

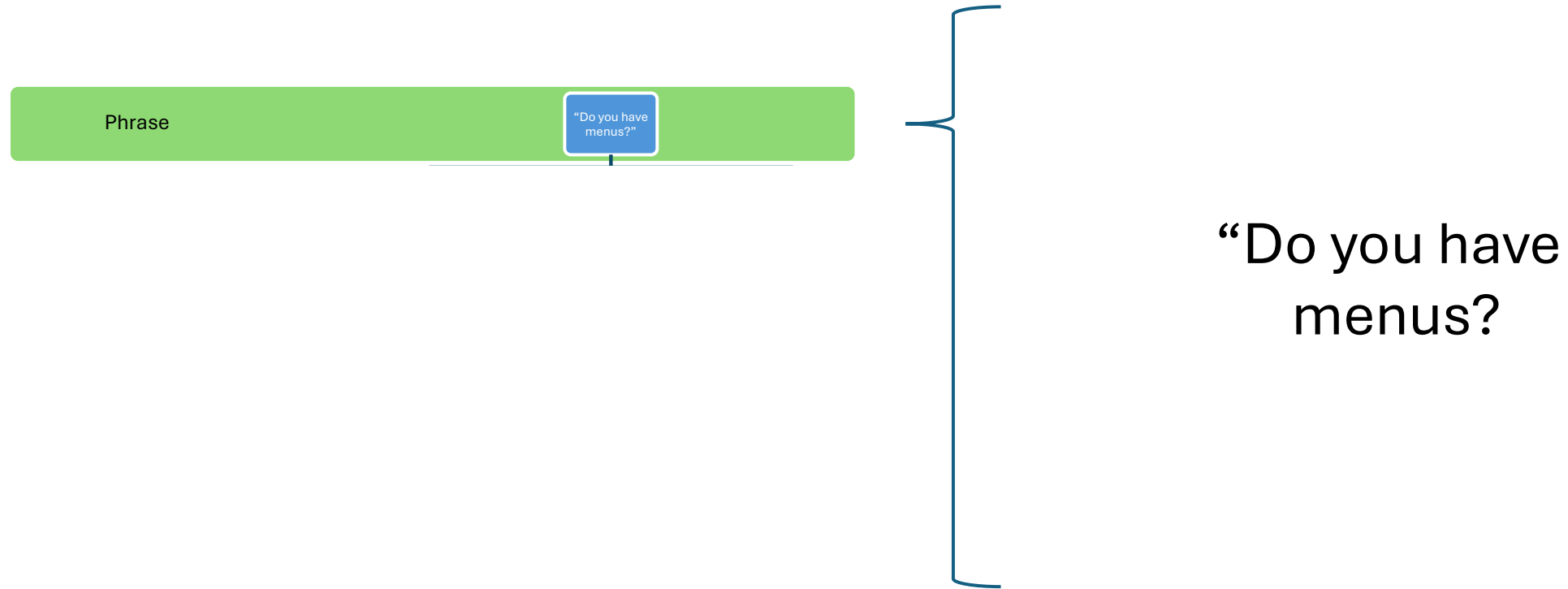
Perplexity Interpretations

Particular and General, Literal and Pragmatic

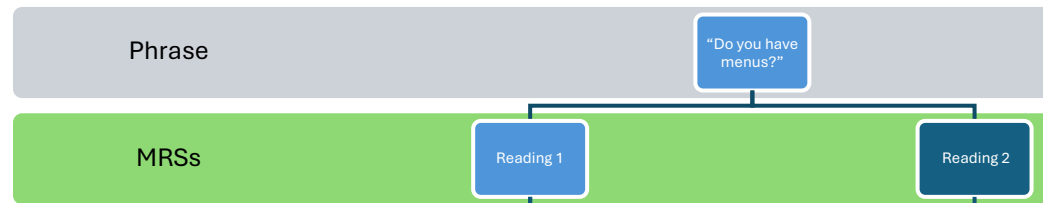
Review: Perplexity Execution Flow

Using the English Restaurant example ...

Review: Perplexity Flow



Review: Perplexity Flow



["Do you have menus?"

