lec 21-22

4.4.2025 - 7.4.25

<u>Property</u>

The subject-construction theorem :

Let Δ be a TA_λ-deduction of a formula $\Gamma \mapsto M : \tau$.

(i) If we remove from each formula in Δ everything except its subject, Δ changes to a tree of terms which is exactly the construction-tree for M.

(ii) If M is an atom, $M \equiv x$, then $\Gamma = \{x : \tau\}$ and Δ contains only one formula, namely the axiom

$$x : \tau \mapsto x : \tau$$

(iii) If $M \equiv PQ$ the last step in Δ must be an application of $(\to E)$ to two formulae with form

$$\Gamma \restriction P \mapsto P : \sigma \to \tau \qquad \Gamma \restriction Q \mapsto Q : \sigma \quad \text{for some } \sigma.$$

restriction $\text{Subjects}(\Gamma) = FV(P)$

(iv) If $M \equiv \lambda x . P$ then τ must have form $\rho \to \sigma$

if $x \in FV(P)$ the last step in Δ must be an application of $(\to I)_{main}$ to

$$\Gamma, x : \rho \mapsto P : \sigma$$

if $x \notin FV(P)$ the last step in Δ must be an application of $(\to I)_{vac}$ to

$$\Gamma \mapsto P : \sigma$$

Deductions in TA_λ may not be unique

<u>Example</u> :-

$$\Delta_M \quad \dfrac{\dfrac{\dfrac{y : a \mapsto y : a}{\mapsto (\lambda y . y) : a \to a}(\to I)}{\mapsto (\lambda x . \lambda y . y) : (\sigma \to \sigma) \to (a \to a)}(\to I) \qquad \dfrac{z : \sigma \mapsto z : \sigma}{\mapsto (\lambda z . z) : \sigma \to \sigma}(\to I)}{\mapsto (\lambda x . \lambda y . y)(\lambda z . z) : a \to a}(\to E)$$

here $M \equiv (\lambda x . \lambda y . y)(\lambda z . z)$ $\quad \tau \equiv a \to a$ $\quad \Gamma = \phi$

here σ can be anything and this makes the Δ not unique.

(Property)

Uniqueness of deductions for normal forms.

Let M be a β-nf and Δ a TA_λ-deduction of $\Gamma \mapsto M:\tau$.

Then (i) every type in Δ has an occurrence in τ or in a type in Γ,

(ii) Δ is unique, i.e., if Δ' is also a deduction of $\Gamma \mapsto M:\tau$ then $\Delta' \equiv \Delta$.

Subject reduction and expansion (Property)

If P has type τ we can think of P as being in some sense "safe".

If P represents a stage in some computation which continues by β-reducing P then all later stages in the computation are also "safe". (~~Unsafe~~ means mismatch of types.)

Subject-reduction theorem :-

If $\Gamma \vdash_\lambda P:\tau$ and $P \rhd_\beta Q$ then $\Gamma \vdash_\lambda Q:\tau$

$\vdash_\lambda$ means there is a deduction of $\langle \Gamma, P, \tau\rangle$ in TA_λ.

~~Exact~~ Proof :-

$P \equiv (\lambda x.M)N \qquad Q \equiv M[N/x]$

let $x \in FV(M)$, then by the subject-construction theorem the lower steps of Δ must have the form

$$\dfrac{\dfrac{\Gamma_1,\ x:\sigma \mapsto M:\tau}{\Gamma_1 \mapsto (\lambda x.M):\sigma\to\tau}(\to I)\text{ main} \qquad \Gamma_2 \mapsto N:\sigma}{\Gamma_1 \cup \Gamma_2 \mapsto \underbrace{((\lambda x.M)N)}_{Q}:\tau}\quad Q:\tau$$

Now $\Gamma = \Gamma_1 \cup \Gamma_2$ and $\text{Subjects}(\Gamma) = FV(P)$

so we have a deduction for $\Gamma \mapsto P:\tau$

but $(\lambda x.M)N \rhd_\beta Q$. i.e $P \rhd_\beta Q$.

So we also have a deduction for $\Gamma \mapsto Q:\tau$. $\square$.

Subject-expansion theorem :-

If $\Gamma \vdash_\lambda Q:\tau$ and $P \rhd_\beta Q$ [*] then $\Gamma \vdash_\lambda P:\tau$.

[*] by non-duplicating and non-cancelling contractions.

The above condition [*] is very important. Removing it will make the conclusion false.

4.4.2025

Defn: (β-contraction)

beta-redux is that which can reduce.

a β-redex is any term $(\lambda x . M)N$

its contractum is $M[N/x]$

its rewrite rule is $(\lambda x \cdot M)N \triangleright_{1\beta} M[N/x]$

Iff P contains a β-redex-occurrence $R \equiv (\lambda x . M)N$ and Q is the result of replacing this by $M[N/x]$, we say P β-contracts to Q ($P \triangleright_{1\beta} Q$)

we call (P, R, Q) a β-contraction of P.

