Preliminaries

myuon

2017 年 10 月 15 日

目次

概要

他のノートを読むにあたって前提とされていることの一部をここにまとめておく.

1 Categorical Semantics

type theory とはその名の通り “型” とは何かということを定めたものである. 以下では項の文法としてラムダ計算を取り扱うことにして, あまり type theory と型付きラムダ計算を区別せずに話をするが, 必ずしも type theory は型付きラムダ計算を指すわけではない.

1.1 Informal Definitions

type theory の定義や定式化はものによって様々なので formal な定義を与えるのは難しい. よってここでは, 出来る限り標準的な ($\neq$ 一般的な)type theory について言えることを述べることにする. 以下の定義が上手く機能しないようなものを考えるときは適宜定義を修正して考えることになる.

定義 1. type theory T とは, type, term, そしてそれらを関係づける rule からなるものである. それぞれ次のような代表的な定め方がある:

- type formation rule: 型コンストラクタが満たすべきルールを $\vdash A : \mathrm{Type}$ のようにして記述したもの. 例えば, function type constructor $\to$ が満たすべきルールは,

$$\vdash A : \mathrm{Type}, \vdash B : \mathrm{Type} \Longrightarrow \vdash A \to B : \mathrm{Type}$$

  のようになる. ただし, 型は型コンストラクタによって自由生成されたもの (特別な制約がないもの) が多いので, そのような場合には $A ::= A_1 \to A_2$ のように BNF で書いてしまっても十分なことが多い.
- term introduction/elimination rule: 項コンストラクタが満たすべきルールを, 各型コンストラクタに対してそれを導入する (結論に用いられる) 規則とそれを除去する (前提に用いられる) 規則からなる (両方あるとは限らない). これらはいずれも項コンストラクタの導入規則であることに注意. 例として, function type constructor$\to$ の intro/elim ルールは, 次のような

abs/app ルールである:

$$\text{abs(funI)}\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

$$\text{app(funE)}\ \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

- conversion rule: 項の equality を external に与えるようなルールで, これが実際にラムダ計算に構造を与える. type system によっては項の equality を internal に定義できる (実際に type system 上で equal であることを証明できる) 場合があるが, ここでいう conversion rule はそのようなものは含まない. あくまで external に与えられたルールを指していう. 例として, function type $\to$ に関するルールとして次の β/η ルールがある:

$$\begin{aligned}(\beta) \quad & \vdash (\lambda x.\, M)N = M[N/x] \\ (\eta) \quad & \vdash (\lambda x.\, Mx) = M\end{aligned}$$

- computation rule: 項の計算・評価を行う際に用いられるルールで, これは rule と言っているが項を別の項へ変換する関数である. 主にラムダ計算をプログラミング言語として見る場合に定義される. 例として, function type $\to$ に関する computation rule として次のような β-reduction ルールがある:

$$(\lambda x.\, M)N \rightsquigarrow M[N/x] \qquad (\beta)$$

定義 2. type theory T に対して, 次のようにして与えた category を T の **syntactic category** とよび, Syn(T) とかく.

- object: T の context $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$
- hom(Γ,Δ): type judgement の組 $\Gamma \vdash_T t_i : \Delta_i$ が存在するもの (を, T の項に関する同値関係で割ったもの).

context を用いるのは external に直積 (組) の扱いができることと同じなので, 組を type theory が internal に扱えるならば, すなわち T が product type をもつならば, 次のように定めても同じことである[*1].

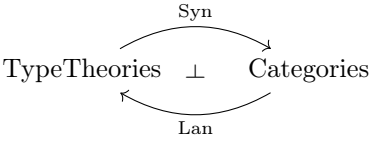
定義 3. product type をもつ type theory T に対して, 次のようにして与えた category を T の **syntactic category** とよび, Syn(T) とかく.

- object: T の type
- hom(A,B): type judgement $x : A \vdash t : B$(を T の項に関する同値関係で割ったもの).
- conversion: また, このとき T の一般の judgement $\{x_1 : A_1, \ldots, x_n : A_n\} \vdash M : B$ は $z : A_1 \times \cdots \times A_n \vdash M[\mathrm{pr}_i(z)/x_i] : B$ によって射とみなせる.

以後は product type をもつ type theory を扱う際は後者を主に使っていく.

定義 4. 適当な type theory のなす category TypeTheories と適当な category のなす category Categories が与えられているとする. **categorical semantics** とは, 次の形の随伴 (圏同値) のことである:

[*1] 同じことであると言っているが, T が product type をもつとき, 前者では external な組と internal な組を別のものとして区別して扱うことができるが後者はそれができないので同じではない. context を product 以外の方法でエンコードしたいのであればこれらは区別する必要があるが, このノートではそのようなことはおそらくしないと思う.



ここで, Syn は syntactic category(を与える functor), Lan は internal language のことである.

定義 5. type theory T の (categorical) **モデル (model)** とは, category $\mathbb{C}$ と morphism $\mathrm{Syn}(T) \to \mathbb{C}$ のことをいう. この morphism を T の $\mathbb{C}$ での **interpretation** とよぶ. また, 単に $\mathbb{C}$ のことをモデルということもある.

T と $\mathrm{Syn}(T)$ を自然に同一視して, T から $\mathbb{C}$ への変換を interpretation と呼ぶことも多い. この場合は T の context を object へ, T の judgement を morphism へ対応させるような変換である.

また, 今考えている type theory T の type と term からなり, T の構造 (射の同値性) と同じ構造をもつモデルを term model とよぶ. categorical model でいえば, 上で構成した syntactic category がそれに当たる.

