nML Programming Guide

Sukyoung Ryu and Oukseh Lee

ROPAS, KAIST

March 6, 2001

nML의 특징

- 작고 간단하다.
- 타입 검사를 통해 안전한 프로그램만 받아들인다.
- 타입은 다르지만 하는 일이 같은 함수들을 하나의 함수로 표현할 수 있다.
- 함수를 다른 값들과 구분없이 사용한다.
- 다양한 데이타 구조를 쉽게 만들 수 있다.
- 예외상황 관리 메카니즘을 갖고 있다.
- 정제된 모듈 구조를 갖고 있다.

N/Caml: nML 컴파일러

```
ropas > nml
nML compiler 0.9 (built at 2001/3/14) by H. Eo, J. Kim, and O. Lee
- Copyright(c) 2000-2001 Research On Program Analysis System
- http://ropas.kaist.ac.kr/n
- Based on Objective Caml 3.00 Code

# 1 + 3;;
- : int = 4
#
```

기본적인 것들

- 상수

```
# 1 + 2;;
- : int = 3
# 1.0 + 2.0;;
- : real = 3
# 'a';;
- : char = 'a'
# "하나";;
- : string = "하나"
```

- 이름 붙이기

```
# val puppy = "류석영";;
val puppy : string = "류석영"
# val 하나 = 1;;
val 하나 : int = 1
```

튜플

- 튜플 만들기

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

- 튜플 쪼개기

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

리스트

- 리스트 만들기

```
# val 첫번째 = [3,5];;
val 첫번째 : int list = [3, 5]
# val 두번째 = 1::첫번째;;
val 두번째 : int list = [1, 3, 5]
# val 세번째 = 두번째@[7,9];;
val 세번째 : int list = [1, 3, 5, 7, 9]
```

- 리스트 쪼개기

```
# case 세번째 of [] => -1
               | (hd::tl) => hd;;
- : int = 1
```

레코드

- 레코드 선언

```
# type 조교 = {첫번째: string, 두번째: string};;
type 조교 = { 첫번째: string, 두번째: string }
```

- 레코드 만들기

```
# val cs320 = {첫번째="류석영", 두번째="이욱세"};;
val cs320 : 조교 = {첫번째="류석영", 두번째="이욱세"}
```

- 레코드 쪼개기

```
# val 다음 = cs320.두번째;;
val 다음 : string = "이욱세"
```

타입 (type)

- 타입이 맞는 프로그램은 잘못 수행되지 않는다.

- 타입:
  - 기본 타입 (int, real, string, char, bool, unit)
  - 튜플 타입 ($\tau * \tau$), 레코드 타입 ($\{l{:}\tau, \cdots\}$), 리스트 타입 (τ list), 포인터 타입 (τ ref)
  - 사용자 정의 데이타 타입

- 타입 검사: 프로그램을 수행하기 전에 타입이 맞는지 검사한다.
  - 타입 검사 예

    $$\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3$$

    에서 e_1은 bool 타입이어야 하고, e_2와 e_3의 타입은 같아야 한다.
  - 타입 정보를 사용자가 주지 않아도 알아서 유추하여 검사한다.

패턴 (pattern)

- 패턴의 종류
  상수, 변수, 레코드, 튜플, 리스트, 사용자 정의 데이타 타입, ...

- 패턴의 사용
  `val`, `fun`, `fn`, `case`, `handle`

- 사용 예

```
val (정수,실수) = (1,2.0);;
fun length [] = 0 | length (h::t) = 1 + length t;;
fn x => x + 1;;
case 3 of 0 => true | _ => false;;
```

예외상황 관리 (exception mechanism)

- 예외상황
  - 에러 처리 (eg. `Division_by_zero`)
  - 복잡한 구조를 한번에 빠져나오는 경우

- 예외상황 선언

```
exception Fail
exception Error of string
```

- 예외상황 발생

```
raise (Error "Illegal Input")
```

- 예외상황 처리

e `handle Fail =>` e_1

```
       | Error "Illegal Input" =>
       | Error _ =>
```
e_2

e_3

메모리 사용 (imperative features)

