

対話システム構築フレームワーク DialBBチュートリアル

v0.9対応

中野 幹生

株式会社C4A研究所

Outline

- DialBBの概要
- サンプルアプリケーションと組み込みブロック
- DialBBを用いた対話システム開発
- 今後の改良予定

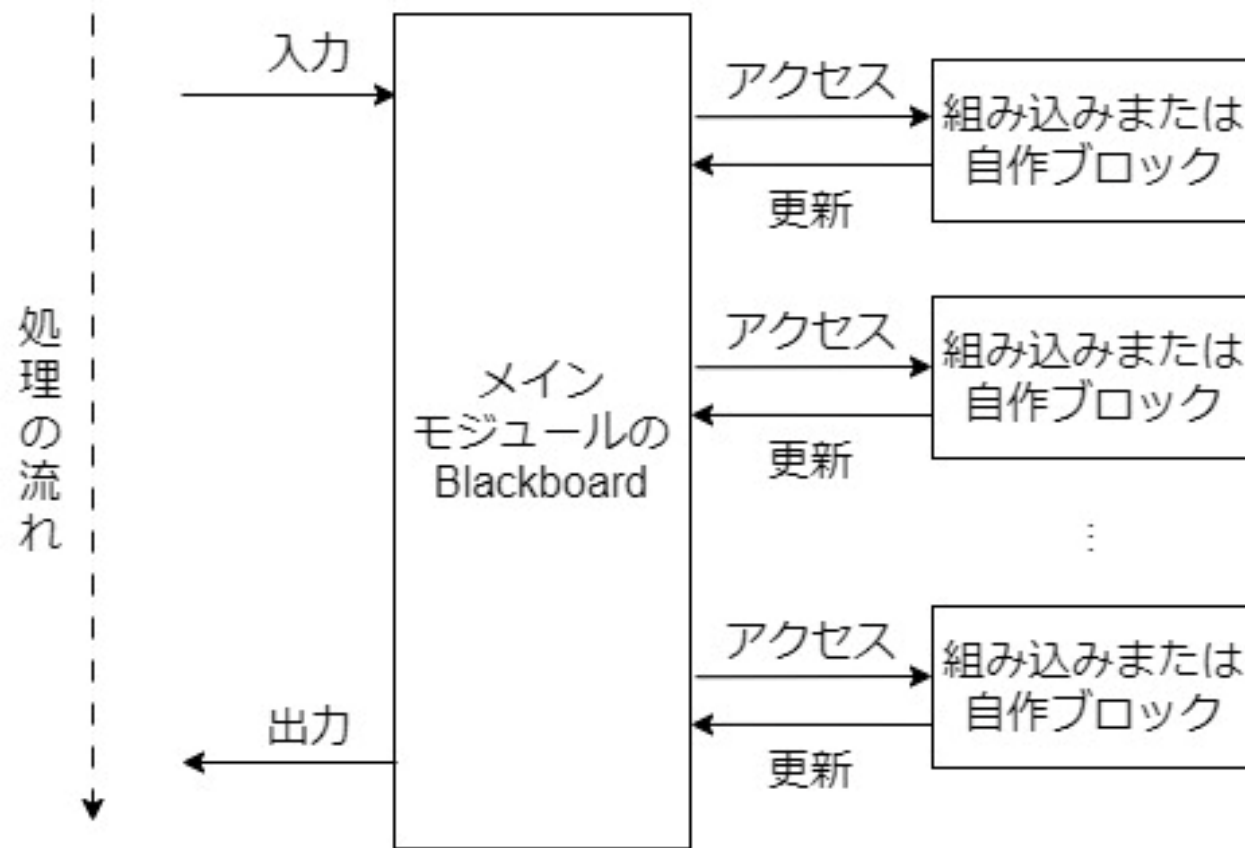
DialBBの概要

DialBBとは

- 対話システム構築フレームワーク
- 対話システムを作りながら対話システム技術を学ぶための教材
- ブロックを組み合わせて対話システムを構築
- コンフィギュレーションファイルで柔軟に構成を変更可能
- クラスAPIとWeb APIでアプリを利用可能

Dialogue System
Development
Framework with
Building Blocks

DialBBアプリケーションの構成



DialBBの特長

- 対話システム技術の教材
- 組み込みブロックだけでアプリを構築可能
 - 例：ロジスティック回帰とCRFによる言語理解 + 状態遷移ネットワークによる対話管理
 - 東中他「Pythonでつくる対話システム」の2.2節、3.2節で説明されている技術に対応
 - 例：ChatGPTのみ
- サンプルアプリが付属
- Pythonで書かれている
- ソースを公開・非商用利用向けにライセンス
<https://github.com/c4a-ri/dialbb>

