Fondamenti di informatica

UniVR - Dipartimento di Informatica

Fabio Irimie

1° Semestre 2025/2026

Indice

1 Introduzione

1.1 Cos'è l'informatica?

È una scienza che studia la calcolabilità, cioè cerca di capire che problemi si possono risolvere con un programma. Nasce dall'unione di matematica, ingegneria e logica. Il computer è solo uno strumento, mentre la matematica è il linguaggio con cui si creano algoritmi che permettono di risolvere i problemi.

1.2 Origini dell'informatica

Hilbert, nel 1900, si pose l'obiettivo di formalizzare tutta la matematica con un insieme finito e non contraddittorio di assiomi. Nel 1931, invece, Gödel dimostrò che l'informatica non potrà mai rappresentare tutta la matematica, perché ci saranno sempre proposizioni vere ma non dimostrabili tramite il calcolo. Ci si iniziò a chiedere se esistessero modelli di calcolo meccanici in grado di risolvere tutti i problemi. Nel 1936, Turing propose la macchina di Turing, una **sola** macchina programmabile in grado di risolvere tutti i problemi risolvibili:

$$\mathrm{Int}(P, x) = \begin{cases} P(x) & \text{se } P(x) \text{ termina} \\ \uparrow & \text{se } P(x) \text{ non termina} \end{cases}$$

dove P è un programma e x è un input. La macchina di Turing è un modello teorico di calcolatore, che non esiste fisicamente, ma è in grado di simulare qualsiasi altro calcolatore. Da questo modello deriva la concezione di calcolabilità, cioè se un problema è intuitivamente calcolabile, allora esiste un programma in grado di risolverlo.

Altri modelli di calcolo che sono stati proposti sono:

- Lambda-calcolo
- Funzioni ricorsive
- Linguaggi di programmazione (Turing-completi)

Definizione utile 1.1. La Turing-completezza è la proprietà di un linguaggio di programmazione di essere in grado di simulare una macchina di Turing, cioè di poter risolvere qualsiasi problema risolvibile.

1.2.1 Calcolabilità

Un programma è calcolabile se termina, ma non è detto che termini in un tempo ragionevole. Non esistono algoritmi che possono dire se un programma termina o meno. Questo è un esempio di problema non calcolabile.

I problemi non calcolabili sono infinitamente più numerosi di quelli calcolabili

2 Funzioni calcolabili

Un insieme è una proprietà ed è rappresentato da una funzione che indica se un elemento appartiene o meno all'insieme. I problemi da risolvere (in questo corso) hanno come soluzione una funzione sui naturali:

$$f : \mathbb{N} \to \mathbb{N}$$

Questo tupo di funzione è un **insieme** di associazioni input-output. Un esempio è la funzione quadrato:

$$f = \text{quadrato} = \{(0,0),(1,1),(2,4),(3,9),\ldots\} = \{(n, n^2) | n \in \mathbb{N}\}$$

Quindi f è un insieme di coppie in $\mathbb{N} \subseteq \mathbb{N} \times \mathbb{N}$ la cui cardinalità è: $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$. Di conseguenza la funzione è un sottoinsieme di $\mathbb{N} \times \mathbb{N}$:

$$f \subseteq \mathbb{N} \times \mathbb{N} \quad f \in \mathcal{P}(\mathbb{N} \times \mathbb{N})$$

Dove $\mathcal{P}(\mathbb{N} \times \mathbb{N})$ è l'insieme delle parti di $\mathbb{N} \times \mathbb{N}$, cioè l'insieme di tutti i sottoinsiemi di $\mathbb{N} \times \mathbb{N}$.

Il numero di funzioni è:

$$f : \mathbb{N} \to \mathbb{N} = |\mathcal{P}(\mathbb{N} \times \mathbb{N})| = |\mathcal{P}(\mathbb{N})|$$

Esempio 2.1. Un esempio di insieme delle parti per l'insieme $A = \{1,2,3\}$ è:

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$

E le cardinalità sono:

$$|A| = 3 \quad |\mathcal{P}(A)| = 8 = 2^3$$

$$\downarrow$$

$$|\mathbb{N}| = \omega < \underbrace{|\mathcal{P}(\mathbb{N})| = 2^{\omega}}_{\substack{\text{Insieme delle funzioni,} \\ \text{non numerabile}}} = |\mathbb{R}|$$

Si ha quindi che **l'insieme delle funzioni non è numerabile**

Ci si chiede se queste funzioni sono tutte calcolabili:

Definizione 2.1. Una funzione **intuitivamente calcolabile** è una funzione descrivibile attraverso un algoritmo, cioè una sequenza finita di passi discreti elementari.

