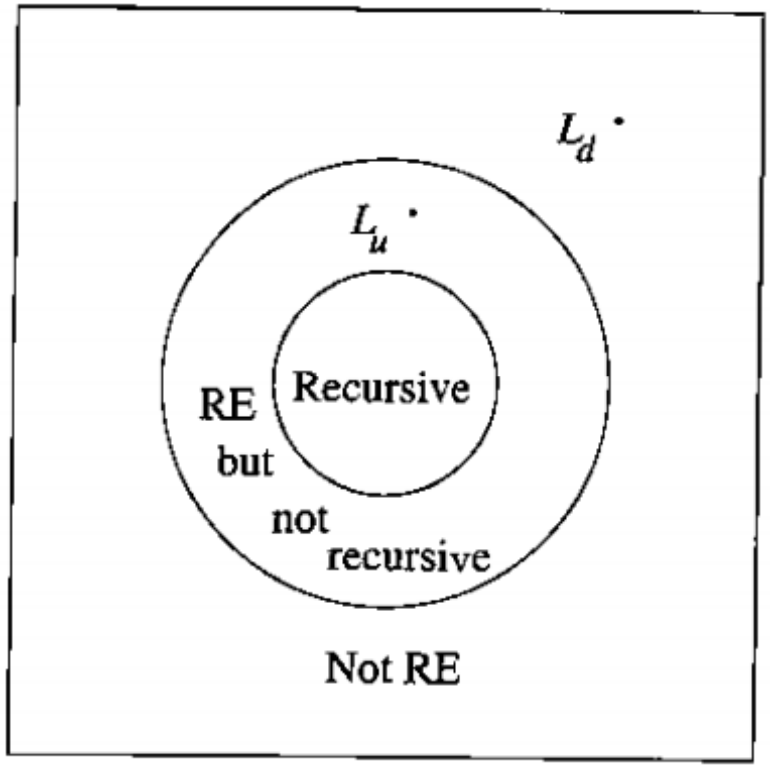
- Oleh karena sebuah persoalan berkoresponden dengan sebuah bahasa, maka persoalan *L* disebut *decidable* jika ia adalah bahasa rekursif. Dengan kata lain, sebuah bahasa disebut *dicidable* jika terdapat algoritma yang:

  (i) berhenti pada setiap input string *w*, dan

  (ii) menyimpulkan apakah $w \in L$ atau $w \notin L$.

- Bahasa yang tidak *decidable* disebut *undecidable language*. Untuk bahasa *undecidable* tidak terdapat algoritma yang memenuhi (i) dan (ii).

