SNU 프로그래밍언어 특강

(2.3)

이 광근

kwangkeunyi.snu.ac.kr

여러모양타입$_{\text{polymorphism}}$의 종류

타입에 상관없는$_{\text{parametric}}$ vs 타입에 민감한$_{\text{ad-hoc}}$

- 타입에 상관없이 고르게 작동하는$_{\text{parametric polymorphism}}$
  - 예) 저울, id, list-length, $g \circ f$ (ftn compose)
  - "진심인" 여러모양타입
- 타입에 민감한데 아닌척하는$_{\text{ad-hoc polymorphism, overloading}}$
  - 예) 1+2, 1.2+3.4, "a"+"b", true+false
  - 예) print 1, print "a", print true
  - 예) myreading.accel, myavante.accel, starship.accel
  - "겉으로만" 여러모양타입

여러모양타입처럼 보이는 계산 지원하기

- 상식: 타입에 따라 다른 계산을 정의하도록

  print x = case type(x) of int → ... | bool → ...

- 상식+디자인: 타입클래스$_{\text{type class}}$
  - 타입클래스 = 타입모음 + 타입마다 같은 이름의 다른 구현을 담음
  - 타입클래스 사용법$_{\text{interface, 접속방안}}$ = 타입인자 + 이름들 타입

    class interface A a = (add: a→a→a, print: a→string)

    (클래스 이름 "A", 타입인자 "a")
  - 구체적인 타입마다 선언한 것들을 구현

    class instance A int = (add n m = n+m, print n = ...)

  - 따로따로 프로그래밍$_{\text{modular pgm'ng}}$: 외부에선 타입클래스 사용법$_{\text{interface, 접속방안}}$만 알고 프로그래밍

타입클래스 선언과 구현

- 타입클래스 선언 = 타입언어로 사용법$_{\text{interface, 접속방안}}$ 정의

  class interface A a = (add: a→a→a, print: a → string)

- 타입클래스를 타입별로 구현하기(타입클래스 식구로 포함시키기)

  class instance A int =
  (add n n' = n+n', print n = int2string n)
  class instance A bool =
  (add b b' = b orelse b', print b = bool2string b)

타입클래스 사용

따로따로 프로그래밍$_{\text{modular pgm'ng}}$ 지원
(부품 속구현 변동 $\not\Rightarrow$ 외부 사용코드 변동)

- 외부에선 타입클래스 사용법$_{\text{interface, 접속방안}}$에만 기대서
- 예)
  - 사용법(타입)
    class interface A a = (add: a→a→a, print: a → string)
  - 외부사용
    show x = print (add x x) $\qquad$ $:\forall a \in A.a \to \text{string}$
    show 3 $\cdots$ show false
  - 타입 상관없는척$_{\text{ad-hoc polymorphism}}$(show)
  - 제한된 여러모양타입$_{\text{bounded polymorphism}}$(show)

타입클래스 예: 모나드$_{\text{monad, 차례차례타입, 계산잇기타입}}$

- 순서가 핵심인 계산(입출력/메모리사용/예외상황처리) 프로그래밍을 위한 타입클래스
  - 유용: 그런게 없는 –계산순서에 무심한– 언어에서
  - 덜유용: 그런게 이미 있는 언어에서는
- 모나드 타입클래스 사용법$_{\text{interface, 접속방안}}$ (타입)

$$
\begin{array}{l}
\text{class interface Monad (a monad)} =\\
(\\
\quad \text{unit: a} \to \text{a monad},\\
\quad \text{>>=: a monad} \to (\text{a} \to \text{a monad}) \to \text{a monad}\\
)
\end{array}
$$

- maybe monad:

```
class interface Monad (a monad) =
(
    unit: a → a monad,
    >>=: a monad → (a → a monad) → a monad
)

class instance Monad (type a monad = Some of a | None) =
(
    unit x = Some x,
      None >>= _ = None
    Some x >>= f = f x
)
div x y = if y = 0 then None else (x / y)
(* code for (a / b) / c is: *)
(div a b) >>= fn r1 (div r1 c) >>= fn r2 r2
```

- state monad:

```
class interface Monad (a monad) =
(
    unit: a → a monad,
    >>=: a monad → (a → a monad) → a monad
)

class instance Monad (type a monad = mem → a × mem) =
(
    unit x = fn m (x,m),
    f >>= g = fn m (let (v1, m1) = f m in
                    let (v2, m2) = g v1 m1 in (v2, m2))
)
(* sequencing of e1 then e2 with memory m0 *)
(e1 >>= e2) m0
```

$\exists a.\tau$ 속구현감춘 데이터타입$_{\text{abstract data type}}$

$$
\begin{array}{llll}
\text{식} & E & \rightarrow & \cdots \\
 & & \mid & \mathsf{abs}\ \tau\ E \quad \text{만들기} \\
 & & \mid & \mathsf{adt}\ x\ E\ E \quad \text{사용하기}
\end{array}
$$

$$
\frac{\Gamma \vdash E : \{\tau/a\}\tau'}{\Gamma \vdash \mathsf{abs}\ \tau\ E : \exists a.\tau'}
$$

$$
\frac{\Gamma \vdash E_1 : \exists a.\tau \quad \Gamma + a + x : \tau \vdash E_2 : \tau'}{\Gamma \vdash \mathsf{adt}\ x\ E_1\ E_2 : \tau'}
$$

abs/adt, package/open, pack/unpack, pack/open, pack/letpack, ...

