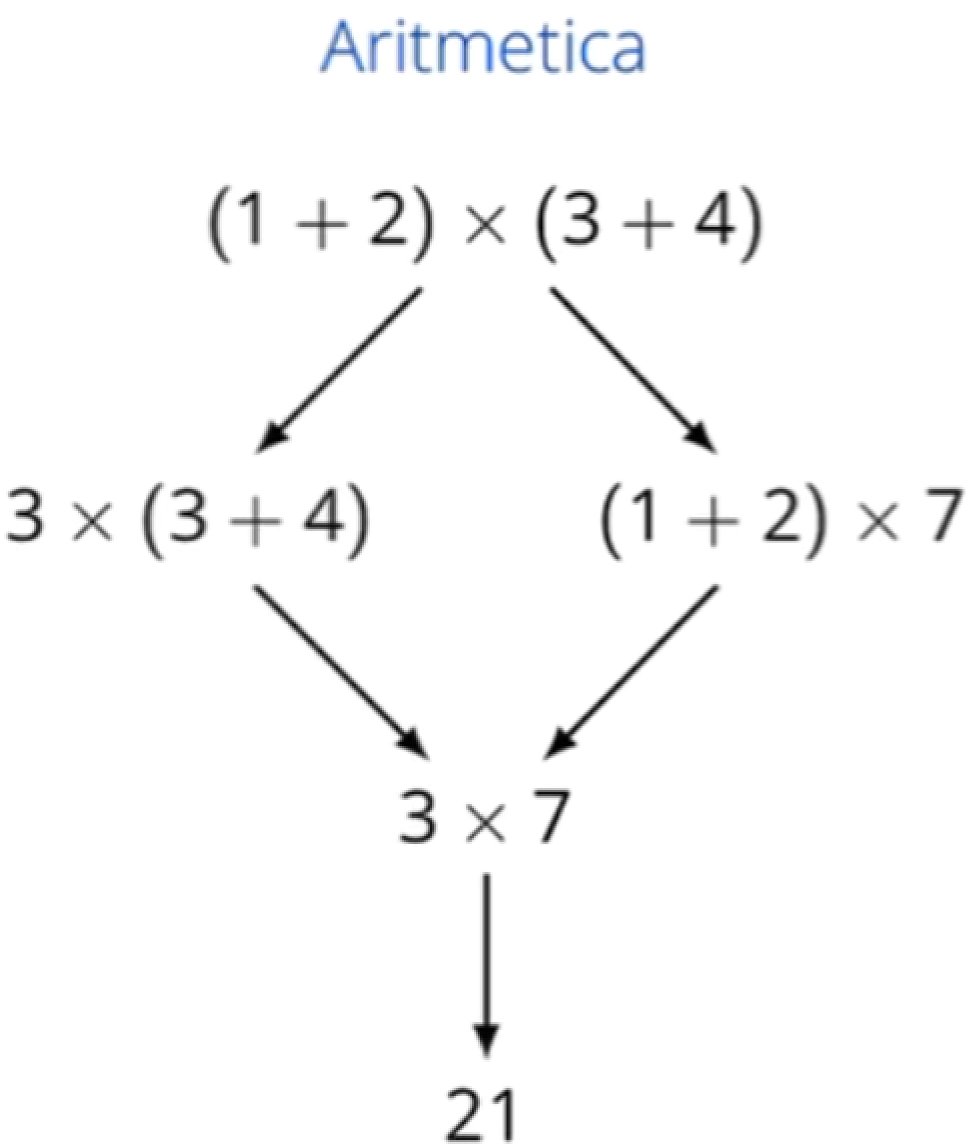
# Confluenza

Dal momento che, data una $\beta$-redex, esistono **più modi** per ridurla allora:

**Teorema (confluenza)**

*Se $M \Rightarrow N_1$ e $M \Rightarrow N_2$ allora esiste $N$ tale che $N_1 \Rightarrow N$ e $N_2 \Rightarrow N$.*

## Esempio



## Forma normale

**Definizione (forma normale)**

*Diciamo che M è in **forma normale** se non può più essere ridotto, ovvero se **non** esiste N tale che $M \rightarrow N$. In tal caso scriviamo $M \not\rightarrow$.*