Hubungan Antara Bahasa Rekursif, RE, dan non-RE

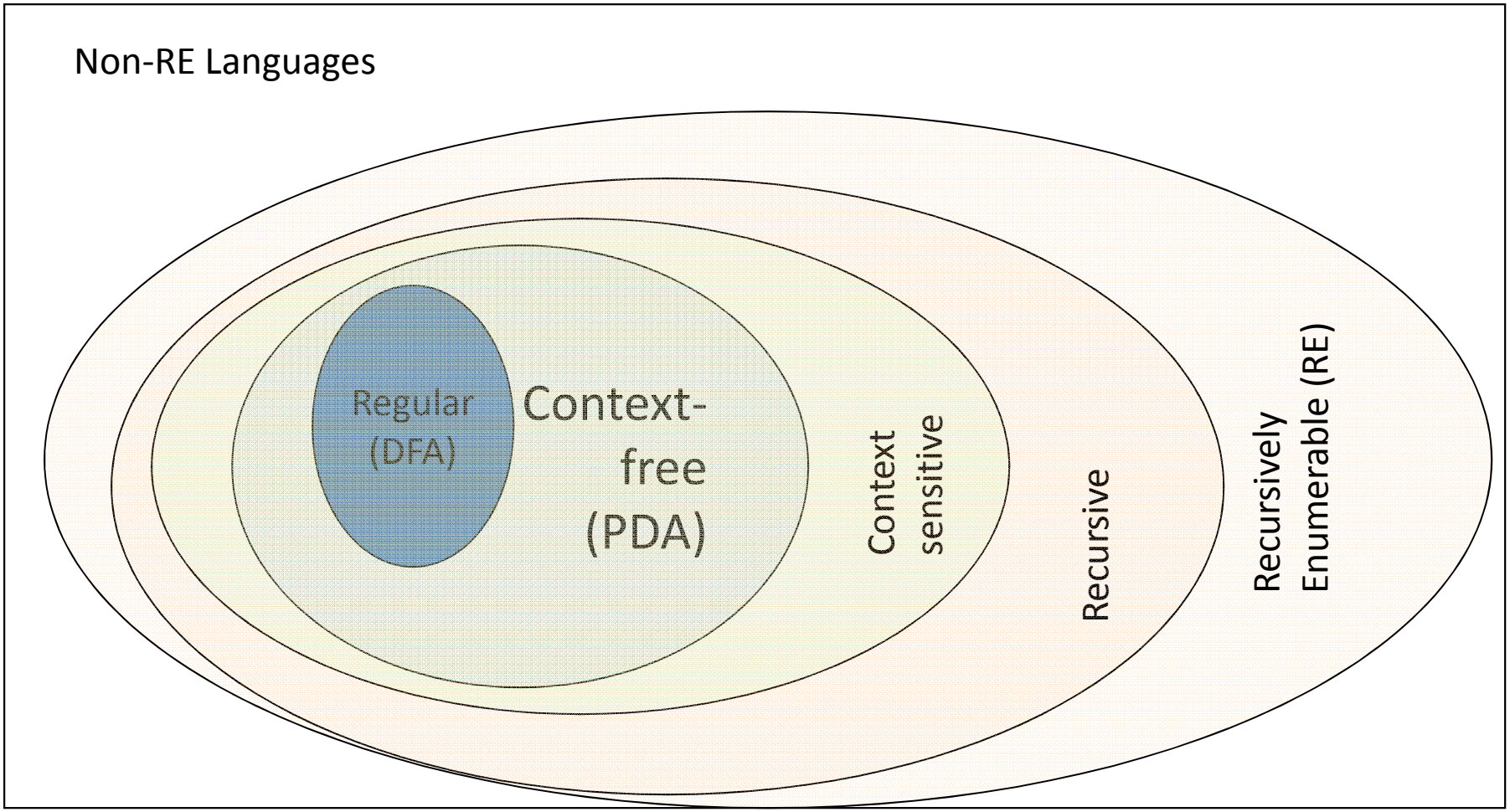


Keteangan:

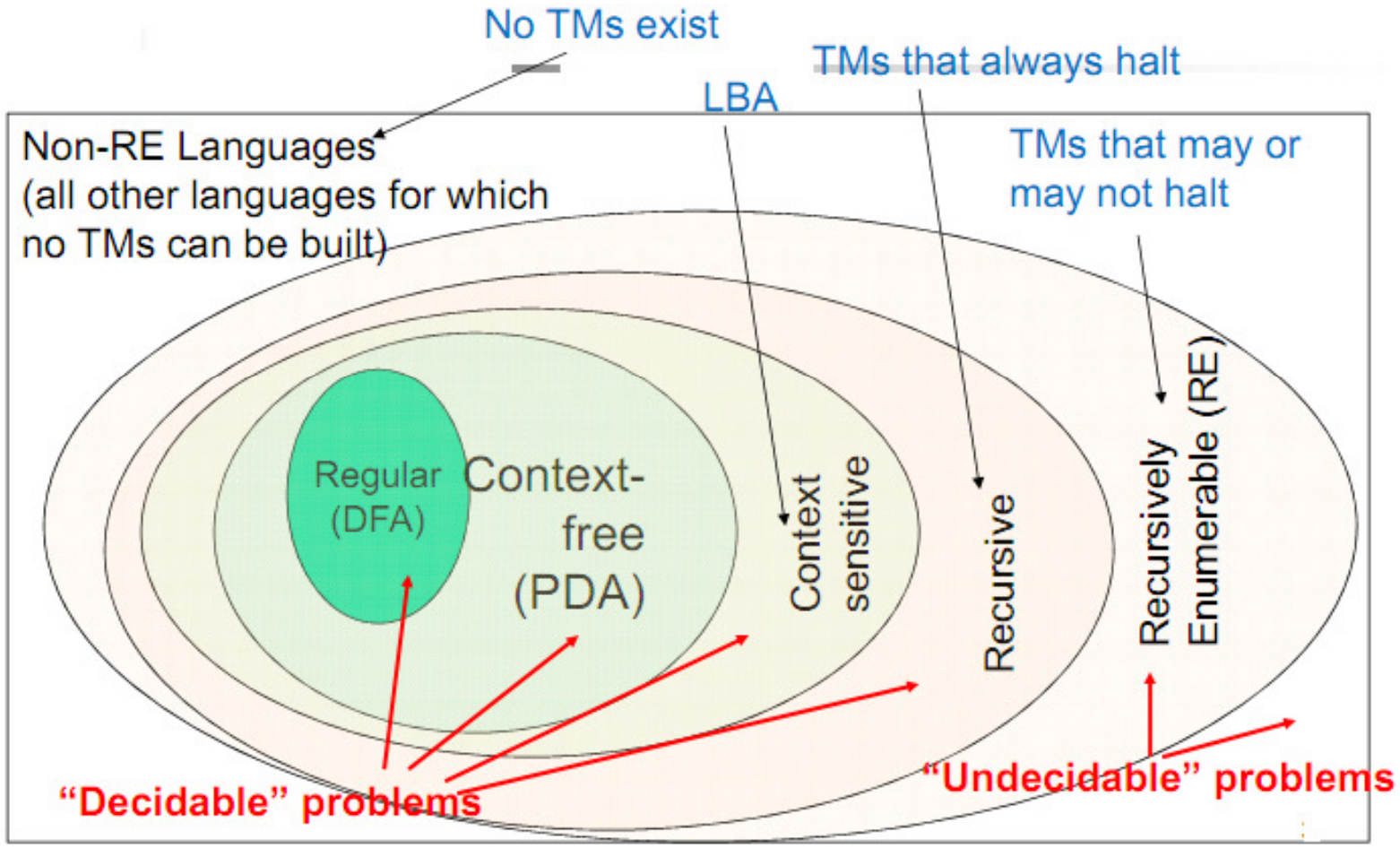
L_d = Bahasa diagonalisasi (akan dijelaskan kemudian)