2.1 Quante funzioni numerabili ci sono?

Consideriamo Σ come un alfabeto finito, cioè una seguenza di simboli utilizzabili per scrivere un programma o algoritmo.

$$\Sigma = \{s_1, s_2, s_3, \ldots, s_n\}$$

$$\downarrow$$

Programma $\subseteq$ sqeuenze di simboli in Σ

Con Σ^* si descrive l'insieme di tutte le sequenze finite di simboli in Σ, quindi l'insieme di tutti i possibili programmi è:

$$\text{Programmi} \subseteq \Sigma^*$$

Esempio 2.2. Se $\Sigma = \{a, b, c\}$ allora la sequenza di tutti i possibili simboli è:

$$\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, \ldots\}$$

dove ϵ è la stringa vuota.
La cardinalità di Σ^* è infinita numerabile (anche se Σ è finito).

$$|\Sigma^*| = |\mathbb{N}|$$

Si ha quindi che l'insieme dei programmi è numerabile:

$$|\text{Programmi in } \Sigma| \leqslant |\Sigma^*| = |\mathbb{N}|$$

e questo implica che l'insieme delle **funzioni calcolabili è numerabile**

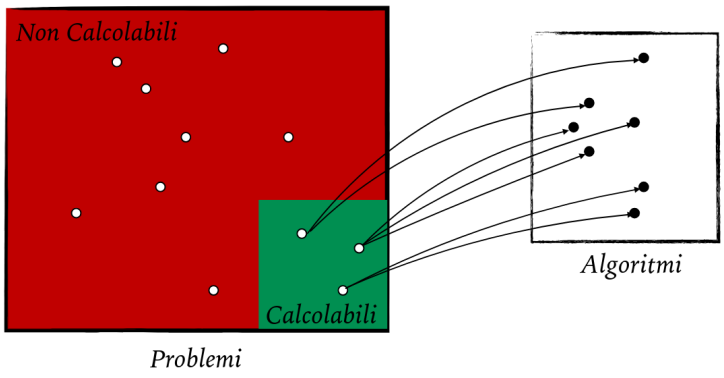

Figura 1: Cardinalità delle funzioni

Esempio 2.3. Prendiamo ad esempio la seguente funzione (serie di fibonacci):

$$f(n) \subseteq \mathbb{N} \times \mathbb{N}$$

Dove:

$$f(0) = 1, f(1) = 1, f(2) = 2$$
$$f(3) = 3, f(4) = 5, f(5) = 8$$
$$f(6) = 13, f(7) = 21, \ldots$$

Definiamo un algoritmo ricorsivo:

$$\begin{cases} f(0) = 1 = f(1) \\ f(x+2) = f(x+1) + f(x) \end{cases}$$

Trovare un algoritmo non è possibile per tutte le funzioni, ma solo per quelle calcolabili.

Nell'insieme delle funzioni calcolabili ci sono:

- Funzioni totali, cioè definite per ogni input $n \in \mathbb{N}$ e terminano sempre
- Funzioni parziali, cioè non definite per ogni $n \in \mathbb{N}$

2.2 Funzioni vs Insiemi

Una funzione può essere vista come un linguaggio $\mathcal{L}_f$ tale che:

$$f : \mathbb{N} \to \mathbb{N} \quad \leftrightarrow \quad \mathcal{L}_f = \left\{ 1^{f(x)} \, | x \in \mathbb{N} \right\} \quad \Sigma = \{1\}$$

Questo linguaggio permette di dire se un input appartiene o meno al linguaggio:

$$\sigma \in \Sigma^*$$

$$\downarrow$$

$$\begin{cases} \sigma \in \mathcal{L}_f & \text{se appartiene al linguaggio} \\ \sigma \notin \mathcal{L}_f & \text{se non appartiene al linguaggio} \end{cases}$$

Parliamo di insiemi invece che di funzioni dove gli elementi dell'insieme dipendono dal calcolo della funzione.

