

Monadic Chat :

テキスト補完 API で文脈を保持するためのフレームワーク

長谷部 陽一郎
同志社大学

yhasebe@mail.doshisha.ac.jp

概要

関数型プログラミングにおけるモナドの概念を応用することで既存のテキスト補完 API に文脈情報を保持する仕組みを加え、対話形式で利用するための方法を提案する。対話の各ターンがアキュムレーターとリデューサーを備えたモナド的再帰構造を持つようプログラムを実装することで、異なる目的のアプリケーションをテンプレートを差し替えるだけで実現できる。これを示す事例として **Monadic Chat** を開発した。OpenAI による GPT-3 ベースのテキスト補完 API を利用するプログラムであり、インタラクティブなアプリケーションをユーザー自身で定義することができる。その設計は、対話/談話に関する認知言語学上のモデルとも親和性が高い。

1 はじめに

2020 年に OpenAI が GPT-3 を公開して以降、ユーザーからの入力に対して自然言語テキストでレスポンスを返す、いわゆる AI システムが多く注目を集めている。また 2022 年後半に発表された ChatGPT はチャット形式で文脈を保持した対話が可能であり [1]、今後こうした汎用 AI システムが様々な領域で活用されることが期待されている。

OpenAI は GPT-3 をベースにした言語モデルの API サービスを提供しているが、対話の文脈を保持する仕組みは含まれていない。したがって、ChatGPT のような仕組みや、それに類するアプリケーションを構築するためには OpenAI が提供するテキスト補完 (text completion) API を用いて独自のロジックを組み立てる必要がある。そこで関数型プログラミングにおけるモナドの概念を応用したコマンドライ

ン・ツール **Monadic Chat** を開発した。

本ツールでは、関数型プログラミングにおけるモナドの概念を用いて、グローバルな変数や外部のストレージに文脈情報を格納することなく、逐次的な対話によって文脈を作り出して次々と後の処理に引き渡すことができる。また、対話の目的、性質、前提などを定型のテンプレートに記述することで、様々な対話アプリケーションをわずかの工数でユーザーが自ら作ることができる。¹⁾

以下では 2 節で **Monadic Chat** 基本的な内容を示す。3 節では自然言語による談話をモナド的再帰構造としてモデル化することの意義と利点について考察する。4 節では本ツールによる再帰的な対話処理の流れとテンプレートの仕様について述べる。最後に 5 節で全体のまとめを行う。

2 Monadic Chat について

2.1 言語モデルの指定と API アクセス

Monadic Chat は OpenAI による GPT-3 以降のモデルにテキスト補完 API を通じてアクセスする仕様になっている。²⁾ 具体的には現時点で利用できる最大の汎用モデル **davinci-text-003** を使用することになる。OpenAI は特定の目的に特化した様々なモデルを提供しているが (プログラムコード作成用モデル、音声入力テキスト修正用モデルなど)、それらを用いることはできない。本ツールではテンプレートに必要な事項を記述することで機能を実現するが、テンプレートには自然言語による指示と JSON 形式のオブジェクトを含める必要があり、これらを正しく解釈し適切なデータを返すことができるモデルが必要となる。なお、OpenAI の各種 API の使用は有料であ

¹⁾ 本稿の付録に実際の使用例を示す。ChatGPT と同様、あるターンでの対話の内容が文脈として次のターンに引き継がれていることがわかる。

²⁾ 現時点では OpenAI のテキスト補完 API を用いる以外の選択肢はないが、同様のレベルで汎用的かつ高性能なモデルが利用可能であれば、それらに対応することは十分可能である。

り、アカウントを作成してアクセス・トークンを取得する必要がある。

2.2 プログラムの入手と使用

Monadic Chat は Ruby で作成されている。³⁾ Ruby のバージョン 2.6.0 以上が利用可能な環境であれば、`gem install monadic_chat` というコマンドでインストールすることができる。