L_u = Bahasa Universal (akan dijelaskan kemudian)

- Hirarkhi bahasa formal menurut Chomsky:



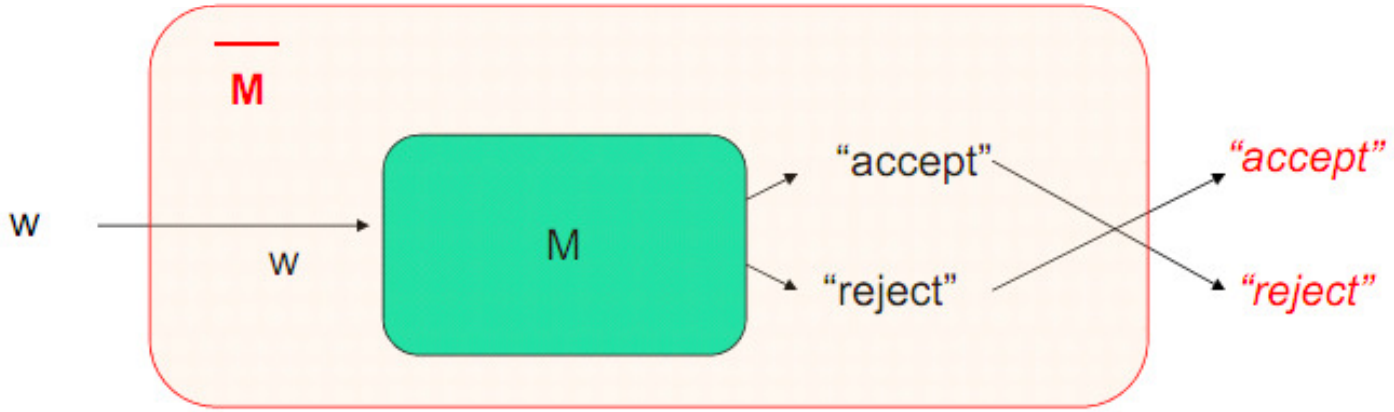
- *Recursive, RE, Undecidable languages*



Kaidah Tentang Komplemen

- Misalkan *L* adalah sebuah bahasa pada alfabet {0, 1} dan $\overline{L}$ adalah komplemennya.

  <u>Kaidah 1</u>: Jika *L* rekursif, maka $\overline{L}$ juga rekursif



Kaidah 2: Jika L dan $\overline{L}$ adalah RE, maka L adalah rekursif.

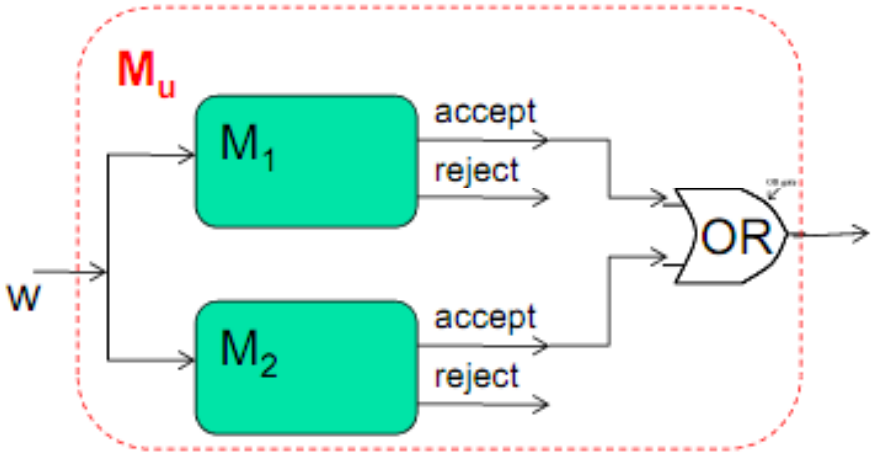
Menurut Kiadah 1, $\overline{L}$ juga rekursif.

Misalkan $L_1 = L(M_1)$ dan $L_2 = \overline{L} = L(M_2)$

Let M_u = TM for L_1 U L_2

M_u construction:

1. Make 2-tapes and copy input w on both tapes
2. Simulate M_1 on tape 1
3. Simulate M_2 on tape 2
4. If either M_1 or M_2 accepts, then M_u accepts
5. Otherwise, M_u rejects.



Undecidable Problems

- Beberapa persoalan tergolong *undecidable*, yang berarti tidak terdapat mesin Turing yang dapat menyelesaikan instans persoalan tersebut.

