ADVISER Workshop

A Dialog System Framework for Education & Research

Developed by Lindsey, Daniel the Most Equal and Dirk the Hungriest

Outline

- Getting Started
- Ontology Creation
- NLU and Regexes
- Beliefstate Tracker
- Handcrafted Policy
- NLG and Templates

Getting Started

Links

- Paper: https://www.aclweb.org/anthology/P19-3016/
- Full code: https://github.com/DigitalPhonetics/adviser
- Tutorial Code: https://github.com/sheogorath15/adviser_workshop

Installation

Downloading the code:

- Go to your working directory using the Terminal
  > `cd path_to_your_working_directory`
- Download the code
  > `git clone https://github.com/sheogorath15/adviser_workshop.git`
- Enter the pydial folder
  > `cd adviser`
- Set up/ activate Virtual environment
  > `python3 -m venv wrkshp_env`
  > `source wrkshp_env/bin/activate`
- Install requirements
  > `pip install -r requirements.txt`

Testing ADIVSER

Testing the installation:

- The easiest way to test is to run a dialog
- Type:

```
> python run_chat.py -d ImsCourses -dp
```

- This should allow you to start a chat with the system about courses at the IMS
- To start, type “hi” or provide info about:
  - ***Term:*** *summer* or *winter*
  - ***Credits:*** *3 credits* or *6 credits*
  - ***Related to:*** *speech* or *deep learning* or *linguistics* or …
  - ***Master:*** *yes* or *no*
  - etc.

Creating a New Domain

Adding a New Domain

ADVISER provides **tools/ontology/create_ontology.py** in order to add a new domain, if

- The data is stored in an SQLite database (DB)
- The DB has only one table, whose name is the domain identifier (eg. **superhero**)
- Each table attribute is a slot
- Binary slots are represented as 1: True and 0:False

```
> python tools/ontology/create_ontology.py resources/databases/superhero.db
```

Types of slots

- *Informable*: information the user can inform the system about
- *System requestable*: information the system can actively ask the user
- *Requestable*: information the user can request from the system
- *Binary*: information which is in the form of a yes/no or true/false

Superhero Slots

Superhero
Slot
name
primary_uniform_color
main_superpower
last_known_location
loyalty
description
real_name

- All Slots should be of type str
- Database contains 15 entries
- Name is the primary key

Superhero Domain

User Informs:

- *I want a superhero who is primarily green*
- *I want a superhero who is good at martial arts*
- *I can only work with a superhero who is part of the Avengers*

User Requests:

- *Can you give me a description of the hero?*
- *What is their primary color? I don't want them to clash with the rest of the team*
- *What is their real name?*

Exercise: Adding a New Domain

Slots vs slot types. Mark the slots according to this table, confirm using ENTER

	Slot	User Informable	System requestable	User Requestable	Binary
1	🔑 name	✔		✔	
2	primary_uniform_color	✔	✔	✔	
3	main_superpower	✔	✔	✔	
4	last_known_location			✔	
5	loyalty	✔	✔	✔	
6	description			✔	
7	real_name			✔	

The new ontology is created in:

