nML 프로그래밍 소개

2004년 9월 9일

신재호 <netj@ropas.snu.ac.kr>
프로그래밍 연구실 (ROPAS)
서울대학교 컴퓨터공학부

- 2004년 봄, 4190.102A 수업에서 김재황님이 사용했던 것을 다듬은 자료입니다.

ML의 특징

- 값 중심의 프로그래밍 언어

small & simple language	작고 간단
type safety	타입 검사를 통해 안전한 프로그램만 수행
polymorphism	타입은 다르지만 하는 일이 같은 함수를 하나로 작성 가능
higher-order function	함수를 다른 값과 구분 없이 사용
abstract data type	다양한 데이터 구조 쉽게 사용
exception handling	예외 상황 관리 구조
module system	정제된 모듈 구조

ML 컴파일러 시스템

- Standard ML of New Jersey
  - Bell Lab. & Princeton, USA
- Objective Caml
  - INRIA, France

- nML
  - SNU/KAIST, Korea
  - 한글 식별자 (identifier) 사용 가능

nML 바로 실행 컴파일러: nml

```
ropas> nml
nML 컴파일러 0.92b (2004/04/19)
  Copyright(c) 2000-2004 KAIST/SNU 프로그램 분석 시스템 연구단
  (과기부 창의적 연구진흥사업 1998-2003)
  http://ropas.snu.ac.kr/n
  Objective Caml 3.04 코드를 기반으로 작성
# val x = 1+2;;
val x: int = 3

# x+1;;
val it: int = 4

# #use "file.n";;
...
```

Unix, MS Windows 에서 수행 가능
http://ropas.snu.ac.kr/n

nML 컴파일러: nmlc

- 컴파일 그리고 수행

```
ropas> nmlc file.n
ropas> a.out
```

- 의존 순서대로 분리 컴파일

```
ropas> nmlc -c a.n
ropas> nmlc -c b.n
ropas> nmlc -o a.out a.cmo b.cmo
```

- 모듈 이름은 모두 달라야 한다.
- 모듈로 포장한 것들만 다른 파일에서 쓸 수 있다.

- 자동 Makefile 생성

```
ropas> nmakegen
ropas> make
ropas> run
```

값 중심의 프로그래밍?

- 값은 변하지 않는다.

```
int fac (int n) {
  int i, result = 1;
  for (i=1; i<=n; i++)
    result = result * i;
  return result;
}
```

```
fun fac n = if n = 0 then 1
                     else n * fac (n - 1)
```

타입 (type) !

- 타입이 맞는 프로그램은 오류가 발생하지 않는다.

```
1 + 1.0 (X)     (real_of_int 1) + 1.0 (O)
```

- 자동으로 타입을 유추하므로 타입을 명시할 필요가 없다.

```
fun fac n = if n = 0 then 1 else n * fac (n - 1)
```

- 다양한 타입 제공
  - 기본, 튜플, 레코드, 리스트, 사용자 정의 데이터 타입
  - 다형 타입 (polymorphic type)

정의 (이름 붙이기)

- nML 프로그램은 정의의 모임이다.

```
# val kwang = "이광근";;
val kwang: string = "이광근"

# val result = let val 둘 = 2
                   val 셋 = 3
               in  둘 * 셋
               end;;
val result: int = 6
```

바로 실행 컴파일러에서는 "정의" 대신 "프로그램 식"을 쓰면 편의상 "val it = "를 앞에 붙여 준다.

기본적인 값

- 정수, 실수, 문자, 문자열, 논리값, ...

```
# (1 + 0xc) % 2 << 2;;
val it: int = 4

# 'a';;
val it: char = 'a'

# (true && false) || true;;
val it: bool = true

# ();;
val it: unit = ()
```

```
# 1.0 + 2.0;;
val it: real = 3

# "하" ^ "나";;
val it: string = "하나"
```

튜플 (tuple)

- 값들의 모임

```
# val 하나들 = (1, 1.0, "하나");;
val 하나들: int * real * string = (1, 1, "하나")

# 하나들.2;;
val it: string = "하나"

# val (정수, 실수, 문자열) = 하나들;;
val 정수: int = 1
val 실수: real = 1
val 문자열: string = "하나"
```

레코드 (record)

- 꼬리표 (label) 붙은 값들의 모임

```
# val 값 = {첫째="이광근", 둘째="nML"};;
val 값: {첫째: string, 둘째: string}
   = {첫째="이광근", 둘째="nML"}

# 값.둘째;;
val it: string = "nML"

# val {첫째=교수님, 둘째=언어} = 값;;
val 교수님: string = "이광근"
val 언어: string = "nML"
```

리스트 (list)

- 같은 타입 값들의 나열

```
# val 첫째 = [3,5];;
val 첫째: int list = [3, 5]
# val 둘째 = 1 :: 첫째;;
val 둘째: int list = [1, 3, 5]
# val 셋째 = 둘째 @ [7,9];;
val 셋째: int list = [1, 3, 5, 7, 9]

# case 셋째 of []    => 0
              | h::t => h;;
val it: int = 1
```

[3,5] 는 3::5::[]과 같다.