**Corollario**

*La forma normale di M, se esiste, è unica (a meno di $\alpha$-conversioni).*

**Dimostrazione.**

Supponiamo che $M$ abbia due forme normali $N_1$ ed $N_2$, ovvero $M \Rightarrow N_1 \not\rightarrow$ e $M \Rightarrow N_2 \not\rightarrow$. Per il teorema di confluenza esiste $N$ tale che $N_1 \Rightarrow N$ e $N_2 \Rightarrow N$. Siccome $N_1$ ed $N_2$ sono in forma normale, deve essere $N_1 = N = N_2$ (a meno di $\alpha$-conversioni). □

## Strategie di riduzione

Due principali strategie:

1. **Ordine applicativo:**
    - Redex più a sinistra e **più interno**
    - $(\lambda x.\, x)\underline{((\lambda y.\, y)z)} \rightarrow \underline{(\lambda x.\, x)z} \rightarrow z$
        - In questo caso, riduciamo prima la redex più interna, quindi riduciamo $(\lambda y.\, y)z$ che diventa $z$, e poi applico $z$ alla redex più esterna, ottendendo $z$
2. **Ordine normale:**
    - Redex più a sinistra e **più esterno**
    - $\underline{(\lambda x.\, x)((\lambda y.\, y)z)} \rightarrow \underline{(\lambda y.\, y)z} \rightarrow z$
        - In questo caso, riduciamo la redex più esterna, quindi riduco tutta la lambda espressione.

Applico $((\lambda y.\, y)z)$ a $(\lambda x.\, x)$ ottendendo $(\lambda y.\, y)z$ , infine ci applico $z$

## Eager evaluation e Lazy evaluation

Quasi tutti i linguaggi di programmazione, specialmente quelli **imperativi**, usano l'**ordine applicativo**, ovvero quando passiamo un parametro ad una funzione, prima riducono il parametro ad una **forma normale**, e solo **poi** lo applicano alla funzione.
Questi linguaggi sono detti **zelanti** o **eager.**

Al contrario, alcuni linguaggi, tra cui **Haskell**, prima sostituiscono il parametro nel corpo della funzione, e solo poi, **quando serve**, lo riducono ad una **forma normale.**
Questi linguaggi sono detti **pigri** o **lazy**

Entrambe le strategie, a causa del **Teorema della confluenza**, *confluiscono* verso lo stesso risultato, tuttavia **non sono equivalenti.**

# Esempio in cui conviene essere *pigri*

## Ordine applicativo

$$(\lambda x.y)\ (\underline{(\lambda z.z)\ (\lambda z.z)}) \to \underline{(\lambda x.y)\ (\lambda z.z)} \to y$$

## Ordine normale

$$\underline{(\lambda x.y)\ ((\lambda z.z)\ (\lambda z.z))} \to y$$

## Osservazioni

- l'argomento $x$ non è **mai usato**
- l'ordine normale è più efficiente perché non lo valuta

# Esempio in cui conviene essere *zelanti*

## Ordine applicativo

$$
\begin{aligned}
(\lambda x.x\,x)\,(\underline{(\lambda y.y)\,(\lambda z.z)}) &\to \underline{(\lambda x.x\,x)\,(\lambda z.z)} \\
&\to \underline{(\lambda z.z)\,(\lambda z.z)} \\
&\to \lambda z.z
\end{aligned}
$$

## Ordine normale

$$
\begin{aligned}
\underline{(\lambda x.x\,x)\,((\lambda y.y)\,(\lambda z.z))} &\to (\underline{(\lambda y.y)\,(\lambda z.z)})\,((\lambda y.y)\,(\lambda z.z)) \\
&\to \underline{(\lambda z.z)\,((\lambda y.y)\,(\lambda z.z))} \\
&\to \underline{(\lambda y.y)\,(\lambda z.z)} \\
&\to \lambda z.z
\end{aligned}
$$

## Osservazioni

- l'argomento $x$ è usato **due volte**
- l'ordine applicativo è più efficiente perché lo valuta una volta sola
- ottimizzare l'ordine normale: **salvare** il risultato della prima valutazione dell'argomento e **riusarlo** per le valutazioni successive

### Normalizzazione ![[teorema_normalizzazione.png]] Ciò implica che: - Se esiste una **forma normale** di una espressione, posso ottenerla riducendo l'espressione in **ordine normale** - Questa proprietà **non vale** per l'**ordine applicativo** ![[ordine_applicativo_loop.png]] $\omega \space \omega$ è la funzione **autoapplicazione**