# 関数・論理型プログラミング実験
# 論理型演習第4回
# （通算第14回）

松田 一孝

TA: 武田広太郎 寺尾拓

# これまでの流れ

- 第1回 (6/24)
  - Prologの使い方
- 第2回 (7/1)
  - Prologの評価メカニズム
- 第3回 (7/8)
  - いろいろな探索
- 第4回 (7/15)
  - 関数論理型言語Curry

# Curry?

- 関数論理型言語
  - 関数型言語のように
    高階関数を利用したプログラミング
  - 論理型言語のように
    ビルトインサーチ，単一化を利用可
- Needed Narrowingに基づく操作的意味
  - SLD導出に近いが，
    こちらは名前呼び評価に基づく
- 名前は論理学者Haskell Curryより
  - Haskellもそうなのはいうまでもない

# Curryの特徴

- Haskellに近い構文
  + 単一化

last (_ ++ [x]) = x
とも書ける

```
last xs | _ ++ [x] =:= xs = x where x free
```

```
$ cyi test.curry
...
test> last [1,2,3]
3
More solutions ? [Y(es)/n(o)/a(ll)]
```

cf. "last" in Haskell

```
last [a]    = a
last (_:x) = last x
```

# 起動方法

- MCCでは，
  - cyiがインタプリタ（ghciに相当）
    - よくある起動方法
      - cyi
      - cyi FILENAME.curry
    - 使い方はghciとほぼ同じ，起動してからも
  - cycがコンパイラ
- 処理系によって異なるので注意

# インタプリタ（1/1）

- ghciと同様に式を評価可

```
$ cyi
   _____ __ __
  / ___/ / / /_
 / /    / / / | |
/ /___ / / /  | |
\____//_/   |_|

Muenster Curry Compiler
Version 0.9.11, Copyright (c) 1998-2007

Type :h for help

Prelude> 1 + 2
3
Prelude>
```

  - let f = 1等は書けない
    - cf. ghciではdo記法の中に書けるものが書ける

# インタプリタ（2/2）

- Prologのように問い合わせ

x++[2]++y = [1,2,3]
となるx, yは？

```
test> x ++ [2] ++ y =:= [1,2,3] where x,y free
{x = [1], y = [3]}
More solutions? [Y(es)/n(o)/a(ll)] a

test> x ++ [2] ++ y =:= [1,2,2,3] where x,y free
{x = [1], y = [2,3]}
More solutions? [Y(es)/n(o)/a(ll)] a
{x = [1,2], y = [3]}
```

# 柔軟な宣言

y++c++z=:=xとなるy,z
について(y,z)が返り値

```
separateBy c x
  | y++c++z =:= x = (y,z) where y,z free
```

この宣言の中でy,zは自由
y++c++z =:= xはただの式であることに注意

```
test> separateBy "," "a,b,c"
("a","b,c")
More solutions? [Y(es)/n(o)/a(ll)] a
("a,b","c")
```

# 構文

- f $p_1$ ... $p_n$ | e = e'
  - eはSuccess型の式
    - (=:=)はただのa -> a -> Successの演算子
  - eが成功したならば，e'を評価
    - eが成功するか確認する段階でeやe'中の自由変数は具体化されるかもしれない

```
separateBy c x
  | y++c++z =:= x = (y,z) where y,z free
```

  - cf. Haskellのガード（未説明）
    - eはBool型の式

# Success型を扱う式

- success :: Success
  - 成功を表す

- (&) :: Success -> Success -> Success
  - $x_1$&$x_2$は「$x_1$と$x_2$が両方成功」
    - $x_1$と$x_2$は並行に評価されうる
      (が処理系によってはされない)
      - $x_1$から評価されると止まらない式も
        止まることがある

# Success型を扱う式

- (=:=) :: a -> a -> Success
  - 等しいという制約
  - 型毎に以下のように定義されている気分
    - [] =:= [] = success
      (a:x) =:= (b:y) = (a=:=b) & (x=:=y)
    - 注：Curryにtype classは（今は）ない

- (&>) :: Success -> a -> a
  - c &> xは制約cを解いてxを評価
    - c &> x | c = x
      と定義されているように動作

# 非決定的な計算

```
data Family = Kobo | Koji | Sanae | Iwao | Mine

male = Kobo
male = Koji
male = Iwao

parent(Kobo) = Koji
parent(Kobo) = Sanae

father x | parent(x) =:= y & male =:= y = y
  where y free
```