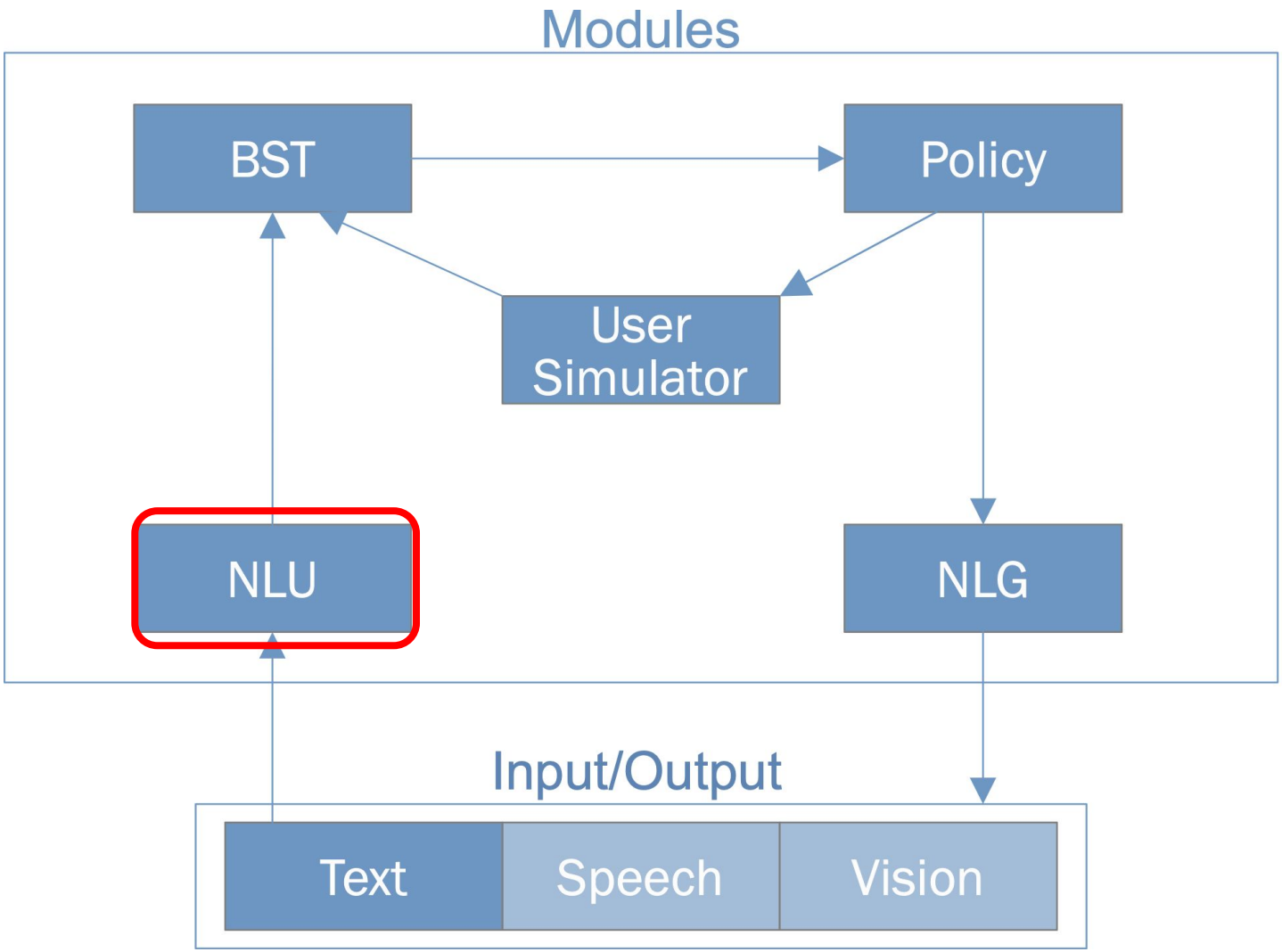
`resources/databases/superhero.json`

```
{
    "discourseAct": [
        "ack",
        "hello",
        "inform",
        ...
    ],
    "informable": {
        "Lastknownlocation": [
            "Avenger\u2019s Tower",
            "Central City",
            ...
        ],
        "Loyalty": [
            "Avengers",
            "Defenders",
            ...
        ],
        "MainSuperpower": [
            "Claws",
            "Gadgets",
            ...
        ], ...
    },
```

```
    "key": "RealName",
    "method": [
        "none",
        "byconstraints",
        ...
    ],
    "requestable": [
        "Name",
        "PrimaryUniformColor",
        ...
    ],
    "system_requestable": [
        "PrimaryUniformColor",
        "MainSuperpower",
        "Loyalty"
    ]
}
```

NLU

Dialog System Architecture



Natural Language Understanding (NLU)

- System decodes information about dialogue act type, which encodes the system or the user intent in a dialog turn, and semantic slots and values

Dialog turn: ***I want a superhero who is primarily green***
Dialog act type: **inform**
Semantic slots: **primary_uniform_color**
Semantic values: **green**
Dialog act: **inform(primary_uniform_color=green)**

Idea: Keyword to intent and slot mapping

Keywords to detect the informable slots (loyalty, main_superpower)?

User informable :

- *I can only work with a superhero who is part of the **Avengers***
- *Please find an **Avengers** superhero*

Keyword → avengers

inform(loyalty=Avengers)

- *I want a superhero who is good at **martial arts***
- *Please find a superhero who knows **karate***

Keywords → martial arts and karate

inform(main_superpower=martial arts)

Idea: Keyword to intent and slot mapping

Keywords to detect the requestable slots (description)?

User requestable:

- *Can you give me a **description** of the hero?*
- *Tell me more **details***

Keyword → description, details

`request(description)`

ADVISER Template Syntax

- To avoid rewriting hundreds of similar regexes, we developed a template syntax
- Write a template which supports synonyms
- The template language supports:
  - Rules
    - Used to map an intent to a series of regexes
  - Functions
    - Allow common parts of regexes to easily be reused