예)

```
adt  counter
     (abs int (new:int = 0, inc x:int = x+1, check x:int=x<10))
     counter.check(counter.inc(counter.new))
```

돌아보기: 타입의 효용 (1/2)

$$
\begin{array}{llll}
\text{타입} & \tau & \to & \iota \mid a \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau \\
 & & \mid & \mu a.\tau \mid \forall a.\tau \mid \forall a \in A.\tau \mid \exists a.\tau \mid \cdots
\end{array}
$$

- 프로그래밍언어를 이해하는/디자인하는 틀
  - 언어 구성자$_{\text{construct}}$ = 특정 타입의 값을 만들고 사용하는 방법
  - 언어 구성자 디자인 가이드 = 타입으로 따지는 의미가 논리증명 규칙과 대응하는지 여부
- 프로그래밍언어 실행전의미$_{\text{static semantics}}$의 어휘 (의미공간$_{\text{semantic domain}}$)

$$\Gamma \vdash E : \tau$$

돌아보기: 타입의 효용 (2/2)

$$
\begin{array}{llll}
\text{타입} & \tau & \to & \iota \mid a \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau \\
& & \mid & \mu a.\tau \mid \forall a.\tau \mid \forall a \in A.\tau \mid \exists a.\tau \mid \cdots
\end{array}
$$

- 프로그램 실행전 검산기술의 단서: 안전한 타입유추$_{\text{sound type inference}}$ 알고리즘

$$\mathcal{M}(\Gamma, E, \alpha)$$

- 따로따로 프로그래밍$_{\text{modular pgm'ng}}$의 기둥: 속구현 감춘, 사용법$_{\text{interface}}$ 서술언어

class interface A a = (add: a→a→a, print: a→string)

커리-하워드 대응$_{\text{curry-howard correspondence}}$

프로그램과 증명은 동전의 양면

$$\begin{array}{rcl} \text{프로그램체계} & \equiv & \text{논리체계} \\ \text{프로그램} & \overset{\text{거울}}{\longleftrightarrow} & \text{증명} \\ \text{타입} & \overset{\text{거울}}{\longleftrightarrow} & \text{명제} \end{array}$$

- 프로그램 $\overset{\text{def}}{=}$ 타입잡고 짜는 값중심언어$_{\text{applicative language}}$식
- 증명 $\overset{\text{def}}{=}$ 각잡고 하는 증명$_{\text{formal proof}}$
  - 정해진+기계적인 증명규칙$_{\text{proof rule}}$만 사용: 논리추론의 징검다리
  - 예) 직관논리$_{\text{intuitionistic logic, constructive logic}}$, 고전논리$_{\text{classical logic}}$

증명하기		프로그램짜기
"논리적인 비약없이 새로운 사실을 확인해가는 과정이다."	$\longleftrightarrow$	공짜없이 새로운 데이터를 만들어가는 과정이다.
"사실을 기반으로 해서 새로운 사실들을 만들어 간다."	$\longleftrightarrow$	이미 만든 데이터를 가지고 새로운 데이터들을 만들어 간다.
"만들어가는 과정은 논리적으로 누구나 수긍하는 추론의 징검다리만을 밟고 가는 과정만 있다."	$\longleftrightarrow$	새 데이터를 만드는 과정은 사용하는 프로그래밍 언어에서 제공하는 프로그램 조립방식만을 써서 만든다.

직관논리intuitionistic/constructive propositional logic

$$\text{논리식} \quad f \quad \to \quad T \mid F \mid f \wedge f \mid f \vee f \mid f \Rightarrow f$$

$$\text{증명규칙} \quad \Gamma \vdash f \qquad (\Gamma \subseteq \text{논리식}, \quad [\![\wedge\Gamma \Rightarrow f]\!] = \text{true인})$$

$$\frac{}{\Gamma \vdash T} \quad \frac{}{\Gamma \vdash f}\, f \in \Gamma \qquad\qquad \frac{\Gamma \vdash F}{\Gamma \vdash f}$$

$$\frac{\Gamma \vdash f_1 \quad \Gamma \vdash f_2}{\Gamma \vdash f_1 \wedge f_2} \qquad\qquad \frac{\Gamma \vdash f_1 \wedge f_2}{\Gamma \vdash f_1}$$

$$\frac{\Gamma \vdash f_1}{\Gamma \vdash f_1 \vee f_2} \qquad \frac{\Gamma \vdash f_1 \vee f_2 \quad \Gamma \cup \{f_1\} \vdash f_3 \quad \Gamma \cup \{f_2\} \vdash f_3}{\Gamma \vdash f_3}$$

$$\frac{\Gamma \cup \{f_1\} \vdash f_2}{\Gamma \vdash f_1 \Rightarrow f_2} \qquad\qquad \frac{\Gamma \vdash f_1 \Rightarrow f_2 \quad \Gamma \vdash f_1}{\Gamma \vdash f_2}$$