  Contoh: *Hello-world problem*

  Halting problem

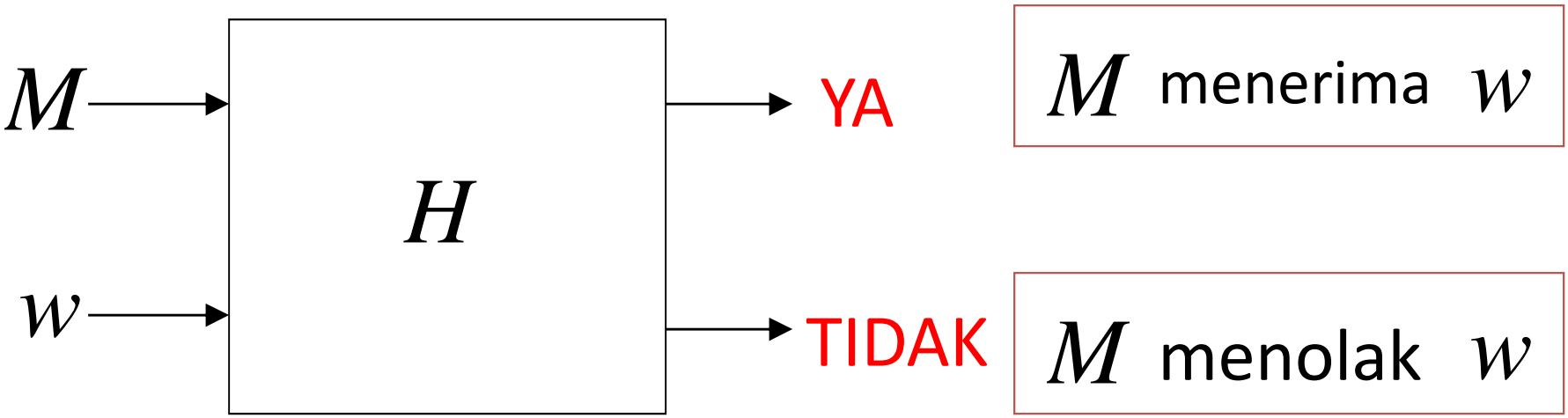
  Membership Problem

Membership Problem

- *Membership problem*: Diberikan mesin Turing *M* dan string *w*. Apakah *M* menerima *w*? Dengan kata lain, apakah *w* termasuk ke dalam bahasa yang dikenali oleh *M*, atau $w \in L(M)$?

- Untuk membuktikan bahwa *membership problem* adalah *undecidable*, maka dilakukan pembuktian dengan kontradiksi sebagai berikut:

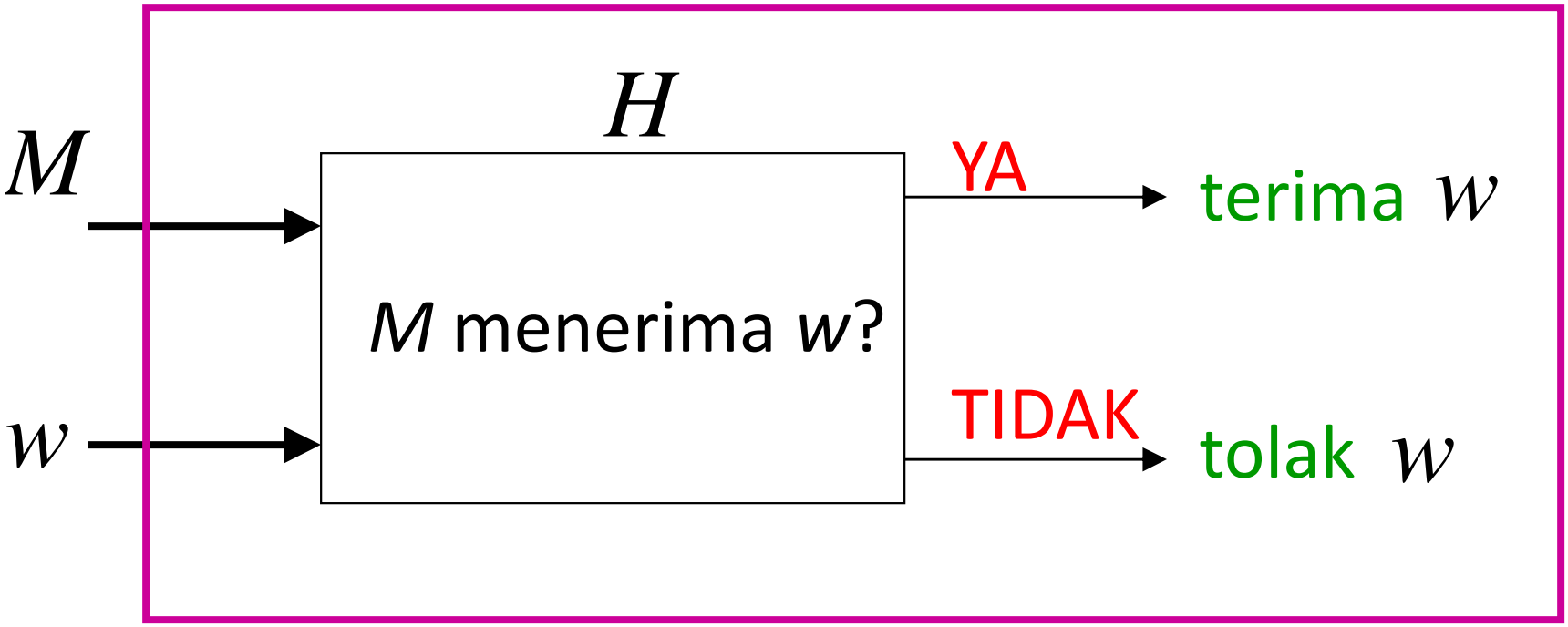
  Misalkan terdapat mesin Turing *H* untuk memecahkan *membership problem*.



