SC22 Dallas, TX | hpc accelerates.

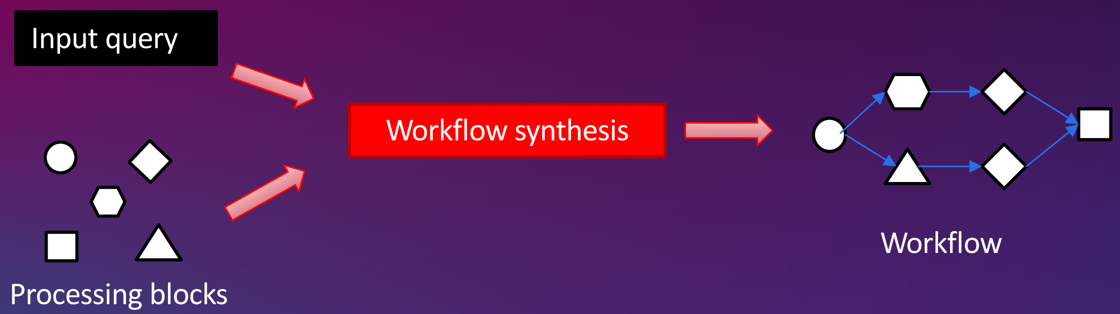
Interactive NLU-Powered Ontology-Based Workflow Synthesis for FAIR Support of HPC

Zifan Nan[1], Mithil Dave[1], Xipeng Shen[1], Chunhua Liao[2], Tristan Vanderbruggen[2], Pei-Hung Lin[2], Murali Emani[3]

[1]North Carolina State University
[2]Lawrence Livermore National Laboratory
[3]Argonne National Laboratory

Workflow Synthesis

- Automatically assembles processing blocks (e.g., scripts, APIs) into a workflow
- The execution of workflow produces results that meet users' input intention
- Benefits: Productivity, automating services



Example in Bioinformatics

Use case	Workflow input	Workflow output	Workflow constraints
No. 1	Mass spectr. spectra in Thermo RAW	Amino acid index (hydropathy) in any format	(i) Use peptide identification; (ii) Use validation of peptide-spectrum matches; (iii) Use retention time prediction; (iv) Do not use protein identification

Synthesized Workflows by PROPHETS

- msconvert → Comet → PeptideProphet → rt4
- msconvert → Comet → PeptideProphet → xml2tsv → SSRCalc
- msconvert → X! Tandem → Tandem2XML → PeptideProphet → rt4
- msconvert → X! Tandem → Tandem2XML → PeptideProphet → xml2tsv → SSRCalc

"Automated workflow composition in mass spectrometry-based proteomics", Bioinformatics, 35(4), 2019, 656–664.

Existing Approaches & Limitations

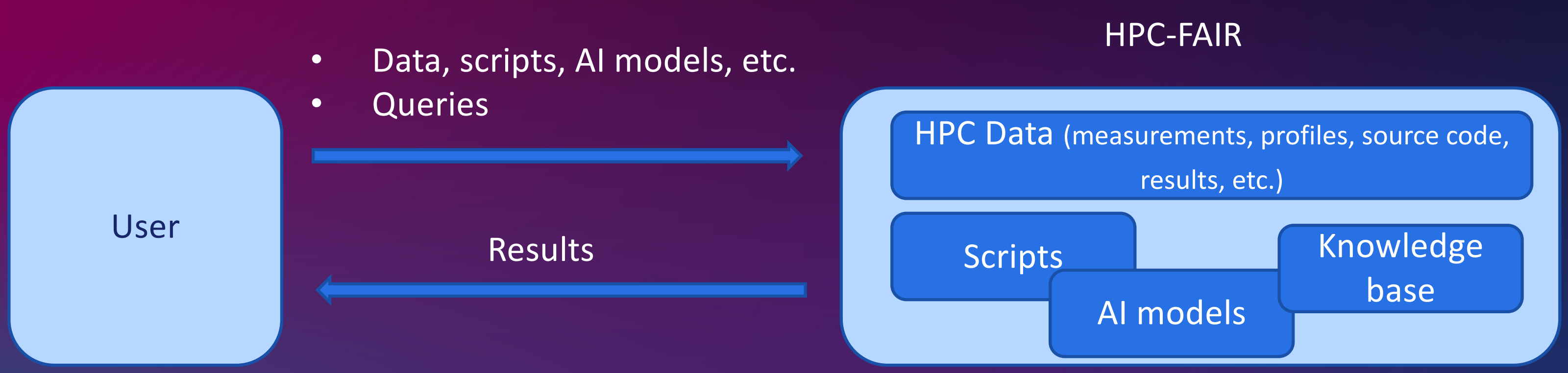
- **Examples**
  - Semantic service composition approaches in *myGrid*
  - OWL-based SADI framework with its SHARE client for web service pipelining
  - The PROPHETS framework that makes use of temporal-logic synthesis
- **Limitations**
  - Input: Queries are restricted in format and vocabulary
  - Domain representation: Rich, consistent annotations in precise terms
  - Domain stableness: Stable domains with a predefined set of concepts and entities.

How to enable workflow synthesis for open domains?

HPC-FAIR http://hpcfair.org/

An open platform for FAIR data support in HPC

- **FAIR: Findability, Accessibility, Interoperability, Reusability**



Features of HPC-FAIR

- A continuously changing domain
- Diverse, boundless needs from users

  The answers often require processing
  of multiple existing datasets

Implications

- Automatic workflow synthesis is desired
- Must allow flexible input queries
- Must handle continuously changing domains

Q: Please get the memory performance data of QCD measured on Pascal and Volta GPUs.

Q: Get the prediction accuracies of the AI models on ImageNet and their inference speed on GPUs; please merge the results into one csv file if possible.

Q: Collect the source code of the nested loops written in C language along with their AST stored in LLVM format.

Solution: INPOWS

Interactive natural language understanding (NLU)-powered ontology-based workflow synthesis.

Superior extensibility

- Adopt **HISyn**, a code synthesizer powered by Natural Language Understanding.
- Only requires semantic and syntax of target domain.
- Easily extensible

Flexibility

- Allow natural language as input quires.
- Offering flexibility to users.

Handling concepts

- Seamless integration with Ontology
- Bridge the gaps between the concepts used in a query and the concepts in datasets.

Handling NL ambiguities

- Interactive design
- Popping up hints and choices when a user inputs her query.
- Helps clarify the intent of the user and simplifies the synthesis.

Background on HISyn

- HISyn: Human Learning-Inspired Code Synthesizer for Natural Language

I would like to find the cheapest flight from Baltimore to Atlanta.

```
MIN_Fare(
    COL_FARE(),
    AtomicRowPredSet(
        AtomicRowPred(
            EQ_DEPARTS(CITY(baltimore), ANY(), ANY(), ANY(), ANY()),
            EQ_ARRIVES(CITY(atlanta), ANY(), ANY(), ANY(), ANY()))))
```

[FSE'2020] *"HISyn: Human Learning-Inspired Natural Language Programming", 2020.*

Background on HISyn

Code Synthesizer from Natural Language Queries

- HISyn: Human Learning-Inspired Code Synthesizer
  - 80% accuracy
  - 0 training data

Learning from the documentation and follow the grammar.