사용자 정의 데이터 타입

- 다양한 구조의 값을 정의 사용 가능하다.

```
# type tree = Leaf
            | Node of int * tree * tree;;

# val 나무 = Node(5,Node(6,Leaf,Leaf),Leaf);;
val 나무: tree = ...

# case 나무 of Leaf    => "잎새"
            | Node _ => "노드";;
val it: string = "노드"
```

타입 정의

- 타입 줄임말 (type abbreviation)

```
type position      = int * int
type student       = {name: string, id: int, age: int}
type ('a, 'b) pair = 'a * 'b
```

- 새로운 데이터 타입 정의

```
type tree = Leaf
          | Node of int * tree * tree

type 'a tree = Leaf
             | Node of 'a * 'a tree * 'a tree

type 'a children = 'a tree * 'a tree
 and 'a tree     = Leaf
                 | Node of 'a * 'a children
```

함수

- 이름없는 함수값

```
fn x => x + 1
fn 0 => true | _ => false
```

- 함수 정의

```
fun incr x = x + 1
fun is_zero 0 = true
  | is_zero _ = false

fun fac n = if n = 0 then 1 else n * fac (n - 1)

fun f (n, i) = if n = 0 then i else f (n - 1, i * n)
and fac n = f (n, 1)
```

패턴 (pattern)

- 값을 맞추어 보는 틀

```
val □ = ...
fun f □ = ... | f □ = ...
fn □ => ... | □ => ...
case ... of □ => ... | □ => ...
... handle □ => ... | □ => ...
```

순서대로 맞추어 보다가
맞는 것이 없으면
예외상황 발생

- 패턴의 종류

```
0    "abc"    x    _
(□, □, ...)    {a=□, b=□, ...}
[]    □::□    [□, □, ...]
Leaf    Node □
```

```
fun fac n = if n = 0 then 1
                     else n * fac (n - 1)
```

```
fun fac 0 = 1
  | fac n = n * fac (n-1)
```

다형 타입 함수 (polymorphic function)

- 여러 타입의 값을 인자로 받는 함수

```
# fun swap (a, b) = (b, a)
val swap: 'a * 'b -> 'b * 'a = <fun>

# fun 머리 (h::t) = h
    | 머리 _      = raise EmptyList
val 머리: 'a list -> 'a = <fun>
```

모든 정의가 다형 타입이 되는 것은 아니다. fun으로 정의한 것은 다형 타입이 될 수 있음을 기억하자.

```
fun f x = ...                    (O)
val f = fn x => x ...            (O)

val ... = ... (fn x => x) ...    (X)
```

고계도 함수 (higher-order function)

- 함수가 인자/결과인 함수

```
# fun add x y = x + y;;    // int -> int -> int
# val incr = add 1;;       // int -> int
# incr 6;;
val it: int = 7

# fun map f [] = []
    | map f (h::t) = f h :: map f t;;
            // ('a -> 'b) -> 'a list -> 'b list
# val map_inc = map (fn x => x + 1);;
# map_inc [1, 2, 3];;
val it: int list = [2, 3, 4]
```

예외상황 (exception)

- 발생하면 처리하는 데까지 빠져 나온다.
  - 오류 처리 (eg. `Division_by_zero`)
  - 복잡한 구조를 한번에 빠져 나오고 싶을 때

```
exception Fail
exception Error of string

if input < 0 then raise Error "Illegal Input"

... handle Fail => ...
         | Error "Illegal Input" => ...
```

참조값(reference)과 배열(array)

- 참조값: 수행 중에 바뀔 수 있는 값

```
# val count = ref 0;;
val count: int ref = ref 0

# count := !count + 1;
  !count;;
val it: int = 1
```

- 배열: 같은 타입의 값들의 변경 가능한 모임

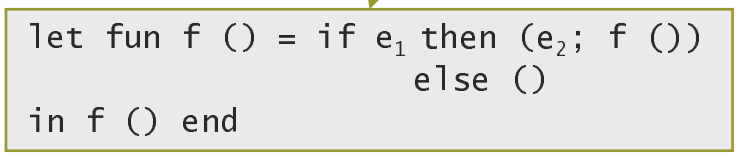
```
# val 배열 = [| 1, 2, 3 |];;
val 배열: int array = ...

# 배열.[1] <- 5;
  배열.[1] + 배열.[2];
val it: int = 8
```

루프 (loop)

- while 문

```
# val x = ref 0;
# while !x < 100 do
    x += 7
  end;
  !x;;
val it: int = 105
```

