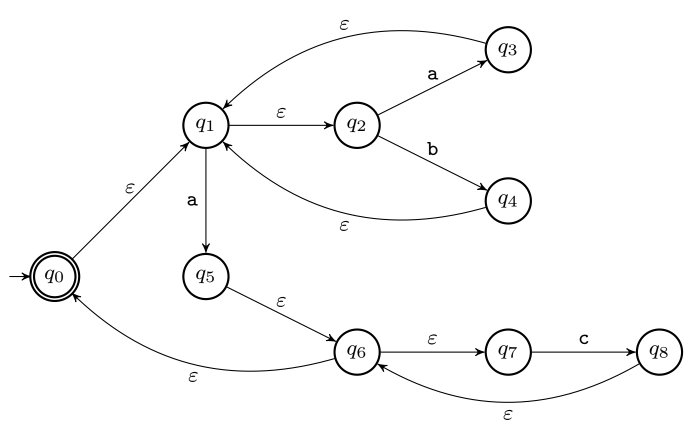
Automi e Linguaggi **(M. Cesati)**

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 24 giugno 2024

Esercizio 1 [6] Determinare un automa a stati finiti deterministico (DFA) per il linguaggio generato dalla REX $((\mathtt{a} \cup \mathtt{b})^*\mathtt{ac}^*)^*$.

Soluzione: Innanzi tutto determiniamo un automa non deterministico equivalente alla REX data. Utilizzando la costruzione canonica si ottiene:

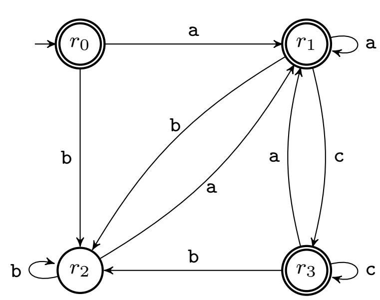


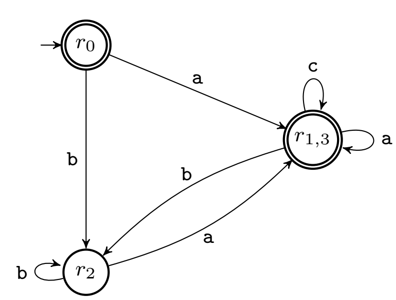
Le chiusure rispetto alle ε-transizioni degli stati di questo automa sono:

$$
\begin{array}{lll}
E(q_0) = \{q_0, q_1, q_2\} & E(q_1) = \{q_1, q_2\} & E(q_2) = \{q_2\} \\
E(q_3) = \{q_1, q_2, q_3\} & E(q_4) = \{q_1, q_2, q_4\} & E(q_5) = \{q_0, q_1, q_2, q_5, q_6, q_7\} \\
E(q_6) = \{q_0, q_1, q_2, q_6, q_7\} & E(q_7) = \{q_7\} & E(q_8) = \{q_0, q_1, q_2, q_6, q_7, q_8\}
\end{array}
$$

Adottando la procedura di conversione dall'automa non deterministico a quello deterministico si ottiene l'automa a sinistra qui sotto:

$r_0 = E(q_0)$
$r_1 = E(q_3) \cup E(q_5)$
$r_2 = E(q_4)$
$r_3 = E(q_8)$





Si può semplificare questo automa osservando che gli stati r_1 e r_3 hanno le stesse transizioni e lo stesso stato di accettazione e quindi possono essere fusi, ottenendo così l'automa a destra. Si osservi anche che r_0 e r_2, pur avendo le stesse transizioni, non possono essere fusi perché r_0 è di accettazione mentre r_2 non lo è.

Esercizio 2 [6] Si consideri il linguaggio $A = \{w\mathtt{0}^h\mathtt{1}^k \mid w \in \{\mathtt{a},\mathtt{b}\}^*, |w| = h + k, \text{ e } |w|_{\mathtt{a}} = h\}$, ove $|w|_{\mathtt{a}}$ indica il numero di simboli '`a`' contenuti in w. Ad esempio, `bab011` $\in A$, `0bab11` $\notin A$, `ab00` $\notin A$. Determinare se A è un linguaggio libero dal contesto (CFL), giustificando la risposta con una dimostrazione.