ただ縦にならべるだけ

```
test> father Kobo
Koji
More solutions? [Y(es)/n(o)/a(ll)] y
No more solutions
```

# 便利な演算

- ?
  - x ? _ = x
    _ ? y = y
    - letやwhere内の利用は注意が（後述）

```
data Family = Kobo | Koji | Sanae | Iwao | Mine

male = Kobo ? Koji ? Iwao

parent(Kobo) = Koji ? Sanae

father x | parent(x) =:= y & male =:= y = y
  where y free
```

# 注意

- オーバラップするパターンには気をつけよう

```
comma []   = ""
comma [x]  = x
comma (x:xs) = x++",  "++comma xs
```

```
test> comma ["A","B","C"]
"A, B, C"
More solutions? [Y(es)/n(o)/a(ll)] a
"A, B, C, "
```

# Choiceのタイミング

# Choiceはいつ起きる？

- 2つの場合
  - call-time choice
    - bindされた時点でchoiceが起こる
      - double coinの返り値は0と2
      - 値呼び的
  - run-time choice
    - 評価が終った時点でchoiceが起こる
      - double coinの返り値は0と1と2
      - 名前呼び的

```
coin = 0 ? 1
double x = x+x
```

# CurryのChoice

- 場所によって異なる
  - top-level定義
    - run-time choice
      - 束縛時にchoiceは起きない
  - それ以外（whereやletも）
    - call-time choice
      - 束縛時にchoiceが起きる
        - 正確にはlazyなので違う

doubleC1のみ
1を返しうる

```
doubleC1 = coin + coin
doubleC2 = c + c where c = coin
doubleC3 = double coin
```

# 注意

- call-type choice≠積極評価
  - choiceは評価されて初めて起きる
    - let x = x?1 in 1は止まる
  - 一旦評価されたら，
    その変数が指している値が置き換わる
    （ように考えられる）
    - let x = 0?1 in (x, x)のxが評価されたら，
      (0, 0)か(1, 1)かになる

# 注意

- η展開は意味を保存しない
  - η展開：M → λx.M xにすること

```
zero 0 = success
one  1 = success
goal f | f x & f y = x + y
   where x, y free
```

```
fcoin1 = zero
fcoin1 = one
```

