# Smullyan のシステムの形式化について

2024.12.1

**SnO2WMaN**

## 目次

メタ情報 ..... 1
1. はじめに ..... 1
2. 本文 ..... 2
2.1. 文字列 ..... 2
2.2. Smullyan モデル ..... 2
2.3. 不動点定理 ..... 7
2.4. 主定理 ..... 8
3. おわりに ..... 9
参考文献 ..... 9

## メタ情報

この文書は証明支援系 Advent Calendar 2024 の 3 日目の記事です.

この文書について

- https://sno2wman.github.io/notes-on-smullyanTP/main.pdf で最新の PDF をダウンロードできます.
- https://github.com/SnO2WMaN/notes-on-smullyanTP で誤植訂正やコメントなどを受け付けています.
- CC-BY-4.0 でライセンスされています.

この文書に載せられたコードの全文は https://github.com/SnO2WMaN/smullyanTP で公開しています.

## 1. はじめに

論理学では自己言及的，パラドキシカルな事実が成り立つことが知られている．この文書はそれらの事実を解説する目的ではないので，詳細に関しては書かずに大雑把に述べることにする．太字の強調部分だけ読めば，とりあえず本文の理解には問題ないだろう．

以下 $T$ は適当な算術の理論とする．

> **定理 1.1** (Gödel-Rosser の第一不完全性定理): 無矛盾な理論 $T$ について，$T$ 上で**証明も反証も出来ない命題が存在する．** すなわち，次のような文 $G_T$ が存在する：$T \nvdash G_T$ かつ $T \nvdash \neg G_T$.

普通は第一不完全性定理ではそのような命題が実際に真であるのかについては言及しない．ただし，真偽について言及するとキャッチーなので，次の形で述べられることもある[1].

> **定理 1.2** (Boolos?): 無矛盾な理論 $T$ について，**正しいが証明できない命題が存在する．** すなわち，次のような文 $G_T$ が存在する：$\mathbb{N} \vDash G_T$ かつ $T \nvdash G_T$.

不完全性定理の証明，あるいはより一般に自己言及的な事実を示すためには，次の不動点補題（対角化補題などいろいろな呼び方がある）が鍵となる．

[1] このような解説の是非については[1]に少し説明がある.

**定理 1.3** (不動点補題): $\varphi(x)$ は $x$ のみを自由変数とする論理式とする．次の文 $F_\varphi$ が存在する．

$$T \vdash F_\varphi \leftrightarrow \varphi\big(\ulcorner F_\varphi \urcorner\big)$$

この意味で，$F_\varphi$ を $\varphi$ の不動点という．

また，次の命題も重要である．

**定理 1.4** (Tarski の定義不可能性定理): **真であるという性質は記述できない**．すなわち，次のような論理式 $\mathrm{True}(x)$ は存在しない：任意の論理式 $\varphi$ について，

$$\mathbb{N} \vDash \varphi \leftrightarrow \mathrm{True}(\ulcorner \varphi \urcorner)$$

論理パズルの一般書などでも有名な論理学者 Smullyan は[2]において， これらの定理を一般の読者に向けて説明するために，文字列の操作だけによる非常に簡単な形式体系を導入した．そして実際に，この形式体系上で，これらのパラドキシカルな定理に対応するような現象が起こることを示した．

この文書では[2]を精査，整理し直した[3]を参考に，Smullyan の形式体系を実際に定理証明支援系である Lean で形式化して定理を証明した． コードは 300 行程度で済んでいるので，興味のある読者は実際にコードを読んだり触ってみることを勧める．

# 2. 本文

定義および証明には，自然言語によるものと，実際に形式化したコードを併記しておいた．自然言語の証明はコードに書かれた順番に沿ってできるだけ気をつけて書き起こしたつもりなので，自然言語と形式化したコードを見比べてみて，大体どれぐらいの対応関係があるのかを確認してみると良いだろう．

## 2.1. 文字列

**定義 2.1.1** (文字列):
- 非空集合 $\Sigma$ をアルファベットとする．
- $\Sigma^*$ は $\Sigma$ 上の文字列の集合とする．
- 空の文字列を $\epsilon$ で表す．
- 文字列 $X, Y \in \Sigma^*$ に対し，$XY$ は $X$ と $Y$ の連結を表す．
- $X$ と $Y$ が記号列として等しいとき，$X \equiv Y$ と書く．

今回の形式化では，文字列はリストとして，すなわち `List α` で表現する．ここでは `α` はアルファベットを表す型である．

## 2.2. Smullyan モデル

**定義 2.2.1** (Smullyan モデル, `SmullyanModel`): 以下を満たす 3 つ組 $\langle \Sigma, \mathrm{Pred}, \Phi \rangle$ を Smullyan モデルという．
1. $\Sigma$ はアルファベット．
2. Pred は $\Sigma^*$ の部分集合であり次を満たす．
   - $H \in \mathrm{Pred}, X \in \Sigma^* \setminus \{\epsilon\}$ に対し，$HX \notin \mathrm{Pred}$ を満たす．
3. $\Phi$ は 写像 $\mathrm{Pred} \to \mathcal{P}(\Sigma^*)$.