Esempio 2.4. Prendiamo ad esempio le seguenti funzioni:

- Funzione costante (Finite)

  $$f(x) = 2 \to \mathcal{L}_f \text{ è finito}$$

- Funzione lineare (Regolari)

  $$f(x) = 2x \to \mathcal{L}_f \text{ è infinito numerabile}$$

  C'è bisogno di una memoria finita per determinare se la stringa appartiene al linguaggio

- (Context free)

  $$f(\sigma) = \sigma\sigma^{\text{reverse}}$$

  $$\sigma = abc \quad \sigma^{\text{reverse}} = cba$$

  Per calcolare questa funzione c'è bisogno di una memoria illimitata, cioè non si può sapere a priori quanta ce n'è bisogno, è sufficiente uno stack.

- Decidibile

$$f(x) = x^2$$

Per calcolare questa funzione c'è bisogno di una memoria illimitata

3 Principio di induzione

Il principio di induzione è un meccanismo di definizione e dimostrazione che funziona **solo su insiemi infiniti**. Esistono due metodi di induzione:

- Induzione matematica
- Induzione strutturale

In questo corso tratteremo solo l'induzione matematica.

Un insieme A infinito con una relazione di ordine non riflessiva (senza l'uguale perchè l'elemento non è in relazione con sè stesso): $< : (A, <)$. $A = \mathbb{N}$ e $<$ è l'ordinamento stretto tra numeri naturali. La relazione di ordine deve essere **ben fondata**, quindi non devono esserci catene discendenti infinite, cioè una sequenza di elementi in ordine decrescente infinita:

$$a_0 > a_1 > a_2 > a_3 > \ldots \rightarrow \text{non ben fondata}$$

Una relazione di ordine riflessiva non è ben fondata, perchè esistono catene infinite:

$$a_0 \geqslant a_1 \geqslant a_2 \geqslant a_3 \geqslant a_3 \geqslant a_3 \geqslant \ldots \rightarrow \text{non ben fondata}$$

b minimale in $A : b \in A$ b è minimale se $\forall b' < b \;.\; b' \notin A$ Ad esempio: $\{1, 2, 3\}$ ha come minimali (di contenimento) $\{1, 2\}$ e $\{2, 3\}$.

Definizione 3.1 (Principio di induzione)**.** Se A è un insisme ben fondato (con ordinamento $<$), e Π è una proprietà definita sugli elementi di $A : \Pi \subseteq A$, allora:

$$\forall a \in A \;.\; \underbrace{\Pi(a)}_{\substack{a \text{ soddisfa } \Pi}} \iff \underbrace{\forall a \in A \;.\; [[\forall b < a \;.\; \Pi(b)] \Rightarrow \Pi(a)]}_{\substack{\text{Se dimostriamo } \Pi \text{ per ogni elemento più piccolo di } a, \\ \text{allora } \Pi \text{ vale anche per } a}}$$

Consideriamo come caso base gli elementi minimali di A:

$$\text{Base}_A = \{a \in A \mid a \text{ minimale}\}$$

Se si dimostra che Π vale per tutti gli elementi minimali di A (la base):

$$\overbrace{\underbrace{\forall a \in \text{Base}_A \;.\; \Pi(a)}_{\substack{\text{Dimostriamo } \Pi \text{ per ogni} \\ \text{elemento minimale di } A}}}^{\text{Base}} \wedge \underbrace{\forall a \in A \setminus \text{Base}_A}_{\text{Passo induttivo}} \;.\; \underbrace{\forall b < a \;.\; \Pi(b)}_{\text{Ipotesi induttiva}} \quad \underbrace{\Rightarrow \Pi(a)}_{\text{Tesi da dimostrare}}$$

Esempio 3.1. Dimostriamo che:

$$\forall n \in \mathbb{N} \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