TOP: h0

INDEX: e2

[e SF: ques TENSE: pres MOOD: indicative PROG: - PERF: -]

RELS: < [pron<3:6> LBL: h4 ARG0: x3 [x PERS: 2 IND: + PT: std]]

[pronoun_q<3:6> LBL: h5 ARG0: x3 RSTR: h6 BODY: h7]

[_have_v_1<7:11> LBL: h1 ARG0: e2 ARG1: x3 ARG2: x8

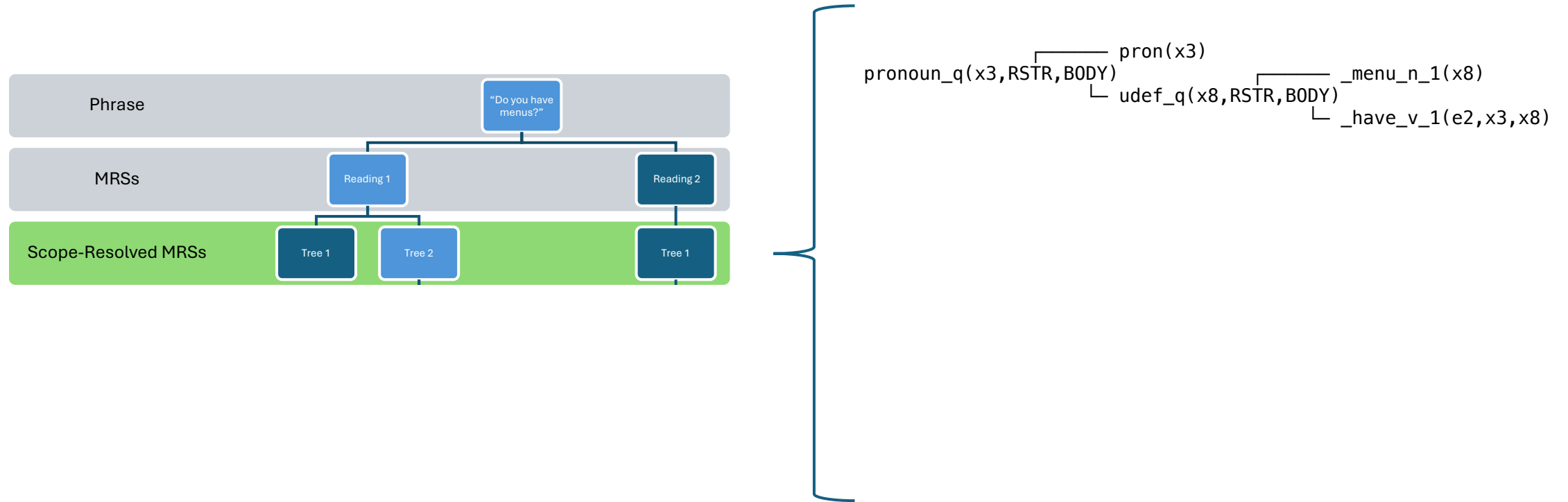
[x PERS: 3 NUM: pl IND: +]]

[udef_q<12:18> LBL: h9 ARG0: x8 RSTR: h10 BODY: h11]