1.2 Typed Lambda Calculus

定義 6. ラムダ計算 λ の **equational theory** とは, λ で証明可能な等式のなす集合のことである: $\mathcal{E} = \{\vdash_\lambda M = N;\ M \text{ と } N \text{ は } \lambda \text{ で well-formed}\}$.

ラムダ計算での等しさの証明可能性とは, 例えば項の equality が conversion rule によって与えられている場合は conversion rule が生成する等式全体になる. 通常 equational theory は, 同値関係の定義である (refl), (sym), (trans) と,

$$\text{(subst)} \quad \vdash M = N \Longrightarrow \vdash L[M/x] = L[N/x]$$

を含めるようにして定義する. 通常 (subst) は, 項のコンストラクタごとに専用の equation を入れる必要がある.

また, 型付きラムダ計算の場合は, well-formed の条件に well-typed(型付け可能である) を要請する.

1.3 Properties

定義 7. ラムダ計算 λ で証明可能な命題 φ が **sound** とは, λ の任意のモデル $\mathbb{C}$(と morphism $m : \mathrm{Syn}(\lambda) \to \mathbb{C}$) に対し, $m(\bar{\varphi})$ が $\mathbb{C}$ で真となることである. ただし, φ の $\mathrm{Syn}(\lambda)$ での対応する命題を $\bar{\varphi}$ とかいた. また, この逆が成り立つ時, φ は **complete** という.

上の φ として equational theory(項の equality) をとったものをよく使うので, もう一度述べなおしておく.

定義 8. ラムダ計算 λ の equational theory $\mathcal{E}$ が **sound** とは, $\vdash_\lambda M = N \in \mathcal{E}$ のとき, λ の任意のモデル $\mathbb{C}$ と interpretation $[\![-]\!] : \lambda \to \mathbb{C}$ に対して $[\![M]\!] = [\![N]\!]$ が $\mathbb{C}$ で成り立つことである. また, この逆を **complete** という.

上はラムダ計算だけでなくその equational theory を指定して初めて意味をなす命題であることに注意.

また, 個々の interpretation に関して sound/complete という言い方もする.

定義 9. interpretation $[\![-]\!] : \lambda \to \mathbb{C}$ が **sound** とは, $\vdash M = N \in \mathcal{E}$ のとき, $[\![M]\!] = [\![N]\!]$ が $\mathbb{C}$ で成り立つことである. また, この逆を **complete** という.

equational theory が sound ならば, interpretation(model) は sound である.

1.4 Logical Formalism

定義 10. ラムダ計算 λ の **signature** Σ とは, 次からなるもの:

- sorts S: 出現する記号がそれぞれ何を表しているかを判別するための記号の集合. 例えば変数, 項, 型を表す sort をそれぞれ v, p, m とすると, このとき $S = \{v, p, m\}$.
- constructor symbols C: type/term constructor を表す記号の集合. また, 各 constructor に対しその domain と codomain の sort を決定する $\mathrm{dom} : C \to S^*$, $\mathrm{cod} : C \to T$ が与えられていて, symbol c の domain と codmain とを合わせて $c : \mathrm{dom}(c) \to \mathrm{cod}(c)$ のように書いて表示する. arity のない constructor ($\mathrm{dom}(c) = \emptyset$ となる c) は constant と呼ばれる. 例えば, 関数抽象, 関数適用を表す constructor symbol は, $\mathrm{Abs} : v \to m \to m$, $\mathrm{App} : m \to m \to m$ などと表す.
- typing rules: 項と型とを関係づけるためのルールの集合. categorical model を考える場合は, object と morphism を関係づけるためにこれを特別に与える必要がある.
- equational theory $\mathcal{E}$: 項の equality のなす集合.

signature とは, ラムダ計算を構成する記号を集めたもの, と思えばよい.
この意味で, 上で定義したモデルを捉え直すことができる.

定義 11. Σ をラムダ計算 λ の signature で, 型と項を表す sort p, m を含むとする. このとき λ の **(categorical) モデル**とは, category $\mathbb{C}$ と次のような interpretation $[\![-]\!]$ をいう:

- sorts:
  - $[\![p]\!] = \mathrm{Obj}(\mathbb{C})$, $[\![m]\!] = \mathrm{Arr}(\mathbb{C})$
  - p, m 以外の sort に関しても適当な (集合またはそれに類するものへの) 割り当てがある.
- constructors: symbol $c : s_1 \to s_2$ に対し, 関数 $[\![c]\!] : [\![s_1]\!] \to [\![s_2]\!]$ なる割り当てがある.
- typing rules: $A \vdash M : B$ を導く rule に対し, 対応する morphism $[\![M]\!] : [\![A]\!] \to [\![B]\!]$ がある. 通常 A は context なので, category が product を持つことを要請した上で射を対応させる.
- equational theory: equality $\Gamma \vdash M = N : A$ があるとき, $\mathbb{C}$ で $[\![A \vdash M : B]\!] = [\![A \vdash N : B]\!]$ が成り立つ.

一般のラムダ計算の signature の sort は多種多様である. categorical model は type, term を表す sort を基本としてモデルを定義しているので, このような sort がない場合や, 他の sort を持ちそこに特別な relation があるような場合にモデルの定義がどうあるべきかを一般的に述べることは出来ない.

今回の signature とモデルの定義は, pure type systems[2] のものに近い.

参考文献

[1] nLab “type theory", `https://ncatlab.org/nlab/show/type+theory`

[2] H. P. Barendregt. 1993. “Lambda calculi with types". In Handbook of logic in computer science (vol. 2), S. Abramsky, Dov M. Gabbay, and S. E. Maibaum (Eds.). Osborne Handbooks

Of Logic In Computer Science, Vol. 2. Oxford University Press, Inc., New York, NY, USA 117-309.