goal fcoin1は
0か2を返す

```
fcoin2 x = zero x
fcoin2 x = one x
```

goal fcoin2は
0か1か2を返す

MCCのUser's Guideより抜粋

# Curryの探索

# DFS

ただし，処理系によってはsearch treeを
取り出し別の探索が可能

cf. MCCのAllSolutionsモジュール

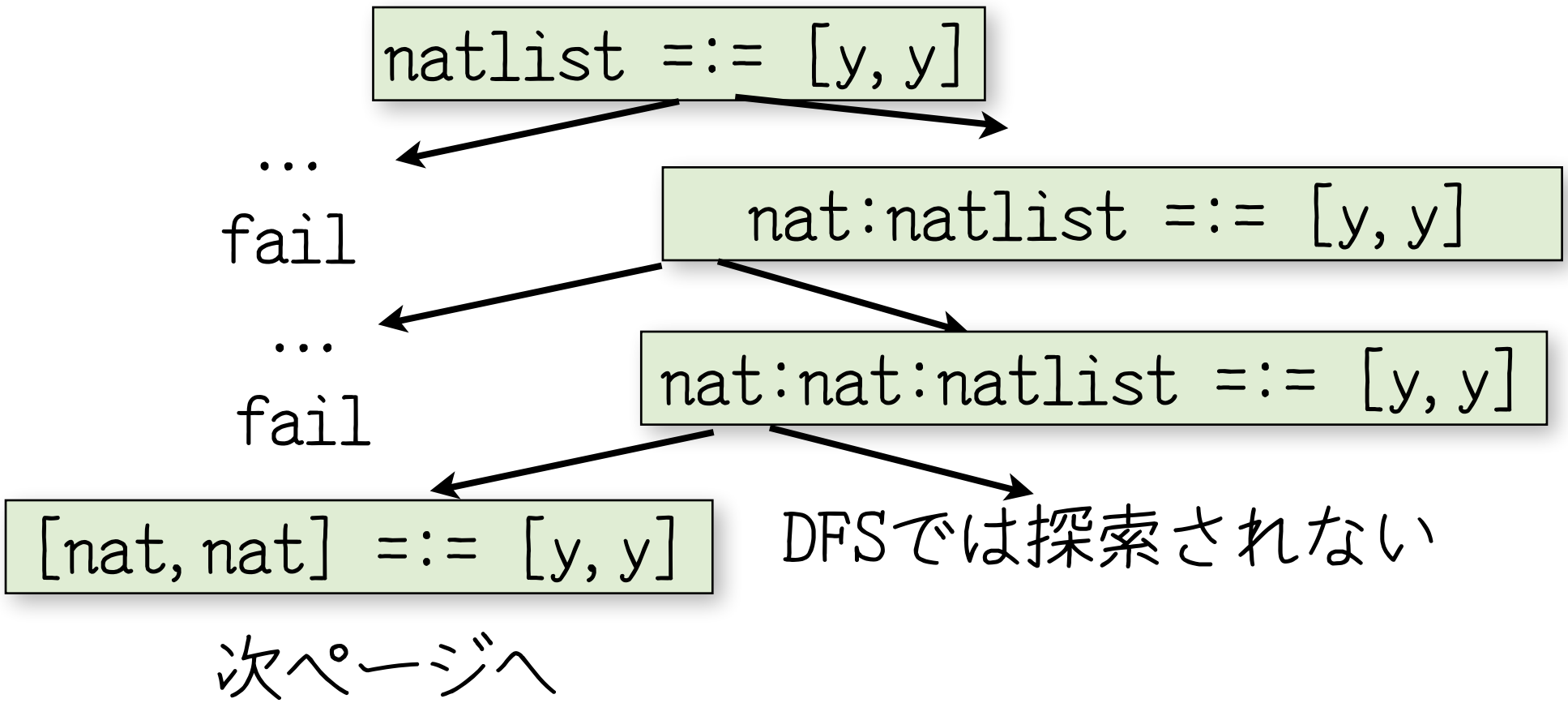
# Prologとの大きな違い

- Curryは（非決定的な）関数を持つ
  - Prologは述語だけ
  - それらの関数は遅延評価される
    - 探索の停止性や効率に影響することがある（課題の問5参照）
    - コピーされた変数は同期して展開される
      - call-time choiceの実現のため

# 例

```
data Nat = Z | S Nat
```

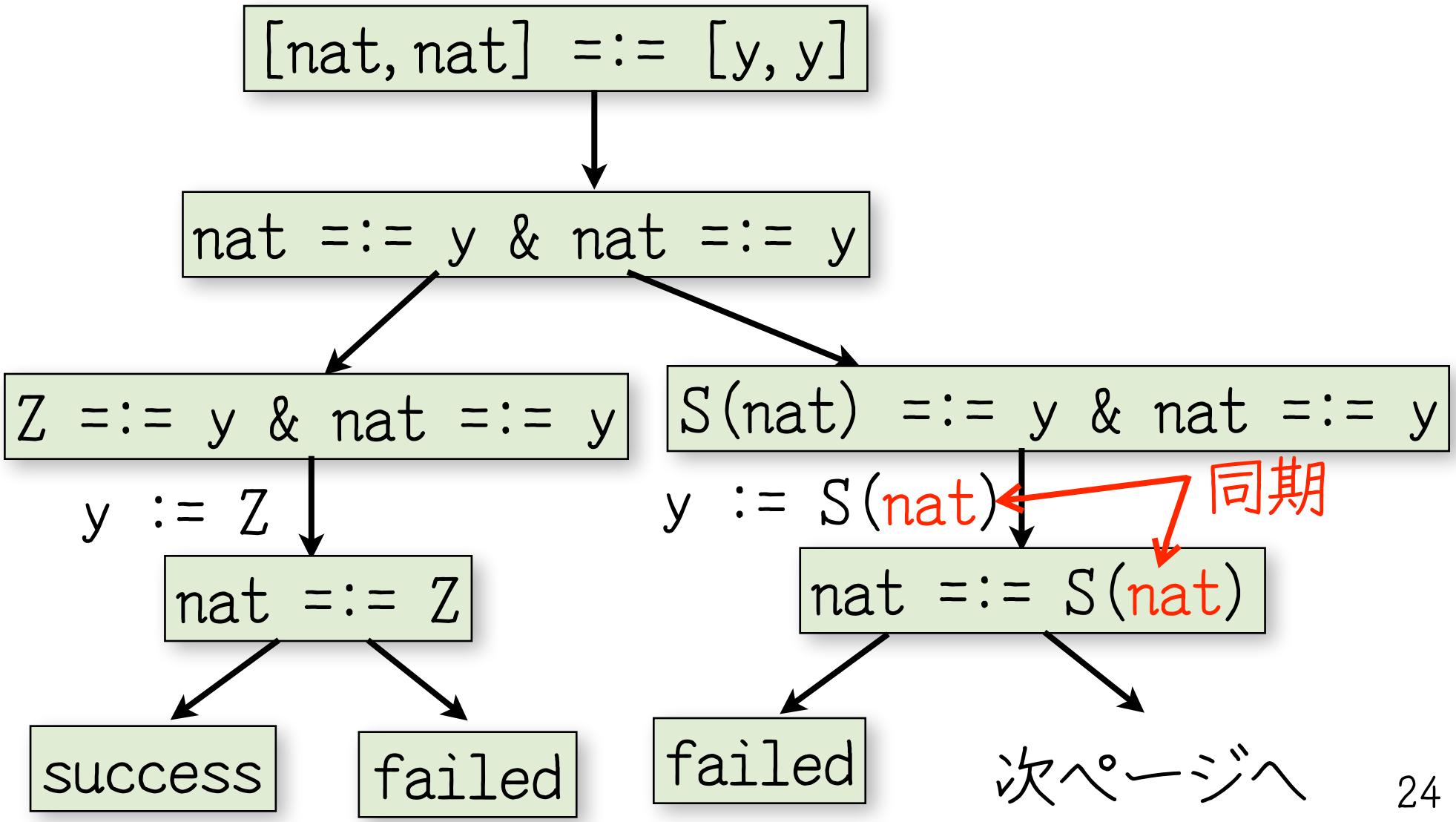
```
nat = Z
nat = S nat
natlist = []
natlist = nat : natlist
```