Smullyan モデル $M = \langle \Sigma, \mathrm{Pred}, \Phi \rangle$ の構成要素 $\Sigma, \mathrm{Pred}, \Phi$ をそれぞれ $\Sigma_M, \mathrm{Pred}_M, \Phi_M$ と書く． 更に $\Sigma_M^*$ の元を $M$ の語，$\mathrm{Pred}_M$ の元を $M$ の述語と呼ぶ．

さて，これを Lean では `structure` として定義する．

```
structure SmullyanModel where
  α : Type*
  isPredicate : List α → Prop
  isPredicate_spec :
    ∀ H : { H // isPredicate H }, ∀ x ≠ [], ¬isPredicate (H.val ++ x)
  valuation : { H // isPredicate H } → Set (List α)
```

Lean では `α` の素朴な集合の型 `Set α` は，`α` から `Prop` への関数の型 `α → Prop` の略記として定義されることに注意する．Lean では集合の外延表記のように `{ H // isPredicate x }` と書くことで，`List α` の要素 `H` で `isPredicate` を満たすような部分型(Subtype)，すなわち Pred の型を表すことが出来る．

これらを毎回書くのは面倒なので，語の型 `Word` と述語の型 `Predicate` を略記として導入する．$\Sigma_M^*, \mathrm{Pred}_M$ に対応している．

```
abbrev Word (M : SmullyanModel) := List M.α

abbrev Predicate (M : SmullyanModel) := { H // M.isPredicate H }
```

また，$M$ の述語 $H$ に対して $\Phi_M(H)$ を `H.valuated` で表すことにする．

```
def Predicate.valuated {M : SmullyanModel} (H : Predicate M) : Set (Word M) := M.valuation H
```

**定義 2.2.2** (定義する, `Predicate.names`): $H$ を $M$ の述語，$V$ を $M$ の語の集合 とする．$\Phi_M(H) = V$ であるとき，$H$ は $V$ を定義するという．

これは素直に形式化出来る．

```
def Predicate.names {M : SmullyanModel} (H : M.Predicate) (V : Set M.Word) : Prop := H.valuated = V
```

次に，$M$ の文を定義する．

**定義 2.2.3** (文, `Sentence`): $M$ の述語 $H$ と語 $X$ の組 $\langle H, X \rangle$ を $M$ の文という． $M$ 文の集合を $\mathrm{Sent}_M$ と書く． 適宜，文 $M = \langle H, X \rangle$ は $HX$ として語のように扱ってよいことにする．

```
structure Sentence (M : SmullyanModel) where
  pred : M.Predicate
  word : M.Word

-- `S : M.Sentence` を `M.Word` として扱いたい場合は `pred` と `word` を単純に連結したものとする．
def Sentence.toWord : M.Sentence → M.Word := fun ⟨H, X⟩ ⇒ ↑H ++ X

-- `↑S` と書けば `M.Word` として扱えるようになる．
instance : Coe (M.Sentence) (M.Word) := ⟨Sentence.toWord⟩
```

**注意 2.2.4**: ここでは元の論文とは違う方針で文を定義している（本質的には同じ）．元の定義は「$M$ の語 $S$ が $M$ の文であるとは，$S \equiv HX$ となる $H \in \mathrm{Pred}_M$ と $X \in \Sigma_M^*$ が存在することをいう」となっている．

これを素直に定義すると以下のようになる．ここで，`↑H` は `Predicate` 型の要素を `Word` 型の要素にキャストしている[2]と思ってもらえれば良い．

```
def isSentence {M : SmullyanModel} (S : M.Word) : Prop := ∃ (H : M.Predicate) (X : M.Word), S = ↑H ++ X
```