Soluzione: Intuitivamente il linguaggio A non è CFL in quanto per decidere se $x \in A$ è necessario contare sia le occorrenze di '`a`' che di '`b`' nella prima parte della stringa, e confrontarle rispettivamente con il numero di occorrenze di '`0`' e di '`1`' nella seconda parte. Per dimostrarlo formalmente supponiamo per assurdo che A sia CFL, e dunque che per esso valga il Pumping Lemma. Sia dunque $p > 0$ la pumping length, e consideriamo la stringa $s = \mathtt{a}^p\mathtt{b}^p\mathtt{0}^p\mathtt{1}^p$. Ovviamente $s \in A$ e $|s| = 4p$, dunque deve esistere una suddivisione $s = uvxyz$ tale che $|vy| > 0$, $|vxy| \leq p$ e $uv^ixy^iz \in A$ per ogni $i \geq 0$.

Considerando la stringa vy possono verificarsi soltanto questi tre casi:

1. In v non sono presenti simboli '`a`' e '`b`', ossia tutti i simboli '`b`' (e quindi '`a`') cadono entro u. Consideriamo dunque la stringa uv^ixy^iz per qualunque $i \neq 1$: poiché $|vy| > 0$, il numero di '`0`' e/o '`1`' deve variare, ma il numero di '`a`' e '`b`' rimane lo stesso, pertanto la stringa pompata non può far parte di A.

2. In y non sono presenti simboli '`0`' e '`1`', ossia tutti i simboli '`0`' (e quindi '`1`') cadono entro z. Consideriamo dunque la stringa uv^ixy^iz per qualunque $i \neq 1$: poiché $|vy| > 0$, il numero di '`a`' e/o di '`b`' deve variare, ma il numero di '`0`' e '`1`' rimane lo stesso, pertanto la stringa pompata non può far parte di A.

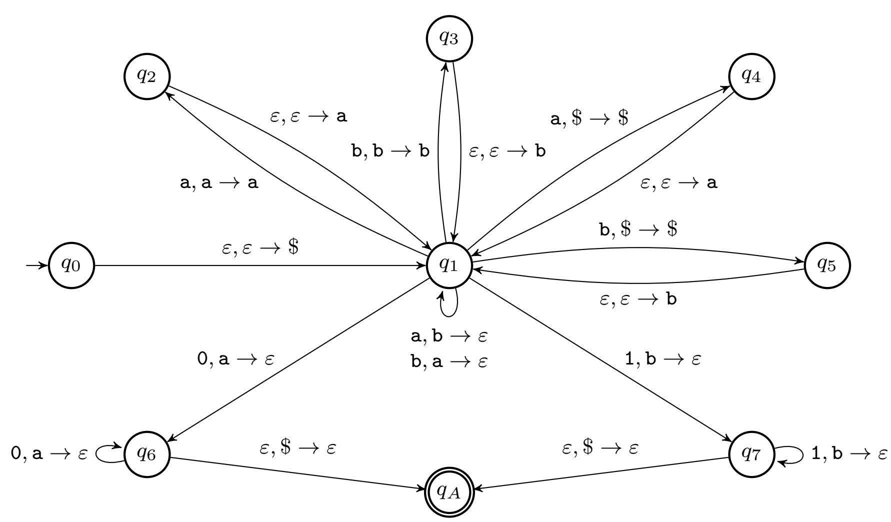
3. In v sono presenti simboli '`a`' e/o '`b`' ed in y sono presenti simboli '`0`' e/o '`1`'. In effetti però, poiché $|vxy| \leq p$, v non può contenere il simbolo '`a`' e y non può contenere il simbolo '`1`', ossia tutte le '`a`' sono in u e tutti gli '`1`' sono in z. Pertanto la stringa uv^ixy^iz per qualunque $i \neq 1$ modifica il numero di '`b`' ma non modifica il numero di '`1`', e modifica il numero di '`0`' ma non modifica il numero di '`a`'; dunque la stringa pompata non può appartenere ad A.

Poiché ogni possibile suddivisione della stringa s non è pompabile, il Pumping Lemma non vale. Ciò implica che A non può essere CFL.