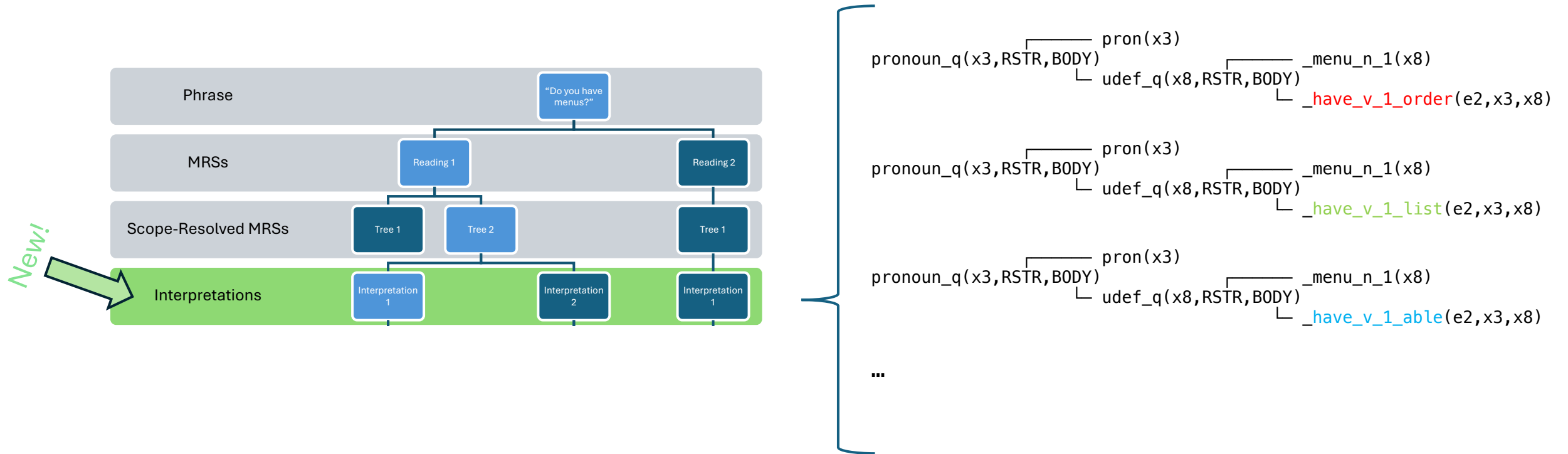
[_menu_n_1<12:17> LBL: h12 ARG0: x8] >

HCONS: < h0 qeq h1 h6 qeq h4 h10 qeq h12 >]

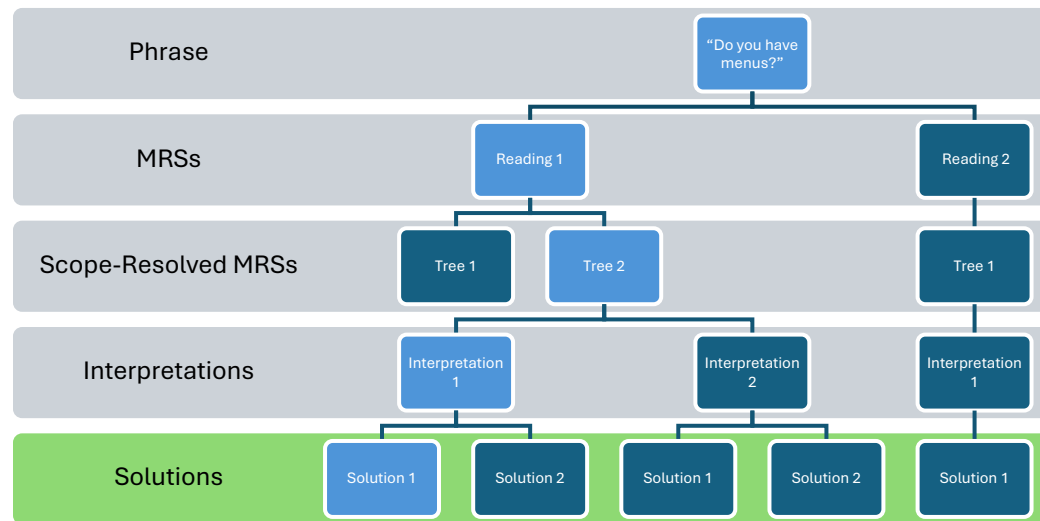
Review: Perplexity Flow



Review: Perplexity Flow



Review: Perplexity Flow



```
pronoun_q(x3,RSTR,BODY)
└─ pron(x3)
   └─ udef_q(x8,RSTR,BODY)
      └─ _menu_n_1(x8)
         └─ _have_v_1_order(e2,x3,x8)
```