- CodeX-12B (by OpenAI)
  - 28.8% accuracy
  - 54M code repositories => 159GB training data

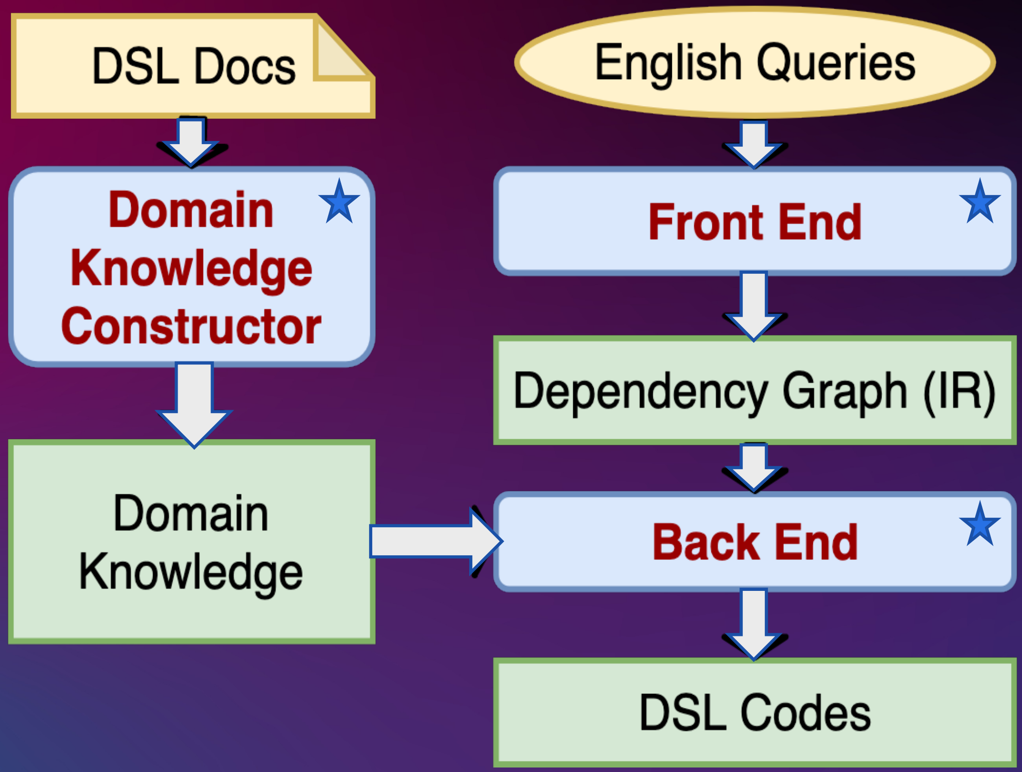
Purely driven by code examples.

Background on HISyn

- Document-based
- Grammar-guided
- NLU-driven (vs example-driven)

Background on HISyn

Understanding the API documents



Understanding the user query

Synthesize the code

Challenges for Workflow Synthesis of HPC-FAIR

1. Formal representation of workflows

- HISyn uses DSL as the abstraction of the set of possible expressions in a target domain.
- A grammar and API documentation are needed to define the DSL of HPC-workflow domain.

GetColumn(columnName(string(“flops”,“frequency”)),datasetName(string(“CPUTrace.csv”)))

2. Treat two entwined search spaces

- One free NL query may ask information about both DSL and Ontology. i.e., two search space.
- How to identify the corresponding search space of the key information inside the NL query?

Get CPU related columns from dataset “CPUTrace.csv”

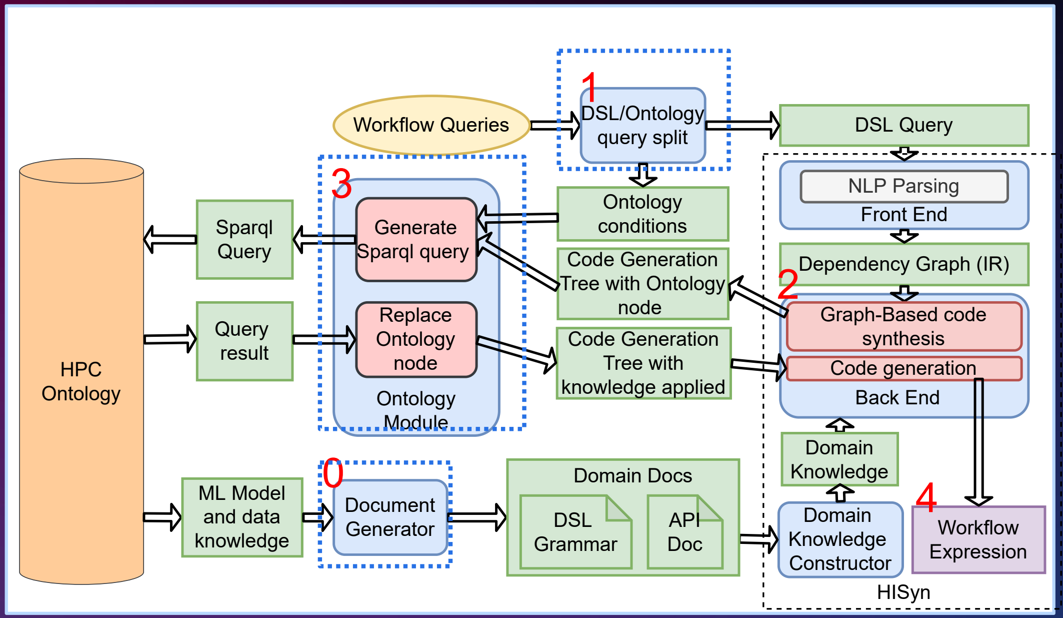
3. Fill knowledge gaps in NL queries through Ontology

- SPARQL query language is usually used in searching for information inside an Ontology.
- How to generate appropriate SPARQL queries from the input queries expressed in NL?

```
SELECT ?colName
  WHERE{?dataset rdf:type hpc:Dataset.
    ?dataset hpc:name "CPUTrace.csv".
    ?dataset hpc:hasColumn ?var.
    ?var hpc:colName ?colName.
    ?var hpc:hasProperty ?colTag.
    {SELECT ?colTag WHERE {
       ?props schema:domainIncludes hpc:cpu.
       ?colTag rdfs:subPropertyOf* ?props}}}
```

Overview of INPOWS

- INPOWS uses 3 new modules to resolve the challenges:
  - **Document generator**
  - **DSL/Ontology query split**
  - **Ontology Module**

