

SNU 4190.210 프로그래밍
원리(Principles of Programming)
Part IV

Prof. Kwangkeun Yi

차례

- 1 안전하게 프로그래밍하기: 손수 vs 자동
- 2 맞는지 확인하기 쉽게 프로그램하기
- 3 대형 프로그래밍을 위한 기술: 모듈 프로그래밍

다음

- 1 안전하게 프로그래밍하기: 손수 vs 자동
- 2 맞는지 확인하기 쉽게 프로그램하기
- 3 대형 프로그래밍을 위한 기술: 모듈 프로그래밍

안전한 프로그래밍



예상하는 데이터/값이 흘러다니는 지를 검사하는 코드를 항상 넣는다. 넣는 장소는 함수 정의내에: 제대로 된 인자가 전달되는 지 확인.

- ▶ 타입체크(*type check*): 함수정의할 때, 함수 타입에 맞는 데이터를 받는지 확인하는 코드를 넣는다.
- ▶ 조건체크(*requirement check*): 함수정의할 때, 함수 타입에 맞는 데이터 중에서 원하는 경우의 데이터인지를 확인하는 코드를 넣는다.

안전한 프로그래밍 예

- ▶ 타입체크(type check)×조건체크(requirement check)

```
(define (nth-child tree n)
  (cond ((not (is-tree? tree)) (error))
        ((not (is-int? n)) (error))
        ((is-leaf? tree) (error))
        ((< n 0) (error))
        (else ...) ; programming in a safe area
  )
)
(define (is-tree? tree)
  (and (pair? tree)
        (or (equal (car x) leaf-tag)
            (and (equal (car x) node-tag)
                 (fold and (map is-tree? (cdr tree)) true)
                )))
)
(define is-int? integer?)
```

안전한 프로그래밍의 단점

타입체크 코드×조건 체크 코드를 꼭 장착시켜야 하는가?

안전한 프로그래밍의 단점

타입체크 코드×조건 체크 코드를 꼭 장착시켜야 하는가?

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴땐 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?”

안전한 프로그래밍의 단점

타입체크 코드×조건 체크 코드를 꼭 장착시켜야 하는가?

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?”
- ▶ “내 프로그램의 실행속도가 문제될 것 같아요. 함수호출하면 매번 타입체크를 하는 데 시간이 소모되잖아요. 문제될 때 곧바로 중단시켜야 문제를 그때 그때 파악하기 쉽긴 할 텐데... 모르겠네요. 더 좋은 방법없을까요?”

안전한 프로그래밍의 단점

타입체크 코드×조건 체크 코드를 꼭 장착시켜야 하는가?

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?”
- ▶ “내 프로그램의 실행속도가 문제될 것 같아요. 함수호출하면 매번 타입체크를 하는 데 시간이 소모되잖아요. 문제될 때 곧바로 중단시켜야 문제를 그때 그때 파악하기 쉽긴 할 텐데... 모르겠네요. 더 좋은 방법없을까요?”
- ▶ “내 프로그램은 KRX-II로켓의 진동보정 시스템에 장착될 겁니다. 실행중에 멈추면 안되는데, 실행중에 문제되면 결국 멈출텐데... 모르겠네요. 더 좋은 방법없을까요?”

미리 타입 검사해주는 프로그래밍 환경

“static type-checking system”

- ▶ 함수의 인자타입에 맞는 데이터/값들이 **항상** 흘러드는지 **미리 자동으로 확인**
 - ▶ “미리”: 실행전에 프로그램 텍스트만을 보면서.
 - ▶ “자동으로”: 프로그래머가 하지 않는다.
- ▶ 프로그래밍을 수월하게/sw개발 생산성 증대
- ▶ 이미 널리 퍼지고 있는 기술(미래? 현재!)

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험
 - ▶ 그밖의 예: Java, C#, F#, Scala, Haskell

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험
 - ▶ 그밖의 예: Java, C#, F#, Scala, Haskell
- ▶ 불안정한 환경

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험
 - ▶ 그밖의 예: Java, C#, F#, Scala, Haskell
- ▶ 불안정한 환경
 - ▶ 타입체크 “ok”, 그러나 실행중에 타입에러

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험
 - ▶ 그밖의 예: Java, C#, F#, Scala, Haskell
- ▶ 불안정한 환경
 - ▶ 타입체크 “ok”, 그러나 실행중에 타입에러
 - ▶ 예: C, C++, Scheme, JavaScript, Python

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험
 - ▶ 그밖의 예: Java, C#, F#, Scala, Haskell
- ▶ 불안정한 환경
 - ▶ 타입체크 “ok”, 그러나 실행중에 타입에러
 - ▶ 예: C, C++, Scheme, JavaScript, Python
 - ▶ 대책?