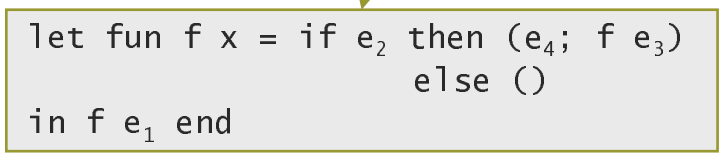
```
while e1 do e2 end
```



- for 문

```
# val x = ref 1;;
# for y = 1; y < 6; y + 1 do x *= y end;
  !x;;
val it: int = 120
```

```
for x = e1; e2; e3 do e4 end
```



nML 프로그래밍 소개

2004년 9월 15일

신재호 <netj@ropas.snu.ac.kr>
프로그래밍 연구실 (ROPAS)
서울대학교 컴퓨터공학부

- 2004년 봄, 4190.102A 수업에서 김재황님이 사용했던 것을 다듬은 자료입니다.

모듈 (structure)

- 값, 타입, 예외상황, 모듈 정의의 모임

```
struct ... end
structure Name = struct ... end
```

이름에 대한 규칙:
- 값, 타입: 소문자 또는 한글로 시작
- 예외상황, 데이터 구성자, 모듈, 모듈 타입, 모듈 함수: 대문자 또는 _ 로 시작

- 예

```
# structure _정수스택 =
   struct
     type 원소 = int and t = 원소 list
     exception _비었다
     val 빈것 = []
     fun 뿌직 s = case s of []=>raise _비었다 | h::t=>(h,t)
     ...
   end;;
# _정수스택.빈것;;
val it: 'a list = []
# open _정수스택;;
# 빈것;;
val it: 'a list = []
```

모듈 타입 (signature)

- 모듈 기술, 모듈 함수의 입출력 기술

```
sig ... end
signature Name = sig ... end
```

- 효과: 값, 타입 가리기, 타입 고정시키기

```
# signature _대기열_ =
    sig
      type 원소 type t
      exception _비었다
      val 빈것: t
      val 뿌직: t -> 원소 * t
      ...
    end;;
# signature _정수열_ = _대기열_ where type 원소 = int;;
# structure _정수스택 : _정수열_ = struct ... end;;
# _정수스택.빈것;;
val it: _정수스택.t = <abstr>
```

모듈 함수 (functor)

- 모듈을 받아 모듈을 결과로 주는 함수

```
functor F (M1:S1, ..., Mn:Sn): S = struct ... end
structure M = F (M1, ..., Mn)
```

- 예

```
functor _빠른큐 (A: _대기열_) : _대기열_ where type 원소 = A.원소 =
  struct
    type 원소 = A.원소
    type t = A.t * A.t
    exception _비었다
    val 빈것 = (A.빈것, A.빈것)
    fun 뿌직 (s1, s2) =
        let val (결과, s1') = A.뿌직 s1
        in  (결과, (s1', s2))
        end
      handle A._비었다 =>
        ignore (A.뿌직 s2);
        뿌직 (A.뒤집기 s2, A.빈것)
      handle A._비었다 => raise _비었다  ...
  end
structure _빠른정수큐 = _빠른큐 (_정수스택)
```

리스트의 기본적인 연산 1/2

```
# val 리스트 = [1,2,3,4,5];;
val 리스트: int list = [1,2,3,4,5]

# List.length 리스트;;                    // 'a list -> int
val it: int = 5

# List.hd 리스트;;                        // 'a list -> 'a
val it: int = 1

# List.tl 리스트;;                    // 'a list -> 'a list
val it: int list = [2, 3, 4, 5]

# List.nth 리스트 3;;              // 'a list -> int -> 'a
val it: int = 4
```

리스트의 기본적인 연산 2/2

```
# val 앞 = [1,2,3];;
# val 뒤 = [4,5];;

# List.rev 앞;;                    // 'a list -> 'a list
val it: int list = [3, 2, 1]

# List.append 앞 뒤;; // 'a list->'a list->'a list
val it: int list = [1, 2, 3, 4, 5]

# List.flatten [앞, 뒤, [], [6]];;
                          // 'a list list -> 'a list
val it: int list = [1, 2, 3, 4, 5, 6]
```

리스트를 뒤져 보는 연산 1/2

```
# val 홀수 = [1,3,5,7,9] and 짝수 = [2,4,6,8,10];;
# val 모두 = 홀수 @ 짝수;;
# fun 짝수니 n = n % 2 = 0;;

# List.for_all 짝수니 모두;;
                // ('a -> bool) -> 'a list -> bool
val it: bool = false

# List.exists 짝수니 모두;;
                // ('a -> bool) -> 'a list -> bool
val it: bool = true

# List.mem 3 짝수;;          // 'a -> 'a list -> bool
val it: bool = false
```