しかしこの定義は使いづらい．なぜならば `S` に対して**具体的に** `H` と `X` が何なのかの情報を持っておらず，ただ存在することだけを主張しているからである．`S.isSentence` という命題からそのような性質を満たす `H` を取り出すには `Exists.choice` を使う必要がある．この操作は Lean においては超越的な/非構成的な操作であり，また単純にこれを毎回書くのは面倒というデメリットがある．したがって，より簡単に扱うために，文を構成する述語と語の組を直接扱うことにした．

重要な補題である文の一意性を示す．

**補題 2.2.5** (文の一意性, `exists_unique_pred_toWord`): $S \in \mathrm{Sent}_M$ に対して $S \equiv HX$ となる述語 $H$ は一意である．

**証明**: 一意でないと仮定して矛盾を導く．$S \equiv HX$ と $S \equiv H'X'$ であり，$H \neq H'$ となる述語 $H, H'$ が存在すると仮定する．

今，一般性を失わずに $H$ は $H'$ を始める真の部分列，すなわち $H' \equiv HY$ となる $Y \neq \varepsilon$ が存在するとしてよい．すると $H$ は述語であるので $HY \notin \mathrm{Pred}_M$ であり，よって $H' \notin \mathrm{Pred}_M$ となる．これはおかしい． ❑

```
lemma exists_unique_pred_toWord (S : M.Sentence) : ∃! H : M.Predicate, ∃ X : M.Word, H ++ X = S.toWord := by
  dsimp only [Sentence.toWord];
  apply exists_unique_of_exists_of_unique;
  . use S.pred, S.word;
  . rintro H₁ H₂ ⟨W₁, h₁⟩ ⟨W₂, h₂⟩;
    rw [←h₂] at h₁; clear h₂;
    wlog h : (H₁.val <+: H₂.val);
    . rcases List.IsProperPrefix.trichnomy h₁ with h | h | h;
      . exact Subtype.eq h;
      . contradiction;
      . exact this S H₂ H₁ W₂ W₁ h₁.symm h ▷.symm;
    obtain ⟨t, ht, h⟩ := h;
    have := M.isPredicate_spec H₁ t ht;
    simp [h, isPredicate_predicate] at this;
```

**例 2.2.6**: 例えば $\langle \mathtt{ab}, \mathtt{cde} \rangle$ は $M$ の文であるとすると，この文は語 `abcde` として表される．補題 2.2.5 は，`abcde` となるような文の候補として $\langle \mathtt{abc}, \mathtt{bc} \rangle$ や $\langle \mathtt{a}, \mathtt{bcde} \rangle$ があるが，これらは $M$ の文にはなり得ないということを保証する．

[2]Coercion と呼ぶ．今，`X` は型 `M.Word` の要素だが，`H` は型 `M.Predicate` の要素である．`++`すなわち関数 `List.append` の型は `List α → List α → List α` であるため，`H ++ X` とすると型が合わない．そのため Coercion が必要となる．

**注意 2.2.7**: 更に形式化上では，後の証明で使う次の重要な補題が得られるが，これは形式化上の問題であり，数学的な問題ではないので，詳細は割愛する．

```
/--
  `Sentence.toWord` は単射である．
  すなわち：
    - `S₁.toWord = S₂.toWord → S₁ = S₂`
    - `S₁` と `S₂` が文字列として等しいならば，`S₁` と `S₂` の構成要素 `pred, word` は互いに等しい．
-/
lemma toWord_injective
  : Function.Injective (Sentence.toWord (M := M)) := by ...
```

次に，モデル上の文の真偽を定める．

**定義 2.2.8** (文の真偽): 文の集合 $\mathrm{True}_M$ を次のように定義する．

$$\mathrm{True}_M := \{\langle H, X\rangle \in \mathrm{Sent}_M \mid X \in \Phi_M(H)\}$$

文 $S$ が $S \in \mathrm{True}_M$ のとき，$S$ が（$M$ で）真であるといい $\vDash S$ と書く．逆に $S \notin \mathrm{True}_M$ のとき，$S$ が（$M$ で）真でないという．

これをコードに書き下すと次のようになる．

```
def true_sentences (M : SmullyanModel) : Set M.Sentence := fun ⟨H, X⟩ ⇒ X ∈ H.valuated

def Sentence.isTrue (S : M.Sentence) := S ∈ M.true_sentences

prefix:50 "⊨ " ⇒ Sentence.isTrue
```

次に，Smullyan モデル上で特別な働きを行う `n` および `r` という記号を導入する．

まず，`n` を導入する．`n` は否定(negation)を意図した記号である．

**定義 2.2.9** (n-Smullyan モデル, `SmullyanModel.IsN`): $M$ が n-Smullyan モデルであるとは，次の条件を満たすアルファベット $\mathtt{n} \in \Sigma_M$ が存在することをいう．
1. $H$ が述語なら $\mathtt{n}H$ も述語．
2. $\Phi_M(\mathtt{n}H) = \Sigma_M^* \setminus \Phi_M(H)$.