증명: 증명나무$_{\text{proof tree}}$

$\Gamma \stackrel{\text{let}}{=} \{(A \Rightarrow B) \wedge (A \Rightarrow C), A\}$,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash (A \Rightarrow B) \wedge (A \Rightarrow C)}}{\Gamma \vdash A \Rightarrow B} \qquad \dfrac{}{\Gamma \vdash A}}{\Gamma \vdash B} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash A \Rightarrow C} \quad \dfrac{}{\Gamma \vdash A}}{\Gamma \vdash C}}{\Gamma \vdash B \wedge C}}{\Gamma \setminus \{A\} \vdash A \Rightarrow (B \wedge C)}}{\vdash (A \Rightarrow B) \wedge (A \Rightarrow C) \Rightarrow (A \Rightarrow (B \wedge C))}$$

커리-하워드 대응$_{\text{curry-howard correspondence}}$ I

$$
\begin{array}{llll}
\text{식} & E & \rightarrow & () \mid x \mid \texttt{fn}\, x\, E \mid E\, E \\
 & & \mid & (E,E) \mid E\texttt{.l} \mid E\texttt{.r} \\
 & & \mid & \texttt{inl}\, E \mid \texttt{inr}\, E \mid \texttt{case}\, E\, (\texttt{inl}\, x\, E)\, (\texttt{inr}\, x\, E)
\end{array}
$$

$$
\text{타입} \quad \tau \rightarrow \iota \mid \tau \rightarrow \tau \mid \tau \times \tau \mid \tau + \tau
$$

- 직관논리$_{\text{intuitionistic logic}}$시스템과
  커리-하워드 대응$_{\text{curry-howard correspondence}}$:

$$
\Gamma \vdash E : \tau \Longleftrightarrow |\Gamma| \vdash |\tau|
$$

- $\vdash E : \tau$인 프로그램 E는 곧, $\vdash |\tau|$의 증명나무

참고) "Proofs are Programs: 19th Century Logic and 21st Century Computing", P. Wadler. "Lectures on the Curry-Howard Isomorphism", M. Sørensen and P. Urzyczyn

$$
\begin{array}{rcl}
|\Gamma| & = & \{|\tau| \mid x : \tau \in \Gamma\} \\
|\iota| & = & T \\
|\tau \to \tau'| & = & |\tau| \Rightarrow |\tau'| \\
|\tau \times \tau'| & = & |\tau| \wedge |\tau'| \\
|\tau + \tau'| & = & |\tau| \vee |\tau'|
\end{array}
$$

고전논리classical propositional logic

$$\text{논리식} \quad f \quad \to \quad T \mid F \mid f \wedge f \mid f \vee f \mid f \Rightarrow f$$
$$\mid \quad \neg f \quad (f \Rightarrow F \text{ 의 설탕})$$

$$\text{증명규칙} \quad \Gamma \vdash f \qquad (\Gamma \subseteq \text{논리식}, \quad [\![\wedge\Gamma \Rightarrow f]\!] = \text{true인})$$

$$\frac{}{\Gamma \vdash T} \quad \frac{}{\Gamma \vdash f}\, f \in \Gamma \qquad \frac{\Gamma \vdash \neg\neg f}{\Gamma \vdash f}$$

$$\frac{\Gamma \vdash f_1 \quad \Gamma \vdash f_2}{\Gamma \vdash f_1 \wedge f_2} \qquad \frac{\Gamma \vdash f_1 \wedge f_2}{\Gamma \vdash f_1}$$

$$\frac{\Gamma \vdash f_1}{\Gamma \vdash f_1 \vee f_2} \qquad \frac{\Gamma \vdash f_1 \vee f_2 \quad \Gamma \cup \{f_1\} \vdash f_3 \quad \Gamma \cup \{f_2\} \vdash f_3}{\Gamma \vdash f_3}$$

$$\frac{\Gamma \cup \{f_1\} \vdash f_2}{\Gamma \vdash f_1 \Rightarrow f_2} \qquad \frac{\Gamma \vdash f_1 \Rightarrow f_2 \quad \Gamma \vdash f_1}{\Gamma \vdash f_2}$$

$$\frac{\Gamma \vdash \neg\neg f}{\Gamma \vdash f}$$ 하나면 아래 규칙들 불필요:

$$\frac{\Gamma \vdash F}{\Gamma \vdash f} \qquad \frac{\Gamma \cup \{f\} \vdash F}{\Gamma \vdash \neg f} \qquad \frac{\Gamma \vdash f \quad \Gamma \vdash \neg f}{\Gamma \vdash F}$$