Individuals

waiter

menu1

menu2

Facts

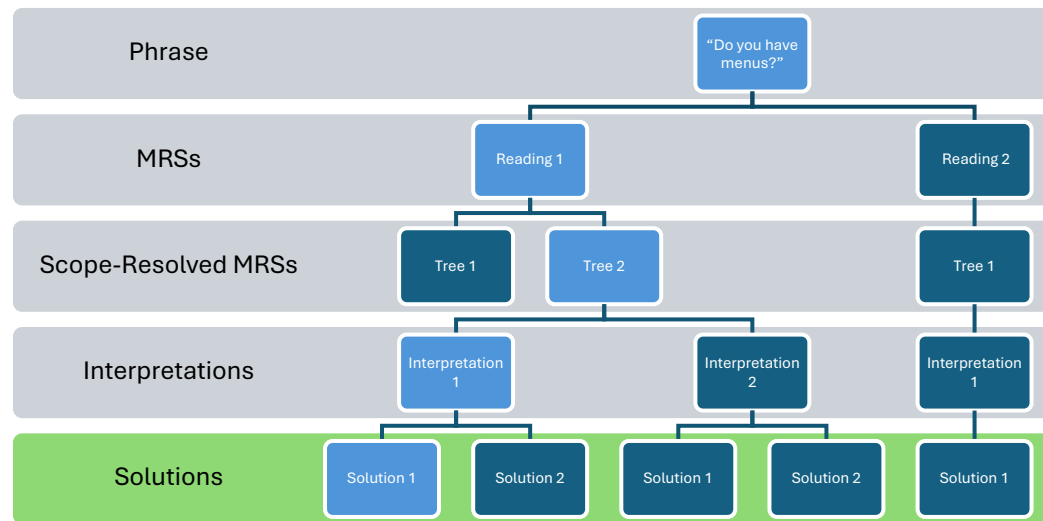
waiter has menu1

waiter has menu2

Solver Input:

- Scope-resolved MRS with a specific interpretation per predication
- World state

Review: Perplexity Flow



pronoun_q(x3,RSTR,BODY) pron(x3)
└─ udef_q(x8,RSTR,BODY) └─ _menu_n_1(x8)
 └─ **_have_v_1_order**(e2,x3,x8)

Individuals

waiter

menu1

menu2

Facts

waiter has menu1

waiter has menu2

Solution 1

x3 [waiter]

x8 [menu1]

Solution 2

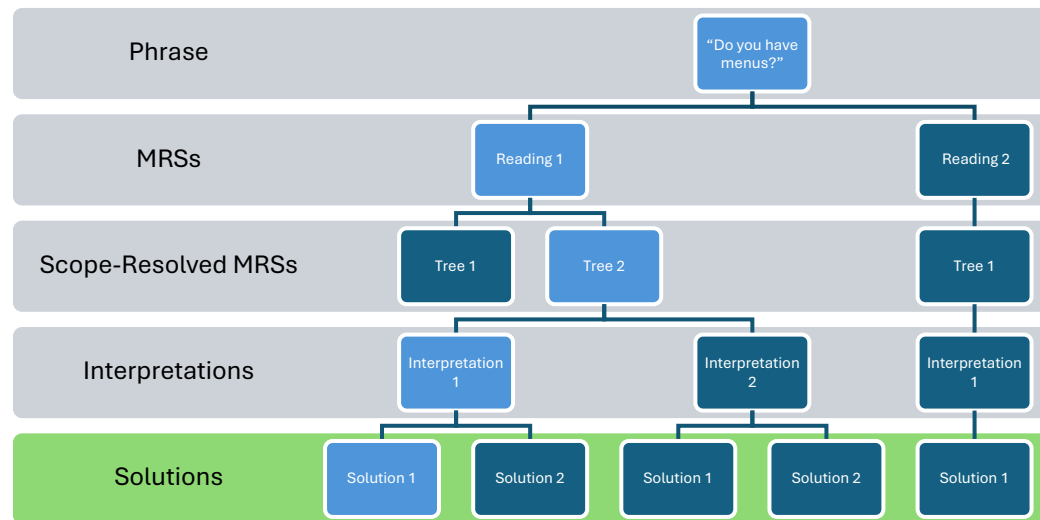
x3 [waiter]

x8 [menu2]

Solver Phase 1:

- Generate all solutions
- Solution: Single assignment of a value to every variable such that every predication succeeds
- Ignore cross-solution criteria, such as plurality
- A solution is independently true but needs to be combined with others to match uttered phrase