Esercizio 3 [7] Si considerino i linguaggi $B = \{w\mathtt{0}^h \mid w \in \{\mathtt{a},\mathtt{b}\}^* \text{ e } |w|_\mathtt{a} - |w|_\mathtt{b} = h > 0\}$ e $C = \{w\mathtt{1}^h \mid w \in \{\mathtt{a},\mathtt{b}\}^* \text{ e } |w|_\mathtt{b} - |w|_\mathtt{a} = h > 0\}$, ove $|w|_\mathtt{x}$ indica il numero di occorrenze del carattere x nella stringa w. Determinare un automa a pila deterministico (DPDA) che riconosca il linguaggio $B \cup C$, od in alternativa dimostrare che tale automa non esiste.

Soluzione: Il linguaggio $B \cup C$ contiene elementi costituiti dalla concatenazione di stringhe in $\{\mathtt{a},\mathtt{b}\}^*$ e da un numero in unario che rappresenta la differenza del numero di occorrenze dei due simboli nella prima parte. In particolare, se il numero di '`a`' è maggiore del numero di '`b`' allora la differenza è codificata in unario utilizzando il simbolo '`0`'; al contrario se il numero di '`b`' è maggiore del numero di '`a`' allora la differenza è codificata in unario utilizzando il simbolo '`1`'. Un automa a pila che riconosca $B \cup C$ dovrebbe innanzitutto calcolare la differenza nel numero di simboli nella prima parte della stringa, e poi confrontare tale differenza con il numero di '`0`' o '`1`', come appropriato, nella seconda parte. Si osservi che $B \cup C$ non contiene alcuna stringa avente lo stesso numero di '`a`' e di '`b`'.

Consideriamo dunque il seguente automa a pila P (le transizioni omesse si intendono sempre entranti in uno stato di non accettazione con ogni transizione uscente che rientra sullo stesso stato):



Dimostriamo che $L(P) = B \cup C$. L'automa inizializza lo stack con il simbolo '\$', poi nello stato interno q_1 legge la prima parte della stringa in input, contenente solo caratteri '`a`' e '`b`'. Supponiamo di leggere dall'input x il carattere '`a`': vi sono tre possibili casi:

1. Sulla cima dello stack vi è il carattere '`a`': ciò significa che prima della lettura corrente valeva $|x|_\mathtt{a} > |x|_\mathtt{b}$; dunque P rimette sullo stack il carattere '`a`' letto in q_1 e aggiunge tramite q_2 un ulteriore carattere '`a`': la differenza (ossia il numero di '`a`' sullo stack) aumenta di una unità.

2. Sulla cima dello stack vi è il carattere '\$': ciò significa che prima della lettura corrente valeva $|x|_{\mathtt{a}} = |x|_{\mathtt{b}}$; dunque P rimette sullo stack il carattere '\$' letto in q_1 e aggiunge tramite q_4 un carattere '`a`' sullo stack (è stata letta in totale una '`a`' in più delle '`b`').

3. Sulla cima dello stack vi è il carattere '`b`': ciò significa che prima della lettura corrente valeva $|x|_{\mathtt{a}} < |x|_{\mathtt{b}}$; dunque P *non* aggiunge nuovamente sullo stack il carattere '`b`' letto in q_1, in modo da ridurre la differenza tra le '`b`' e le '`a`' di una unità.

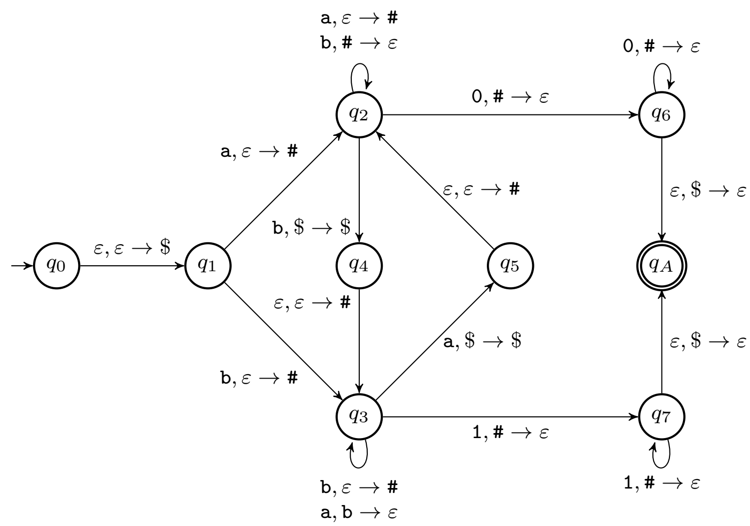
Analogo meccanismo è implementato quando si legge un carattere '`b`' dall'input, per mezzo degli stati interni q_1, q_3 e q_5.