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험
 - ▶ 그밖의 예: Java, C#, F#, Scala, Haskell
- ▶ 불안정한 환경
 - ▶ 타입체크 “ok”, 그러나 실행중에 타입에러
 - ▶ 예: C, C++, Scheme, JavaScript, Python
 - ▶ 대책?
 - ▶ 프로그래머의 책임

미리 타입 검사해주는 프로그래밍 환경

이슈: “안전하게 확인” vs “불안전하게 확인”

- ▶ 안전하게 자동 확인해주는 환경
 - ▶ 대표 예: ML(OCaml)프로그래밍 실습을 통해서 경험
 - ▶ 그밖의 예: Java, C#, F#, Scala, Haskell
- ▶ 불안정한 환경
 - ▶ 타입체크 “ok”, 그러나 실행중에 타입에러
 - ▶ 예: C, C++, Scheme, JavaScript, Python
 - ▶ 대책?
 - ▶ 프로그래머의 책임
 - ▶ 안전한 프로그래밍(defensive programming) 준수

미리 타입 검사해주는 프로그래밍 환경

안전한 “static type-checking system” 덕택에

미리 타입 검사해주는 프로그래밍 환경

안전한 “static type-checking system” 덕택에

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?”

미리 타입 검사해주는 프로그래밍 환경

안전한 “static type-checking system” 덕택에

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking

미리 타입 검사해주는 프로그래밍 환경

안전한 “static type-checking system” 덕택에

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?” **Static Type-Checking**
- ▶ “내 프로그램의 실행속도가 문제될 것 같아요. 프로시저실행중에 매번 타입체크를 하는 데 시간이 소모되잖아요. 문제될 때 곧바로 중단시켜야 문제를 그때 그때 파악하기 쉽긴 할 텐데... 모르겠네요. 더 좋은 방법없을까요?”

미리 타입 검사해주는 프로그래밍 환경

안전한 “static type-checking system” 덕택에

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking
- ▶ “내 프로그램의 실행속도가 문제될 것 같아요. 프로시저실행중에 매번 타입체크를 하는 데 시간이 소모되잖아요. 문제될 때 곧바로 중단시켜야 문제를 그때 그때 파악하기 쉽긴 할 텐데... 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking

미리 타입 검사해주는 프로그래밍 환경

안전한 “static type-checking system” 덕택에

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking
- ▶ “내 프로그램의 실행속도가 문제될 것 같아요. 프로시저실행중에 매번 타입체크를 하는 데 시간이 소모되잖아요. 문제될 때 곧바로 중단시켜야 문제를 그때 그때 파악하기 쉽긴 할 텐데... 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking
- ▶ “내 프로그램은 KRX-II로켓의 진동보정 시스템에 장착될 겁니다. 실행중에 멈추면 안되는데, 실행중에 문제되면 결국 멈출텐데... 모르겠네요. 더 좋은 방법없을까요?”

미리 타입 검사해주는 프로그래밍 환경

안전한 “static type-checking system” 덕택에

- ▶ “분명히 필요없는 경우가 있긴한데. 그럴때 필요없겠죠... 그런데 불안하긴 해요. 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking
- ▶ “내 프로그램의 실행속도가 문제될 것 같아요. 프로시저실행중에 매번 타입체크를 하는 데 시간이 소모되잖아요. 문제될 때 곧바로 중단시켜야 문제를 그때 그때 파악하기 쉽긴 할 텐데... 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking
- ▶ “내 프로그램은 KRX-II로켓의 진동보정 시스템에 장착될 겁니다. 실행중에 멈추면 안되는데, 실행중에 문제되면 결국 멈출텐데... 모르겠네요. 더 좋은 방법없을까요?” Static Type-Checking

```
(define (nth-child tree n)
  (cond ((not (is-tree? tree)) (error))
        ((not (is-int? n)) (error))
        ((is-leaf? tree) (error))
        ((in 0) (error))
        (else ...) ; programming in a safe area
  )
)
```

```
(define (is-tree? tree)
  (and (pair? tree)
       (or (equal (car x) leaf-tag)
           (and (equal (car x) node-tag)
                (fold and (map is-tree? (cdr tree)) true)
                )))
  )
)
```

```
(define is-int? integer?)
```

```
(define (nth-child tree n)
  (cond
    ((not (is-int? n)) (error))
    ((is-leaf? tree) (error))
    ((= 0) (error))
    (else ...) ; programming in a safe area
  )
)
```

```
(define (is-tree? tree)
  (and (pair? tree)
    (or (equal (car x) leaf-tag)
        (and (equal (car x) node-tag)
              (fold and (map is-tree? (cdr tree)) true)
            )))
  )
)
```

```
(define is-int? integer?)
```

```
(define (nth-child tree n)
  (cond
    ((is-leaf? tree) (error))
    ((in 0) (error))
    (else ...) ; programming in a safe area
  )
)
```

```
(define (is-tree? tree)
  (and (pair? tree)
    (or (equal (car x) leaf-tag)
        (and (equal (car x) node-tag)
              (fold and (map is-tree? (cdr tree)) true)
            )))
  )
)
```

```
(define is-int? integer?)
```

```
(define (nth-child tree n)
  (cond
    ((is-leaf? tree) (error))
    ((in 0) (error))
    (else ...) ; programming in a safe area
  )
)
```