General Regexes

- To map non-domain specific utterances to regexes

```
# general rules

rule hello(): "(\\b|^|\\ )(hi|hello|howdy|hey)\\b"

rule bye(): "(\\b|^|\\ )(bye(-)?(bye)?|good(-)?bye|that'?s?\\ (is\\ )*all)(\\s|$|\\ |\\.)"

rule deny(): "((\\b|^|\\ )(n(o)?|wrong|incorrect|error|nope)|(not\\ (true|correct|right)))(\\s)?$"

rule affirm(): "((yes|yeah|(\\b|^)ok\\b|(\\b|^)OK|okay|sure|^y$|(\\b|^)yep(\\b|$)|(that('?s| is)
)?(?<!not\\ )(?<!no\\ )(right|correct|confirm)))(\\s)?$"

rule thanks(): "(?=.*(^(\\ )*)(((great|good|awesome|nice|cool)\\ )?((that(')?s|\\ (is|was))\\ (very\\
)?helpful)\\ )?(thank|thank(s|\\ you)(\\ (very|so)\\ much)?)(\\ (that(')?s|\\ (is|was))\\ (very\\
)?helpful))?|((great|good|awesome|nice|cool)\\ )?(that(')?s|\\ (is|was))\\ (very\\
)?helpful)|(great|good|awesome|nice|cool))((\\ )*))(?=^(?:(?!bye).)*$).*$"

rule repeat(): "(\\b|^|\\ )(repeat((\\ that )|(\\ it))?)|(say((\\ that )|(\\ it))\\ again)|(again)"

rule reqalts(): "(\\b|^|\\ )((something|anything)\\ else)|(different(\\ one)*)|(another\\
one)|(alternatives*)|(other options*)|((don'*t|do not) (want|like)\\ (that|this)(\\ one)*)"

rule ack(): "{IMPOSSIBLEREGEX()}"

rule bad(): "{IMPOSSIBLEREGEX()}"
```

General Regexes

- To map non-domain specific utterances to regexes

```
# general rules

rule hello(): "(\\b|^|\\ )(hi|hello|howdy|hey)\\b"
```

intent

regex

```
rule bye(): "(\\b|^|\\ )(bye(-)?(bye)?|good(-)?bye|that'?s?\\ (is\\ )*all)(\\s|$|\\ |\\.)"

rule deny(): "((\\b|^|\\ )(n(o)?|wrong|incorrect|error|nope)|(not\\ (true|correct|right)))(\\s)?$"

rule affirm(): "((yes|yeah|(\\b|^)ok\\b|(\\b|^)OK|okay|sure|^y$|(\\b|^)yep(\\b|$)|(that('?s| is)
)?(?<!not\\ )(?<!no\\ )(right|correct|confirm)))(\\s)?$"

rule thanks(): "(?=.*(^(\\ )*)(((great|good|awesome|nice|cool)\\ )?((that(')?s|\\ (is|was))\\ (very\\
)?helpful)\\ )?(thank|thank(s|\\ you)(\\ (very|so)\\ much)?)(\\ (that(')?s|\\ (is|was))\\ (very\\
)?helpful))?|((great|good|awesome|nice|cool)\\ )?(that(')?s|\\ (is|was))\\ (very\\
)?helpful)|(great|good|awesome|nice|cool))((\\ )*))(?=^(?:(?!bye).)*$).*$"

rule repeat(): "(\\b|^|\\ )(repeat((\\ that )|(\\ it))?)|(say((\\ that )|(\\ it))\\ again)|(again)"

rule reqalts(): "(\\b|^|\\ )((something|anything)\\ else)|(different(\\ one)*)|(another\\
one)|(alternatives*)|(other options*)|((don'*t|do not) (want|like)\\ (that|this)(\\ one)*)"

rule ack(): "{IMPOSSIBLEREGEX()}"

rule bad(): "{IMPOSSIBLEREGEX()}"
```

Constants

- General form regexes that gets repeated a lot

```
# constants

function domain_vocab(): "(lecturer|teacher)"
function IMPOSSIBLEREGEX(): "^\b$"
function WHAT(): "(what(\\')?(s)?|which|does|where)(\\ (its|the))*"
function IT(): "(it\\'*s*|it\\ have|is\\ it\\'*s*|is\\ (the|their))(\\ for)*"
function CYTM(): "(can\\ you\\ tell\\ me\\ (the|it\\'*s|their))"
function CIG(): "(can\\ I\\ get\\ (the|it\\'*s|their))"
```

Synonyms

- Allow for multiple values to map to one slot or one value

```
function slot_synonyms(slot)
    "{IMPOSSIBLEREGEX()}"

    if slot = "name"
        "name"
        "{domain_vocab()}('s)? name"
        "name of the {domain_vocab()}"
    if slot = "department"
        "department(s)?"
        "institute"
    if slot = "office_hours"
        "(office |consultation )?(hour(s)?)"
        "Sprechstunde(n)?"
    if slot = "mail"
        "email"
        "mail"
        "e\-mail"
    if slot = "phone"
        "telephone"
        "phone( number)?$"
        "number"
    if slot = "room"
        "room"
        "(?=.*(room|office))(?=^(?:(?!hours).)*$).*$"
    if slot = "position"
        "(administrative )?position"
```