Misalkan *L* adalah bahasa *recursively enumerable* dan misalkan *M* adalah mesin Turing yang menerima *L*.

Kita akan buktikan bahwa *L* juga adalah bahasa rekursif.

Kita gambarkan mesin Turing yang menerima *L* dan berhenti pada sembarang masukan apapun yang diberikan:



Oleh karena itu *L* adalah bahasa rekursif

Karena *L* adalah bahasa yang dipilih sembarang, maka setiap bahasa *recursively enumerable* juga adalah bahasa rekursif.

Tetapi, terdapat bahasa *recursively enumerable* yang bukan bahasa rekursif.

Kontradiksi!

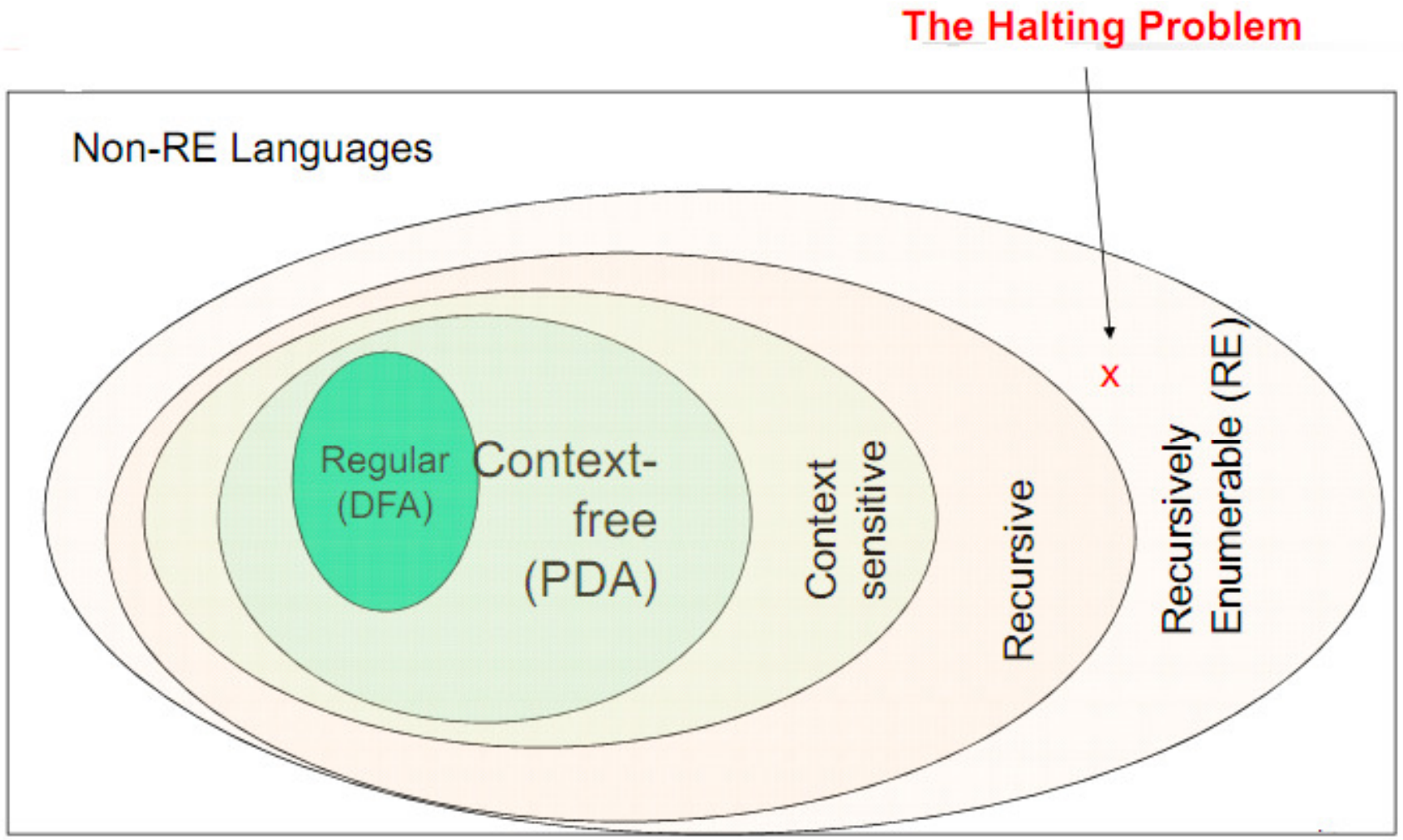
Halting Problem

- *Halting problem* adalah persoalan *recursively enumerable* yang juga *undecidable.*

  Halting Problem: Diberikan sebuah program *P* dan masukannya, *I*. Tentukan apakah *P* akan berhenti pada input *I*?