Review: Perplexity Flow



$\text{pronoun_q}(x3, \text{RSTR}, \text{BODY})$
 $\begin{matrix} \text{pron}(x3) \\ \text{L} \end{matrix}$
 $\text{udef_q}(x8, \text{RSTR}, \text{BODY})$
 $\begin{matrix} \text{_menu_n_1}(x8) \\ \text{L} \end{matrix}$
 $\text{_have_v_1_order}(e2, x3, x8)$

Individuals

waiter
menu1
menu2

Facts

waiter has menu1
waiter has menu2

Solution Group

Solution 1

x3	[waiter]
x8	[menu1]

Solution 2

x3	[waiter]
x8	[menu2]

Solver Phase 2:

- Group solutions into *Solution Groups*
- Group must meet cross-solution criteria (e.g. plurality) using one of the counting algorithms (collective, distributive, cumulative)

Literal Response

Customer: “Do you have menus?”

Waiter: “Yes!”

Limitations to Current Approach

- *Particular* objects are simple
 - Developer checks truth of facts
 - Solver handles cross-solution criteria (e.g. plurality)
- *General* objects are harder
 - "I want a steak", "Do you have menus?", "I want a well-done ostrich steak"
 - Need to represent something akin to the *referring expression* to manipulate *general object*

Introducing “Concepts”

- Previously: Perplexity only had *particular* objects
 - Individuals like: menu1, menu2
 - General notion of “lunch menu I saw last time” unrepresentable
- Now: *General* objects represented by “Concept” object
 - Represents something akin to a referring expression
 - Opaque to the solver
 - Implementation is app specific
- Solver records (but does not check) Concept cross-solution constraints
 - Developer must check cross-solution constraints based on interpretation
 - E.g. “Do you have menus?”
 - Developer must check if there are enough menus to give the customer
 - Party of 1 only needs 1 menu (even though they said “menus”)

“Do you have [particular or general] menus?”

Create two `_menu_n_1` interpretations:

- `_menu_n_1_instances`: “menu1”, “menu2”, etc.
- `_menu_n_1_concepts`: yields a single object: `Concept(“menu”)`

“Do you have [particular or general] menus?”

Create two `_menu_n_1` interpretations:

- `_menu_n_1_instances`: “menu1”, “menu2”, etc.
- `_menu_n_1_concepts`: yields a single object: `Concept(“menu”)`

Create all the needed `_have_v_1` interpretations:

- `_have_v_1_particular_literal`(e2, x3, x8): “is x3 holding/carrying a menu?”
- `_have_v_1_general_pragmatic` (e2, x3, x8): “give me a menu.”
- etc.

“Do you have [particular or general] menus?”

Create two `_menu_n_1` interpretations:

- `_menu_n_1_instances`: “menu1”, “menu2”, etc.
- `_menu_n_1_concepts`: yields a single object: `Concept(“menu”)`

Create all the needed `_have_v_1` interpretations:

- `_have_v_1_particular_literal`(e2, x3, x8): “is x3 holding/carrying a menu?”
- `_have_v_1_general_pragmatic`(e2, x3, x8): “give me a menu.”
- etc.

Because it doesn’t know which interpretation is meant, the solver tries all combinations:

- ~~`pronoun_q(x3,pron(x3),undef_q(x8,_menu_n_1_concepts(x8),_have_v_1_particular_literal(e2,x3,x8)))`~~
- `pronoun_q(x3,pron(x3),undef_q(x8,_menu_n_1_instances(x8),_have_v_1_particular_literal(e2,x3,x8)))`
- `pronoun_q(x3,pron(x3),undef_q(x8,_menu_n_1_concepts(x8),_have_v_1_general_pragmatic(e2,x3,x8)))`
- ~~`pronoun_q(x3,pron(x3),undef_q(x8,_menu_n_1_instances(x8),_have_v_1_general_pragmatic(e2,x3,x8)))`~~

Which runs first is context dependent.

First success wins!