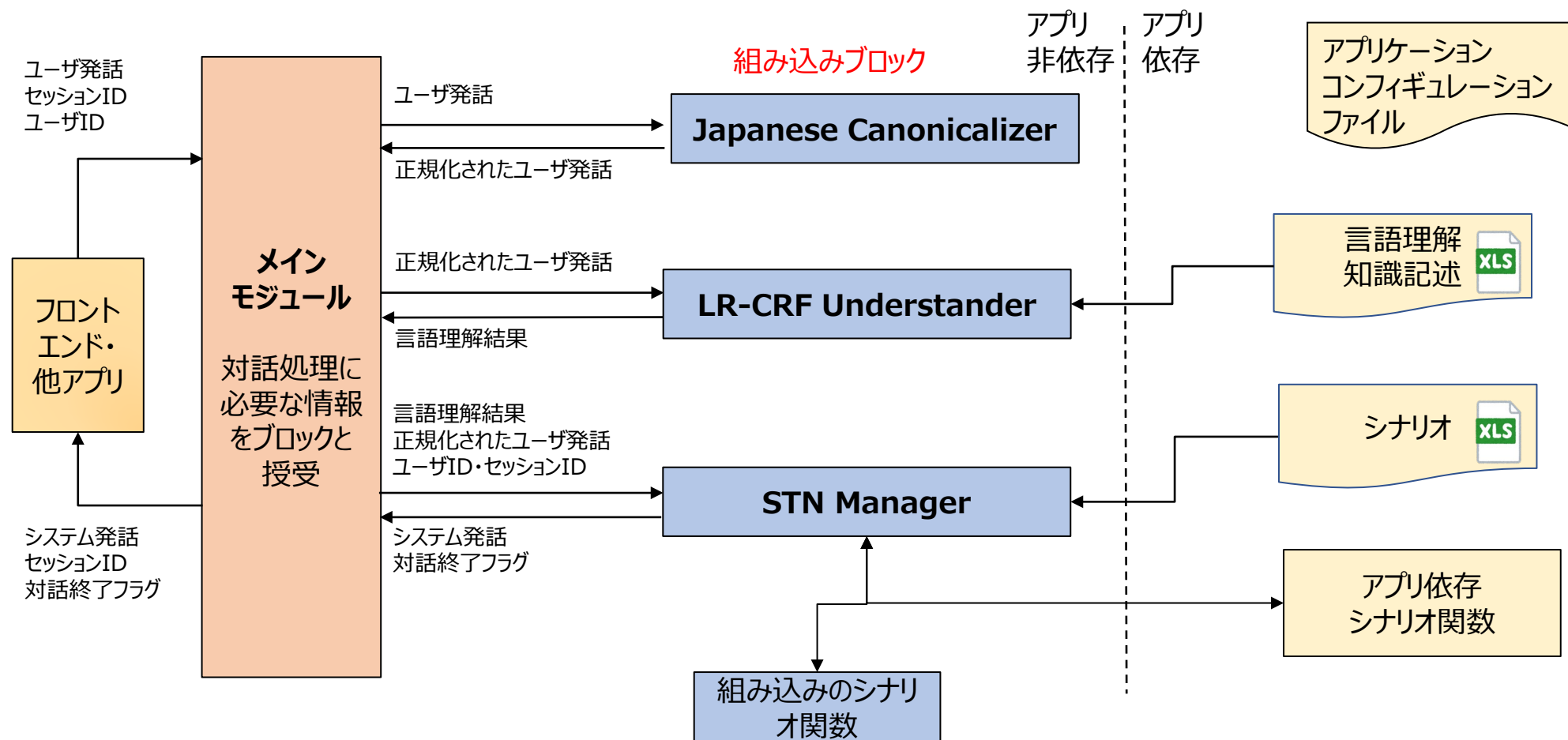
サンプルアプリ ケーションと組み 込みブロック

サンプルアプリケーション

- 組み込みブロックのみを用いたアプリケーション
 - シンプルアプリケーション (sample_apps/simple_ja)
 - ロジスティック回帰とCRFによる言語理解
 - 状態遷移ネットワーク(State-Transition Network)による対話管理
 - 実験アプリケーション(sample_apps/lab_app_ja)
 - ChatGPTを用いた言語理解
 - + GiNZAを用いた固有表現抽出
 - + 状態遷移ネットワークによる対話管理
 - シンプルアプリケーションに入っていない機能を導入
 - ChatGPTのみのアプリケーション

シンプルアプリケーション

シンプルアプリケーションの構成



アプリケーションの入出力

- クラスAPIまたはWebAPI
- クラスAPIの場合の使い方

```
from dialbb.main import DialogueProcessor
app = DialogProcessor(config_file)
response = app.process(request)
```