- Mengingat mesin Turing dapat dipandang sebagai sebuah program dan pita sebagai data masukan, maka deskripsi *Halting Problem* di atas ekivalen dengan *Halting Problem* pada mesin Turing, seperti yang sudah dipelajari sebelumnya:

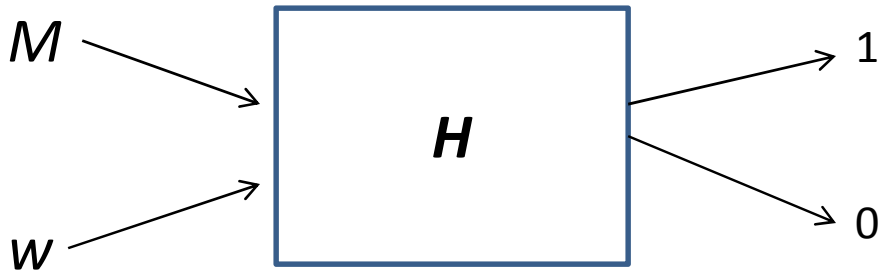
  Halting Problem: Diberikan mesin Turing *M* dan *string* masukan *w*. Tentukan apakah *M* akan berhenti pada input *w*?


The Halting Problem
Non-RE Languages
Regular (DFA)
Context-free (PDA)
Context sensitive
Recursive
Recursively Enumerable (RE)
x

- Akan dibuktikan bahwa *Halting Problem* adalah *undecidable*.

- Untuk membuktikannya, asumsikan bahwa *Halting Problem* adalah *decidable*, yaitu terdapat mesin Turing universal *H* yang mensimulasikan perilaku mesin Turing *M*.

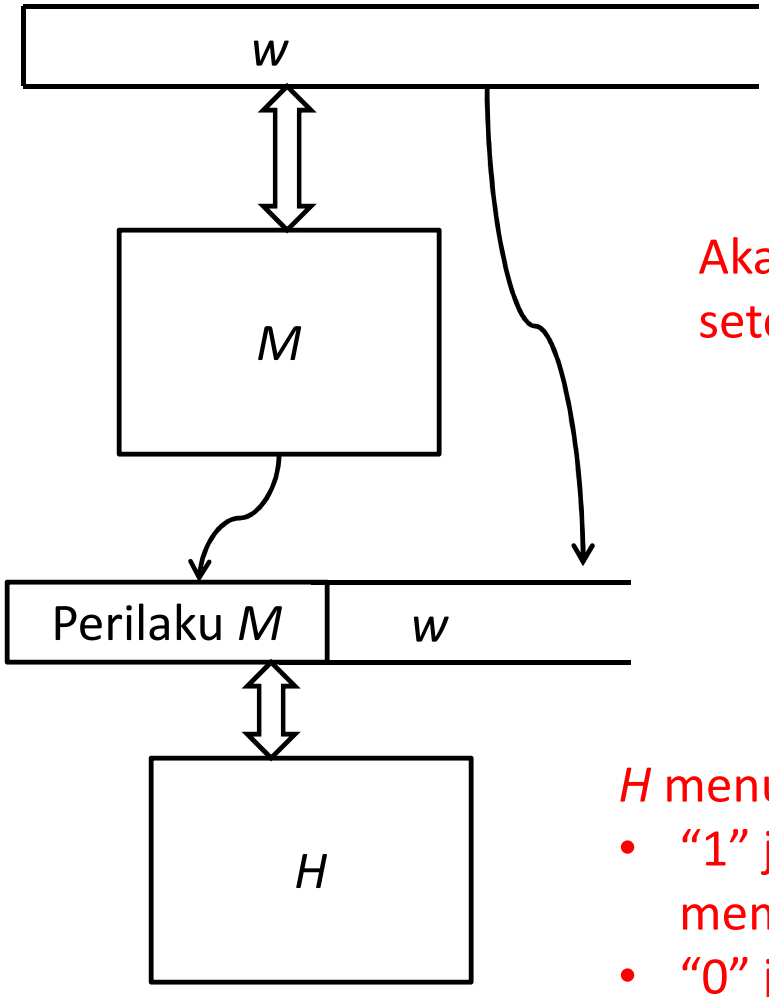
- *H* menerima input berupa perilaku *M* dan data *w*. *H* memberikan jawaban "1" jika *M* berhenti setelah mengolah *w*, atau "0" jika *M* tidak berhenti setelah mengolah *w*.



