Confluence

It doesn't matter in what order we do reductions

If t is a λ-term and

$t \rightsquigarrow^* t_1$
$t \rightsquigarrow^* t_2$

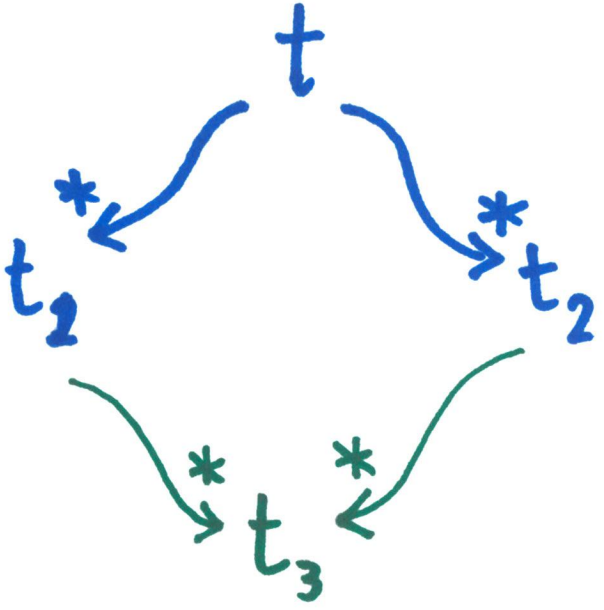
} I can reduce t to two different terms using different reduction sequences

Then there is a term t_3 s.t.

$t_1 \rightsquigarrow^* t_3$
$t_2 \rightsquigarrow^* t_3$

} I can find a common reduct

In pictures:



Consequence:

Normal Forms are Unique

A term can have only one normal form

(But some terms don't have one)

Evaluation Strategies

for λ-terms

How do we choose which redex to reduce

- **Full β-reduction**

  reduce any redex you want

- **Normal order**

  reduce the leftmost redex (top redex in AST)

  normalizing strategy:

  if there is a normal form, it will find it

- **Call-by-name**

  - If the term is an application

    $(f\ u)$

    reduce f until it becomes an abstraction

    $f \rightsquigarrow^* \lambda x.t$

    then reduce the main redex

    $(\lambda x.t)\ u \rightsquigarrow t[x := u]$

  - Never reduce under abstraction

    $\lambda f.t$ is considered a value

  Lazy evaluation (Haskell)

  The same with term-graphs and sharing

• Call-by-value

In an application

$$(f\ u)$$

- reduce f to an abstraction

$$f \rightsquigarrow^* \lambda x.t$$

- reduce u to a value

$$u \rightsquigarrow^* v$$

Then reduce the main redex

$$(\lambda x.t)\,v \rightsquigarrow t[x := v]$$

Idea:

- in call-by-name
  the argument u is passed to the function without evaluation:
  we use its "name", ie its syntactic expression
- in call-by-value
  we evaluate the argument to a value before passing it to the function