왜) $\neg f \stackrel{\text{def}}{=} (f \Rightarrow F)$ 이고,

$$\cfrac{\cfrac{\cfrac{\Gamma \vdash F}{\Gamma \cup \{\neg f\} \vdash F}}{\Gamma \vdash \neg\neg f}}{\Gamma \vdash f}$$

참고) $\vdash f \vee \neg f$ (모아니면도$_{\text{excluded middle}}$) 증명가능, 직관논리에서는 불가능.

$\dfrac{\Gamma \vdash \neg\neg f}{\Gamma \vdash f}$ 대신, 아래 두개로 대체가능:

$$\frac{\Gamma \vdash F}{\Gamma \vdash f} \qquad \frac{\Gamma \cup \{f \Rightarrow F\} \vdash f}{\Gamma \vdash f}$$

왜)

$$\Gamma \vdash \neg\neg f$$

즉

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash f \Rightarrow F \Rightarrow F}{\Gamma \cup \{f \Rightarrow F\} \vdash f \Rightarrow F \Rightarrow F \qquad \Gamma \cup \{f \Rightarrow F\} \vdash f \Rightarrow F}}{\Gamma \cup \{f \Rightarrow F\} \vdash F}}{\Gamma \cup \{f \Rightarrow F\} \vdash f}}{\Gamma \vdash f}$$

커리-하워드 대응$_{\text{curry-howard correspondence}}$ II

$$
\begin{array}{llll}
\text{식} & E & \to & \cdots \\
 & & | & \texttt{catch}\,x\,E \mid \texttt{throw}\,x\,E
\end{array}
$$

$$
\text{타입} \quad \tau \ \to \ \iota \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau
$$

- 고전논리$_{\text{classical logic}}$시스템과
  커리-하워드 대응$_{\text{curry-howard correspondence}}$:

$$\Gamma \vdash E : \tau \iff |\Gamma| \vdash |\tau|$$

-

$$
\frac{\Gamma + x : \tau \to \tau' \vdash E : \tau'}{\Gamma \vdash \texttt{catch}\,x\,E : \tau} \quad \overset{\text{거울}}{\longleftrightarrow} \quad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}
$$

$$
\frac{\Gamma(x) = \tau \to \tau' \qquad \Gamma \vdash E : \tau}{\Gamma \vdash \texttt{throw}\,x\,E : \tau''} \quad \overset{\text{거울}}{\longleftrightarrow} \quad \frac{\Gamma \vdash F}{\Gamma \vdash A}
$$

- $\vdash E : \tau$인 프로그램 E는 곧, $\vdash |\tau|$의 증명나무

참고) "A Formulae-as-Types Notion of Control", POPL'90, T. Griffin

다시: 어느 디자인이 “좋은가”? (“맞을까”?)

$$C[\mathtt{catch}\,x\,E] \;\to\; C[\{\lambda v.C[v]/x\}E]$$

vs

$$C[\mathtt{catch}\,x\,E] \;\to\; \{\lambda v.C[v]/x\}E$$

- 두번째 디자인:

$$\frac{\Gamma + x : \tau \to \tau' \vdash E : \tau'}{\Gamma \vdash \mathtt{catch}\,x\,E : \tau} \quad \overset{\text{거울}}{\longleftrightarrow} \quad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$

- 첫번째 디자인:

$$\frac{\Gamma + x : \tau \to \tau' \vdash E : \tau}{\Gamma \vdash \mathtt{catch}\,x\,E : \tau} \quad \overset{\text{거울}}{\longleftrightarrow} \quad \frac{\Gamma \cup \{A \Rightarrow F\} \vdash A}{\Gamma \vdash A}$$

커리-하워드 대응$_{\text{curry-howard correspondence}}$

$$\vdash E : \tau$$

이면

- 프로그램 E는, 증명

$$\frac{\nabla}{\vdash |\tau|}$$

  를 표현
- τ는 증명한 명제 $|\tau|$

따라서,

- 증명하기 = 프로그램짜기
- 증명이 맞는지 검사하기 = 프로그램 타입 검사하기
- 증명한 명제 = 프로그램의 타입

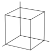
값에기댄 타입$_{\text{dependent type}}$: 쓰임새

- 증명도우미 시스템$_{\text{proof-assistant system}}$, 증명중심 프로그래밍$_{\text{proof-oriented pgm'ng}}$에서
- 증명대상인 명제가·타입이 값(계산식)도 품을 수 있어야
  - inc : $\forall n \in \mathbb{Z}.\text{int}(n) \to \text{int}(n+1)$
    inc 프로그램
    = 증명[정수 n이 있다고하자. 그러면 정수 $n+1$을 만들수있다]
  - mul : $\forall m, n, l \in \mathbb{N}.\text{mat}(m, n) \to \text{mat}(n, l) \to \text{mat}(m, l)$
    mul 프로그램
    = 증명[$m \times n, n \times l$ 행렬이 있으면 $m \times l$ 행렬을 만들수있다]
  - sort : $\forall n \in \mathbb{N}.\ i$:int array$(n) \to$
    o:int array$(n) \times$ sorted(o) $\times$ permute(i,o)
    sort 프로그램
    = 증명[크기 n인 정수열이 있으면 정열된 결과를 만들수 있다]