Synonyms

- Allow for multiple values to map to one slot or one value

```
function slot_synonyms(slot)
    "{IMPOSSIBLEREGEX()}"

    if slot = "name"
        "name"
        "{domain_vocab()}('s)? name"
        "name of the {domain_vocab()}"
    if slot = "department"
        "department(s)?"
        "institute"
    if slot = "office_hours"
        "(office |consultation )?(hour(s)?)"
        "Sprechstunde(n)?"
    if slot = "mail"
        "email"
        "mail"
        "e\-mail"
    if slot = "phone"
        "telephone"
        "phone( number)?$"
        "number"
    if slot = "room"
        "room"
        "(?=.*(room|office))(?=^(?:(?!hours).)*$).*$"
    if slot = "position"
        "(administrative )?position"
```

```
# synonyms

function synonyms(slot, value)
    "{value}"

    add_if slot = "department"
        if value = "phonetics"
            "phonetic(s)?"
            "speech"
        if value = "theory"
            "theoretical"
            "statistical"
            "statistics"
        if value = "foundations"
            "foundation(s)?"
            "fundamental"
        if value = "external"
            "(not at|outside) (the )?ims"
            "informatic(s)?"
            "(computer)? science"
            "\\bcs$"
            "informatik"
            "linguistic"
            "linguistik"
```

Request Regexes

- Map intent/slot combinations to regexes

```
rule request(department)
    "{rREQUEST()} {slot_synonyms("department")}"
    "(?<!{DONTCAREWHAT()})(?<!want ){IT()} {slot_synonyms("department")}"
    "(?<!{DONTCARE()}){WHAT()} {slot_synonyms("department")}"
    "{WANT()}.*{slot_synonyms("department")}"
    "(the )?{slot_synonyms("department")}"

rule request(mail)
    "{rREQUEST()} {slot_synonyms("mail")}"
    "(?<!{DONTCAREWHAT()})(?<!want ){IT()} {slot_synonyms("mail")}"
    "(?<!{DONTCARE()}){WHAT()} {slot_synonyms("mail")}"
    "{WANT()}.*{slot_synonyms("mail")}"
    "(the )?{slot_synonyms("mail")}"
    "what .*{slot_synonyms("name")}"
```

Inform Regexes

- Map intent / slot combinations to regexes

```
rule inform(name)
    "{rINFORM()} {synonyms("name", name)}"
    "{synonyms("name", name)}{WBG()}"
    "(\\ |^){synonyms("name", name)}(\\ (please|and))*"

rule inform(department)
    # "\\\\b{department}\\\\b"
    "{rINFORM()} {synonyms("department", department)}"
    "{synonyms("department", department)}{WBG()}"
    "(\\ |^){synonyms("department", department)}(\\ (please|and))*"

rule inform(position)
    # "\\\\b{position}\\\\b"
    "{rINFORM()} {synonyms("position", position)}"
    "{synonyms("position", position)}{WBG()}"
    "(\\ |^){synonyms("position", position)}(\\ (please|and))*"
```