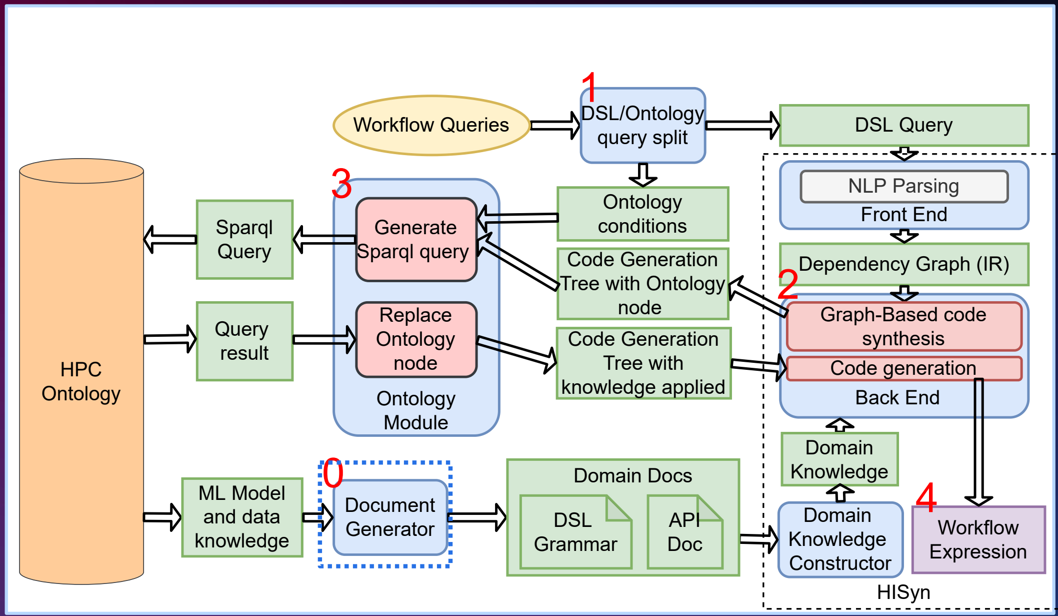


INPOWS - Interactive NLU-powered ontology-based workflow synthesis

Overview of INPOWS

1. **Document generator**

- Read API, script and data information from Ontology
- Generate the grammar and API documentation for the DSL of HPC-workflow domain.

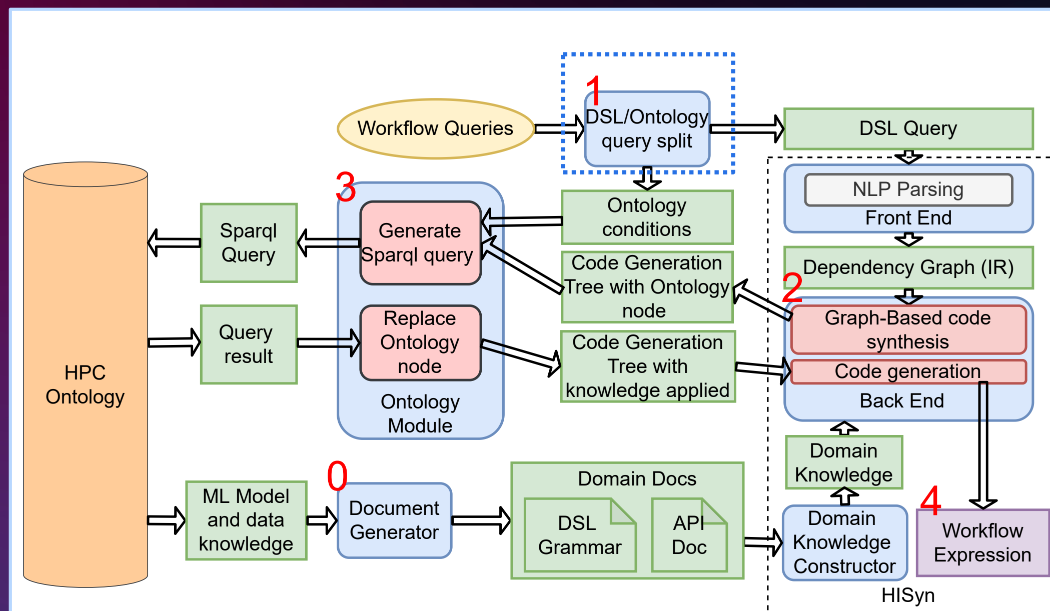


Framework of INPOWS

Overview of INPOWS

2. DSL/Ontology query split

- **Identify and replace** the words and phrases related to information in Ontology
- Split the original NL query into **DSL query** (only related with DSL APIs) and **Ontology conditions** (only related with Ontology)

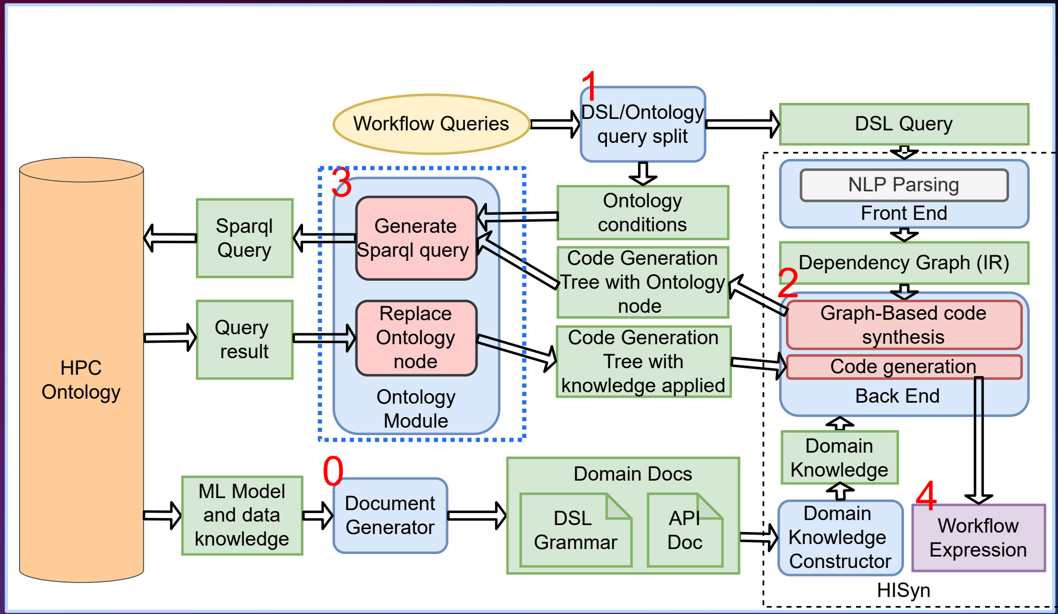


Framework of INPOWS

Overview of INPOWS

3. Ontology Module

- Get information from both Ontology conditions and code generation tree.
- Create the corresponding SPARQL query and get the results.
- Add the results into the code generation tree.