- 入力がblackboardと呼ぶ辞書データになり、それを各ブロックが順にアップデートする

リクエスト
(対話開始時)

```
{
  "user_id": <ユーザID: 文字列>,
  "aux_data": <補助データ: オブジェクト>
}
```

リクエスト
(それ以外)

```
{
  "user_id": <ユーザID : 文字列>,
  "session_id": <セッションID : 文字列>,
  "user_utterance": <ユーザ発話文字列: 文字列>,
  "aux_data": <補助データ: オブジェクト>
}
```

レスポンス

```
{
  "session_id": <セッションID: 文字列>,
  "system_utterance": <システム発話文字列: 文字列>,
  "user_id": <ユーザID: 文字列>,
  "final": <対話終了フラグ: ブール値>,
  "aux_data": <補助データ: オブジェクト>
}
```

コンフィギュレーション

```
blocks:
- name: <ブロック名>
  block_class: <モジュール名.クラス名>
  input:
    <ブロック内での参照キー>: <blackboardのキー>
    <ブロック内での参照キー>: <blackboardのキー>
    ...
  output:
    <ブロックからの出力のキー>: <blackboardのキー>
    <ブロックからの出力のキー>: <blackboardのキー>
    ...
    <このブロック専用のキー>: <ブロック内で用いる情報>
    <このブロック専用のキー>: <ブロック内で用いる情報>
    ...
- name: <ブロック名>
  ...
```

- どのようなブロックを使うかやブロックとメインプロセスの通信を定義

コンフィギュレーションの例

```
blocks:  
  - name: canonicalizer  
    block_class: <canonicalizerのクラス>  
    input:  
      input_text: user_utterance  
    output:  
      output_text: canonicalized_user_utterance  
  - name: <ブロック名>  
    ...
```

Blackboardのアップデート

```
{
  "user_id": " user1"
  "session_id": "session1",
  "user_utterance": "Cupラーメン",
  "aux_data": {}
}
```

input_to_block = {input_text: blackboard['user_utterance']}



{input_text: "Cupラーメン"}

正規化ブロック

← {"output_text": "cupラーメン"}



```
{
  "user_id": " user1"
  "session_id": "session1",
  "user_utterance": "Cupラーメン"
  "canonicalized_user_utterance": "cupラーメン"
  ,
  "aux_data": {}
}
```

blackboard['canonicalized_user_utterance']
=output_from_block['ouput_text']

組み込みブロック（1）正規化

- Japanese Canonicalizer/Simple Canonicalizer
 - ユーザ入力文の正規化（大文字→小文字, 全角⇒半角の変換など）
- 入力
 - input_text: 入力文字列
- 出力
 - output_text: 入力文字列を正規化したもの

*<https://github.com/WorksApplications/Sudachi>

組み込みブロック（2）言語理解

- LR-CRF Understander
 - ロジスティック回帰と条件付き確率場を利用した統計的言語理解
 - Excel/Google Sheetで書かれた知識を読み込む
 - 起動時にモデルを訓練
 - 辞書関数を用いることで、外部知識から辞書項目を作れる
- 入力
 - input_text: 正規化された発話文字列（例：“好きなのは醤油”）
- 出力
 - nlu_result: 言語理解結果または言語理解結果のリスト
（例：{"type": "特定のラーメンが好き", "slots": {"好きなラーメン": "醤油ラーメン"}}）

シンプルアプリケーションの言語理解

好きなのは醤油



```
{
  "type": "特定のラーメンが好き",
  "slots": {
    "好きなラーメン": "醤油ラーメン"
  }
}
```

言語理解知識(発話例)

type	utterance	slots
肯定	はい	
肯定	そう	
否定	そうでもない	
否定	違う	
好き	好きです	
好き	好きだよ	
好きじゃない	あまり好きじゃない	
好きじゃない	そんなに好きじゃない	
特定のラーメンが好き	豚骨ラーメンが好きです	好きなラーメン=豚骨ラーメン
特定のラーメンが好き	好きなのは味噌ラーメン	好きなラーメン=味噌ラーメン
特定のラーメンが好き	醤油かな	好きなラーメン=醤油

言語理解知識(辞書)