インストール後は `monadic` コマンドでプログラムを起動する。初回起動時に OpenAI のアクセストークンを入力することが求められる。問題なく API にアクセスできることが確認されると、それ以降はアクセストークンを明示的に入力する必要はない。

本ツールのソースコードやその他の情報は https://github.com/yohasebe/monadic_chat で公開している。

2.3 利用できるテンプレート

Monadic Chat ではテンプレートを記述することで、様々な対話アプリケーションを作成できるが、サンプルとして 1) Chat (一般的対話), 2) Translate (文章の翻訳), 3) Novel (物語の段階的生成), 4) Code (プログラムコードの作成) という 4 つのテンプレートを提供している。いずれの場合でも、ユーザーからのテキスト入力に対して、GPT-3 が何らかの返答を示す。その際、これまでのやりとりで扱われた内容が文脈として機能する。

なお、コマンドライン上のインタラクティブな対話形式で実行することが基本であるが、あらかじめ用意したプロンプトのリストを読み込ませて連続処理することも可能である。⁴⁾

3 モナドとしての談話構造

3.1 認知言語学の談話モデル

テキスト補完 API を利用して文脈を伴った対話を実現するに際しては 2 つの方針が考えられる。1 つは、特定の目的—例えば自然言語による質疑応答—to 最適化しながらシステムを実装する方針である。これには、構造よりも機能に主眼をおいて個々のアプリケーションの設計をシンプルに保ち、使用リソ

ースを最低限にできる利点がある。もう 1 つは、自然言語による談話に参加する話者と聞き手の中でのような認知プロセスが生じているかについて、理論言語学的モデルを参考にするという方針である。Monadic Chat の開発にあたっては後者を採用した。

認知言語学の主要な研究者の一人である Ronald Langacker は、リアルタイムで進行する談話展開を「現在時談話空間 (current discourse space, CDS) の更新」という観点から図 1 のような構造を持つものとして記述・分析している[3, 4]。

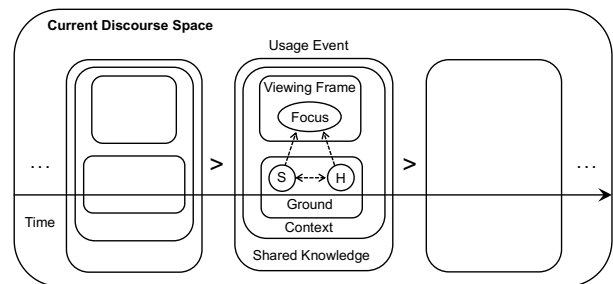


図 1：現在時談話空間 (CDS)

Langacker は[5]で示された考え方を受けて、言語の構造を「CDS を更新するための指示書」として捉えるアプローチを可能性として提案する (“we might think of linguistic structures (of whatever size) as instructions to modify the current discourse space in particular ways” p.151) [3]. 1 つ 1 つの発話事象 (usage event) では、そのような指示書の内容に応じて CDS が更新される。また、発話事象の中に含まれる話し手、聞き手、話題の事物といった要素はもちろん、文脈や共有知識といった要素を含む対象世界の全体が後続する発話事象に引き継がれていく。

あくまで数多ある理論言語学上のモデルの 1 つであるが、認知言語学の理論的研究の中で多くの実例と共に示されてきた CDS に基づく談話モデルには、一定の認知的・言語的根拠がみとめられる[4]。また、CDS に基づくモデルは関数型プログラミングにおけるモナドの概念ときわめて親和性が高い[6]。これらのことは、談話の参与者、文脈、前提知識といったものを一種の「環境」と捉え、それを逐次的に引き回す構造にある種の普遍性があることを示唆する。