L'insieme è:

$$A = \mathbb{N} \setminus \{0\} = \{1, 2, 3, \ldots\}$$

- **Base**

$$\text{Base}_A = \{1\}$$

  - dimostriamo la base

$$\sum_{i=1}^{1} i = n(n+1)/2 = 1(1+1)/2 = 1$$

- **Passo induttivo**: Prendo $n \in \mathbb{N}$

  - Ipotesi induttiva, cioè per ogni $m < n$ vale la proprietà:

$$\forall_{\substack{m \\ \in \mathbb{N}}} < n \; . \; \sum_{i=1}^{m} i = \frac{m(m+1)}{2}$$

    Dobbiamo dimostrare la proprietà per n:

$$\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n$$

$n - 1 < n$ quindi vale l'ipotesi induttiva

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{(n-1)n}{2}$$

$$\downarrow$$

$$\begin{aligned}
\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n \\
&= \frac{(n-1)n}{2} + n \\
&= \frac{(n-1)n + 2n}{2} \\
&= \frac{n^2 - n + 2n}{2} \\
&= \frac{n(n+1)}{2} \quad \square
\end{aligned}$$

È quindi dimostrato che:

$$\forall n \in \mathbb{N} \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

3.1 Linguaggi formali

Definizione 3.2. Un linguaggio formale è un insieme di stringhe costruite su un alfabeto finito Σ .

Solitamente un linguaggio formale $\mathcal{L}$ è un sottoinsieme di Σ^*, tipicamente infiniti, ma non necessariamente:

$$\mathcal{L} \subseteq \Sigma^*$$

I linguaggi sono divisi in:

- Linguaggi finiti
- Linguaggi regolari, il modello utilizzato è l'automa a stati finiti.

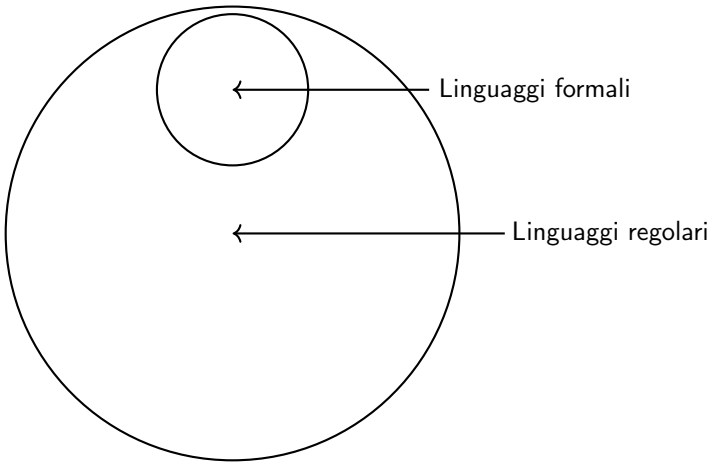

Figura 2: Linguaggi formali e linguaggi regolari

3.2 Linguaggi regolari (automi a stati finiti)

Il meccanismo più semplice per una memoria finita è l'automa a stati finiti

Consideriamo il linguaggio:

$$\mathcal{L}_f = \left\{1^2 n \,|\, n \in \mathbb{N}\right\}$$

ha bisogno di due stati q_0 e q_1. Lo stato q_0 rappresenta l'informazione di essere di lunghezza pari, mentre lo stato q_1 rappresenta l'informazione di essere di lunghezza dispari.

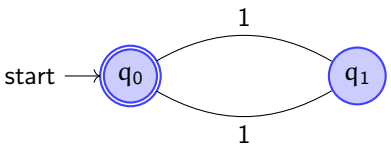

Figura 3: Automa a stati finiti per il linguaggio $\mathcal{L}_f = \left\{1^{2n} \,|n \in \mathbb{N}\right\}$

L'automa a stati finiti **deterministico** è definito come una tupla:

$$M = (Q, \Sigma, \delta, q_0, F)$$

dove:

- Q è un insieme **finito** di stati. Ogni stato rappresenta un'informazione
- Σ è un insieme **finito** di simboli (alfabeto). Ogni simbolo è un elemento atomico che posso leggere e che compone le stringhe da riconoscere
- $q_0 \in Q$ è uno stato e identifica lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati finali (di accettazione)
- $\delta : Q \times \Sigma \to Q$ È una **funzione di transizione** che dato uno stato e un simbolo, restituisce lo stato successivo ed è come se fosse una tabella che associa ad ogni coppia (stato, simbolo) uno stato:

$\Sigma \setminus Q$	q_0	q_1
1	q_1	q_0

  La funzione deve essere **totale**, cioè deve essere definita per ogni coppia $(q, a) \in Q \times \Sigma$, quindi la tabella deve essere completa.