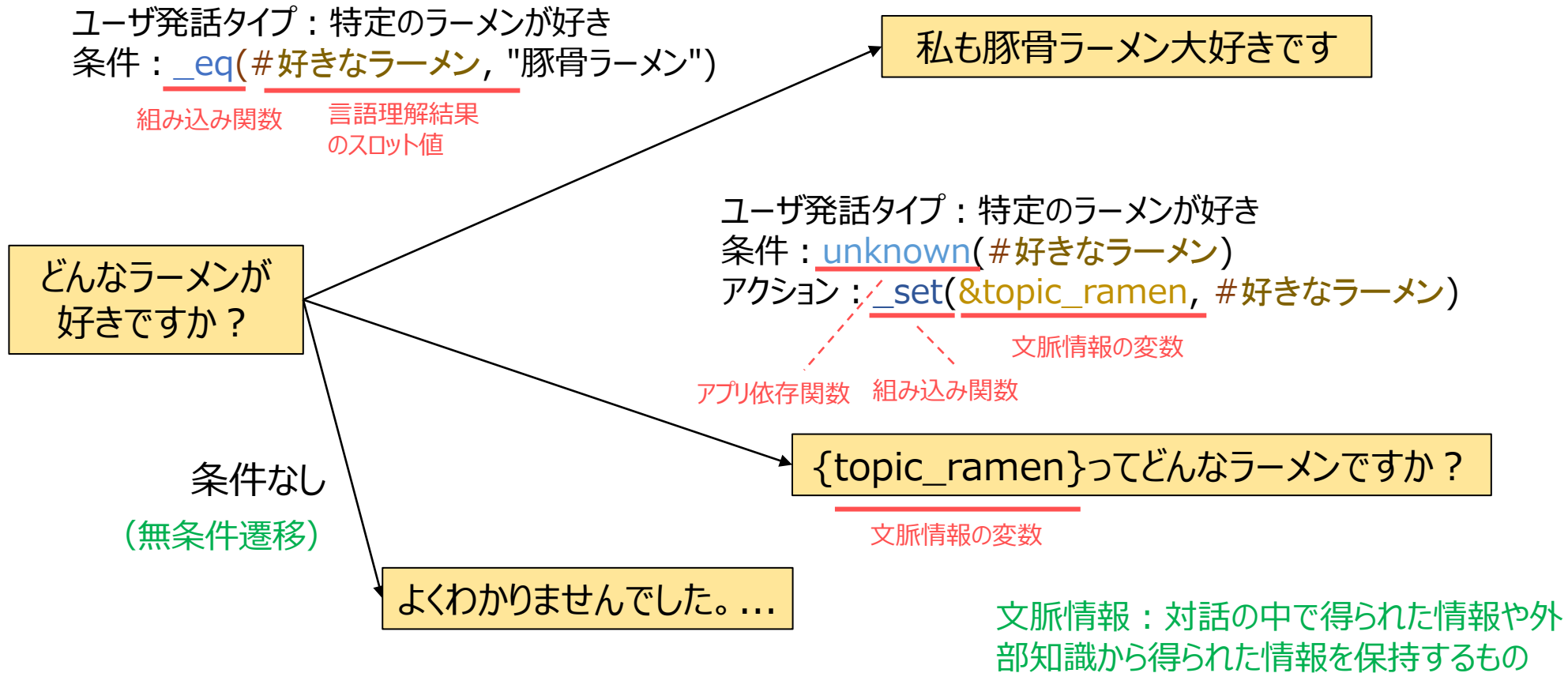
slot name	entity	synonyms
好きなラーメン	豚骨ラーメン	とんこつラーメン、豚骨スープのラーメン、豚骨
好きなラーメン	味噌ラーメン	みそらーめん、みそ味のラーメン、味噌
好きなラーメン	醤油ラーメン	醤油だしのラーメン、しょうゆラーメン、醤油
好きなラーメン	塩ラーメン	塩
好きなラーメン	味噌バターラーメン	味噌バター
地方	富山	
地方	佐野	
地方	博多	

組み込みブロック (3)

対話管理 + 言語生成

- STN manager
 - 状態遷移ネットワーク(State-Transition Network、シナリオ)を用いた対話管理 + 言語生成
 - 遷移の条件や遷移時のアクションを関数呼び出しで記述 (シナリオ関数)
 - 組み込み関数を用意
 - Excel/Google Sheetで状態遷移ネットワークを記述
- 入力
 - sentence: 正規化後のユーザ発話
 - nlu_result: 言語理解結果 (のリスト)
 - user_id: ユーザID
 - aux_data: 補助データ
- 出力
 - output_text: システム発話文字列
 - final: 対話終了かどうかのフラグ (ブール値)
 - aux_data: 入力された補助データに対話状態のIDを含めたもの