Dal momento in cui P legge un carattere diverso '`a`' e '`b`' si lascia lo stato interno q_1 per non rientrarvi più. Di conseguenza, ogni stringa malformata in cui un carattere '`a`' o '`b`' segue uno '`0`' od un '`1`' non può essere accettata. Similarmente, si osservi che qualunque stringa malformata iniziante con '`0`' o '`1`' non può essere accettata perché sulla cima dello stack si troverebbe '\$', ma non esistono transizioni da q_1 verso q_6 e q_7 definite quando la cima dello stack contiene '\$'; per il medesimo motivo, nessuna stringa che contenga un ugual numero di '`a`' e '`b`' può essere accettata.

Se il primo carattere diverso da '`a`' e '`b`' è '`0`', allora la cima dello stack deve contenere '`a`' (quindi ci si aspetta una eccedenza di '`a`' in quanto $x \in B$): lo stato q_6 continua a leggere gli '`0`' nella parte finale di x rimuovendo man mano le '`a`' dallo stack. Se lo stack contiene '\$' si può entrare nello stato di accettazione q_A, ma la stringa x viene accettata soltanto se P è riuscito a leggere tutti gli '`0`' nella parte finale, ossia se questi codificano in unario il valore $|x|_{\mathtt{a}} - |x|_{\mathtt{b}}$. Analogo discorso si applica se il primo carattere diverso da '`a`' e '`b`' è '`1`': ci si aspetta che $x \in C$ e che nella parte finale di x vi siano tanti '`1`' quanti sono i caratteri '`b`' sullo stack, ossia in quantità pari a $|x|_{\mathtt{b}} - |x|_{\mathtt{a}}$.

Osservando l'automa P è immediato verificare che esso è deterministico. Infatti gli unici stati con più di una transizione uscente sono q_1, q_6 e q_7. Tutte le transizioni uscenti da q_1 forzano la lettura di un simbolo in input ed una "pop" dallo stack, quindi simbolo in input e simbolo sulla cima dello stack definiscono deterministicamente la transizione da applicare. Lo stato q_6 ha solo due transizioni uscenti, ma l'applicazione di queste è deterministica perché dipende dal simbolo sulla cima dello stack (se '`a`' oppure '\$'). Analogo discorso vale per lo stato q_7. Dunque, P è in effetti un DPDA.

Una soluzione alternativa dell'esercizio è costituita dal seguente DPDA. La differenza sostanziale rispetto alla soluzione precedente è che l'informazione su quale simbolo sia in ogni momento eccedente non è memorizzata entro lo stack (il simbolo sulla cima indicante il carattere attualmente eccedente) ma dagli stati interni (q_2 e q_6 per le eccedenze di '`a`', q_3 e q_7 per le eccedenze di '`b`').



Esercizio 4 [7] Sia L un linguaggio Turing-riconoscibile (ossia ricorsivamente enumerabile) e sia $L^s = \{x \mid \exists\, y \in L \text{ tale che } x \text{ è una sottostringa di } y\}$. Dimostrare che L^s è Turing-riconoscibile.

Soluzione: Il linguaggio L^s è costituito da tutte le sottostringhe degli elementi di L. Se ad esempio $abcd \in L$, allora certamente a, b e bc fanno parte di L^s.

Per dimostrare che L^s è ricorsivamente enumerabile è sufficiente esibire un enumeratore E' per esso, ossia una TM che produce in uscita tutti e soli gli elementi del linguaggio, anche se possibilmente con ripetizioni e senza ordine prefissato. In effetti, poiché L è ricorsivamente enumerabile, esiste un enumeratore E per esso, ed è semplice costruire E' basandosi su E:

E'= "On any input:
1. Simulate the enumerator E for L
   2. for each element $x \in L$ printed by E:
      3. for each substring y of x:
         4. print y"

Sia $y \in L^s$; dunque y è una sottostringa di un elemento $x \in L$, ossia $x = wyz$, con w e z eventualmente uguali a ε. Poiché E enumera L, dopo un tempo finito genererà l'elemento x di L. Pertanto E' stamperà tutte le sottostringhe di x, e dunque anche y. Viceversa, se E' stampa una stringa y, allora necessariamente tale stringa è una sottostringa di una stringa x stampata da E, e poiché E è un enumeratore per L, $x \in L$. Pertanto $y \in L^s$. Concludiamo che E' è un enumeratore per L^s, e dunque L^s è ricorsivamente enumerabile.