Framework of INPOWS

Workflow query and expression examples

- Data manipulation

- Query: Merge dataset "X.csv" and dataset "Y.json"
- Workflow expr: *MergeDataset(csvFile(string("X.csv")), json2csv(jsonFile(string("Y.json"))))*

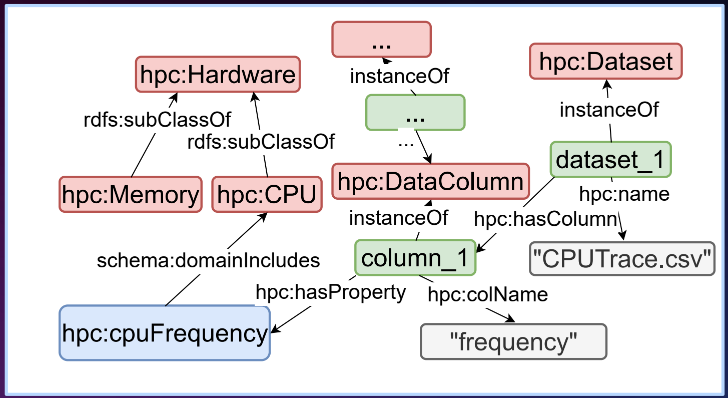
- Ontology query

- Query: Get datasets whose subject is "GPGPU"
- Workflow expr: *GetDataset(datasetName(string("lassen_overhead, performance_results_dataset")))*

- Combination

- Query: Get CPU related columns from dataset "CPUTrace.csv"
- Workflow expr: *GetColumn(columnName(string("flops","frequency")),datasetName(string("CPUTrace.csv")))*

Running Example



Example information stored in HPC Ontology

Document generation

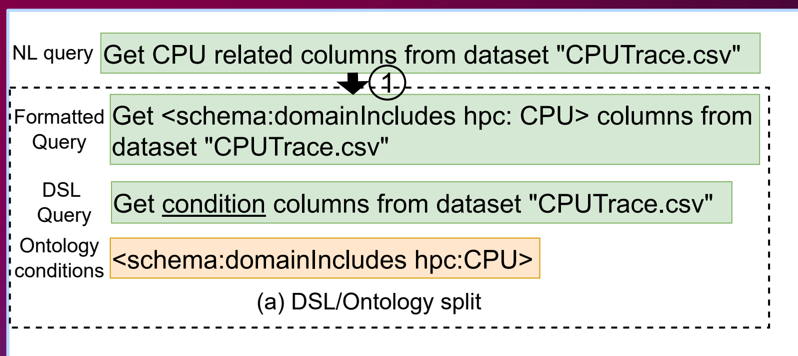
1. **Data manipulation APIs**
   - APIs that handle the data processing tasks.
   - Such as fetching data, transferring file type.
   - E.g., *json2csv(_jsonFile)*
2. **Project APIs**
   - Transformed from user uploaded scripts.
   - Named with script name and project name.
   - E.g., logsToDataset_XPlacer(_nsight_log)
3. **Ontology APIs**
   - mapped with the Ontology related content
   - serves as a placeholder inside the code generation trees (CGT)

```
_workflow := _string | _list | _array | _file |_number | ...
_string := string(STRING) | GetHardware(_hwName) | ...
_array := GetColumn(_columnName,_datasetName)                1
          | MergeDataset(_csvFile, _csvFile) | …
_csvFile := json2csv(_jsonFile) | csvFile(_string)
          | logsToDataset_XPlacer(_nsight_log) |...          2
_datasetName := datasetName(_ontology_arg)
_columnName := columnName(_ontology_arg)
_hwName := hardwareName(_ontology_arg)
_ontology_arg:= _string | _ontology
_ontology := ontology()                                      3
...
```

API: MergeDataset *description*: Merge data in two csv files	*API*: datasetName *description*: matches dataset names *ontology_class*: hpc:dataset *ontology_subject*: hpc:name

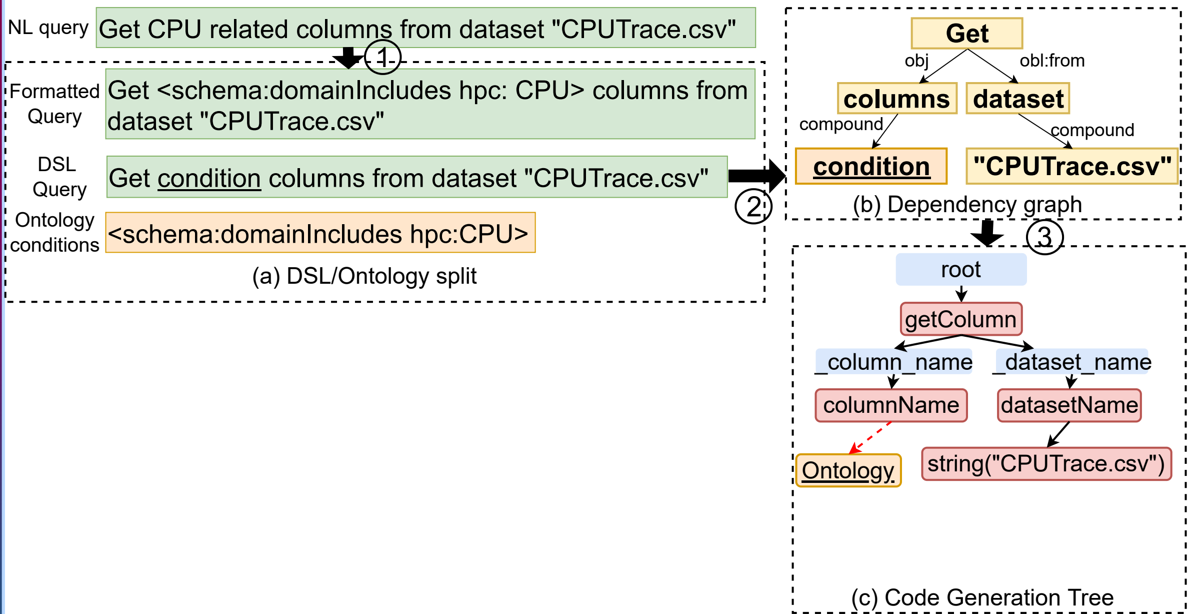
Workflow DSL Grammar

Workflow synthesis running example



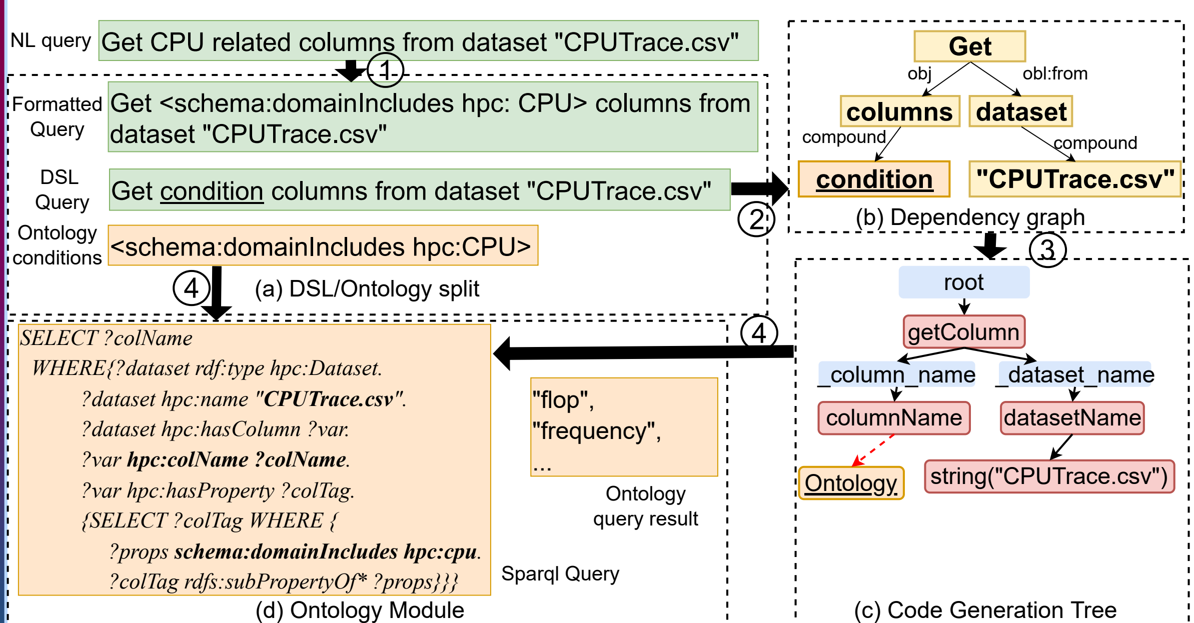
1 Transform the NL query into formatted query, and separate it into DSL query and Ontology condition.