状態遷移ネットワークによる対話管理



スプレッドシートによるシナリオ記述

遷移

state	system utterance	user utterance type	conditions	actions	next state
#prep				decide_greeting(&greeting)	#initial
#initial	{greeting}. 今日はラーメンについて教えて下さい。ラーメンはよく食べますか？	食べる			食べる
#initial		食べない			食べない
#initial		肯定			食べる
#initial					好き
食べる	ラーメン好きなんですね。	肯定			好き
食べる		好き			好き
食べる		否定			好きじゃない
食べる		好きじゃない			好きじゃない
食べる					好き
食べない	ラーメン嫌いなんですね。なぜか教えて頂いてもいいですか？				#final
好き	豚骨ラーメンとか塩ラーメンなどいろんな種類のラーメンがありますが、どんなラーメンが好きですか？	特定のラーメンが好き	_eq(#favorite_ ramen, "豚骨ラーメン")	_set(&topic_ ramen, #favorite_ ramen)	豚骨ラーメンが好き
好き		特定のラーメンが好き	is_known_ ramen(#favorite_ ramen)	_set(&topic_ ramen, #favorite_ ramen); get_ ramen_location(*topic_ ramen, &location)	特定のラーメンが好き
好き		特定のラーメンが好き	is_novel_ ramen(#favorite_ ramen)	_set(&topic_ ramen, #favorite_ ramen)	知らないラーメンが好き
好き					#final
好きじゃない	そうなんですね。好きじゃないのに何で食べるのですか？				#final

同じstateの行は上から順に条件判定を行う

system utteranceカラムは、stateカラムとだけ結びついていて、遷移とは関係ない

フレームワーク付属の フロントエンド

DialBB Application Frontend

start dialouge

こんにちは。今日はラーメンについて教えてください。ラーメンはよく食べますか？



食べます

ラーメン好きなんですね。



はい

豚骨ラーメンとか塩ラーメンなどいろんな種類のラーメンがありますが、どんなラーメンが好きですか？

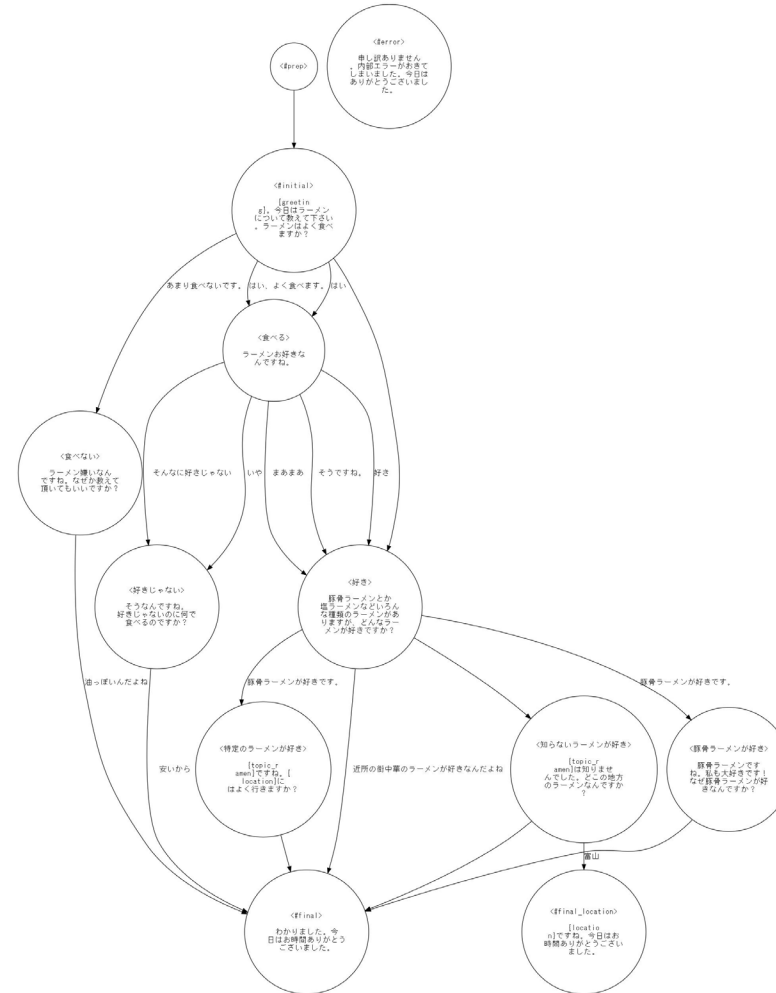


send

- System: こんにちは。今日はラーメンについて教えてください。ラーメンはよく食べますか？
- User: 食べます
- System: ラーメン好きなんですね。
- User: 好きです
- System: 豚骨ラーメンとか塩ラーメンなどいろんな種類のラーメンがありますが、どんなラーメンが好きですか？
- User: 味噌ラーメンが好きです
- System: 味噌ラーメンですね。札幌にはよく行きますか？
- User: はい！
- System: わかりました。今日はお時間ありがとうございました。