[Demo] “Having Menus” Interpretations

`_have_v_1(e2, x3, x8)`

The solver generates all combinations of values for x3 and x8:

x3	x8
Concept(you)	Concept(menu)
Concept(you)	menu1
Person1	Concept(menu)
Person1	menu1

Every combination is tried with each interpretation of `_have_v_1` (in an order based on context):

Interpretation	Phrase
<code>_have_v_1_order</code>	I will have a [general] menu
<code>_have_v_1_request_order</code>	Do you have a [general] menu?
<code>_have_v_1_request_order</code> (special case)	Do you have any [general] vegetarian items?
<code>_have_v_1_fact_check</code>	Do I have a [particular] menu?
<code>_have_v_1_able</code>	Can I have a [general] menu?

[Demo] “Having Menus” Interpretations

`_have_v_1(e2, x3, x8)`

The solver generates all combinations of values for x3 and x8:

X3	x8
Concept(you)	Concept(menu)
Concept(you)	menu1
Person1	Concept(menu)
Person1	menu1

Every combination is tried with each interpretation of `_have_v_1` (in an order based on context):

Interpretation	Phrase	Pragmatic Behavior	Literal (for reference)
<code>_have_v_1_order</code>	I will have a [general] menu	“Give me a menu”	“OK! Let me know when.”
<code>_have_v_1_request_order</code>	Do you have a [general] menu?	“Give me (maybe us) a menu”	“Yes”
<code>_have_v_1_request_order</code> (special case)	Do you have any [general] vegetarian items?	“Describe the list of vegetarian dishes”	“Yes”
<code>_have_v_1_fact_check</code>	Do I have a [particular] menu?	“Yes”	“Yes”
<code>_have_v_1_able</code>	Can I have a [general] menu?	“Give me a menu”	“Yes”

Implementing Interpretations

Particular and General, Literal and Pragmatic

The English Restaurant Concept Object

- State stored as triples: (“*object*”, “*relationship*”, “*target*”)
 (“soup”, “on”, “menu”)
 (“salmon”, “isAdj”, “grilled”)
- Concept has criteria: e.g. Concept(“my menu”)
 (object, “is”, “menu”)
 &
 (“person1”, “has”, object)
- Operations:
 - List all individuals that meet criteria
 - Concept entails X
 - E.g. “Brunch Menu” entails “Menu”?
 - Prove it “formally” (in very limited cases)
 - .. or *approximate* using induction:
 exhaustively check if all instances of this concept are also instances of X

The English Restaurant Planner

- Hierarchical Task Network (HTN)
- Well researched, simple planner, expressive enough for restaurant
 - Erol, Kutluhan. *Hierarchical task network planning: formalization, analysis, and implementation*. University of Maryland, College Park, 1995.
 - D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. "SHOP: Simple Hierarchical Ordered Planner." In IJCAI-99, pp. 968-973, 1999. Describes SHOP.
 - Georgievski, Ilche, and Marco Aiello. "An overview of hierarchical task network planning." *arXiv preprint arXiv:1403.7426* (2014).
- <https://blog.inductorsoftware.com/blog/htnoverview>

Anatomy of an Interpretation

_have_v_1_order(e, actor, object)

1. Declare required predicates and properties

Anatomy of an Interpretation

_have_v_1_order(e, actor, object)

1. Declare required predicates and properties

```
@Predication(vocabulary,
              names=["_have_v_1"],
              phrases={"Let's have a menu": {'SF': 'comm', 'TENSE': 'pres',
                                              'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'},
                      "I will have a steak": {'SF': 'prop', 'TENSE': 'fut',
                                              'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}})

def _have_v_1_order(context, state, e_introduced, x_actors, x_objects):
    """
```

Anatomy of an Interpretation

_have_v_1_order(e, actor, object)

2. “Solution handler” evaluates predication appropriateness
 - Actor is one or more customer individuals
 - Object is a Concept that entails something orderable:
Food, Drink, Table, Menu, Bill

Anatomy of an Interpretation

_have_v_1_order(e, actor, object)