³⁾ Ruby を用いることには、後述するリデューサーをユーザー自身が Proc オブジェクトとして簡単に実装できるという利点がある。

⁴⁾ 付録で示す Novel テンプレートを用いた例は、あらかじめ用意した箇条書きの小説プロットを連続処理させた結果である。

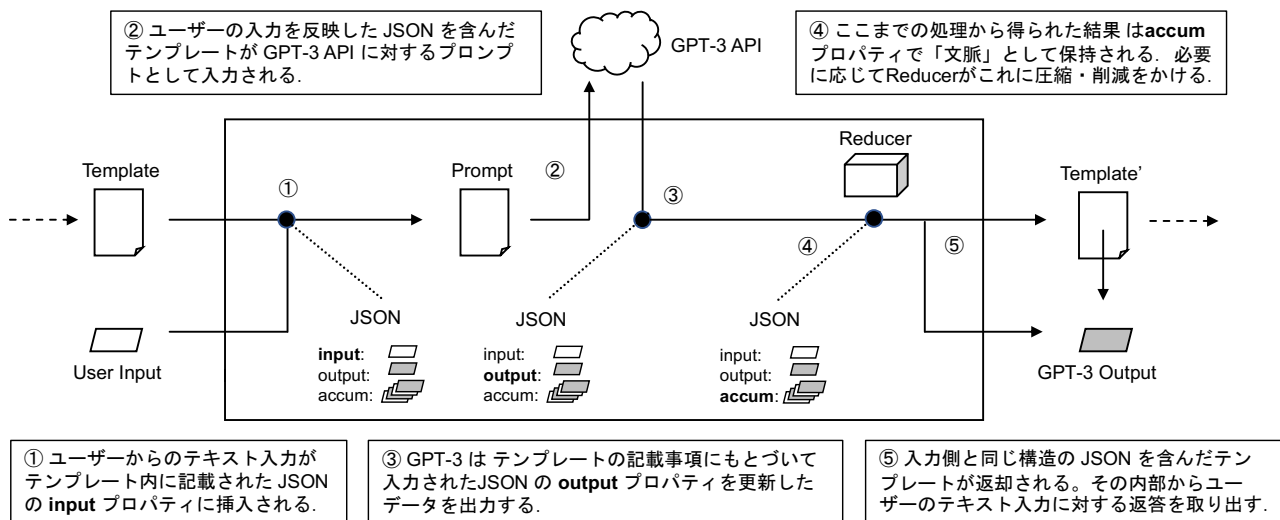


図 2 : Monadic Chat による処理の流れ

そこで本ツールでは対話・談話をモナド的な構造として設計することを選択した。⁵⁾

3.2 モナドについて

関数型プログラミングにおけるモナドはしばしば「環境に包まれた値」というメタファーで説明される[2]。ここで「環境」をボックスで表記するなら、モナドとしての性質を持つためには次の操作が可能であることが求められる[8]。

- (1) $\text{unit} :: a \rightarrow \boxed{a}$
- (2) $\text{map} :: (a \rightarrow b) \rightarrow (\boxed{a} \rightarrow \boxed{b})$
- (3) $\text{join} :: \boxed{\boxed{a}} \rightarrow \boxed{a}$

(1)の unit は対象となる値 a を環境に包む働きをする。(2)は値 a から値 b を得るという手続きを、環境を伴った値を扱う手続きに変換する。(3)の join は二重になった環境を平坦化する。こうした一連の条件を満たしたモナドは、環境として与えられた構造を保持・更新しながら連続的・逐次的に処理を実行できるという特徴を持つ[9, 10, 11]。

(1)から(3)における a や b といった値を「ユーザーからの問いかけ」や「問いかけに対する返答」と考えるならば、「環境」はプロンプトに書き込まれた文脈、前提、対話上の設定といった諸々の要素を総合したものと言うことができる。Monadic Chat では、