Esercizio 5 [7] Sia $\mathcal{D} = \{\langle M \rangle \mid M$ è una macchina di Turing deterministica per la quale esiste una stringa x tale che $M(x)$ accetta in meno di $|\langle M \rangle|$ passi$\}$. In altri termini, $\langle M \rangle \in \mathcal{D}$

se e solo se esiste una stringa x tale che $M(x)$ accetta in un numero di passi inferiore alla lunghezza della codifica di M. $\mathcal{D}$ è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Innanzitutto osserviamo che non è consentito utilizzare il Teorema di Rice, in quanto la proprietà che caratterizza l'appartenenza al linguaggio $\mathcal{D}$ dipende strettamente da una caratteristica della macchina di Turing particolare (la lunghezza della sua codifica) e non è una proprietà del linguaggio associato alla TM. Infatti, potremmo considerare una TM M il cui linguaggio associato è costituito da una singola stringa w e tale che il numero di passi della computazione $M(w)$ è superiore alla lunghezza della codifica $|\langle M\rangle|$. D'altra parte, è sempre possibile ottenere una TM M' aggiungendo alla TM M stati e transizioni inutili in modo da aumentare la dimensione della codifica fino a renderla superiore al numero di passi di $M(w)$. Pertanto avremmo $L(M) = L(M')$, $\langle M\rangle \in \mathcal{D}$ e $\langle M'\rangle \notin \mathcal{D}$.

In effetti il linguaggio $\mathcal{D}$ è decidibile poiché le computazioni associate alla proprietà che definisce il linguaggio sono di lunghezza limitata. Per dimostrarlo formalmente, consideriamo $\langle M\rangle \in \mathcal{D}$. Per definizione esiste dunque una stringa w che è accettata da M con un numero di passi inferiore a $|\langle M\rangle|$. È immediato osservare che se w avesse lunghezza maggiore o uguale a $|\langle M\rangle|$, allora esisterebbe un'altra stringa w' di lunghezza inferiore a $|\langle M\rangle|$ che è accettata da M in meno di $|\langle M\rangle|$ passi. Infatti, solo $|\langle M\rangle| - 1$ simboli di w possono essere letti da M prima di accettare la stringa, quindi qualunque prefisso di w di lunghezza $|\langle M\rangle| - 1$ deve ugualmente essere accettato.

Possiamo dunque considerare un decisore T per $\mathcal{D}$ basato su una TM universale che esegue M per meno di $|\langle M\rangle|$ passi su tutte le stringhe di lunghezza inferiore a $|\langle M\rangle|$. Formalmente, sia S l'insieme di tutte le stringhe sull'alfabeto di input Σ di M di lunghezza inferiore a $|\langle M\rangle|$. Definiamo T come segue:

T= "On input $\langle M\rangle$, where M is a Turing machine:
1. For every string $w \in S$:
    2. Run $M(w)$ for at most $|\langle M\rangle| - 1$ steps
        3. If $M(w)$ accepts, then accept
4. Reject"

L'insieme S contiene $\sum_{k=0}^{|\langle M\rangle|-1} |\Sigma|^k$ stringhe, quindi il ciclo esterno viene eseguito un numero finito di volte. In ogni iterazione la simulazione di $M(w)$ viene interrotta dopo $|\langle M\rangle| - 1$ passi, quindi $T(\langle M\rangle)$ termina sempre. Se $T(\langle M\rangle)$ accetta, allora esiste una stringa w che è accettata da M in un numero di passi inferiore a $|\langle M\rangle|$, e quindi per definizione $\langle M\rangle \in \mathcal{D}$. Viceversa, se $T(\langle M\rangle)$ rifiuta, allora per tutte le stringhe $w \in S$ la computazione $M(w)$ non accetta entro $|\langle M\rangle| - 1$ passi. Per quanto osservato precedentemente non può esistere alcuna stringa $w \in \Sigma^*$ che è accettata da M in meno di $|\langle M\rangle|$ passi, pertanto $\langle M\rangle \notin \mathcal{D}$. Dunque, la TM T è un decisore per il linguaggio $\mathcal{D}$.