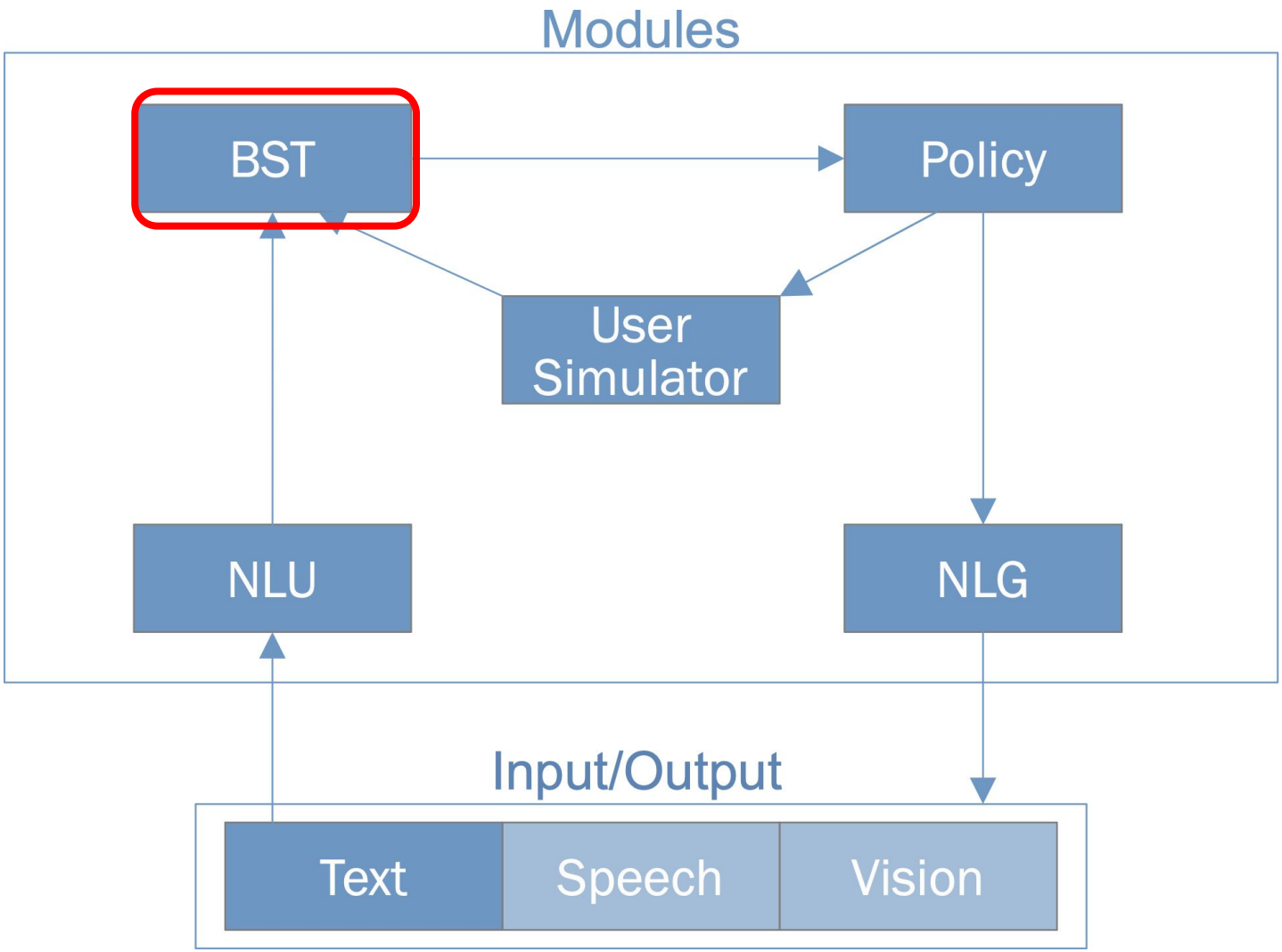
Practice

- Look at the file `resources/regexes/ImsLecturers.nlu`
  - As the code blocks are very repetitive, we recommend using copy/past ;)
- Use this as a basis to define your own `superhero.nlu` file:
  - Synonyms
  - Requests
  - Informs
- Since this is only for the user, ONLY consider user informs/user requests
- After you are done, you can call :