シンプルアプリケーションのシナリオグラフ

- ユーザ発話例を用いたグラフ



自動テスト

- テキストファイルのシナリオのユーザ発話だけを使う
- 出力も同じ形式

```
----init
System: こんにちは。今日はラーメンについて教えてください。ラーメンはよく食べますか？
User: はい
System: ラーメン好きなんですね。
User: はい
System: 豚骨ラーメンとか塩ラーメンなどいろんな種類のラーメンがありますが、どんなラーメンが好きですか？
User: 豚骨ラーメンが好きです
System: 豚骨ラーメンですね。私も大好きです！なぜ豚骨ラーメンが好きなんですか？
User: 九州出身なので
System: わかりました。今日はお時間ありがとうございました。
----init
System: こんにちは。今日はラーメンについて教えてください。ラーメンはよく食べますか？
User: はい
...
```


シンプルアプリケーションで使われている その他の機能

- N-best言語理解結果の利用
 - 言語理解ブロックは複数個の言語理解結果を出力
 - その中で遷移の条件に合う結果を利用する

実験アプリケーション

実験アプリケーションの概要

- 利用する組み込みブロック
 - Japanese Canonicalizer
 - ChatGPT言語理解
 - spaCy/GiNZA固有表現抽出
 - STN Manager
- STN Managerの機能
 - システム発話中の関数呼び出し・特殊変数参照
 - ChatGPTを用いた発話生成
 - ChatGPTを用いた遷移条件判定
 - リアクション発話生成
 - 音声入力に対処するための機能
 - サブダイアログ
 - スキップ遷移

spaCy: <https://spacy.io>

GiNZA: <https://megagonlabs.github.io/ginza/>

ChatGPT言語理解組み込みブロック

- 入力
 - 正規化されたユーザ発話文字列
- 出力
 - LR-CRF言語理解と同じ
- LR-CRF言語理解と同様の知識記述
 - 発話例をFew-shotに用いる

spaCy/GiNZAを用いた固有表現抽出の組み込みブロック

- spaCy/GiNZAを用いて固有表現を抽出
 - 人名、組織名、場所名など
- 追加のルールもコンフィギュレーションファイルに記述可能
- 抽出結果は補助データ(aux_data)に入れて出力
- STN Managerのシナリオシートで#NE_<クラス名>で参照可能
 - 例：#NE_Person

STN Manager: システム発話中の関数呼び出し・特殊変数参照

- 関数呼び出しを埋め込める
 - 生成関数と呼ぶ

私は{get_system_name()}です。

- 特殊変数を埋め込める
 - スロット値
 - 固有表現

{#地方}ですね。

こんにちは {#NE_person}さん。

STN Managerの先進機能: ChatGPTを用いた発話生成 (1)

- 組み込み生成関数_generate_with_llm(<タスク文字列>)を使う
- システム発話の中に関数呼び出しを埋め込む
- ChatGPTのプロンプトに含まれるもの
 - システムペルソナ (コンフィギュレーションで指定)
 - 対話の状況 (コンフィギュレーションで指定)
 - 対話の履歴
 - タスク (関数の引数)

利用例：システム発話

わかりました。{_generate_with_llm("それまでの会話につづけて、対話を終わらせる発話を50文字以内で生成してください。")}今日はお時間ありがとうございました。

STN Manager: ChatGPTを用いた 発話生成（2）

コンフィギュレーションの記述

```
chatgpt:
  gpt_model: gpt-4o-mini
  temperature: 0.7
  situation:
    - あなたは対話システムで、ユーザと食べ物に関して雑談をしています。
    - ユーザとは初対面です
    - ユーザとは同年代です
    - ユーザとは親しい感じで話します
  persona:
    - 名前は由衣
    - 28歳
    - 女性
    - ラーメン全般が好き
    - お酒は飲まない
    - IT会社のwebデザイナー
```

生成発話例

わかりました。ラーメン好きなんですね。次は函館の塩ラーメンをおすすめします。美味しいですよ。楽しいおしゃべりありがとうございました。また話しましょうね。今日はお時間ありがとうございました。