Workflow synthesis running example



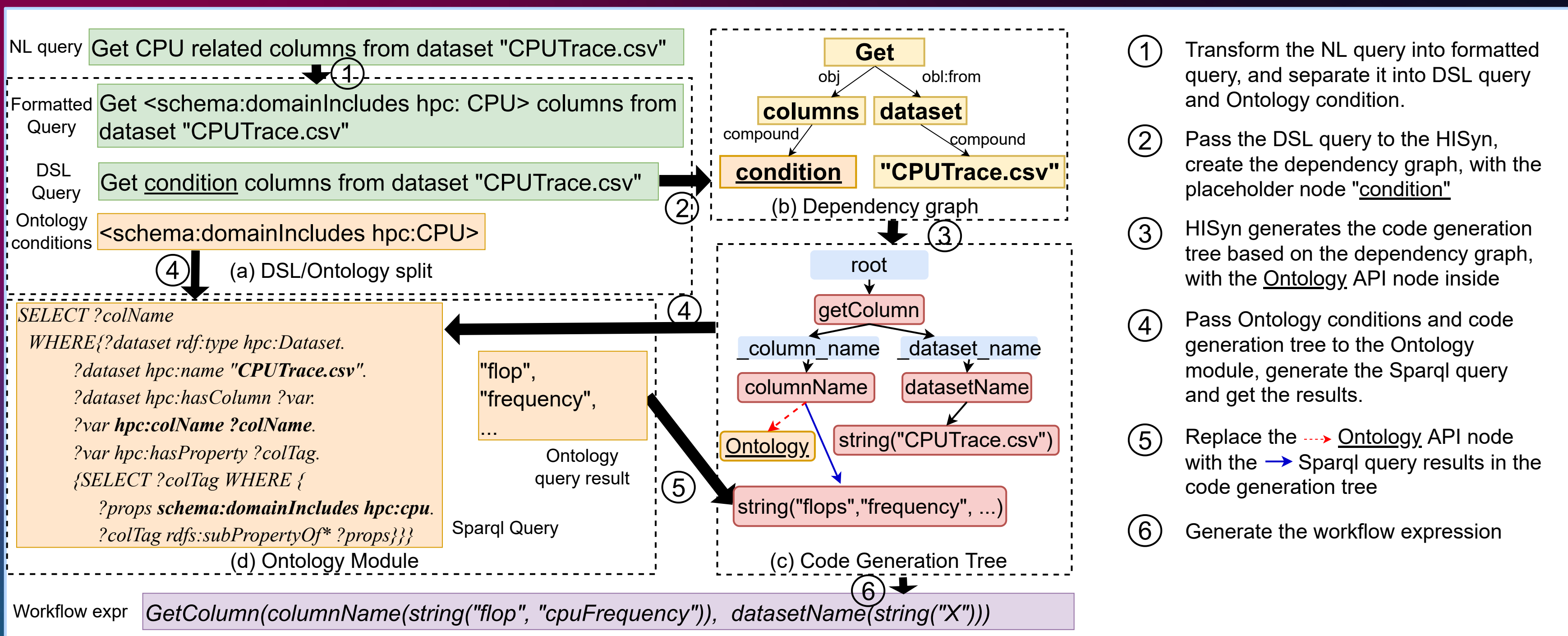
1. Transform the NL query into formatted query, and separate it into DSL query and Ontology condition.
2. Pass the DSL query to the HISyn, create the dependency graph, with the placeholder node "condition"
3. HISyn generates the code generation tree based on the dependency graph, with the Ontology API node inside

Workflow synthesis running example



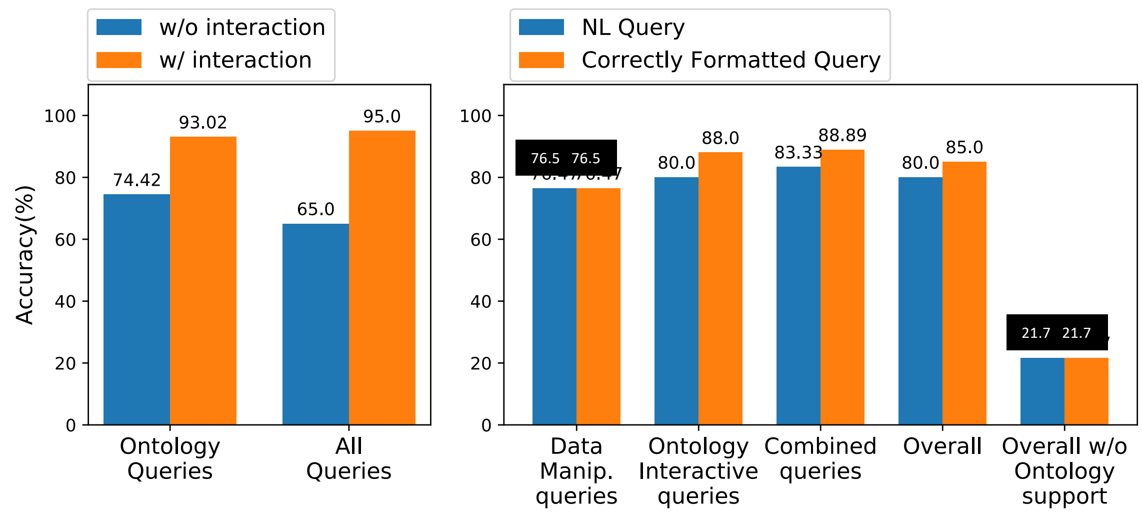
1. Transform the NL query into formatted query, and separate it into DSL query and Ontology condition.
2. Pass the DSL query to the HISyn, create the dependency graph, with the placeholder node "condition"
3. HISyn generates the code generation tree based on the dependency graph, with the Ontology API node inside
4. Pass Ontology conditions and code generation tree to the Ontology module, generate the Sparql query and get the results.

Workflow synthesis running example



Experiment results

- 60 NL queries.
  - 17 data manipulation queries, 25 Ontology queries, 18 combined queries.
- 80% overall accuracy with user interaction.



(a)Split Accuracy (b)Synthesis Accuracy

User interaction is helpful for resolving the NL ambiguity.