값에기댄 타입$_{\text{dependent type}}$의 완결판

CoC$_{\text{calculus of constructions}}$ $\stackrel{\text{def}}{=}$ $\lambda \times \omega \times F \times \Pi$

- 하나된 계산세계: 값/타입 구분없이 계산 대상/결과
- {값, 타입}이 {값, 타입}계산에
- 증명도우미$_{\text{proof assistant}}$ 시스템의 핵

* 람다큐브$_{\text{lambda cube}}$  (값과 타입을 섞는 8가지)
  - 원점: 값이 값계산에 λ
  - x축: 타입이 타입계산에 ω
  - y축: 타입이 값계산에 F
  - z축: 값이 타입계산에 Π

참고) "Lambda Calculi with Types", H. Barendregt, in *Handbook of Logic in Computer Science, Vol.II*, 1992. "The calculus of constructions", T. Coquand and G. Huet, 1986, 1988.

구분: 값, 타입, 타입의^{i}타입

$$\vdots$$

$$\mathsf{Type}_2, \cdots$$

$$\mathsf{Type}_1, \mathsf{Type}_1 \to \mathsf{Type}_1, \cdots$$

$$\mathsf{nat}, \quad \mathsf{bool}, \quad \mathsf{nat} \to \mathsf{bool}, \quad \mathsf{nat} \times \mathsf{bool}, \quad \mathsf{nat} + \mathsf{bool}$$

$$\forall a.a \to \mathsf{nat}, \quad \exists a.a \times \mathsf{nat}, \quad \mu a.\mathsf{nat} + a, \quad \lambda a.a\ \mathsf{list}, \quad \cdots$$

$$1, \quad 2, \quad \mathsf{true}, \quad (1, \mathsf{false}), \quad \mathsf{Leaf}\ 8$$

$$\lambda x.x, \quad \lambda x.\lambda y.x + y, \quad [1, 2], \quad \cdots$$

구분: 값, 타입, 타입의^{i}타입

$$\Box$$

Type, Type $\to$ Type, $\cdots$

---

nat, bool, nat $\to$ bool, nat $\times$ bool, nat $+$ bool

$\forall a.a \to$ nat, $\exists a.a \times$ nat, $\mu a.$nat $+ a$, $\lambda a.a$ list, $\cdots$

1, 2, true, $(1,$ false$)$, Leaf 8

$\lambda x.x$, $\lambda x.\lambda y.x + y$, $[1, 2]$, $\cdots$

하나로: 값, 타입, 타입의^{i}타입

$$\Box$$

$$\mathsf{Type},\ \mathsf{Type} \to \mathsf{Type}, \cdots$$

$$\mathsf{nat}, \quad \mathsf{bool}, \quad \mathsf{nat} \to \mathsf{bool}, \quad \mathsf{nat} \times \mathsf{bool}, \quad \mathsf{nat} + \mathsf{bool}$$

$$\forall x{:}A.x \to \mathsf{nat}, \quad \exists x{:}A.x \times \mathsf{nat}, \quad \lambda x{:}A.x\ \mathsf{list}, \quad \cdots$$

$$1, \quad 2, \quad \mathsf{true}, \quad (1, \mathsf{false}), \quad \mathsf{Leaf}\ 8$$

$$\lambda x{:}A.x, \quad \lambda x{:}A.\lambda y{:}B.x + y, \quad [1, 2], \quad \cdots$$

하나로: 값, 타입, 타입의^{i}타입

$$\square$$

$$\text{Type}, \text{Type} \to \text{Type}, \cdots$$

$$\text{nat}, \quad \text{bool}, \quad \text{nat} \to \text{bool}, \quad \text{nat} \times \text{bool}, \quad \text{nat} + \text{bool}$$

$$\forall x{:}A.x \to \text{nat}, \quad \exists x{:}A.x \times \text{nat}, \quad \lambda x{:}A.x\ \text{list}, \quad \cdots$$

$$1, \quad 2, \quad \text{true}, \quad (1, \text{false}), \quad \text{Leaf}\ 8$$

$$\lambda x{:}A.x, \quad \lambda x{:}A.\lambda y{:}B.x + y, \quad [1, 2], \quad \cdots$$