(Property)

Lemma: $P \triangleright_{1\beta} Q \implies FV(P) \supseteq FV(Q)$

Defn (β-reduction)

a β-reduction of a term P is a finite or infinite sequence of β-contractions with form

$$\langle P_1, R_1, Q_1 \rangle, \ldots \langle P_2, R_2, Q_2 \rangle \ldots$$

$$P_1 \equiv_\alpha P \qquad Q_i \equiv_\alpha P_{i+1} \quad (i = 1, 2, \ldots).$$

Iff there is a reduction from P to Q we say P β-reduces to Q

$$P \triangleright_\beta Q$$

NB: α-conversions are allowed in a β-reduction.

Defn: (β-conversion)

Iff we can change P to Q by a finite sequence of β-reductions and reversed β-reductions we say P β-converts to Q or P is β-equal to Q.

$$P =_\beta Q$$

a reversed β-reduction is called β-expansion.

$P \triangleright_\beta Q$: P β-reduces to Q

$P =_\beta Q$: P β-reduces to Q and Q β-reduces to P.

* let $X_F = YF$ X_F beta-converts to YF

Prove: $F X_F =_\beta X_F$

(i) ~~EXF.~~ $F X_F \triangleright_\beta X_F$

(ii) $X_F \triangleright_\beta F X_F$

Y here is Y combinator

(i) $\cancel{F X_F} = F(YF)$
$= F(F(\underline{YF}))$
$= F(F X_F)$
$= \cancel{FX}$

$F X_F = F(YF)$
$= YF$ (property of "Y")
$= X_F$

(ii) $X_F = YF$
$= F(\underline{YF})$ (property of "Y".)
$= F X_F$

Defn (β-normal form) : β-nf

A β-nf is a term that contains no β-redexes.

The class of all β-nf's is called β-nf.

We say a term M has β-nf N iff

$M \triangleright_\beta N$ and $N \in \beta$-nf

A reduction can be thought of as a computation and a β-nf as a result.

(Terms need not have β-nf) eg $Sa\, Sa$ $\quad Sa = (\lambda x . xx)$

$\forall M$ s.t. $M : \tau$ (for some τ), $\Rightarrow$ M has a β-nf.

$\boxed{\text{Every term with a type has a } \beta\text{-nf}}$.

Defn : A β-contraction $(\lambda x . M)N \triangleright_{1\beta} M[N/x]$ is said to cancel N iff x does not occur free in M; it is said to duplicate N iff x has at least two free occurrences in M.

A β-reduction is non-duplicating iff none of its contractions duplicates; it is non-cancelling iff none cancells.

7.4.2025.

Definition : Types(M)

If M is closed, define Types(M) to be the set of all τ s.t. $\vdash_\lambda M : \tau$

x in sa will not have any type.

Note: Types(M) is either empty or infinite.

if the types(M) is empty, then there doesn't exists deduction tree for M

Property

Lemma: Let P be closed. Then

(i) $P \rhd_\beta Q \Rightarrow$ Types(P) $\subseteq$ Types(Q)

(ii) if $P \rhd_\beta Q$ by a non-cancelling and non-duplicating reduction, then Types(P) = Types(Q)

Note We need not always have $M =_\beta N \Rightarrow$ Types(M) = Types(N)

Note It could be that $M =_\beta N$ but Types(M) $\cap$ Types(N) = $\emptyset$

Note Types(M) is more than Types(N) means Types(M) is a larger set than Types(N). that Types(M) is a This means that there is a type of M that is less constrained than a type of N. (See (i) of the above Lemma).

The Typable terms :-

TA_λ divides the λ-terms into two complimentary classes: those which can receive types (eg, $\lambda x.\lambda y.\lambda z.\ x(yz)$) — Safe and those which cannot (eg, $\lambda x.\ xx$).

Definition :- A term M is called (TA_λ-) typable iff there exist Γ and τ s.t. $\Gamma \vdash_\lambda M : \tau$

If a deduction proof exists for a lambda term with a given type, then it will be typable.

Property

Lemma: The class of all TA_λ-typable terms is closed under the foll. operations
(i) taking subterms (all subterms of a typable term are typable)
(ii) β-reduction
(iii) non-cancelling and non-duplicating β-expansion
(iv) λ-abstraction (i.e. if M is typable so is $\lambda x . M$).

Theorem: The class of all TA_λ-typable terms is decidable.
i.e., there is an algorithm which decides whether a given term is typable in TA_λ.

Proof: The Principal-type algorithm (PT-algorithm).

Weak Normalization Theorem :- (property). (WN Thm).
Every TA_λ-typable term a β-nf. (excluding ~~η-reduction~~ η-nf)

Strong Normalization (SN) Thm :- (property) If M is a TA_λ-typable term, every β-reduction that starts at M is finite. (excluding η-reduction).

SN $\Rightarrow$ WN but the proof of Turing in WN is simpler, and mostly we used WN.

strong normalization will imply weak normalization.

Theorem (Property)
There is a decision procedure for β-equality of TA_λ-typable terms i.e an algorithm which, given any typable terms P and Q, will decide whether $P =_\beta Q$.

Proof: by WN, reduce P, Q to their nf's. Then check whether they differ.

—x—

Principle type algorithm will give Principle type

Principle type is the simplest type which can't be further simplified.
PT(id) = a->a
All other types are obtained by substitution over a->a.