다음

- 1 안전하게 프로그래밍하기: 손수 vs 자동
- 2 맞는지 확인하기 쉽게 프로그램하기
- 3 대형 프로그래밍을 위한 기술: 모듈 프로그래밍

귀납적으로 차근차근 프로그래밍하기 (inductive refinement)

예) “어울리지않아” 프로그래밍

$\text{smatch} : \text{스트링} * \text{코드} \rightarrow \text{bool}$

▶ 코드

$c \rightarrow 0 \mid 1 \mid \dots \mid 9 \mid c \cdot c \mid c \mid c \mid c? \mid c*$

▶ 보조자료 참고

다음

- 1 안전하게 프로그래밍하기: 손수 vs 자동
- 2 맞는지 확인하기 쉽게 프로그램하기
- 3 대형 프로그래밍을 위한 기술: 모듈 프로그래밍

대형 프로그래밍의 상황

바깥에 드러나는 정의된 이름들이 점점 많아진다



대형 프로그래밍의 상황

바깥에 드러나는 정의된 이름들이 점점 많아진다



해결책?

- ▶ 보따리 포장: 수많은 정의된 이름들을 그룹별로 나누고
- ▶ 보따리 이름: 보따리마다 이름을 붙이자
- ▶ “어느 보따리의 무엇”으로 호칭

모듈 프로그래밍

대형 프로그래밍을 위한

모듈 프로그래밍

대형 프로그래밍을 위한

- ▶ 모듈 = 정의들 + 보따리포장 + 보따리이름
 - ▶ “module”, “.c file”, “structure”, “package”

모듈 프로그래밍

대형 프로그래밍을 위한

- ▶ 모듈 = 정의들 + 보따리포장 + 보따리이름
 - ▶ “module”, “.c file”, “structure”, “package”
- ▶ 모듈타입 = 박스안에 있는 정의들의 타입들
 - ▶ “module type”, “.h file”, “signature”, “interface”

모듈 프로그래밍

대형 프로그래밍을 위한

- ▶ 모듈 = 정의들 + 보따리포장 + 보따리이름
 - ▶ “module”, “.c file”, “structure”, “package”
- ▶ 모듈타입 = 박스안에 있는 정의들의 타입들
 - ▶ “module type”, “.h file”, “signature”, “interface”
- ▶ 일반화된 모듈 = 모듈을 인자로 받는 모듈
 - ▶ “functor”, “parameterized module”, “class template”, “generic package”

모듈 프로그래밍

대형 프로그래밍을 위한

- ▶ 모듈 = 정의들 + 보따리포장 + 보따리이름
 - ▶ “module”, “.c file”, “structure”, “package”
- ▶ 모듈타입 = 박스안에 있는 정의들의 타입들
 - ▶ “module type”, “.h file”, “signature”, “interface”
- ▶ 일반화된 모듈 = 모듈을 인자로 받는 모듈
 - ▶ “functor”, “parameterized module”, “class template”, “generic package”
- ▶ 타입 시스템의 똑똑한 도움(대세)
 - ▶ 모듈 정의대로 외부에서 사용하는 지 미리 검사해 준다
 - ▶ 길으로 드러내기로 한 내용만 사용하는 지 미리 검사해 준다

모듈의 정의

Modules in ML

이 보따리 이름은 `Box`

```
val x = ...  
type t=A|B  
...
```

```
module Box =  
  struct  
    val x = ...  
    type t = ...  
  end
```

`Box.x ... Box.A`

`module`(보따리)는 정의한(이름붙인) 것들을 하나로 모아놓고 이름붙여 놓은 것 입니다.