- $\hat{\delta} : Q \times \Sigma^* \to Q$ Descrive lo stato che raggiungono leggendo una sequenza di simboli

$$\begin{cases} \hat{\delta}(q, \epsilon) = q \\ \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a) \end{cases} \qquad w \in \Sigma^*, \quad a \in \Sigma$$

  È quindi la **chiusura transitiva** di δ

3.2.1 Come si dimostra che un linguaggio è regolare?

Esempio 3.2. Prendiamo in considerazione il seguente linguaggio:

$$L = \{\sigma \,|\sigma \text{ contiene almeno due } 1\}$$

$$\Sigma = \{0, 1\}$$

Con le seguenti stringhe si ha:

- $\sigma = 011 \in \mathcal{L}$
- $\sigma = 1000100 \in \mathcal{L}$

- $\sigma = 00010 \notin \mathcal{L}$

L'informazione che codifica lo stato iniziale deve essere coerente con ε (stringa vuota)

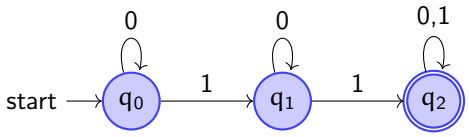

Figura 4: Automa a stati finiti per il linguaggio $\mathcal{L}$

Un lingauggio L è riconosciuto da $M = (Q, \Sigma, \delta, q_0, f)$ (DFA, Deterministic Finite Automaton) se: $L = L(M)$ dove $L(M)$ è il linguaggio di M definito come:

$$L(M) = \left\{ \sigma \in \Sigma^* \,\middle|\, \hat{\delta}(q_0, \sigma) \in F \right\}$$

Cioè sono tutte le stringhe che partendo da q_0 fanno raggiungere uno stato finale.

Definizione 3.3. Per dimostrare che L è regolare dobbiamo costruire M (almeno un M) e **dimostrare che** $L = L(M)$

$L = L(M)$ è un uguaglianza insiemistica e si dimostra con due contenimenti:

$$L = L(M) \equiv L \subset L(M) \wedge L(M) \subset L$$

- Se un elemento si trova nel primo insieme, allora si trova anche nel secondo

$$L \subseteq L(M) \equiv \sigma \in L \Rightarrow \sigma \in L(M) \equiv \sigma \in L \Rightarrow \hat{\delta}(q_0, \sigma) \in F$$

- Se un elemento si trova nel secondo insieme, allora si trova anche nel primo

$$L(M) \subseteq L \equiv \sigma \in L(M) \Rightarrow \sigma \in L \equiv \hat{\delta}(q_0, \sigma) \in F \Rightarrow \sigma \in L$$

  o per contrapposizione:

$$\sigma \notin L \Rightarrow \hat{\delta}(q_0, \sigma) \notin F$$

Questo dimostra che il linguaggio è regolare perchè è riconosciuto da un automa.

Esempio 3.3. Riprendendo l'esempio precedente:

$$L = \{\sigma \,|\, \sigma \text{ contiene almeno due } 1\}$$

$$\Sigma = \{0, 1\}$$

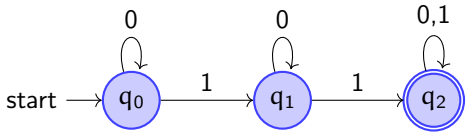

Figura 5: Automa a stati finiti per il linguaggio $\mathcal{L}$

Dimostriamo per induzione sulla lunghezza delle stringhe $\sigma \in \Sigma^*$ che se $x \in L$ allora $\hat{\delta}(q_0, x) \in F$ e se $x \notin L$ allora $\hat{\delta}(q_0, x) \notin F$.