표기법

- $A \to B$, $\forall x{:}A.B$ 은 설탕 모두 $\Pi x{:}A.B$로
  - $A \to B \ \equiv\ \Pi x{:}A.B \quad (x \notin B$ 이면$)$
- $A \times B$, $\exists x{:}A.B$, $A + B$ 은 설탕 모두 $\Sigma x{:}A.B$로
  - $A \times B \ \equiv\ \Sigma x{:}A.B \quad (x \notin B$ 이면$)$
  - $C(l) + C(r) \ \equiv\ \Sigma x{:}\text{Tag}.C(x) \quad (\text{Tag} = \{l, r\})$

CoC$_{\text{calculus of constructions}}$: 하나된 계산법

값식$_{\text{value term}}$ 타입식$_{\text{type term}}$ 구분없이 E

식	E	$\rightarrow$	$n \mid$ nat	기본값 | 기본타입
		$\mid$	$\Pi x{:}E.E \mid$ Type $\mid \square$	타입 | 타입의$^+$타입
		$\mid$	x	값|타입 변수
		$\mid$	$\lambda x{:}E.E$	값|타입 함수
		$\mid$	$E\ E$	값|타입 적용

제약사항:

* $(\Pi \mid \lambda)x{:}E.E'$ 에서 $x \notin E$
* 묶인이름은 모두 고유함 $(x \cdots \lambda y \cdots \Pi z \cdots)$

식 중에서 타입식$_{\text{type term}}$은 A, B 로

CoC 식의 예

- 값으로 값

  $\lambda x{:}\mathsf{nat}.x \quad : \quad \Pi x{:}\mathsf{nat}.\mathsf{nat}$

- 타입으로 값

  $\lambda a{:}\mathsf{Type}.\lambda x{:}a.x \quad : \quad \Pi a{:}\mathsf{Type}.\Pi x{:}a.a$

- 타입으로 타입

  $\lambda a{:}\mathsf{Type}.a\ \mathsf{list} \quad : \quad \Pi a{:}\mathsf{Type}.\mathsf{Type}$

- 값으로 타입

  $\lambda n{:}\mathsf{nat}.\mathsf{int\ list}(n) \quad : \quad \Pi n{:}\mathsf{nat}.\mathsf{Type}$

- 값계산

  $(\lambda a{:}\mathsf{Type}.\lambda x{:}\mathsf{a}.x)\ \mathsf{nat}\ 3 \ \to^* \ 3$

- 타입계산

  $(\lambda a{:}\mathsf{Type}.\lambda n{:}\mathsf{nat}.a\ \mathsf{list}(n+1))\ \mathsf{int}\ 3 \ \to^* \ \mathsf{int\ list}(4)$

- inc :

  $\Pi n{:}\mathsf{nat}.\Pi x{:}\mathsf{nat}(n).\mathsf{nat}(n+1)$

  즉, $\forall n \in \mathbb{N}.\mathsf{nat}(n) \to \mathsf{nat}(n+1)$
- mul :

  $\Pi m, n, l{:}\mathsf{nat}.\Pi x{:}\mathsf{mat}(m, n).\Pi y{:}\mathsf{mat}(n, l).\mathsf{mat}(m, l)$

  즉, $\forall m, n, l \in \mathbb{N}.\mathsf{mat}(m, n) \to \mathsf{mat}(n, l) \to \mathsf{mat}(m, l)$
- cons :

  $\Pi n \in \mathsf{nat}.\Pi x{:}\mathsf{int}.\Pi l{:}\mathsf{int\ list}(n).\mathsf{int\ list}(n+1)$

  즉, $\forall n \in \mathbb{N}.\mathsf{int} \to \mathsf{int\ list}(n) \to \mathsf{int\ list}(n+1)$

- sort :

  $\Pi n{:}\mathsf{nat}.\Pi x{:}\mathsf{array}(n).\mathsf{sorted\text{-}array}(x)$

  즉, $\forall n \in \mathbb{N}.x{:}\mathsf{array}(n) \ \rightarrow \ \mathsf{sorted\text{-}array}(x)$
- sort :

  $\Pi n{:}\mathsf{nat}.\Pi x{:}\mathsf{array}(n).\Sigma y{:}\mathsf{array}(n).\mathsf{permuted\text{-}sorted}(x, y)$

  즉,

  $\forall n \in \mathbb{N}.x{:}\mathsf{array}(n) \rightarrow (y{:}\mathsf{array}(n) \wedge \mathsf{permuted\text{-}sorted}(x, y))$
- sort :

  $\Pi n{:}\mathsf{nat}.\Pi x{:}\mathsf{array}(n).\Sigma y{:}\mathsf{array}(n).\Sigma z{:}\mathsf{sorted}(y).\mathsf{permuted}(x, y)$

  즉, $\forall n \in \mathbb{N}.x{:}\mathsf{array}(n) \ \rightarrow$

  $(y{:}\mathsf{array}(n) \ \wedge \ \mathsf{sorted}(y) \ \wedge \ \mathsf{permuted}(x, y))$