  > `python tools/regextemplates/templatestojson superhero.nlu`

BST

Dialog System Architecture

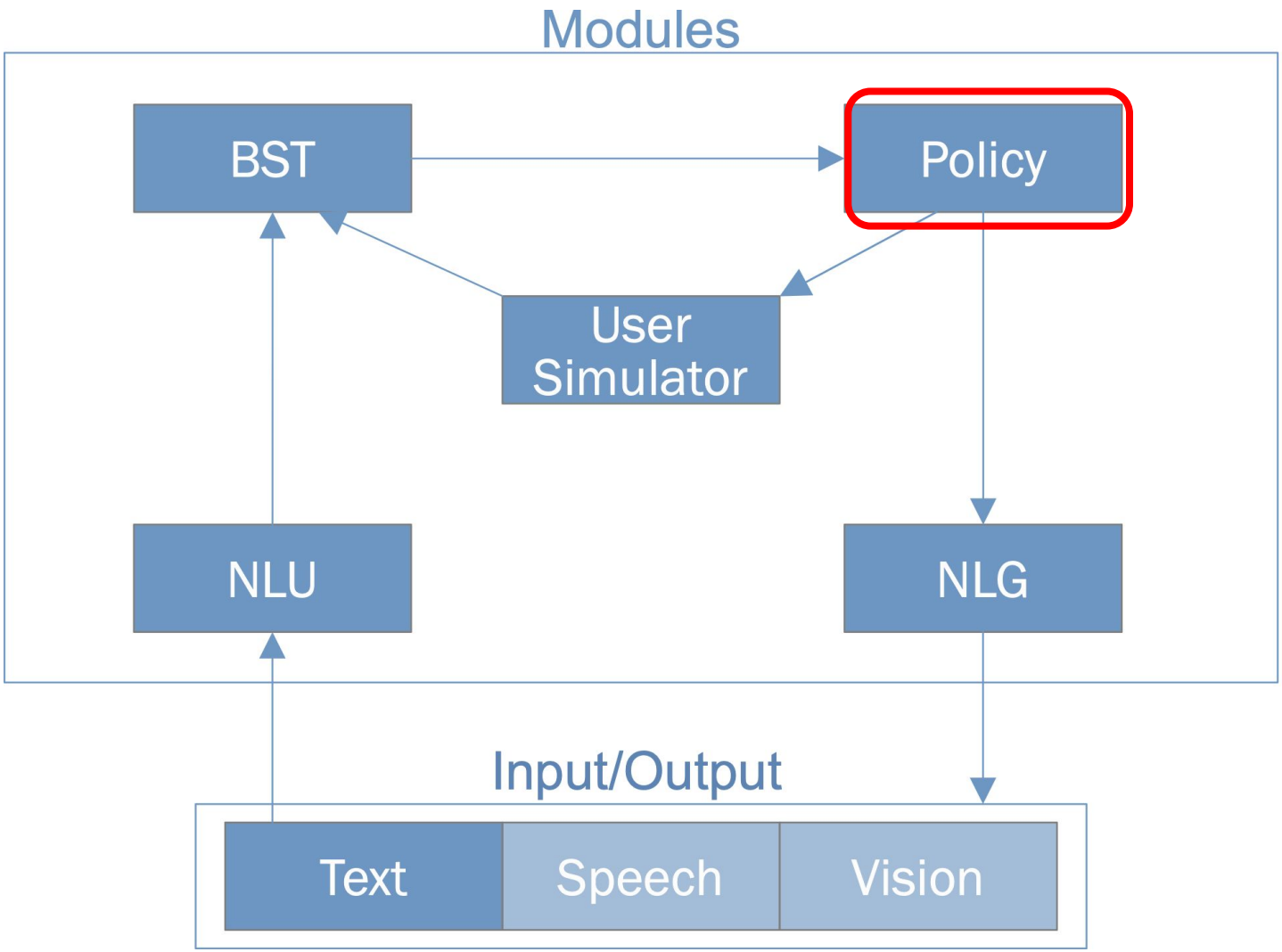


Belief Tracker

- The Belief Tracker maintains the internal dialog state representation
- The rule-based implementation is domain independent
- All domain-specific information is drawn from the ontology
- Updates belief probabilities based only on current user act and last system act

Policy

Dialog System Architecture



Decision Making

- The policy is responsible for choosing the appropriate response to the dialog act that has been decoded.
- Input: Semantic representation of the user utterance (dialog act)

```
inform(semester=winter, program=bachelor)
```

- Output: The response is encoded as system act

```
request(language)
```

Decision Making

- The response is chosen from a set of possible actions (hello(), bye(), request(), inform()...) according to a policy (π)

$$a \in A$$

- The response depends on the semantic input, also called observation because it encodes what the system observes about the user

$$o \in O$$

- The last system response and context (e.g. full dialog history) also play an important role to maintain an internal representation of the full observation sequence, called dialog state or ***belief state***