Conclusions

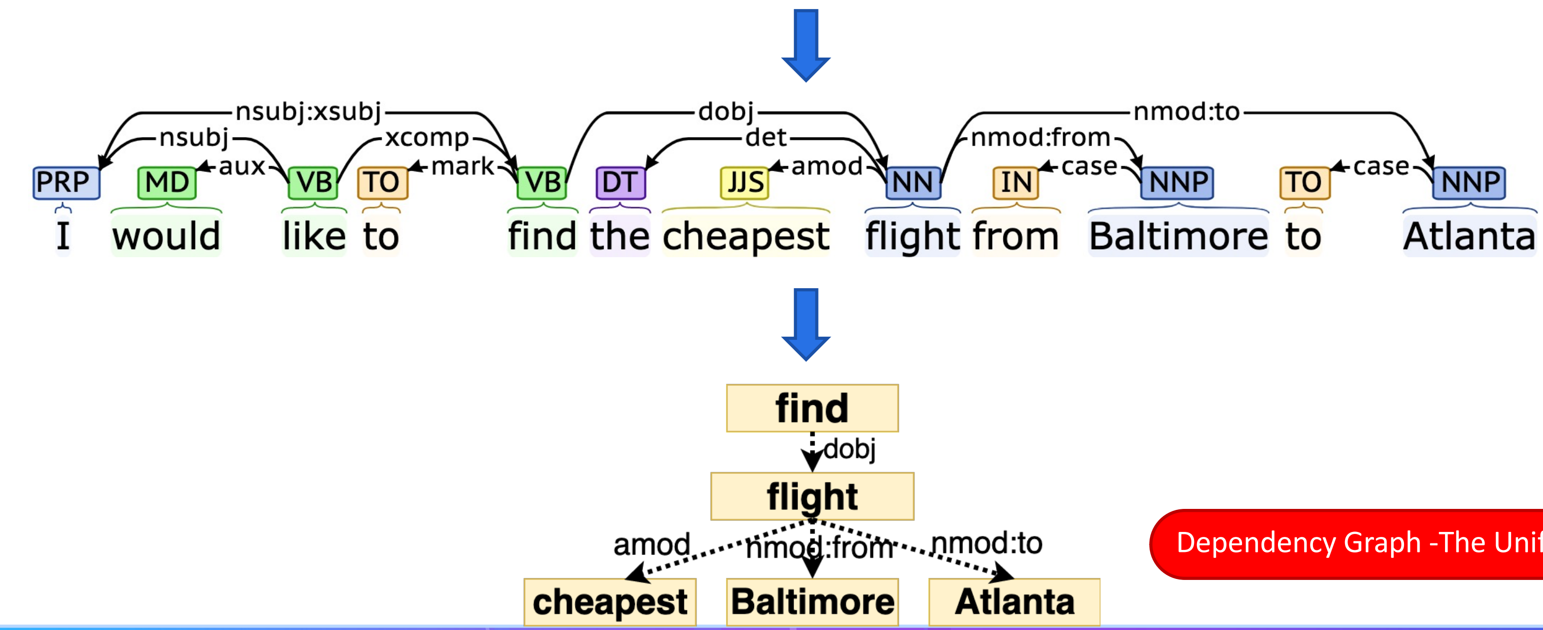
- INPOWS: A new approach to automatic workflow synthesis for open domains
  - HISyn enabled easy domain extensibility without the need of training data
  - Combination with Ontology to deal with hidden relations
  - Interactions are useful for disambiguate NL queries

Thanks!

Questions?

HISyn: Front end

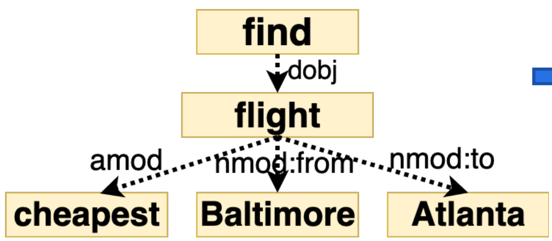
I would like to find the cheapest flight from Baltimore to Atlanta.



Dependency Graph -The Unified IR

HISyn: Semantic Mapping

Find **a list of APIs** that semantically related to the words in dependency nodes.



Dependency graph

flight: ['PredSet', 'MAX', 'MAX_TIME', 'MIN_TIME', '**MIN_FARE**']
cheapest: ['**MIN_FARE**']
from: ['EQ_DPRT']
Baltimore: ['CITY(Baltimore)']
to: ['EQ_ARR']
Atlanta: ['CITY(Atlanta)']

Semantic mapping results

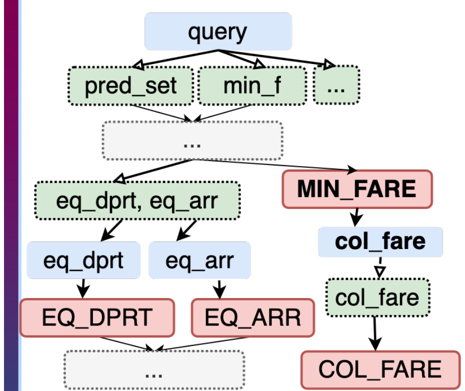
The final code must:

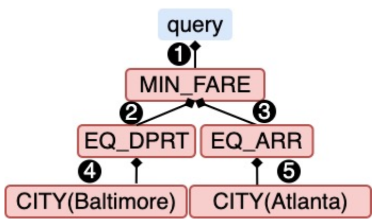
(1) contain one of APIs from each node.

(2) contain the unmapped APIs as less as possible.

HISyn: Reversed all-paths search

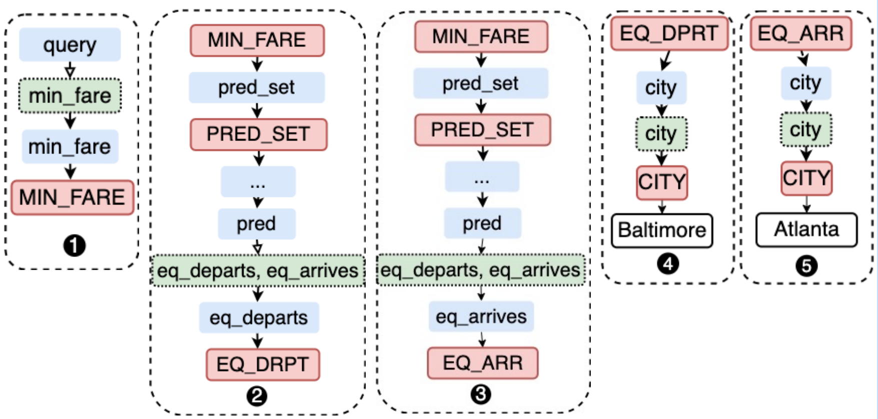
Find a subgraph in grammar graph that:
(1) covers one API nodes from each dependency node. (2) has minimum numbers of API nodes.