文 $S = \langle H, X\rangle$ に対して，その否定の文 $\mathtt{n}S = \langle \mathtt{n}H, X\rangle$ とする．

**定義 2.2.10** (`Sentence.isNegatedTrue`): $\vDash \mathtt{n}S$ を $\nvDash S$ と書き，$S$ の否定が真であることを表す．

これらをコードに書き下すと次のようになる．`nH` のようにそのまま記号として使うとよくわからなくなるので，形式化の上では `~H` で表すことにする．

```
class SmullyanModel.IsN (M : SmullyanModel) where
  n : M.α
  n_spec₁ : ∀ H : M.Predicate, (M.isPredicate (n :: H))
  n_spec₂ : ∀ H : M.Predicate, M.valuation ⟨n :: H, n_spec₁ H⟩ = (H.valuated)ᶜ

variable [M.IsN] -- 以下 `M` は `IsN` であると仮定する

-- `n ++ H`（定義）と，実際にそれは `M.Predicate` である（証拠）の組で述語の否定を定義．
```

```
def Predicate.neg (H : M.Predicate) : M.Predicate := ⟨IsN.n :: H.val, IsN.n_spec₁ H⟩

-- `~H` で述語の否定を表すこととする.
prefix:90 "~" ⇒ Predicate.neg

-- 元の `pred` の部分を否定して文の否定とする.
def Sentence.neg (S : M.Sentence) : M.Sentence := ⟨~S.pred, S.word⟩

-- `~S` で文の否定を表すこととする.
prefix:90 "~" ⇒ Sentence.neg

def Sentence.isNegatedTrue (S : M.Sentence) := ⊨ ~S
prefix:50 "⊭ " ⇒ Sentence.isNegatedTrue
```

直感的には，$S$ の否定が真であることと，$S$ が真でないことは一致してほしい．次の補題は実際そうなるということを示している．

**補題 2.2.11** (Sentence.iff_isNegTrue_not_isTrue): $\nvDash S \iff S \notin \mathrm{True}_M$.

**証明**: $S = \langle H, X \rangle$ とする．定義に沿って変形していく．

$$\begin{aligned}
\nvDash \langle H, X \rangle &\iff \vDash \mathtt{n}\langle H, X \rangle \\
&\iff \vDash \langle \mathtt{n}H, X \rangle \\
&\iff X \in \Phi_M(\mathtt{n}H) \\
&\iff X \in \Sigma_M^* \setminus \Phi_M(H) \\
&\iff X \notin \Phi_M(H) \\
&\iff \langle H, X \rangle \notin \mathrm{True}_M
\end{aligned}$$

以上より $\nvDash S \iff S \notin \mathrm{True}_M$ である． ❑

証明を見ればわかるように，これは定義上明らかなので，形式化上では単純に simp と tauto だけで示すことが出来る．

```
lemma Sentence.iff_isNegTrue_not_isTrue : ⊭ S ↔ ¬⊨ S := by
  simp only [Sentence.isNegatedTrue, Sentence.neg, Sentence.iff_isTrue, Predicate.eq_neg_valuated];
  tauto;
```

更に，元の文の否定の否定が真なら，元の文も真である．すなわち二重否定は元に戻ることも確認出来る．

**補題 2.2.12** (Sentence.iff_isNegTrue_neg_isTrue): $\nvDash \mathtt{n}S \iff \vDash S$.

**証明**: 定義に沿って変形すればよい． ❑

同様に，これも形式化上では simp だけで示すことが出来る．

```
lemma Sentence.iff_isNegTrue_neg_isTrue : ⊭ ~S ↔ ⊨ S := by
  simp [Sentence.iff_isTrue, Sentence.isNegatedTrue];
```

次に $\mathtt{r}$ を導入する．$\mathtt{r}$ は繰り返し(repeated)を意図した記号であり，この記号によって不動点の構成が可能になる．

**定義 2.2.13** ($\mathtt{r}$-Smullyan モデル, SmullyanModel.IsR): $M$ が $\mathtt{r}$-Smullyan モデルであるとは，次の条件を満たすアルファベット $\mathtt{r} \in \Sigma_M$ が存在することをいう．
1. $H$ が述語なら $\mathtt{r}H$ も述語.