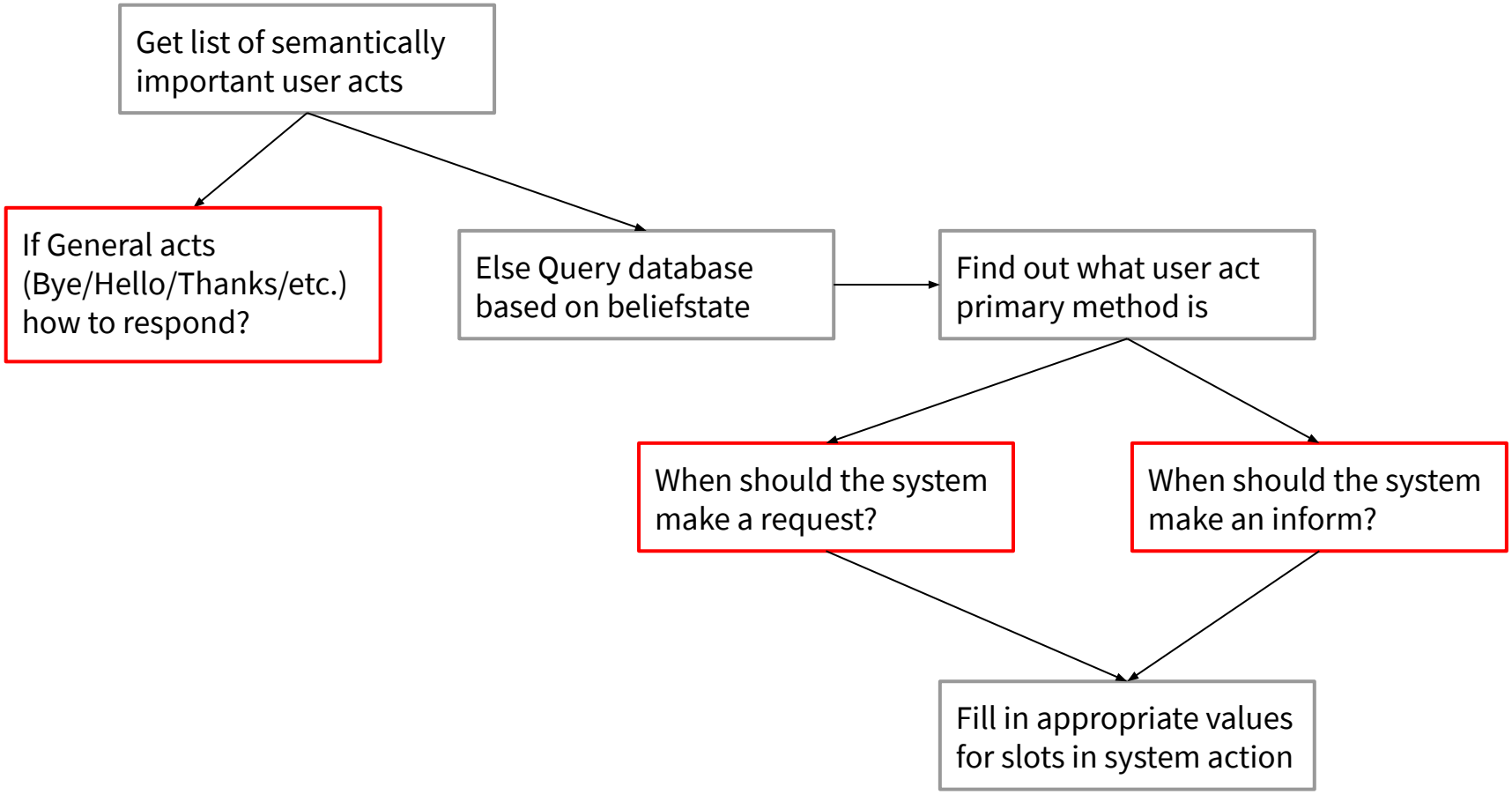
$$b \in B$$

Policy

- The *dialog policy*, π, defines what the system does in each belief state
- π defines a probability distribution over the actions $\Pi(A)$
- π : A mapping function from belief states to actions

$$\pi : B \rightarrow \Pi(A)$$

Handcrafted Policy Flow

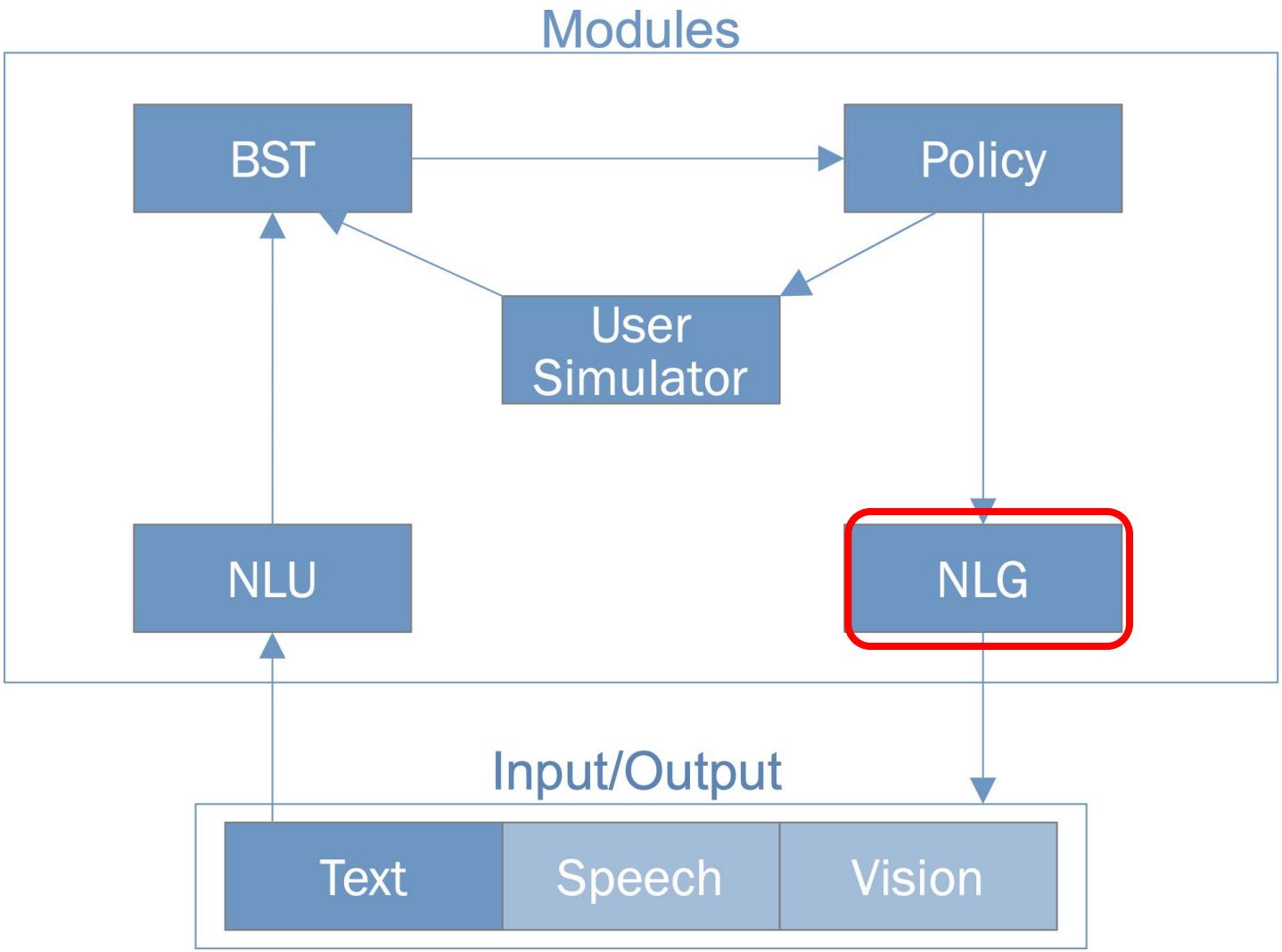


Practice

- Open the file `modules/policy/policy_handcrafted_student.nlu`
- This contains the skeleton of a policy you can use to create your own handcrafted policy
- This class inherits from HandcraftedPolicy which defines several helper functions used inside
- Areas which need to be implemented are marked with a comment box
- As you implement, consider the decision flow your policy should make
    - When to output a general action (and which one)
    - When to output a **request** vs. an **inform**
    - etc.