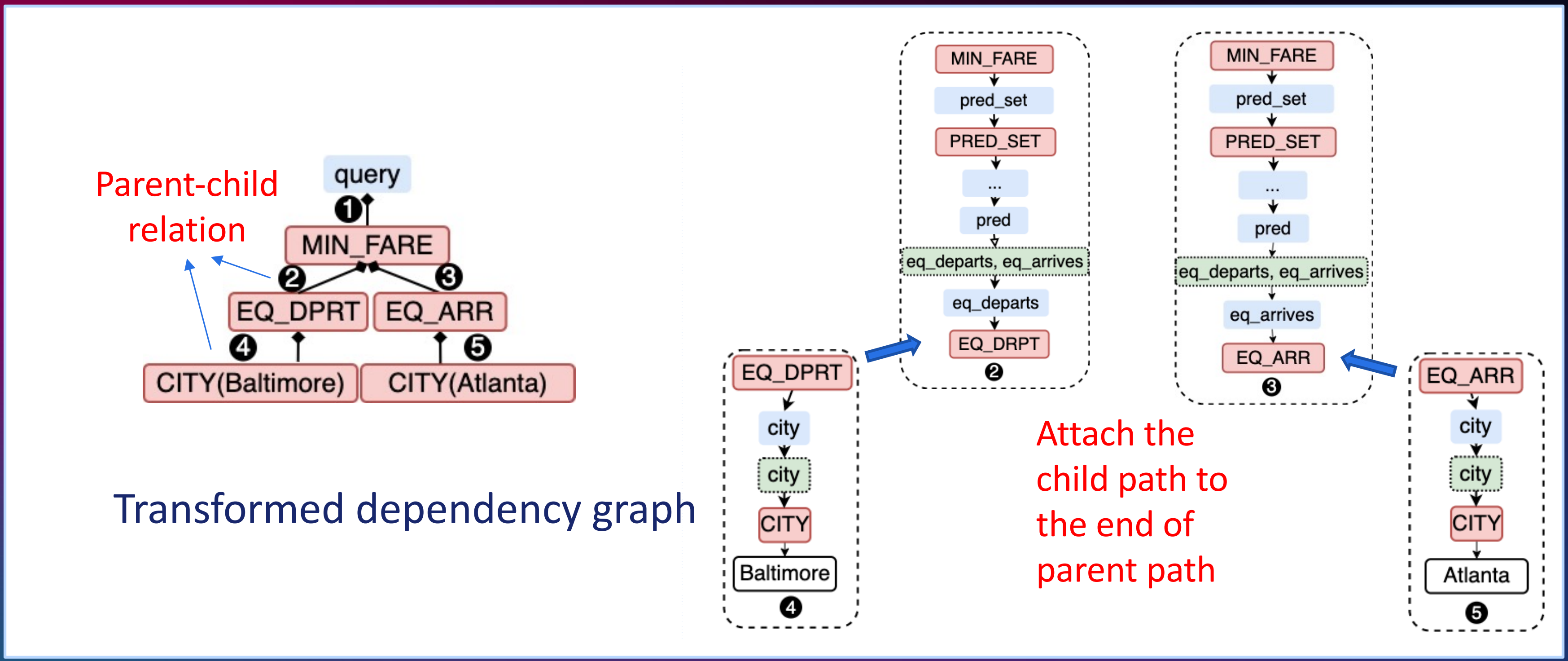
Transformed dependency graph

Find **all the paths** that can connect the mapped APIs in each dependency edge.

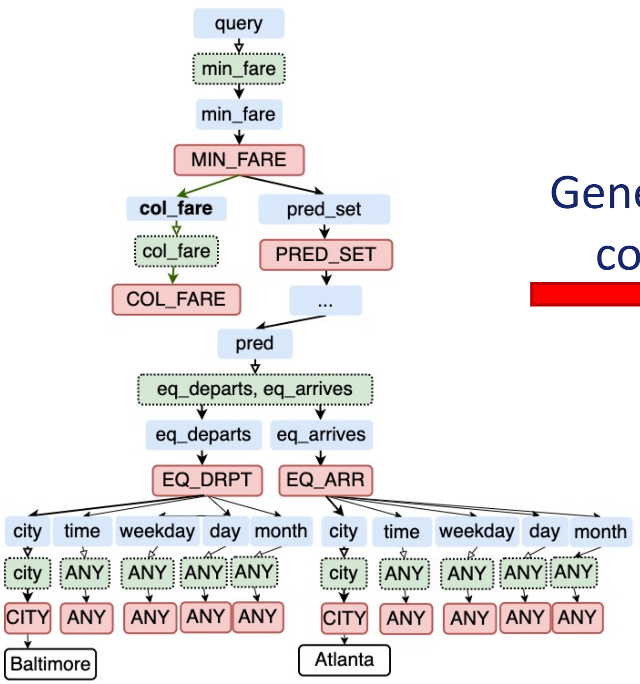


One set of path search result sets

HISyn: Path combination



HISyn: Code Generation



Generate code

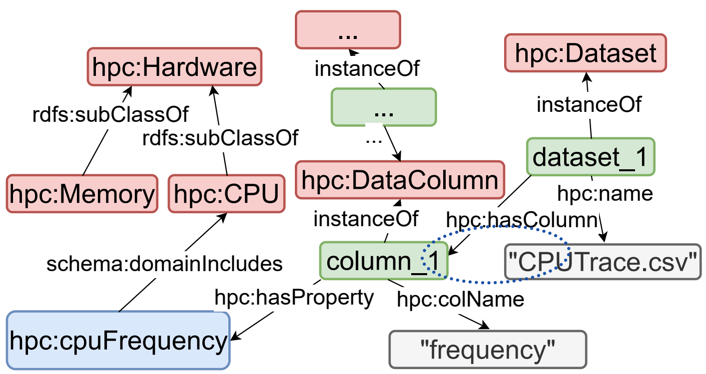
MIN_Fare(
COL_FARE(),
AtomicRowPredSet(
AtomicRowPred(
EQ_DEPARTS(CITY(baltimore), ANY(), ANY(), ANY(), ANY()),
EQ_ARRIVES(CITY(atlanta), ANY(), ANY(), ANY(), ANY()))))

Synthesized ATIS domain specific code expression

Ontology query types.

1. Property query

- Searches for the subject with certain properties.
- Corresponding to Ontology properties.
- E.g., *hpc:name, hpc:subject*
- Query: Get datasets whose *subject is "GPGPU"*



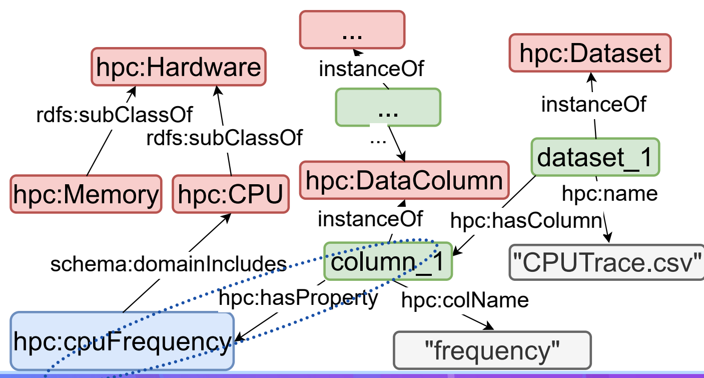
Ontology query types.