$|\sigma| = 0$ non è **mai** sufficiente come base, ma è eventualmente la base **solo** per una delle due dimostrazioni. Bisogna quindi prendere la lunghezza più piccola che permette di avere sia $\sigma \in L$ che $\sigma \notin L$, in questo caso è $|\sigma| = 2$. Per ogni σ tale che $|\sigma| < 2 \quad \sigma \notin L$ perchè non può contenere due 1 e non è riconosciuta da M dove il primo stato finale è raggiunto leggendo almeno due simboli.

$$\varepsilon \in L \quad \varepsilon \notin L$$

- **Base**: Controlliamo ogni stringa di lungheezza minima nel linguaggio per provare il caso base

$$\begin{cases} \sigma = 11 \in L \text{ e } \hat{\delta}(q_0, 11) = q_2 \in F \\ \sigma = 10 \notin L \text{ e } \hat{\delta}(q_0, 10) = q_1 \notin F \\ \sigma = 01 \notin L \text{ e } \hat{\delta}(q_0, 01) = q_1 \notin F \\ \sigma = 00 \notin L \text{ e } \hat{\delta}(q_0, 00) = q_0 \notin F \end{cases}$$

- **Passo induttivo**: Dimostriamo l'**ipotesi induttiva**, cioè la tesi con un limite fissato:

$$\forall \sigma \in \Sigma^* \,.\, |\sigma| \leqslant n \,.\, \begin{cases} \sigma \in L \Rightarrow \hat{\delta}(q_0, \sigma) \in F \\ \sigma \notin L \Rightarrow \hat{\delta}(q_0, \sigma) \notin F \end{cases}$$

  Vogliamo dimostrare se $|\sigma| = n + 1$ (la successiva stringa che posso considerare) allora vale $\begin{cases} \sigma \in L \Rightarrow \hat{\delta}(q_0, \sigma) \in F \\ \sigma \notin L \Rightarrow \hat{\delta}(q_0, \sigma) \notin F \end{cases}$

  Dimostrazione:

$$|\sigma| = n + 1 \rightarrow \sigma = \sigma' 1 \ \vee \ \sigma = \sigma' 0$$

  – Supponiamo che σ appartiene al linguaggio e termini con 1:

$$\sigma \in L \wedge \sigma = \sigma' 1 \rightarrow$$

  * Se $\sigma' \in L$ applico l'ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_2$$

$$\begin{aligned}\hat{\delta}(q_0, \sigma) &\overset{\sigma=\sigma'1}{=} \hat{\delta}(q_0, \sigma'1) \\ &= \delta(\hat{\delta}(q_0, \sigma'), 1) \\ &= \delta(q_2, 1) = q_2 \in F\end{aligned}$$

* Se $\sigma' \notin L$ allora σ' contiene esattamente un 1:

$$\hat{\delta}(q_0, \sigma') = q_1$$

$$\hat{\delta}(q_0, \sigma'1) = \delta(q_1, 1) = q_2$$

– Supponiamo che σ appartiene al linguaggio e termini con 0:

$$\sigma \in L \wedge \sigma = \sigma'0$$

Per definizione di L abbiamo che

$$\sigma \in L \wedge \sigma = \sigma'0 \Rightarrow \sigma' \in L$$

Dimostriamo l'ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_2$$

allora

$$\begin{aligned}\hat{\delta}(q_0, \sigma) &= \hat{\delta}(q_0, \sigma'0) \\ &= \delta(\hat{\delta}(q_0, \sigma'), 0) \\ &= \delta(q_2, 0) = q_2 \in F\end{aligned}$$

– Supponiamo che σ non appartiene al linguaggio e termini con 1:

$$\sigma \notin L \wedge \sigma = \sigma'1 \text{ (ha esattamente un 1)}$$

$$\Downarrow$$

$$\sigma \notin L \text{ non ha 1}$$

– Supponiamo che σ non appartiene al linguaggio e termini con 0:

$$\sigma \notin L \wedge \sigma = \sigma'0$$

$$\Downarrow$$

$$\sigma' \notin L$$