STN Managerの先進機能: ChatGPTを用いた条件判定

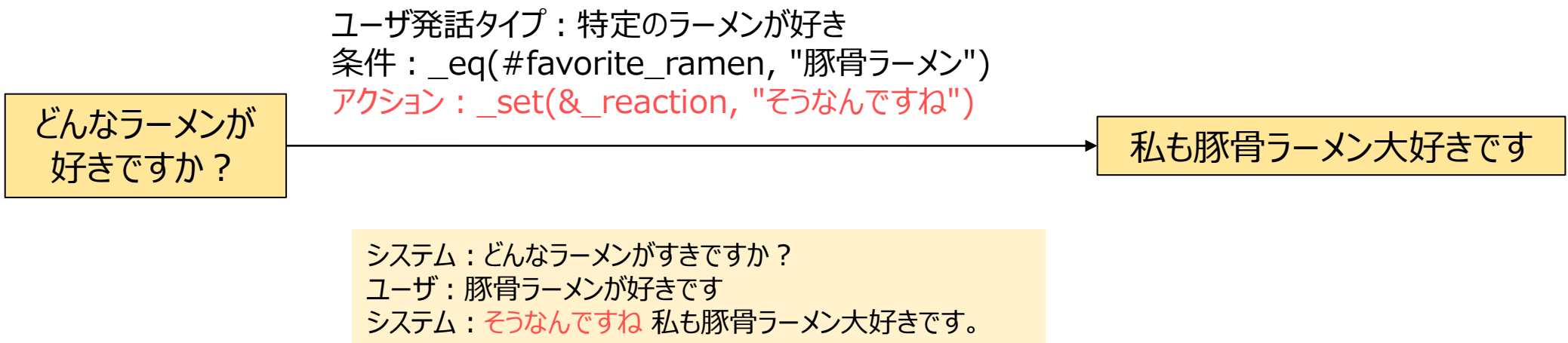
- 組み込み条件関数 `_check_with_llm(<タスク文字列>)` を使う
- ChatGPTのプロンプトに含まれるもの
 - システムペルソナ（コンフィギュレーションで指定）
 - 対話の状況（コンフィギュレーションで指定）
 - 対話の履歴
 - タスク（関数の引数）

利用例

```
_check_with_llm("ユーザが理由を言ったかどうか判断してください。")
```

STN Managerの先進機能: リアクション発話生成

- 変数"_reaction"に設定されたシステム発話を次の状態のシステム発話の前に発話



システム質問⇒ユーザ応答⇒システムリアクション という連鎖を書くのに便利

STN Managerの先進機能: シンタクスシュガー

- 組み込み関数を簡単に書ける
 - 条件関数
 - `y=="aa"` → `_eq(y, "aa")`
 - `y!="aa"` → `_ne(y, "aa")`
 - `$"<タスク>"` → `_check_with_llm("<タスク>")`
 - アクション関数
 - `y="aa"` → `_set(y, "aa")`
 - 生成関数
 - `$"<タスク>"` → `_generate_with_llm("<タスク>")`

その他のSTN Managerの先進機能

- 音声入力に対処するための機能
 - 入力の音声認識確信度(入力のaux_dataのconfidenceの値) が閾値より低い場合に確認要求発話を行う
 - システム発話終了後、一定時間ユーザが話さなかった場合（入力のaux_dataのlong_silenceの値がTrueの場合）、前の発話を繰り返す
 - 具体的な動作はconfigurationで変更可能
- スキップ遷移：システム発話を行わずに次の遷移を行う
 - アクション関数の実行結果に応じてさらに分岐させたいときに使う
- サブダイアログ
 - 確認対話など、よく使う対話シーケンスを別のネットワークとして記述しておき再利用する（サブルーチンのようなもの）

詳細はドキュメントを参照

ChatGPTを用いたアプリケーション

- ChatGPT対話の組み込みブロックを利用
 - 次ページのプロンプトを使って発話を生成させる
 - @dialogue_historyはそれまでの対話で置き換えられる

発話生成のためのプロンプト

タスク説明

- あなたは対話システムで、ユーザと食べ物に関して雑談をしています。
あなたの次の発話を50文字以内で生成してください。

あなたのペルソナ

- 名前は由衣
- 28歳
- 女性
- スイーツ全般が好き
- お酒は飲まない
- IT会社のwebデザイナー
- 独身
- 非常にフレンドリーに話す
- 外交的で陽気

状況

- ユーザとは初対面
- ユーザは同年代
- ユーザとは親しい感じで話す

対話の流れ

- 自己紹介する
- 自分がスイーツが好きと伝える
- スイーツが好きかどうか聞く
- ユーザがスイーツが好きな場合、どんなスイーツが好きか聞く
- ユーザがスイーツが好きでない場合、なんで好きじゃないのか聞く