NLG

Dialog System Architecture



Natural Language Generation (NLG)

NLG transforms a system dialog act into a human-readable sentence

System dialog act → System text output

ADVISER Template Syntax

General outputs

```
template welcomemsg(): "Welcome to the IMS lecturer chat bot. How may I help you?"
template welcomemsg(help): "Sorry, I cannot understand you. Please tell me again what you are looking for."
template welcomemsg(more) : "Can I help you with anything else?"
template bad() : "Sorry I am a bit confused; please tell me again what you are looking for."
# repeat() : "Could you please repeat that?"
template closingmsg() : "Thank you, goodbye."
```

Request(slot)

```
template request(department): "To which chair offered at the IMS shall the lecturer belong?"
template request(position): "Which position does the lecturer hold at the IMS (e.g. study adviser)?"
```

Natural Language Generation (NLG)

NLG transforms a system dialog act into a human-readable sentence

System dialog act → System text output

ADVISER Template Syntax

General outputs

intent

```
template welcomemsg(): "Welcome to the IMS lecturer chat bot. How may I help you?"
template welcomemsg(help): "Sorry, I cannot understand you. Please tell me again what you are looking for."
template welcomemsg(more) : "Can I help you with anything else?"
template bad() : "Sorry I am a bit confused; please tell me again what you are looking for."
# repeat() : "Could you please repeat that?"
template closingmsg() : "Thank you, goodbye."
```

message

Request(slot)

```
template request(department): "To which chair offered at the IMS shall the lecturer belong?"
template request(position): "Which position does the lecturer hold at the IMS (e.g. study adviser)?"
```

Natural Language Generation (NLG)

Inform(slot-value pairs)

inform(name="Capitan America", primary_uniform_color="blue", loyalty="Avengers") →

Capitan America is your superhero! They have a blue uniform and belongs to Avengers . What do you want to know about this hero?

System Inform Templates


```
template inform(name)
        "There is a lecturer named {capitalised(name)}. What do you want to know about {obj_pron(name.gender)}?"

        if name = "none": "I'm sorry, I could not find the lecturer you specified."

template inform(name, *slots)
        "{for_entry(slots, "info", ", ", " and ", name)}."

        special_case name = "none"
                "There is no such lecturer {for_entry(slots, "info", ", ", " and ", "who")}."

template inform_byname(name)
        "There is a lecturer named {capitalised(name)}. What do you want to know about {obj_pron(name.gender)}?"

        if name = "none": "I'm sorry, I could not find the lecturer you specified."

template inform_byname(name, *slots)
        "{for_entry(slots, "info", ", ", " and ", name)}."

        special_case name = "none"
                "There is no such lecturer {for_entry(slots, "info", ", ", " and ", "who")}."
```

System Inform Templates

Helper functions

```
# --------- Helper Functions -------- #

function genitive(name)
        "{genitive_s(name)}"
        if name = "who": "whose"

function obj_pron(gender)
        if gender = "female": "her"
        if gender = "male":   "him"

function capitalised(name)
        "{name.cap_name}"
        if name = "who": "who"
```

System Inform Templates

Functions

```
function info(slot, value, name)
        if slot = "office_hours": "{genitive(capitalised(name))} office hours take place {value}"
        if slot = "mail": "{genitive(capitalised(name))} e-mail address is {value}"
        if slot = "phone": "{genitive(capitalised(name))} phone number is {value}"
        if slot = "room": "{genitive(capitalised(name))} office is in room {value}"

        if slot = "department"
                if value = "na":          "{genitive(capitalised(name))} chair is unknown"
                if value = "phonetics":   "{capitalised(name)} belongs to the chair 'Digital Phonetics'"
                if value = "theory":      "{capitalised(name)} belongs to the chair 'Theoretical Computational Linguistics'"
                if value = "foundations": "{capitalised(name)} belongs to the chair 'Foundations of Computational Linguistics'"
                if value = "external":    "{capitalised(name)} belongs to a chair outside the IMS"
```

Practice

- Look at the file `resources/templates/ImsCourses.nlg`
  - As the code blocks are similar, we recommend using copy/past ;)

- Use this as a basis to define your own `superhero.nlg` file:
  - General
  - Requests
  - Informs

- Since this is only for the system acts, ONLY consider system informs/system requests

References

- Stefan Ultes, Lina M. Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Pawel Budzianowski, Nikola Mrksić, Tsung-Hsien Wen, Milica Gasic, and Steve Young. PyDial: A Multidomain Statistical Dialogue System Toolkit. In Proceedings of ACL 2017, System Demonstrations, pages 73–78, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-4013.
- Blaise Thomson. Statistical methods for spoken dialogue management. University of Cambridge, 2009. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.227.5305&rep=rep1&type=pdf