2. $\Phi_M(\mathbf{r}H) = \{K \in \mathrm{Pred}_M \mid KK \in \Phi_M(H)\}$.

同様に，rH のようにそのまま記号として使うとよくわからなくなるので，形式化の上では□H で表すことにする．

```
class IsR (M : SmullyanModel) where
  r : M.α
  r_spec₁ : ∀ H : M.Predicate, (M.isPredicate (r :: H))
  r_spec₂ : ∀ H : M.Predicate, M.valuation ⟨r :: H, r_spec₁ H⟩ = { K : M.Predicate | K.val ++ K.val ∈ H.valuated }

variable [M.IsR]

def Predicate.rep (H : M.Predicate) : M.Predicate := ⟨IsR.r :: H.val, IsR.r_spec₁ H⟩
prefix:90 "□" ⇒ Predicate.rep

def Sentence.rep (S : M.Sentence) : M.Sentence := ⟨□S.pred, S.word⟩
prefix:90 "□" ⇒ Sentence.rep
```

## 2.3. 不動点定理

冒頭で，自己言及的な事実を示すためには定理 1.3 が重要であると述べた．このシステム上では定理 2.3.2 がその定理に対応する．

**定義 2.3.1** (不動点, fixpoint): $M$ は $\mathbf{r}$-モデルとする． $H$ を $M$ の述語として，文 $F = \langle \mathbf{r}H, \mathbf{r}H\rangle$ を $H$ の不動点という．

```
def Predicate.fixpoint [M.IsR] (H : M.Predicate) : M.Sentence := ⟨□H, □H⟩
```

**定理 2.3.2** (不動点定理, fixpoint_spec): 任意の述語 $H$ に対し， $H$ の不動点 $F$ は次の性質が成り立つ．

$$\vDash F \Longleftrightarrow \vDash HF$$

**証明**: 定義に沿って次の同値が成り立つ． $\vDash \mathbf{r}H\mathbf{r}H \iff \mathbf{r}H \in \Phi_M(\mathbf{r}H) \iff \mathbf{r}H\mathbf{r}H \in \Phi_M(H) \Longleftrightarrow \vDash H\mathbf{r}H\mathbf{r}H$. ❑

やはりこれも形式化上では simp だけで示すことが出来る．

```
lemma fixpoint_spec [M.IsR] : ⊨ H.fixpoint ↔ ⊨ (⟨H, H.fixpoint⟩) := by
  simp [Predicate.fixpoint, Sentence.iff_isTrue];
```

**注意 2.3.3**: 元の論文では不動点定理は「$\vDash F \Longleftrightarrow \vDash HF$ となる $F$ が任意の $H$ に対して存在する」という形で言及されている． しかし 注意 2.2.4 でも述べたように，存在するという形の言明は実際に構成できるならば構成したほうが扱いやすくてよいという方針で形式化を進めている．

最後に，主定理の証明において頻繁に用いる，述語の否定の不動点に関しての補題を示す．

**補題 2.3.4** (iff_isTrue_neg_fixpoint): $M$ は $\mathbf{nr}$-モデル とする． 任意の述語 $H$ に対し，$\mathbf{n}H$ の不動点を $F$ とする． このとき $\vDash F \iff F \notin \Phi_M(H)$.

**証明**: まず，不動点の定義より次の同値が成り立つ．

$$\vDash F \Longleftrightarrow \vDash \mathbf{n}HF \Longleftrightarrow \nvDash HF$$

補題 2.2.11 から $\nvDash HF \iff HF \notin \mathrm{True}_M \iff F \notin \Phi_M(H)$ が成り立つのでよい． ❑

```
lemma iff_isTrue_neg_fixpoint [M.IsNR] : ⊨ (~H).fixpoint ↔ ↑(~H).fixpoint ∉ H.valuated := by
  simp [Predicate.fixpoint, Sentence.iff_isTrue];
```