CoC$_{\text{calculus of constructions}}$: 하나된 계산법

값식$_{\text{value term}}$ 타입식$_{\text{type term}}$ 구분없이 E

식	E	$\rightarrow$	n | nat	기본값 | 기본타입
		|	$\Pi x{:}E.E$ | Type | $\square$	타입 | 타입의$^+$타입
		|	x	값|타입 변수
		|	$\lambda x{:}E.E$	값|타입 함수
		|	$E\ E$	값|타입 적용

제약사항:

- $(\Pi \mid \lambda)x{:}E.E'$ 에서 $x \notin E$
- 묶인이름은 모두 고유함 $(x \cdots \lambda y \cdots \Pi z \cdots)$

식 중에서 타입식$_{\text{type term}}$은 A, B 로

CoC 실행전의미$_{\text{static semantics, type rules}}$ $\Gamma \vdash E : A$

$\Gamma \in \mathsf{Var} \xrightarrow{\text{fin}} \mathsf{TypeTerm}$ $\qquad s \in \{\mathsf{Type}, \Box\}$

$$\frac{}{\Gamma \vdash n : \mathsf{nat}} \qquad \frac{}{\Gamma \vdash \mathsf{nat} : \mathsf{Type}} \qquad \frac{}{\Gamma \vdash \mathsf{Type} : \Box}$$

$$\frac{\Gamma \vdash A : s \quad \Gamma + x{:}A \vdash B : s'}{\Gamma \vdash \Pi x{:}A.B : s'}$$

$$\frac{x{:}A \ \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma + x{:}A \vdash E : B \qquad \Gamma \vdash \Pi x{:}A.B : s}{\Gamma \vdash \lambda x{:}A.E : \Pi x{:}A.B}$$

$$\frac{\Gamma \vdash E_1 : \Pi x{:}A.B \qquad \Gamma \vdash E_2 : A}{\Gamma \vdash E_1\ E_2 : \{E_2/x\}B}$$

$$\frac{\Gamma \vdash E : A \qquad \Gamma \vdash B : s \qquad A =_\beta B}{\Gamma \vdash E : B}$$

바꿔치기$_{\text{substitution}}$ $\{E'/x\}E$

$$
\begin{aligned}
Sn \mid S\mathsf{nat} \mid S\mathsf{Type} &= n \mid \mathsf{nat} \mid \mathsf{Type} \\
Sx &= \begin{cases} E & \text{if } E/x \in S \\ x & \text{if } E/x \notin S \end{cases} \\
S(\Pi x{:}E.E') &= \Pi x{:}(SE).(SE') \quad (x \notin S) \\
S(\lambda x{:}E.E') &= \lambda x{:}(SE).(SE') \quad (x \notin S) \\
S(E_1 E_2) &= (SE_1)(SE_2)
\end{aligned}
$$

- 타입시스템은 안전함$_{\text{type safety}}$
  - 타입갖춘 식은 항상 끝나고$_{\text{strong normalization}}$

    타입이 유지됨$_{\text{type preservation}}$

CoC 실행의미$_{\text{dynamic semantics}}$ $E \rightarrow E'$

$$\frac{}{(\lambda x{:}A.E)\ E' \rightarrow \{E'/x\}E}$$

$$\frac{E_1 \rightarrow E_1'}{E_1\ E_2 \rightarrow E_1'\ E_2} \qquad \frac{E_2 \rightarrow E_2'}{E_1\ E_2 \rightarrow E_1\ E_2'}$$

$$\frac{A \rightarrow A'}{\Pi x{:}A.E \rightarrow \Pi x{:}A'.E} \qquad \frac{E \rightarrow E'}{\Pi x{:}A.E \rightarrow \Pi x{:}A.E'}$$

계산은

- 베타계산$_{\beta\text{-reduction}}$뿐이고
- 순서 상관없이 만나게 됨$_{\text{confluence}}$

CoC 쓰임새: 증명도우미$_{\text{proof assistant}}$ 시스템의 실현

커리-하워드 대응$_{\text{curry-howard correspondence}}$과
값에기댄 타입$_{\text{dependent type}}$ 덕에

$$\begin{aligned} \text{CoC의 풍부한 타입} &= \text{풍부한 명제} \\ \text{그 타입의 CoC 식} &= \text{그 명제의 증명} \\ \text{그 CoC 식의 타입검사} &= \text{그 증명의 검산} \end{aligned}$$

CoC 타입으로 논리명제를 표현

- 성질$_{\text{predicate}}$ P = 어떤 집합 X의 원소들에 대한 명제
  CoC에서 P는 아래 타입의 이름

$$P : X \to \mathsf{Type}$$

- 따라서,

명제	CoC 타입
$\forall x \in X.P(x)$	$\Pi x : X.P(x)$

커리-하워드 대응$_{\text{curry-howard correspondence}}$

CoC $\xleftrightarrow{\text{거울}}$ 명제논리$_{\text{predicate logic, 술어논리, 성질논리, 모든어떤논리}}$

CoC타입규칙	논리증명규칙

$$\frac{\Gamma + x{:}X \vdash P(x)}{\Gamma \vdash \forall x{:}X.P(x)}$$

$$\frac{\Gamma \vdash \forall x{:}X.P(x) \qquad \Gamma \vdash t : X}{\Gamma \vdash P(t)}$$

주의) 정확한 양방향 대응은 아님 (회색 $\xleftrightarrow{\text{거울}}$ 인 이유). $\longleftarrow$는 사실이지만 $\longrightarrow$는 아님.