모듈타입의 정의

Modules in ML

그러한 보따리의 타입:

```
val x: int -> int  
type t
```

```
val x: int -> int  
type t = A|B
```

```
val x: int -> int
```

```
module type S =  
  sig  
    ...  
  end
```

- ▶ 모듈정의에서 자동유추 될 수 있기도하고
- ▶ 모듈타입을 정의할 수도 있고

모듈타입의 용도

- ▶ “signature matching”(모듈 타입과 어울리기)을 통해서
 - ▶ 어울린 모듈은 모듈타입에 드러난 내용만 바깥에 알려짐
 - ▶ 모듈이 어느 모듈타입과 어울리려면 그 타입에 드러난 정의는 최소한 있어야

```
module Box:S = struct ... end
```

```
module Box' = struct ... end:S
```

모듈함수의 정의

- ▶ 함수: 값을 받아서 값을 만드는 함수

```
fun f(x,y) = x+y
```

- ▶ 모듈함수: 모듈을 받아서 모듈을 만드는 함수

```
module F(X:S1, Y:S2) = struct ... end
```


모듈함수의 정의

- ▶ 함수: 값을 받아서 값을 만드는 함수

```
fun f(x,y) = x+y
```

- ▶ 모듈함수: 모듈을 받아서 모듈을 만드는 함수

```
module F(X:S1, Y:S2) = struct ... end
```

- ▶ 참고: 인자로 받는 모듈의 타입을 자동유추할 수 있을까?

모듈함수의 정의

- ▶ 함수: 값을 받아서 값을 만드는 함수

```
fun f(x,y) = x+y
```

- ▶ 모듈함수: 모듈을 받아서 모듈을 만드는 함수

```
module F(X:S1, Y:S2) = struct ... end
```

- ▶ 참고: 인자로 받는 모듈의 타입을 자동유추할 수 있을까?
- ▶ 아직 불가능. 예)

```
module F(X) = struct let f x = X.g x
```

- ▶ X.g의 타입?

모듈함수의 정의

- ▶ 함수: 값을 받아서 값을 만드는 함수

```
fun f(x,y) = x+y
```

- ▶ 모듈함수: 모듈을 받아서 모듈을 만드는 함수

```
module F(X:S1, Y:S2) = struct ... end
```

- ▶ 참고: 인자로 받는 모듈의 타입을 자동유추할 수 있을까?
- ▶ 아직 불가능. 예)

```
module F(X) = struct let f x = X.g x
```

- ▶ X.g의 타입?
- ▶ 알려면, 만들어지는 모듈이 어디에서 어떻게 f를 사용하는 지 알아야.

모듈 프로그래밍

```
module type Animal =  
  sig  
    val age: int  
    val think: string -> bool  
    val feel: string -> bool  
  end
```

```
module Couple (Beauty: Animal, Beast: Animal) =  
  struct  
    let age = Beauty.age + Beast.age  
    let think x = (Beauty.think x)  
                  || (Beast.think x)  
    let feel x = (Beauty.feel x)  
                 && (Beast.feel x)  
  end
```

모듈 프로그래밍

```
module type CAR =  
  sig  
    type speed  
    type fuel  
    val accelerator: int -> speed  
    val break: int -> speed  
    val fill_tank: int -> fuel  
  end
```

```
module Porche =  
  struct  
    type speed = int  
    type fuel = EMPTY | FULL of int  
    let accelerator n = n**n  
    let break n = n/10  
    let fill_tank n = FULL n  
  end
```

모듈 프로그래밍

```
module type CAR =  
  sig  
    type speed  
    type fuel  
    val accelerator: int -> speed  
    val break: int -> speed  
    val fill_tank: int -> fuel  
  end
```

```
module DriverSchool(Car: CAR) =  
  struct  
    let speed_up n = Car.accelerator n  
    let slow_down n = Car.break n  
    let get_ready n = Car.fill_tank n  
  end
```

```
module TicoDriver = DriverSchool(Tico)  
module PorcheDriver = DriverSchool(Porche)
```

모듈 프로그래밍

```
module type STACK =  
  sig  
    type atom  
    type 'a stack  
    val empty_stack: atom stack  
    val push: atom * atom stack -> atom stack  
  end
```

```
module MakeStack(S: sig type t end) =  
  struct  
    type atom = S.t  
    type 'a stack = 'a list  
    let empty_stack = []  
    let push (x, stk) = x::stk  
  end
```

모듈 프로그래밍

```
structure IntStk = MakeStack(struct  
    type t = int  
end)
```

```
structure StrStk = MakeStack(struct  
    type t = string  
end)
```

```
structure PairStk =  
    MakeStack(struct  
        type t = int * string  
    end)
```