# 例

```
data Nat = Z | S Nat
```

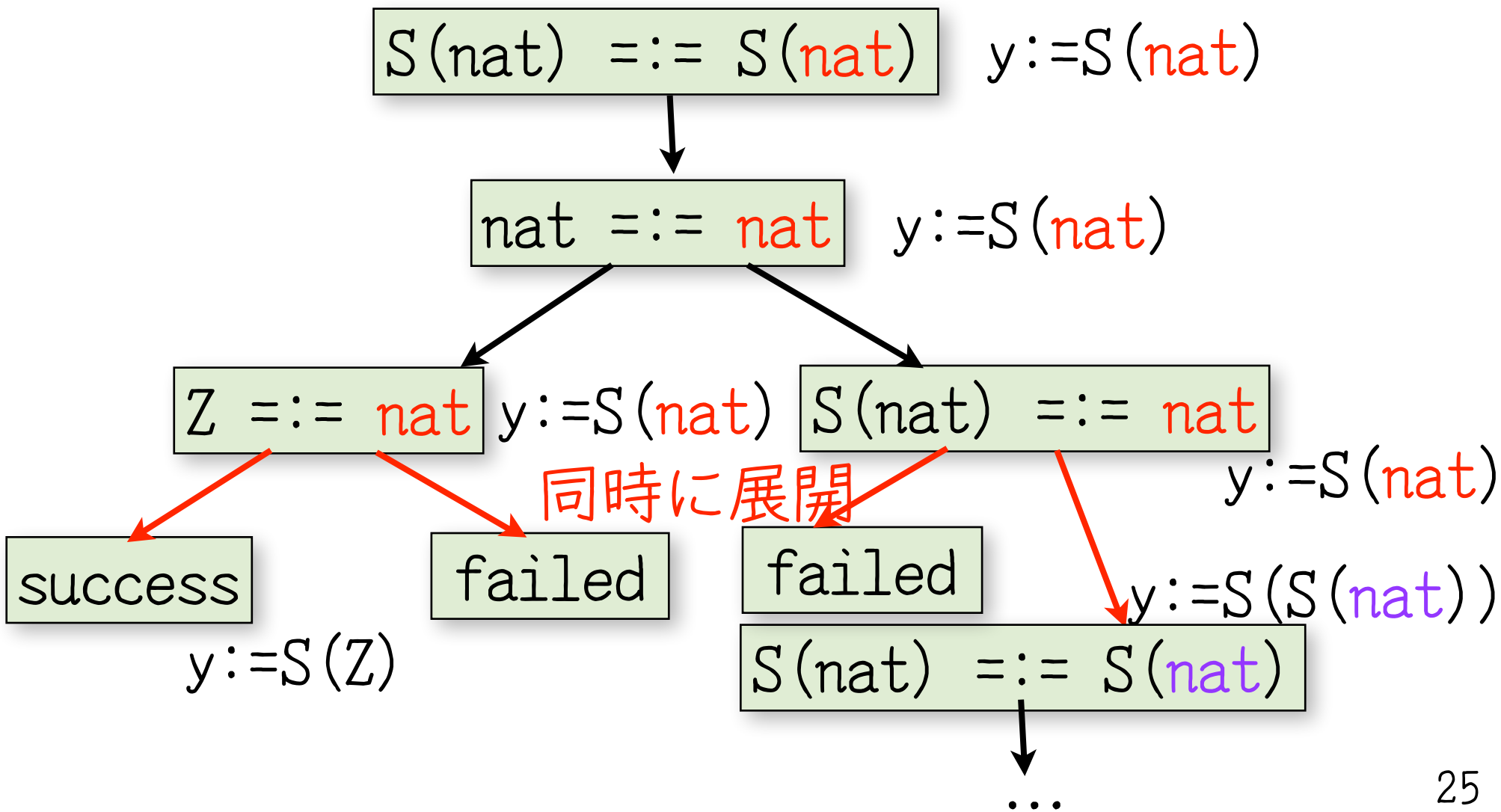
```
nat = Z
nat = S nat
natlist = []
natlist = nat : natlist
```