예를들어, CoC 식에서 $\Pi a{:}\mathrm{Type}$의 a는 자기 타입을 포함한$_{\text{impredicative}}$ 모든 타입에 걸치지만, 명제논리에서 $\forall a$의 a는 하위 공간에만$_{\text{predicative}}$ 걸침.

커리-하워드 대응$_{\text{curry-howard correspondence}}$

CoC $\xleftrightarrow{\text{거울}}$ 명제논리$_{\text{predicate logic, 술어논리, 성질논리, 모든어떤논리}}$

CoC타입규칙	논리증명규칙
$\dfrac{\Gamma + x{:}A \vdash E : B \qquad \Gamma \vdash \Pi x{:}A.B : s}{\Gamma \vdash \lambda x{:}A.E : \Pi x{:}A.B}$	$\dfrac{\Gamma + x{:}X \vdash P(x)}{\Gamma \vdash \forall x{:}X.P(x)}$
$\dfrac{\Gamma \vdash E_1 : \Pi x{:}A.B \qquad \Gamma \vdash E_2 : A}{\Gamma \vdash E_1\ E_2 : \{E_2/x\}B}$	$\dfrac{\Gamma \vdash \forall x{:}X.P(x) \qquad \Gamma \vdash t : X}{\Gamma \vdash P(t)}$

주의) 정확한 양방향 대응은 아님 (회색 $\xleftrightarrow{\text{거울}}$ 인 이유). $\longleftarrow$는 사실이지만 $\longrightarrow$는 아님.
예를들어, CoC 식에서 $\Pi a{:}$Type의 a는 자기 타입을 포함한$_{\text{impredicative}}$ 모든 타입에 걸치지만, 명제논리에서 $\forall a$의 a는 하위 공간에만$_{\text{predicative}}$ 걸침.

커리-하워드 대응curry-howard correspondence

CoC $\xleftrightarrow{\text{거울}}$ 명제논리predicate logic, 술어논리, 성질논리, 모든어떤논리

CoC타입규칙	논리증명규칙
$\dfrac{\Gamma + x{:}A \vdash E : P(x) \qquad \Gamma \vdash \Pi x{:}A.P(x) : s}{\Gamma \vdash \lambda x{:}A.E : \Pi x{:}A.P(x)}$	$\dfrac{\Gamma + x{:}X \vdash P(x)}{\Gamma \vdash \forall x{:}X.P(x)}$
$\dfrac{\Gamma \vdash E_1 : \Pi x{:}A.P(x) \qquad \Gamma \vdash E_2 : A}{\Gamma \vdash E_1\ E_2 : P(E_2)}$	$\dfrac{\Gamma \vdash \forall x{:}X.P(x) \qquad \Gamma \vdash t : X}{\Gamma \vdash P(t)}$

주의) 정확한 양방향 대응은 아님 (회색 $\xleftrightarrow{\text{거울}}$ 인 이유). $\longleftarrow$는 사실이지만 $\longrightarrow$는 아님.

예를들어, CoC 식에서 $\Pi a{:}\text{Type}$의 a는 자기 타입을 포함한impredicative 모든 타입에 걸치지만, 명제논리에서 $\forall a$의 a는 하위 공간에만predicative 걸침.

CoC 식으로 명제증명을 표현

- 명제 $(\forall x{:}X.\forall y{:}X.P\ x\ y) \Rightarrow (\forall x{:}X.P\ x\ x)$ 는
  CoC 타입으로 $(\Pi x{:}X.\Pi y{:}X.P\ x\ y) \to (\Pi z{:}X.P\ z\ z)$
- 위 명제의 증명은

$$\bigtriangledown \text{ (칠판)}$$

이고, CoC 식으로는

$$(\lambda k{:}(\Pi x{:}X.\Pi y{:}X.P\ x\ y).\lambda z{:}X.k\ z\ z) \stackrel{\text{let}}{=} E$$

이다. 왜냐면,

$$\begin{array}{l} X{:}\text{Type}, P{:}X \to X \to \text{Type} \\ \vdash \\ E : (\Pi x{:}X.\Pi y{:}X.P\ x\ y) \to (\Pi z{:}X.P\ z\ z) \end{array}$$