M berhenti setelah mengolah *w*

M tidak berhenti setelah mengolah *w*

Konstruksi *H*

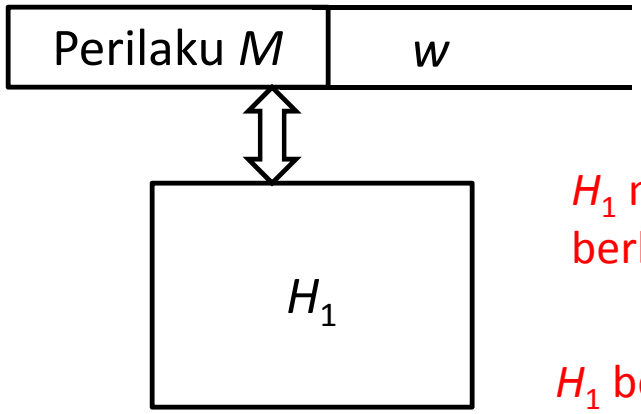


Akankah *M* berhenti setelah mengolah *w*?

H menuliskan:

- "1" jika *M* berhenti setelah membaca *w*
- "0" jika tidak

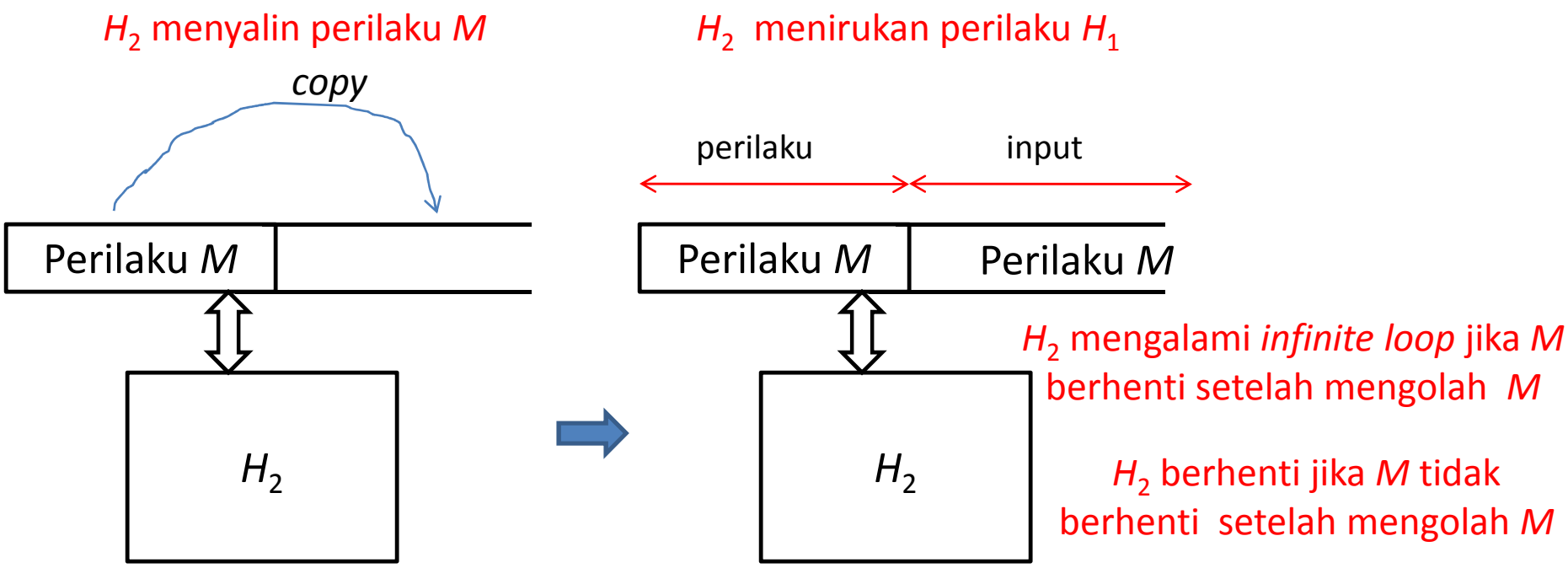
- Misalkan mesin *H* dimodifikasi menjadi mesin H_1 sedemikian sehingga H_1 memiliki perilaku seperti *H*, namun bedanya:
  - H_1 mengalami *infinite loop* jika *M* berhenti setelah mengolah *w*
  - H_1 berhenti jika *M* tidak berhenti (mengalami *infinite loop*) setelah mengolah *w*