# 例

```
data Nat = Z | S Nat
```

```
nat = Z
nat = S nat
natlist = []
natlist = nat : natlist
```



# まとめ

- 関数論理型言語Curry
  - 関数型言語のように高階関数を用いたプログラミング
  - 論理型言語のように
    ビルトインサーチ，単一化を利用可
- Needed Narrowingによる
  より無駄のない探索

# 第14回レポート課題
# 締切 7/29 13:00

# 問1

- 次の関数をCurryの機能を活用し書け
  - take :: Int -> [a] -> [a]
    - 例：take 3 [1,2,3,4] = [1,2,3]
      - 長さ3未満のリストに対して未定義でよい
  - drop :: Int -> [a] -> [a]
    - 例：drop 3 [1,2,3,4] = [4]
      - 長さ3未満のリストに対して未定義でよい
  - sillyUnparen :: String -> String
    - sillyUnparen "(12)()" = "12)("
    - sillyUnparen "(12)(" = "(12)("
      - 外側に括弧があるときのみ外す
      - 上以外の解は返さない

# 問2

- 次の関数unparenを実装せよ
  - unparen :: String -> Paren
    - data Paren = PSeq Paren Paren | PEmp
    - paren p = s ⇔
      unparen sの返り値の一つがp
      となるように
      - paren PEmp = ""
        paren (PSeq p1 p2)
        = "("++paren p1++")"++paren p2
- unparen xが成功するときのxは何かを説明せよ

# 問3

- 第11回課題の問1，2をCurryで．

# 問4

- 第11回課題の問4をCurryで.

# 問5（1/3）

- 次のPrologプログラムと…

```
ins(X, Y, [X|Y]).
ins(X, [A|Y], [A|Z]) :- ins(X, Y, Z).

perm([], []).
perm([A|X], Z) :- perm(X, Y), ins(A, Y, Z).

sorted([]).
sorted([_]).
sorted([A, B|X]) :- (A =< B), sorted([B|X]).

mysort(X, Y) :- permutation(X, Y), sorted(Y), !.
```

# 問5（2/3）

- …次のCurryプログラムを考える．

```
insert x xs     = x:xs
insert x (y:xs) = y:insert x xs

perm []     = []
perm (x:xs) = insert x (perm xs)

isSorted []  = True
isSorted [a] = True
isSorted (a:b:x) = (a <= b) && isSorted (b:x)

mysort xs
  | isSorted ys = ys
    where ys = permutation xs
```

# 問5（3/3）

- 二つのプログラムを実行し比較せよ
  - 特に，リストの長さに応じて，それぞれの実行時間はどう変化するか？
  - その差はどこから来るのか，述べよ

# 問6

- 問6のプログラムの実行時間は改善したとはいえ指数オーダである．が，以下のように変更すると$O(n^2)$となる．なぜか

```
import List

...

insertions xs = scanl (flip insert) (reverse xs)

mysort xs
  | all isSorted ys = ys
     where ys = insertions xs
```

# 問7

- 第12回課題の問3をCurryで

# 発展1

- Needed Narrowingについて調べ，
  実装を行え
- それを元に関数論理型言語を作成せよ

# 発展2

- Curryでは p :: X -> Bool に対し p(x)がTrueとなるxを列挙することができる
  - これを活用し，QuickCheckのようなテスト用ライブラリを作成せよ
  - 参考
    * EasyCheck
    * (Lazy) SmallCheck

# 次回は最終回