Esercizio 6 [7] Sia $\mathcal{L} = \{\langle M \rangle \mid M$ è una macchina di Turing deterministica tale che esiste almeno una stringa di input x per la quale $M(x)$ non termina$\}$. $\mathcal{L}$ è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Osserviamo innanzi tutto che non è possibile ricorrere al Teorema di Rice per risolvere l'esercizio, poiché la proprietà che caratterizza il linguaggio $\mathcal{L}$ non è propria del linguaggio delle macchine di Turing che appartengono a $\mathcal{L}$. Infatti, sia M una macchina di Turing che termina su ogni input e rifiuta almeno una stringa $\overline{x}$. È facile derivare da M una TM M' che per ogni input x dapprima controlla se $x = \overline{x}$, ed in tal caso entra in un ciclo senza fine; altrimenti, se $x \neq \overline{x}$, $M'(x)$ simula $M(x)$. Pertanto $L(M) = L(M')$, ma $\langle M \rangle \notin \mathcal{L}$ mentre $\langle M' \rangle \in \mathcal{L}$.

Per dimostrare che $\mathcal{L}$ è non decidibile è sufficiente mostrare una riduzione da un problema indecidibile; è naturale prendere in considerazione $\mathcal{A}_{\mathrm{TM}}$. Possiamo in effetti mostrare che $\mathcal{A}_{\mathrm{TM}} \leq_{\mathrm{T}} \mathcal{L}$ tramite la seguente TM con oracolo $\mathcal{L}$ che decide $\mathcal{A}_{\mathrm{TM}}$:

$M^{\mathcal{L}}$= "On input $\langle T, x \rangle$, where T is a TM and x is a string:

1. Build from $\langle T, x \rangle$ the following TM:

   M'= "On input y, where y is a string:
   (a) If $y \neq x$, then halt
   (b) Run T on input x"
   (c) Halt"

2. Ask the oracle whether $\langle M' \rangle \in \mathcal{L}$
3. If the answer is YES, reject
4. Otherwise, if the answer is NO, run T on input x
5. If $T(x)$ accepts, then accept; otherwise, reject"

Sia $\langle T, x \rangle \in \mathcal{A}_{\mathrm{TM}}$; dunque $M'(y)$ termina sempre (al passo (a) se $y \neq x$, al passo (c) se $y = x$). Perciò $\langle M' \rangle \notin \mathcal{L}$, e quando la computazione $M^{\mathcal{L}}(\langle T, x \rangle)$ interroga l'oracolo, questo risponderà NO. Dunque $M^{\mathcal{L}}(\langle T, x \rangle)$ simula $T(x)$ ed accetta poiché $T(x)$ accetta. Supponiamo invece che $\langle T, x \rangle \notin \mathcal{A}_{\mathrm{TM}}$. Esistono due casi: $T(x)$ rifiuta e $T(x)$ entra in un ciclo senza fine senza terminare. Nel caso in cui $T(x)$ rifiuta, l'oracolo risponde NO perché $T(x)$ termina, quindi non esiste alcun input sul quale M' non termina. $M^{\mathcal{L}}(\langle T, x \rangle)$ arriva dunque a simulare $T(x)$ e, quando quest'ultima computazione termina rifiutando, rifiuta. Se infine $T(x)$ non termina, $M'(x)$ non termina, dunque $\langle M' \rangle \in \mathcal{L}$ e l'oracolo risponde YES; perciò $M^{\mathcal{L}}(\langle T, x \rangle)$ rifiuta al passo 3. Riassumendo, $M^{\mathcal{L}}$ decide $\mathcal{A}_{\mathrm{TM}}$, e quindi $\mathcal{A}_{\mathrm{TM}} \leq_{\mathrm{T}} \mathcal{L}$; poiché $\mathcal{A}_{\mathrm{TM}}$ è indecidibile, resta dimostrato che anche $\mathcal{L}$ è indecidibile.