1. Property query

- Searches for the subject with certain properties.
- Corresponding to Ontology properties.
- E.g., *hpc:name, hpc:subject*
- Query: Get datasets whose *subject is "GPGPU"*

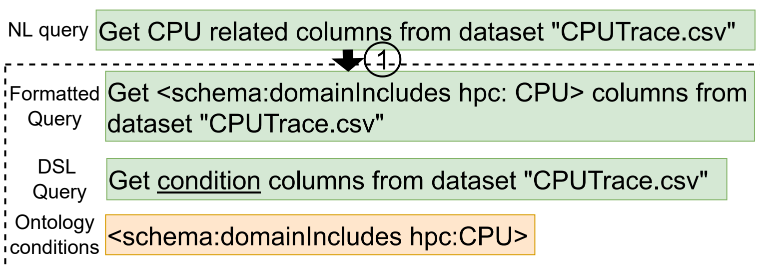
2. Concept query

- Searches for the hierarchical information from the Ontology concepts.
- Usually link to certain columns of datasets as related property domains of the columns.
- E.g., *(column_1, hpc:hasProperty, hpc:cpuFrequency)*
- Query: Get CPU related columns from dataset "CPUTrace.csv"

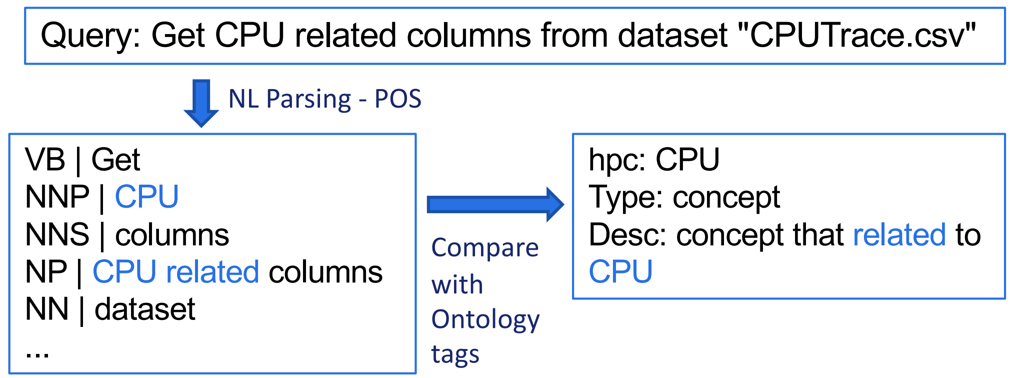


DSL/Ontology query split

Goal: split the contents in NL query to DSL portion and Ontology portion.



(a) DSL/Ontology split



1. Identify and map Ontology components

DSL/Ontology query split

Goal: split the contents in NL query to DSL portion and Ontology portion.

NL query	Get CPU related columns from dataset "CPUTrace.csv"
Formatted Query	Get <schema:domainIncludes hpc: CPU> columns from dataset "CPUTrace.csv"
DSL Query	Get condition columns from dataset "CPUTrace.csv"
Ontology conditions	<schema:domainIncludes hpc:CPU>

(a) DSL/Ontology split

1. Identify and map Ontology components
2. Generate formatted query
3. Interactive selection

Query: Get CPU related columns from dataset "CPUTrace.csv"

hpc: CPU
Type: concept
Desc: concept that related to CPU

Create Ontology conditions based on map

Ontology conditions:
<schema:domainIncludes hpc:CPU>

Replace Ontology components with conditions

Get <schema:domainIncludes hpc:CPU> columns from dataset "CPUTrace.csv"

Interactive query selection

Ontology Module

Goal: generate SPARQL queries to acquire information from Ontology.

- **Template for property queries.**

```
SELECT ?var
  WHERE {
     ?x_item rdf:type <class>.
     ?x_item <property tags 1>.
     ?x_item <property tags 2>.
     ...
     ?x_item <subject> ?var
```

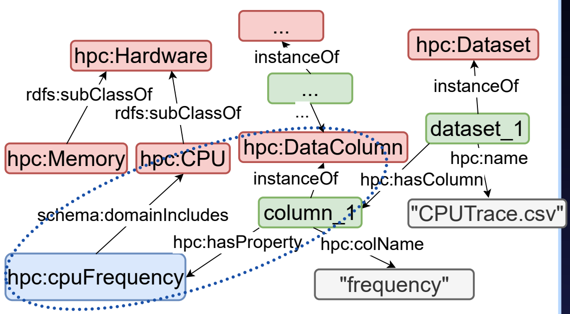
- Query: Get the hardware used by the experiment with name "X"
- Formatted query: Get hardware *<hpc:wasUsedBy <hpc:experiment hpc:name "X">>*

```
SELECT ?var
  WHERE {
     ?x_item rdf:type hpc:Hardware.
     ?x_item hpc:wasUsedBy ?experiment.
     ?experiment hpc:name "X".
     ?x_item hpc:name ?var
}
```

Ontology Module

Goal: generate SPARQL queries to acquire information from Ontology.

- **Template for concept queries.**

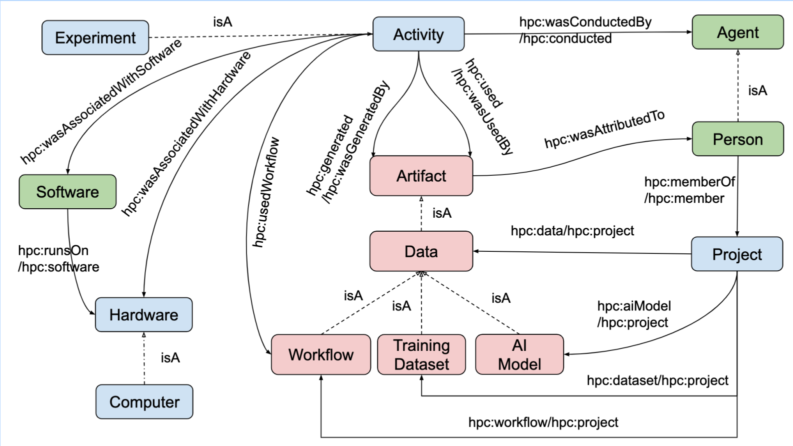


SELECT ?colName
WHERE{?dataset rdf:type hpc:Dataset.
?dataset hpc:name ***"Dataset name"****.*
?dataset hpc:hasColumn ?var.
?var ***hpc:colName ?colName****.*
?var hpc:hasProperty ?colTag.
{SELECT ?colTag WHERE {
?props ***schema:domainIncludes concept****.*
?colTag rdfs:subPropertyOf ?props}}}*

SELECT ?colName
WHERE{?dataset rdf:type hpc:Dataset.
?dataset hpc:name ***"CPUTrace.csv"****.*
?dataset hpc:hasColumn ?var.
?var ***hpc:colName ?colName****.*
?var hpc:hasProperty ?colTag.
{SELECT ?colTag WHERE {
?props ***schema:domainIncludes hpc:cpu****.*
?colTag rdfs:subPropertyOf ?props}}}*

Workflow for HPC-FAIR

- **FAIR: data made findable, accessible, interoperable, and reusable.**
- **HPC-FAIR** is an open platform for FAIR data in HPC.
- **Ontology** in a knowledge graph that stores structural knowledge beyond the NL description
- **HPC-FAIR** stores **training datasets** and **AI models** used for HPC software analyses and optimizations
- **Workflows in HPC-FAIR** cooperate these data and models from different projects, e.g.,
  - Merge dataset “X.csv” and dataset “Y.json”
  - Get datasets whose subject is “GPGPU”
  - Get CPU related columns from dataset “CPUTrace.csv”



Major High-level concepts and relations of the HPC Ontology