2. “Solution handler” evaluates predication appropriateness

- Actor is one or more customer individuals
- Object is a Concept that entails something orderable:
Food, Drink, Table, Menu, Bill

```
@Predication(vocabulary,
              names=["_have_v_1"],
              phrases={"Let's have a menu": {'SF': 'comm', 'TENSE': 'pres',
                                              'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'},
                      "I will have a steak": {'SF': 'prop', 'TENSE': 'fut',
                                              'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}}})

def _have_v_1_order(context, state, e_introduced, x_actors, x_objects):
    valid_requests = [ESLConcept("food"), ESLConcept("drink"), ESLConcept("table"),
                      ESLConcept("menu"), ESLConcept("bill"), ESLConcept("check")]
    if is_user_individual(x_actors) and entails_any(x_objects, valid_requests):
        yield state
```


Anatomy of an Interpretation

_have_v_1_order(e, actor, object)

3. “Solution group handler” checks cross-solution criteria for group of solutions (“Phase 2”), implements behavior

- Criteria check: Make sure the group has as many as they asked for
- Behavior: Planner attempts to give the customers what they asked for if it is available
- Behavior in general: change state or respond to user

Anatomy of an Interpretation

_have_v_1_order(e, actor, object)

3. “Solution group handler” checks cross-solution criteria for group of solutions (“Phase 2”), implements behavior

- Criteria check: Make sure the group has as many as they asked for
- Behavior: Planner attempts to give the customers what they asked for if it is available
- Behavior in general: change state or respond to user

```
@Predication(vocabulary,
              names=["solution_group__have_v_1"],
              handles_interpretation=[_have_v_1_order])
def _have_v_1_order_group(context, state_list, e_group, x_actor_group, x_object_group):
    # All solutions are a customer individual asking to have something orderable
    final_state = do_plan(context,
                          state_list[0],
                          'satisfy_want',
                          x_actor_group,
                          x_object_group,
                          min_from_variable_group(x_object_group)) # Planner checks quantification

    if final_state:
        yield [final_state]
```

`_have_v_1_request_order(e, actor, object)`

Pragmatic ordering via “Do you have ...”

1. Required predicates and properties

"Do you have a table?"	{'SF': 'ques', 'TENSE': 'pres', 'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}
"What do you have?"	{'SF': 'ques', 'TENSE': 'pres', 'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}

2. “Solution handler” evaluates predication appropriateness

- Actor is the waiter individual
- Object is a concept that entails something you can ask for:
Food, Drink, Table, Menu, Bill

3. “Solution group handler” checks cross-solution criteria for group of solutions, implements behavior

- Criteria check: Make sure the group has as many as they asked for
- Ask planner to give the customers what they asked for
 - Special case “What do you have?” to give a menu
 - If all requested objects are menu items, give a menu item
 - If all requested objects are specials describe the specials
 - Otherwise: give what was asked for

`_have_v_1_fact_check(e, actor, object)`

Literal interpretation: Does actor contain/own/carry object?

1. Required predicates and properties

"We have 2 menus"	{'SF': 'prop', 'TENSE': 'pres', 'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}
"Do you have a kitchen?"	{'SF': 'ques', 'TENSE': 'pres', 'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}

2. “Solution handler” evaluates predication appropriateness

- Both arguments must be individuals
- Actor must have a “have” relationship with Object

3. “Solution group handler” checks cross-solution criteria for group of solutions, implements behavior

- Requires individuals so system can check cross-solution criteria
- Thus, no custom handler needed
- Default solution group handler prints: “This is correct” or “That isn’t true”

`_have_v_1_able(e, actor, object)`

Literal and pragmatic answers to “Can x have y?”

1. Required predicates and properties

“Could I have a steak?”	{'SF': 'ques', 'TENSE': 'tensed', 'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}
“Can I have a steak?”	{'SF': 'ques', 'TENSE': 'pres', 'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}
“Can the salad have nuts?”	{'SF': 'ques', 'TENSE': 'pres', 'MOOD': 'indicative', 'PROG': '-', 'PERF': '-'}

2. “Solution handler” evaluates predication appropriateness

- If Actor is a customer: object must be something they can ask for:
Food, Drink, Table, Menu, Bill
- Otherwise: True if any instance of Actor “has” Object because that proves it is “able to”
e.g. “Can a person have an arm?”

3. “Solution group handler” checks cross-solution criteria for group of solutions , implements behavior

- If this is a non-wh question, with Actor = customers, then: it is also a request for something
 - Ask planner to give the customers what they asked for
- Otherwise: literal interpretation: “yes that is true”, etc.