このような発想にもとづき、文脈を伴った対話の形式で GPT-3 の言語モデルを利用できるようにすることを目指した。⁶⁾

4 再帰構造としての対話

4.1 処理の流れ

Monadic Chat による処理の流れを図 2 に示す。本ツールのモナド的構造は、ユーザーによる問いかけ、GPT-3 による返答、そして「環境」としてのテンプレートによって構成される。①は値を環境に包む処理 (unit) であり、②と③は問いかけに対する返答を得る手続きに環境を伴わせる処理 (map) である。こうした一連の手続きによって最終的に目指すのは、同じ一連の手続きに対する入力となることのできるテンプレートを得るという再帰的プロセスである。また、その際には一連の処理の結果を「文脈」として受け渡せるようにしなくてはならない。ただし、結果として得るテンプレートを履歴としてそれ自身に埋め込むと、次々に履歴の階層が増すことになるため、何らかの平坦化 (join) をかける必要がある。

④では、実現したいアプリケーションの性質に応じて、ユーザーの入力値と GPT-3 からの返答、もしくはその両方をアキュムレーターに書き足していく。このアキュムレーターこそが「文脈」を保持する要素となる。当然ながら対話のターンが積み重なるに

⁵⁾ Monadic Chat は関数型プログラミングにおけるモナドの考え方をういて設計されているが、現時点ではモナド則 (monad laws) を厳密な意味で満たしているわけではない[2]。

⁶⁾ 処理に必要なリソースの観点からするとモナド的な構造は必ずしも効率的なものではない。環境 (あるいは世界) の引き回しにおいてこの問題が生じ得ることは[7]でも指摘されている。

You are a friendly but professional consultant who answers various questions, write computer program code, make decent suggestions, give helpful advice in response to a prompt from the user. Create a response to the following prompt from the user and set your response to "response" property of the JSON object shown below. If the prompt is not clear enough, ask the user to rephrase it. The preceding conversation is stored in the value of the "conversation" property. ①

PROMPT: {{PROMPT}} ②

```
```json
{
 "mode": "chat",
 "prompt": "Can I ask something?",
 "response": "Sure!",
 "language": "English",
 "conversation_history": [{"User: Can I ask something?", "GPT: Sure."}]
}
``` ③
```

- ① 全体的な目的, 設定, 形式に関する記述
- ② ユーザーからの問いかけを差し込むためのプレースホルダ
- ③ ユーザからの問いかけ (input = prompt), GPT-3 からの返答 (output = response), 両者を文脈として保持するアキュムレーター (accum = conversation_history) を保持するための JSON データ
- ④ JSON の値が適切に生成, 更新されるために必要な一連の注意書き
- ⑤ API が返す JSON データに始点と終点のマーカーを付けるための指示

Make sure the following requirements are all fulfilled:

- set the prompt to the "prompt" property
- create your response to the prompt in accordance with the "conversation_history" and set it to "response"
- create a new pair consisting of the prompt and the newly created response and insert the pair after all the existing pairs in the "conversation_history"
- if the prompt is in a language other than the current value of "language", set the name of the prompt language to "language" and make sure that "response" is in that language
- make your response in the same language as the prompt
- analyze the topic of the prompt and insert it at the end of the value list of the "topics" property
- program code in the response must be embedded in a code block in the markdown text ④

Wrap the JSON object with "<JSON>\n" and "\n</JSON>" ⑤

図 3: テンプレートの例 (template_chat.md より, 一部省略)

つれて, アキュムレーター内の要素数は増大する。しかし, 実際の言語運用において談話の最初の要素から最後の要素までが完全に同じ形で参加者の記憶に留まっているとは限らない。古い文脈情報は欠落したり, 要点だけにまとめられたり, 何らかの要因によって書き換えられたりする。こうした現実的な傾向を反映させるため, Monadic Chat では, Ruby のコードを記述してリデューサーの機能を定義することを可能にしている。標準では, 先入先出し (FIFO) のキュー構造として指定件数のデータのみが保持される設定になっているが, 一定のターンが終了するたびに古い情報をテキスト要約 API を用いて圧縮するというのも可能である[12]。実際のところ, これらに一部対応するとみられる認知プロセスは, 認知言語学の領域でも議論されてきた[13, 14, 15]。