- nML 문법

e ::=	⋮	
	`ref` e	reference
|	e_1 `:=` e_2	assignment
|	`!` e	dereference

- 사용 예

```
# val count = ref 0;;
val count : int ref = ref 0
# fun inc n = n := !n + 1;;
val inc : int ref -> unit = <fun>
# inc count;;
- : unit = ()
# !count;;
- : int = 1
```

함수 (function)

- 함수 정의와 사용

```
let val f = fn x y => x+y
in  f 1 3
end
```

```
let fun f x y = x+y
in  f 1 3
end
```

- 패턴 매치에 의한 함수 정의

```
let val f = fn 0 => 0
             | x => x+1
in  f 20
end
```

```
let fun f 0 = 0
      | f x = x+1
in  f 20
end
```

- 재귀 함수

```
val rec fac = fn n =>
  if n=0 then 1
         else n * fac(n-1)
```

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

다형 타입 (polymorphism)

- 다형 타입 함수: 여러 타입의 값을 인자로 받는 함수.

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

fun hd (h::t) = h           : 'a list -> 'a
  | hd _ = raise EmptyList
```

- 다형 타입이 되는 경우: 이름 지어진 경우

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

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

함수를 주고받는 함수 (higher-order function)

- 함수를 여타 다른 값과 구별하지 않는다. 즉, 함수값이 함수의 인자 또는 결과가 될 수 있다.

```
# fun map f [] = []
    | map f (h::t) = f h :: map f t

# val map_inc = map (fn x => x+1)

# val map_square = map (fn x => x**2)
```

타입 정의 (type definition)

- 타입 줄임말 (type abbreviation): 복잡한 타입에 이름 지어 주는 것.

```
type position = int * int
type student = {name: string, id: int, age: int}
```

- 데이타 타입 정의: 새로운 데이타 타입을 만들어 주는 것.

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

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
```

데이타 타입 (data type)

- 데이타 생성

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

val y = Node(5,Node(6,Leaf,Leaf),Leaf)
```

- 데이타 사용

```
fun sum Leaf = 0
  | sum (Node(i,t1,t2)) = i + sum t1 + sum t2
```

모듈 (1/2)

- 모듈 (structure): 값과 타입의 모음.

```
structure Int = struct
 type t = Top | Int of int
 fun join Top _ = Top
   | join _ Top = Top
   | join (Int x) (Int y) = if x=y then Int x else Top
end
val x = Int.join Int.Top (Int.Int 20)
```

- 모듈 타입 (signature): 모듈의 타입.

```
signature L = sig
  type t
  val join: t -> t -> t
end
structure IntL = Int : L
```

모듈 (2/2)

- 모듈 함수 (functor): 모듈을 만드는 함수.

```
functor Product(A:L, B:L):L = struct
  type t = A.t * B.t
  fun join (a,b) (c,d) = (A.join a c, B.join b d)
end
structure IntProd = Product(Int,Int)
```

예제: 이진 검색 트리

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

fun find t k = case t of
      Leaf(j,e) =>   if j=k then e else raise Not_found
    | Node(j,l,r) => if j>k then find l k else find r k

fun mem t k = (find t k; true) handle Not_found => false
```

화일 다루기

- 하나의 화일을 컴파일 할 때:

```
ropas> nmlc file.n
```

- 여러 화일을 컴파일 할 때:

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

- 여러 화일을 컴파일 할 때 몇 가지 규칙:
  - 모듈에 쌓인 것만 화일 간에 볼 수 있다.
  - 모듈 이름은 모두 달라야 한다.
- 화일에 적힌 프로그램을 nml에서 읽어 올 때:

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

Makefile 다루기

- Makefile
  - 소스 프로그램에서 실행 화일을 만드는 방법을 설명하는 화일
  - http://www.gnu.org/software/make/make.html

- nmlmake: Makefile 생성기

```
ropas> nmlmake -a "Oukseh Lee" result
```

nML 언어 참고 자료

- nML 언어 참고 자료
  - 웹 페이지: `http://ropas.kaist.ac.kr/n`
  - 정확한 정의: "프로그래밍 언어 nML"

- nML 컴파일러
  - 사용법: "ncaml 시스템"
  - 과기계에 설치: 현재 version 0.9 설치 되어 있음
  - Linux 또는 MS windows 에 설치 가능
  - OCaml 표준 라이브러리 사용 가능: `http://ropas.kaist.ac.kr/n/lib`