Summary: Interpretations in Perplexity

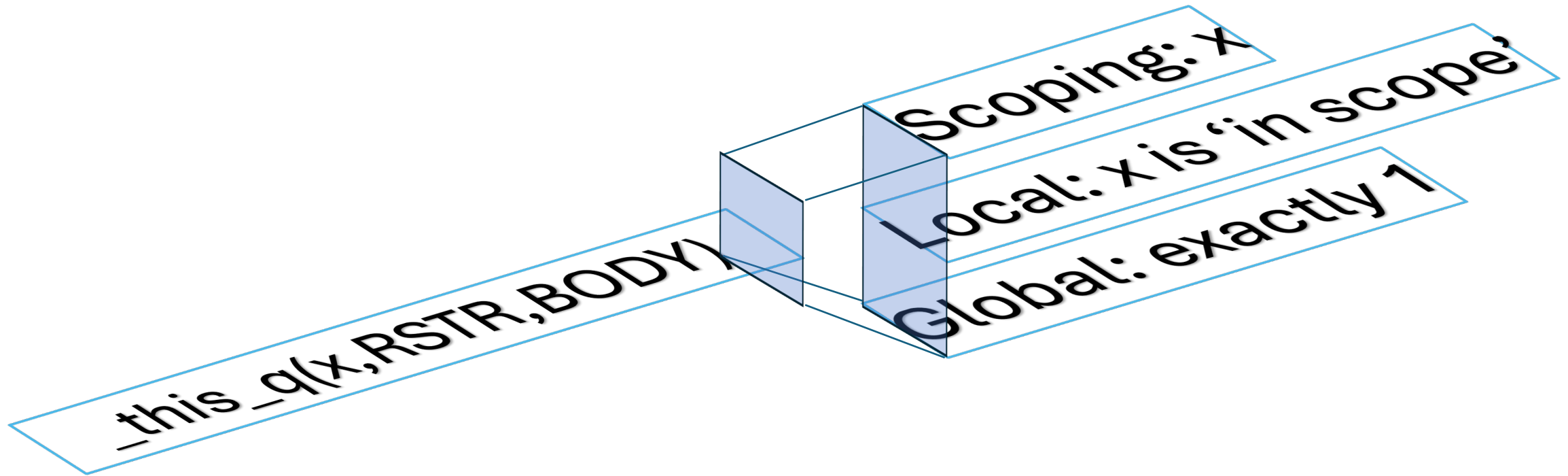
- Support “general” vs. “particular” via the Concept Object
- Support literal and pragmatic interpretations via multiple handlers
- Handlers
 - Phase 1: Solution handlers evaluates predication appropriateness
 - Declarations indicate required properties and predicates
 - Phase 2: Solution Group handlers checks cross-solution criteria for group of solutions and implement behavior

Appendices

Learnings

- The general public thinks natural language interfaces are everywhere and solved
- People have a hard time talking “naturally” to a computer
 - Especially when typing
 - Especially when they are addressing the computer as a computer
- Natural language can be a drag for frequent/complicated tasks
 - Try pretending to use natural language for something on your file system
- People often group communication into multiple sentences at a time, at least when they are addressing another person
 - “Thanks for the menu! I’ll take the salad. My son will take the salmon”

Predication “Semantic Layers”



* Properties like “NUM” can add global restrictions too ...

Predication “Semantic Layers”

Predications can contribute scoping, local or global constraints

Predicate	Scoping	Local	Global
_large_a_1(e,x)	<none>	True for “large” x	<none>
undef_q(x,RSTR,BODY)	x	<none>	<none>
_a_q(x,RSTR,BODY)	x	<none>	Exactly 1
_the_q(x,RSTR,BODY)* *... one of several meanings	x	<none>	1 or more Where all rstr satisfy the body
_this_q(x,RSTR,BODY)	x	True if x is “in scope”* *... among other meanings	Exactly 1
card(CARG,e,x)	<none>	<none>	At least CARG
a_few_a_1(e,x)	<none>	<none>	Between 3 and 5* *top value is debatable
and_c(x,x1,x2)	<none>	<none>	Exactly N where N is total of x1 and x2