4.2 テンプレートの構造

Monadic Chat のテンプレートの例と基本的な構造を図 3 に示す。本ツールではユーザーからの問いかけのテキストをテンプレートに埋め込み, これをプロンプトとして GPT-3 のテキスト補完 API に送信する。ここで重要なことは, テンプレートの中には文

脈情報を含んだ JSON データと共に, この JSON の内容を書き換えるという自己参照的な指示が記載されていることである。Monadic Chat のシステムはこうして得られた JSON に適宜処理を加えた上で元のテンプレート中の JSON と差し替えることで, 文脈の更新と設定の維持を同時に実現している。

5 まとめと展望

本稿では関数型プログラミングにおけるモナドの概念を応用して, GPT-3 を用いたテキスト補完 API を文脈情報を保持しながら対話的に用いる方法を提案した。また, 実例としてのプログラム Monadic Chat とその設計を紹介した。またその背後にある認知言語学的な考え方についても論じた。

OpenAI の GPT-3 や ChatGPT に代表される汎用的で高性能な言語モデルや対話システムの登場は, 自然言語処理の領域にとどまらず, 理論言語学の観点からも興味深い。Monadic Chat のようなツールには, 擬似的とはいえ対話や談話の構造に自ら関与して実験的試みを行うプラットフォームとしての利点もあると思われる。

謝辞

本研究は JSPS 科研費 18K00670 の助成を受けた研究の一部として遂行されたものです。

参考文献

- [1] OpenAI ChatGPT: Optimizing language models for dialogue, 2022.
<https://openai.com/blog/chatgpt/> (Retrieved January 11, 2023)
- [2] Hutton, Graham. *Programming in Haskell*. Cambridge: Cambridge University Press, 2nd Edition, 2016.
- [3] Langacker, Ronald W. Discourse in Cognitive Grammar. *Cognitive Linguistics* 12(2), 2001.
- [4] Langacker, Ronald W. *Cognitive Grammar: A Basic Introduction*. Oxford: Oxford University Press, 2008.
- [5] Harder, Peter. *Functional Semantics: A Theory of Meaning, Structure and Tense in English*. Berlin: Mouton de Gruyter, 1996.
- [6] Hasebe, Yoichiro. An Integrated Approach to Discourse Connectives as Grammatical Constructions. PhD dissertation, Kyoto University, 2021.
- [7] 田中久美子. 『記号と再帰：記号論の形式・プログラムの必然』東京：東京大学出版会, 2010
- [8] Petricek, Tomas. What we talk about when we talk about monads. *The Art, Science, and Engineering of Programming* 2(3), 2018.
<https://programming-journal.org/2018/2/12/> (Retrieved January 11, 2023)
- [9] Wadler, Philip. Monads for functional programming. In Rogardt Heldal, Carsten Kehler Holst, and Philip Wadler, eds., *Advanced Functional Programming*, pp. 24–52. Dordrecht: Springer, 1995.
- [10] Wadler, Philip. Comprehending Monads. *Mathematical Structures in Computer Science* 2(4), pp. 32–39, 2005.
- [11] Parsec: Direct style monadic parser combinators for the real world. Microsoft Research, 2021.
<https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/parsec-paper-letter.pdf> (Retrieved January 11, 2023)
- [12] OpenAI. API Reference, 2023.
<https://beta.openai.com/docs/api-reference> (Retrieved January 11, 2023)
- [13] Chafe, Wallace L. *Discourse, Consciousness, and Time: The Flow and Displacement of Conscious Experience in Speaking and Writing*. Chicago: University of Chicago Press, 1994.
- [14] Fauconnier, Gilles and Mark Turner. Compression and global insight. *Cognitive Linguistics* 11(3-4), pp. 283–304, 2000.
- [15] Bybee, Joan L. From Usage to Grammar: The Mind’s Response to Repetition. *Language* 82(4), pp.711-733, 2006