리스트를 뒤져 보는 연산 2/2

```
# val 뒤질것 = [17,9,24,3,2,213,8,9,0,13];;

# List.find (fn x => x < 10) 뒤질것;;
                    // ('a -> bool) -> 'a list -> 'a
val it: int = 9

# List.filter (fn x => x < 10) 뒤질것;;
               // ('a -> bool) -> 'a list -> 'a list
val it: int list = [9, 3, 2, 8, 9, 0]

# List.partition (fn x => x < 10) 뒤질것;;
  // ('a -> bool) -> 'a list -> 'a list * 'a list
val it: int list * int list =
  ([9, 3, 2, 8, 9, 0], [17, 24, 213, 13])
```

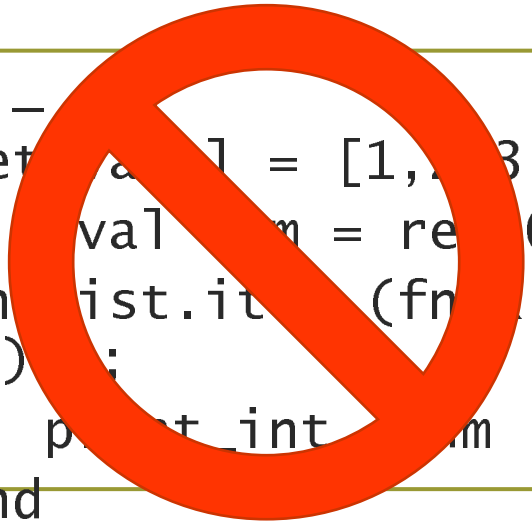
리스트 반복 연산 1/2

```
# List.iter print_int [1,2,3,4];;
                  // ('a -> unit) -> 'a list -> unit
1234val it: unit = ()

# List.map string_of_int [1,2,3,4,5];;
                 // ('a -> 'b) -> 'a list -> 'b list
val it: string list = ["1", "2", "3", "4", "5"]

# List.map (fn i => 10 * i) [1,2,3];;
val it: int list = [10, 20, 30]
```

리스트 반복 연산 1'/2

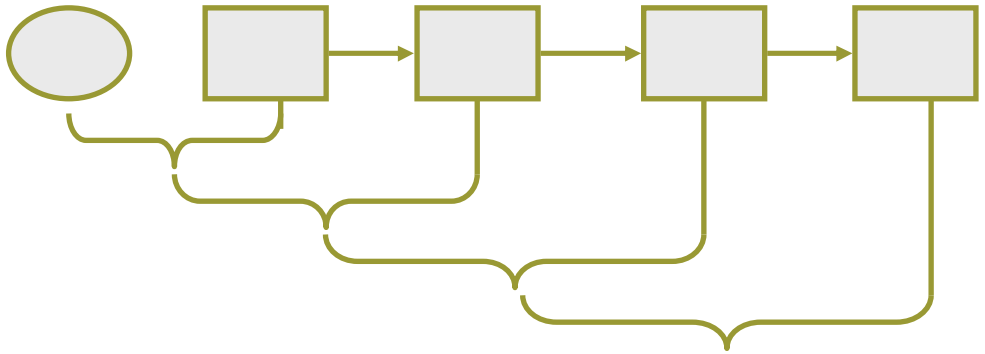
```
main() {
  int l[] = {1,2,3,4,5};
  int i, sum = 0;
  for (i=0; i<5; i++)
    sum += l[i];
  printf("%d", sum);
}
```

```
val _
  let a l = [1,2,3,4,5]
      val m = re 0
  in ist.it (fn => sum +=
   x) ;
      p t_int m
  end
```



리스트 반복 연산 2/2

```
# List.fold_left (fn x y => x*10+y) 0 [1,2,3];;
        // ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
val it: int = 123
```



```
# List.fold_right (fn x y => y*10+x) [1,2,3] 0;;
        // ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
val it: int = 321
```

리스트 반복 연산 2'/2

```
main() {
  int l[] = {1,2,3,4,5};
  int i, sum = 0;
  for (i=0; i<5; i++)
    sum += l[i];
  printf("%d", sum);
}
```

```
val _ =
  let val l = [1,2,3,4,5]
  in  print_int (
        List.fold_left
          (fn x y => x + y) 0 l)
  end
```

참고 자료

- 웹페이지: http://ropas.snu.ac.kr/n
- 정확한 정의: "프로그래밍 언어 nML"
- 참고서:
  - "nML 프로그래밍 걸음마"
  - "nML 프로그래밍 언어 참고서"
  - "nML과 함께하는 프로그래밍 여행"
- 라이브러리: O'Caml 표준 라이브러리
- 읽을거리: 마이크로 소프트웨어 연재기사