## 2.4. 主定理

それではこのシステム上で成り立つ，2 つの主定理を示す．

まずは定理 2.4.1 から． これは定理 1.4 に対応する定理である．

**定理 2.4.1** (Theorem T, tarski): $M$ は $\mathbf{nr}$-モデル であるとする．$\mathrm{True}_M$ を定義する述語は存在しない．

**証明**: 仮に $H$ としてそのような述語が存在するとして矛盾を導く．

$F$ を $\mathbf{n}H$ の不動点とする． 仮定より $\Phi_M(H) = \mathrm{True}_M$ だから，$F \in \Phi_M(H) \Longleftrightarrow \vDash F$ が成立する． 補題 2.3.4 と $F$ の定義より $\vDash F \iff F \notin \Phi_M(H)$ が成り立つ． これらを合わせると $F \in \Phi_M(H) \iff F \notin \Phi_M(H)$ となっておかしい． ❑

```
theorem tarski [M.IsNR] : ¬∃ H : M.Predicate, H.names M.true_sentences := by
  by_contra hC;
  obtain ⟨H, hH⟩ := hC;
  let F := (~H).fixpoint;
  have : ↑F ∈ H.valuated ↔ ⊨ F := iff_of_names_true_sentenes hH F;
  rw [iff_isTrue_neg_fixpoint] at this;
  tauto;
```

次に定理 2.4.2 を示す． これは定理 1.1 と定理 1.2 に対応する定理である．

**定理 2.4.2** (Theorem G, goedel1): $M$ は $\mathbf{nr}$-モデルとし，述語 $H$ は $\Phi_M(H) \subseteq \mathrm{True}_M$ を満たすものとする． このとき，$F \notin \Phi_M(H)$ かつ $\mathbf{n}F \notin \Phi_M(H)$ となる文 $F$ が存在する．

**証明**: $\mathbf{n}H$ の不動点を $F$ とするとこれが所望の文となる．

今，$\vDash F$ としてよい．なぜなら仮に $\vDash F$ でないと仮定すると，補題 2.3.4 より $F \in \Phi_M(H)$ が言えて，$\Phi_M(H) \subseteq \mathrm{True}_M$ であるため $\vDash F$ となっておかしい．

まず補題 2.3.4 より直ちに $F \notin \Phi_M(H)$ が従う．

次に $\mathbf{n}F \notin \Phi_M(H)$ だが，これは $\Phi_M(H) \subseteq \mathrm{True}_M$ だから $\mathbf{n}F \notin \mathrm{True}_M$ を示せば十分．そして補題 2.2.11 と補題 2.2.12 から次の同値関係 $\mathbf{n}F \notin \mathrm{True}_M \Longleftrightarrow \nvDash \mathbf{n}F \Longleftrightarrow \vDash F$ が成り立つのでよい． ❑

```
theorem goedel1 [M.IsNR] (hH : H.valuated ⊆ M.true_sentences) : ∃ S : M.Sentence, ↑S ∉ H.valuated ∧
↑(~S) ∉ H.valuated := by
  let F := (~H).fixpoint;
  use F;
  have h : ⊨ F := by
    by_contra hC;
    have : ↑F ∈ H.valuated := iff_isTrue_not_neg_fixpoint.mp hC;
```

```
    have : ⊨ F := iff_mem_toWord_true_sentence_mem_true_sentence.mp $ hH this;
    contradiction;
  constructor;
  . exact iff_isTrue_neg_fixpoint.mp h;
  . apply Set.not_mem_subset (s := H.valuated) (t := M.true_sentences) hH;
    apply iff_mem_toWord_true_sentence_mem_true_sentence.not.mpr;
    apply Sentence.iff_isNegTrue_not_isTrue.mp;
    apply Sentence.iff_isNegTrue_neg_isTrue.mpr;
    assumption;
```

**注意 2.4.3**: ここではわかりやすく「存在する」と書いたが, もちろん今までの方針と同じ様に補題 2.3.4 の不動点がその性質を満たす具体例であるとして形式化してもよい.

## 3. おわりに

いくつかの証明は定義より明らかであるので, 単純に簡約化(simp)すれば自明に済ませることが出来る. そのため, 今回の形式化では 300 行程度のコードで済んでいる.

[3]ではより精緻な分析や, 実際の算術上で Smullyan モデルを上手く作って冒頭の定理が成り立つことなどを示している. これらの形式化について, とくに後者については別途 Formalized Formal Logic 側の定義や結果などを用いれば可能かもしれないが, 上手くいくかはわからない.

## 参考文献


1. 菊池誠: 不完全性定理. 共立出版 (2014)
2. Smullyan, R. M.: Truth and Provability. The Mathematical Intelligencer. 35, 21–24 (2013). https://doi.org/10.1007/s00283-012-9328-6
3. Kurahashi, T., Tominaga, K.: Smullyan's truth and provability, http://arxiv.org/abs/2410.17895, 参照 2024/10/26. https://doi.org/10.48550/arXiv.2410.17895