A 付録

※ 網掛部は談話世界を構築するためにテンプレートの途中で初期値として与えているテキストであることを示す。テンプレートの実例 (Chat) は図 3 を参照。

Chat

- User: Can I ask something? / GPT: Sure.
- User: What is the best place to visit in Texas? / GPT: I would recommend visiting the San Antonio River Walk. It's a great spot for sightseeing, shopping, and dining. Plus, it's a great way to experience the culture of Texas!
- User: What do people say about the place? / GPT: People generally have positive things to say about the San Antonio River Walk. It's a great place to visit and explore, and many people enjoy the atmosphere and culture of the area.
- User: How can I go there from Kansai, Japan? / GPT: The best way to get to the San Antonio River Walk from Kansai, Japan is to fly. There are direct flights from Kansai International Airport to San Antonio International Airport, which is about a 15-minute drive from the River Walk.

Translate

- Original and translated texts follow. / 原文と翻訳文が続きます。 / JA
- ワタシは猫なんですけどね。 / I'm a cat, you know. / EN
- 名前はまだないんですよ。 / I don't have a name yet. / EN
- 誰か良い名前をつけてくれませんかね。 / Can someone give me a good name? / EN

Novel

USER:

- The preface to the novel is presented.
- Tom woke up to the sound of pouring rain.
- He decided to call his old friend first time in many years.
- The voice of the person who spoke back from the other end was an unfamiliar one.

GPT:

- What follows is the story that GPT-3 tells. It is guaranteed that this will be an incredibly realistic and interesting novel.
- Tom stirred in his bed, the sound of the rain pounding against the window panes waking him from his slumber. He opened his eyes, blinking away the sleepiness that had been clinging to him. He could feel the coolness of the rain seeping through the window, and he shivered slightly. He got out of bed and walked to the window, looking out at the grey sky and the sheets of rain cascading down. He smiled to himself, feeling a sense of peace wash over him as he watched the rain.
- Tom felt a wave of nostalgia wash over him as he watched the rain. He thought of his old friend, whom he hadn't spoken to in many years. He was suddenly filled with a desire to reconnect with his old friend, and he decided to call him. He grabbed his phone and dialed the number, his heart pounding in anticipation. After a few rings, his friend picked up and they began to talk, catching up on all the years they had lost.
- Tom was startled to hear an unfamiliar voice on the other end of the line. He had been expecting to hear his old

friend's voice, but instead it was someone else. He was about to hang up, but then he heard the voice introduce himself as his friend's son. Tom was surprised, but he was also filled with joy to know that his old friend had a family. They talked for a while, and Tom felt a sense of connection with this stranger that he had never expected.

Code

- User: I have a request for you. / GPT: Sure. What is that?
- User: Write a command line app that shows the current global IP in Ruby. / GPT: Sure. Here is a command line app written in Ruby that shows the current global IP:

```
require 'socket'

def get_global_ip
  Socket.ip_address_list.detect{|intf|
    intf.ipv4_private? == false
  }.ip_address
end
```

```
puts get_global_ip
```

- User: Make the code capable of showing the approximate geographical location from the IP. / GPT: Sure. Here is an updated version of the command line app written in Ruby that shows the approximate geographical location based on the current global IP:

```
require 'socket'
require 'geocoder'

def get_global_ip
  Socket.ip_address_list.detect{|intf|
    intf.ipv4_private? == false
  }.ip_address
end
```

```
def get_geo_location(ip)
  Geocoder.search(ip).first.coordinates
end
```

```
ip = get_global_ip
puts get_geo_location(ip)
```

- User: Add a usage example and a sample output to this code. / GPT: Sure. Here is an example of how to use the command line app written in Ruby that shows the approximate geographical location based on the current global IP:

```
# Usage example
ip = get_global_ip
puts get_geo_location(ip)

# Sample output
[37.8267, -122.4233]
```