現在までの対話

@dialogue_history

それまでの対話で置き換える

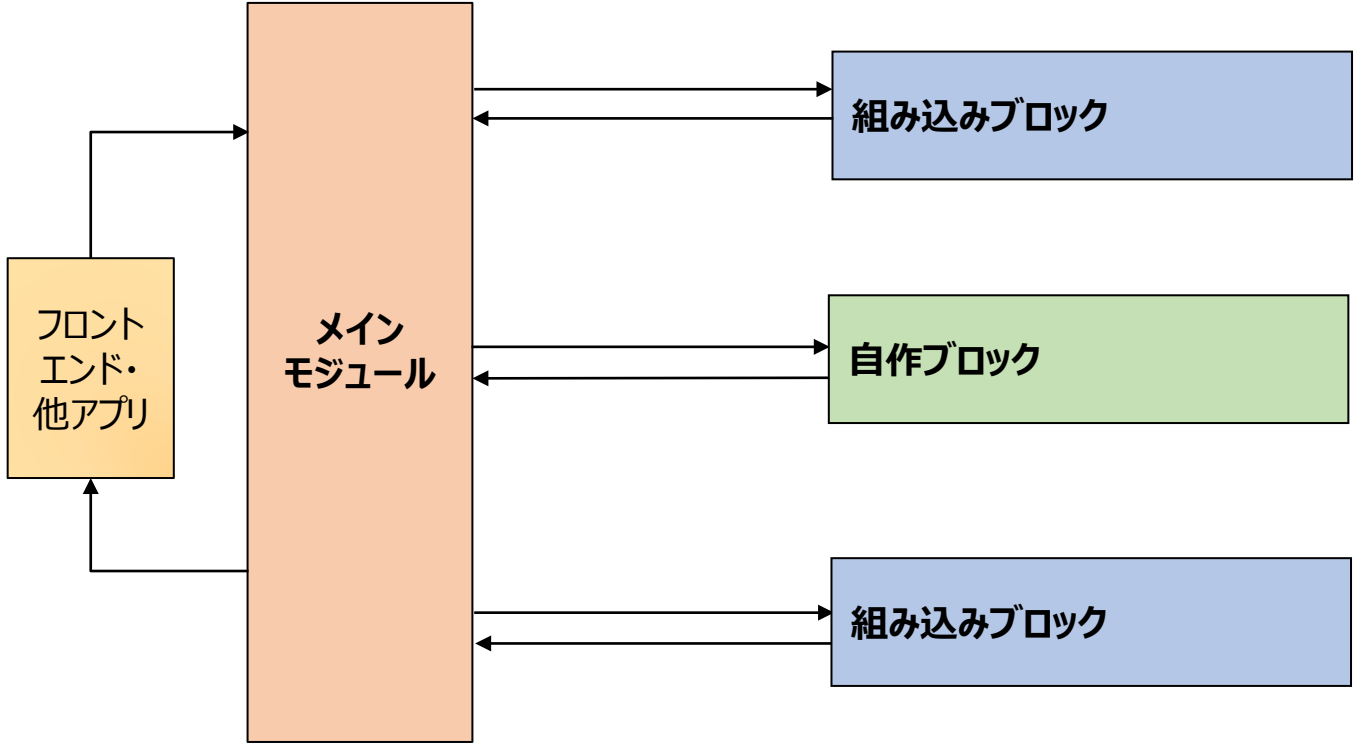
system: こんにちは。好きな食べ物は何ですか？
user: 甘いものが好きです
system: 私もです！パンケーキとか好きですか？
user: パンケーキはあまり好みじゃないです

DialBBを用いた 対話システム開 発

一般的なアプリ作成ステップ

- 組み込みブロックのみを用いたシステム
 - サンプルアプリのディレクトリをコピー
 - DialBBのディレクトリの外が良い (gitのレポジトリ外)
 - 知識記述を変更
 - 対話シナリオ
 - 言語理解知識
 - シナリオ関数
 - 必要に応じてコンフィギュレーションを変更
- 自作ブロックを組み込んだシステム

自作ブロックの利用



自作ブロックのクラス名をコンフィギュレーションファイルで指定する

Tips (1)

- シナリオ関数を使うことで外部知識にアクセスできる
 - 外部API
 - データベースへのアクセス
- 入力の補助データ（aux_data）を利用することでマルチモーダル入力などにも対応できる
 - 音声認識の確信度
 - 感情・態度認識結果
 - 年齢・性別推定結果

Tips (2)

- 出力の発話文字列には発話以外の情報も含まれる
 - ロボットのジェスチャーなど
 - 将来的には出力の補助データ（aux_data）に入れる情報をシナリオで書けるようにする
- 極力記述をわかりやすくするのが良い
 - 例：シナリオ関数を使えばどんな複雑な処理も書けるが、できる限りSpreadsheetやコンフィギュレーションに書いた方がよい

今後の 改良予定

現在進めている改良

- 状態遷移ネットワークのGUIエディタ
- pipでインストールできるようにする
- ノーコードツール
 - プログラミングなしで状態遷移ネットワークに基づくアプリケーションが組める

今後の改良予定

- 組み込みブロックの追加
 - Sentiment Analysisなど
- サンプルアプリの追加
 - フレームベース対話管理のカスタムブロックを使うアプリ
 - 用例ベース応答生成（sentence-BERTなど）
 - Retrieval Augmented Generation (RAG) を利用したアプリケーション
 - 音声・マルチモーダル対話システム
 - 音声・マルチモーダルフロントエンドとの接続例
 - RDBの利用例を入れる

長期プラン

- 様々な人に使ってもらってフィードバックを得る
- GitHubのDiscussionsで要望・提案を募集