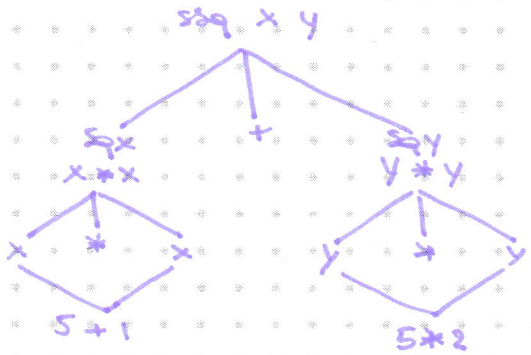
> Modèle de substitut° : Remplacement des param formels par les arguments correspondant.

> Operateurs spéciaux
- → contexte
  - pb : eval non strict
  - solut° : opé primitif spé
- → corollaire
  - quotat° : empêcher l'evaluat° d'une expr.
    - → meta pgmt°
    - → reflexivité
    - → code ⇔ données
    - → macros
    - → calcul symbolique
    - → propagat° de la nott
    - → inférence sur les prédicats

Evaluat° paresseuse → ordre normal
on n'evalue rien à l'avance, seulemt qd on a besoin

> Modèle de substitut° : pareil que stricte
MAIS → arg non eval
→ eval à la demande (aussi vrai pr agregat)
- Cas theorique : reeval pb x de la ie expr
- Cas pratique : en fait non, eval unique → graphe d'evaluat°

sqq (5+1)(5*2) = sq(5+1) + sq(5*2) = (5+1)*(5+1) + (5*2)*(5*2)
= 6*6 + 10*10 = 36 + 100 = 136



⚠ fonctionnel pur uniquemt ! → Pas d'effet de bord

Espce sur integralité du lgg :
- pattern matching : 1° eq. qui matche et on s'arrête (Haskell)
  conservat° de la paresse : eval seulemt arg nécessaires.

Strict VS lazy
- → thm de church Rosser : les 2 sont equivalent
- → strict : interieur gauche, seul en impur
- → lazy : exterieur gauche, abstract° supplémentaires

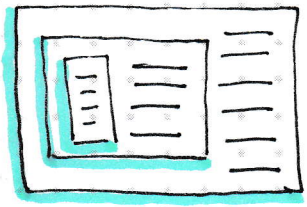
ENVIRONNEMENT

Env global insuffisant.

Contxt locaux implicites :
- arg des fct (α conversion, imbricat°)

Contxt locaux explicites :
- données locales (eviter redondance d'eval)
- fct locales (eviter pollut° des espaces de nom)

> Bloc : ensemble de liaisons { nom = expr }
> env d'eval : struct de blocs imbriqués.
> var liée : def ds contxt local
> var libre : non def localement
> scoping : capture d'une var libre



Fct locales : ce sont des obj de 1° classe
- let & where en Haskell
- labels en Lisp

Scoping → lexical : recherche ds l'env de définit°. Recherche ds code source.
→ dynamique : recherche ds l'env d'appel

Fermeture lexicale : combinaison entre fct et env de def
- opé générq par fct anonymes
- créat° dynq de fct à état local