H_1 mengalami *infinite loop* jika *M* berhenti setelah mengolah *w*

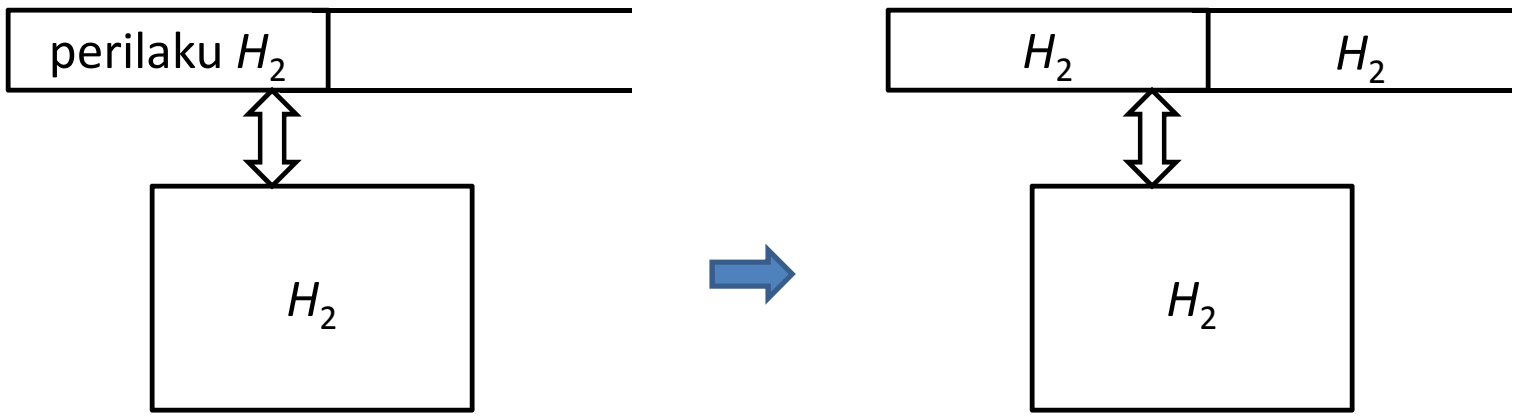
H_1 berhenti jika *M* tidak berhenti (mengalami *infinite loop*) setelah mengolah *w*

- Selanjutnya, mesin H_1 dimodifikasi lagi menjadi mesin H_2. Mesin H_2 membaca hanya deskripsi perilaku *M*, lalu menirukan perilaku H_1.
- Perilaku H_2 dapat digambarkan sebagai berikut: Jika diberikan deskripsi perilaku mesin *M*, maka:
  - H_2 mengalami *infinite loop* jika *M* berhenti setelah mengolah deskripsi perilaku *M* sendiri.
  - H_2 berhenti jika *M* tidak berhenti (mengalami *infinite loop*) setelah mengolah deskripsi perilaku *M* sendiri.



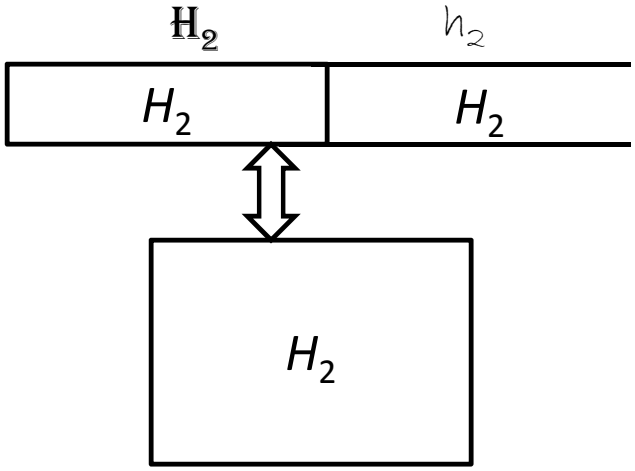
- Akan ditunjukkan bahwa H_2 tidak ada, akibatnya H_1 tidak ada, dan kesimpulannya H juga tidak ada.

- Cara pembuktiannya adalah dengan mengamati apa yang terjadi pada H_2 jika masukan yang diberikan adalah perilaku dirinya sendiri .



Perilaku H_2:

- Mengalami *infinite loop* jika H_2 berhenti setelah mengolah h_2.
- Berhenti jika H_2 tidak berhenti (mengalami *infinite loop*) setelah mengolah h_2.



Pada keadaan yang digambarkan oleh Gambar di atas, dapat ditunjukkan sifat-sifat berikut:

- H_2 mengalami *infinite loop* jika H_2 berhenti setelah mengolah perilakunya sendiri, dan
- H_2 berhenti jika H_2 tidak berhenti (mengalami *infinite loop*) setelah mengolah perilakunya sendiri

KONTRADIKSI!!!

- Dari kedua sifat di atas terlihat adanya hal yang bertentangan:

  H_2 mengalami *infinite loop* jika dan hanya jika H_2 berhenti.

  Hal ini tidak mungkin terjadi atau kontradiksi, sehingga akibatnya dapat disimpulkan:

  H_2 tidak ada ==> H_1 tidak ada ==> *H* tidak ada

- Dengan kata lain, mesin Turing untuk menyelesaikan *Halting Problem* tidak mungkin ada!

- Dengan kata lain, *Halting Problem* adalah *undecidable.*