Semantically Lifted Programming

Eduard Kamburjan

University of Oslo
TCS Seminar, 16.09.23

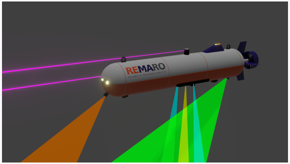
Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems, e.g., for robotics

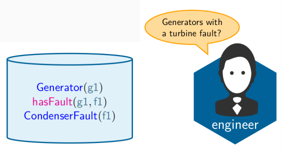


REMARO
RELIABLE AI FOR MARINE ROBOTICS

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry



Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry
- Reasoning for expert systems
  e.g., in the biomedical field

GENEONTOLOGY
Unifying Biology

SNOMED CT
The global
language of
healthcare

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry
- Reasoning for expert systems
  e.g., in the biomedical field
- Data integration
  e.g., as industrial standards

IEEE SA
STANDARDS
ASSOCIATION

READI

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry
- Reasoning for expert systems
  e.g., in the biomedical field
- Data integration
  e.g., as industrial standards

IEEE SA
STANDARDS
ASSOCIATION

READI

Surrounding theories and tools are Semantic Technologies

… and Programs

How to use ontologies in programming?

- Make domain knowledge available to the programmer
- Reduce redundancy between program and other artifacts
- Simplify communication with users/domain experts

… and Programs

How to use ontologies in programming?

- Make domain knowledge available to the programmer
- Reduce redundancy between program and other artifacts
- Simplify communication with users/domain experts

How to program applications around ontologies?

- Using multiple Semantic Web technologies can be tricky
- Programmer must be aware of logical and formal pitfalls
- Correct interplay must be ensures manually

… and Programs

How to use ontologies in programming?

- Make domain knowledge available to the programmer
- Reduce redundancy between program and other artifacts
- Simplify communication with users/domain experts

How to program applications around ontologies?

- Using multiple Semantic Web technologies can be tricky
- Programmer must be aware of logical and formal pitfalls
- Correct interplay must be ensures manually

This Talk

- First results, challenges, on-going research
- Use ontologies in programming to enable Digital Twins.

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Peter a Person. Paul a Person. Maria a Person.
Peter hasChild Paul. Paul hasChild Maria.
```

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Peter a Person. Paul a Person. Maria a Person.
Peter hasChild Paul. Paul hasChild Maria.
```

OWL:

`hasChild` **some** (`hasChild` **some** `Person`)

subClassOf `GrandParent`

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Peter a Person. Paul a Person. Maria a Person.
Peter hasChild Paul. Paul hasChild Maria.
```

OWL:

`hasChild` **some** (`hasChild` **some** `Person`)

subClassOf `GrandParent`

SPARQL: `SELECT ?x WHERE { ?x a GrandParent }`

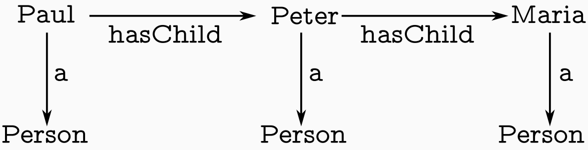
Knowledge Graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.



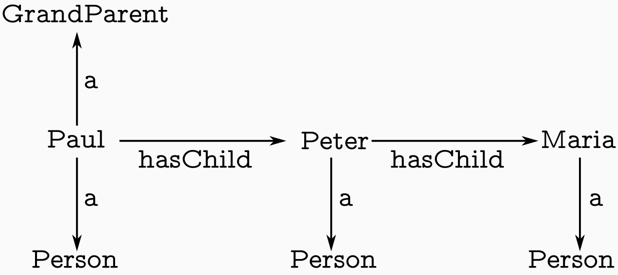
Knowledge Graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.



Semantically Lifted Programs and Digital Twins

Semantically Lifted States

A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.

Semantically Lifted States

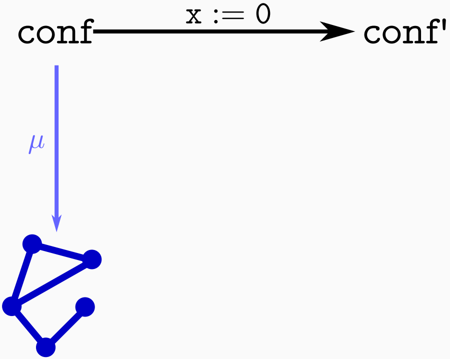
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.

$$\text{conf} \xrightarrow{\texttt{x} := 0} \text{conf'}$$

Programming and Debugging with Semantically Lifted States, Kamburjan et al. [ESWC'21]

Semantically Lifted States

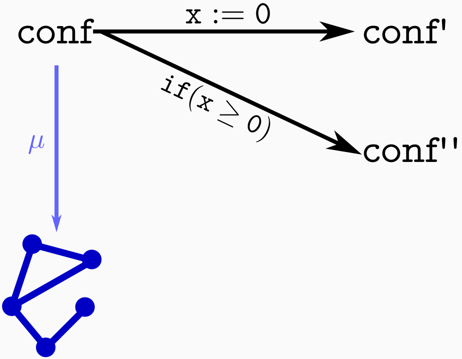
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



Programming and Debugging with Semantically Lifted States, Kamburjan et al. [ESWC'21]

Semantically Lifted States

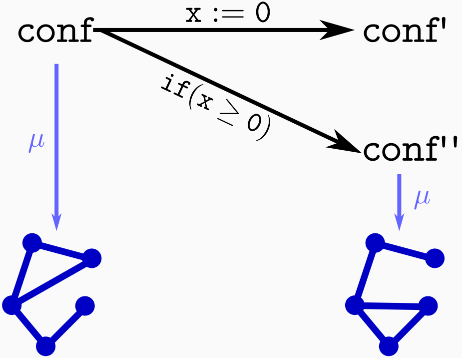
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



Programming and Debugging with Semantically Lifted States, Kamburjan et al. [ESWC'21]

Semantically Lifted States

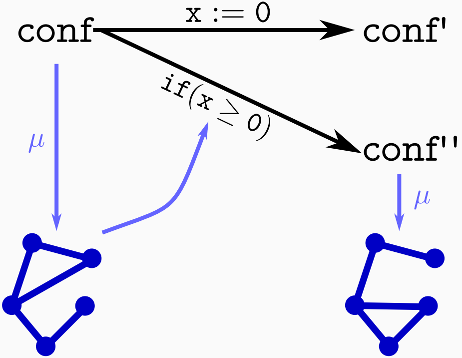
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



Programming and Debugging with Semantically Lifted States, Kamburjan et al. [ESWC'21]

Semantically Lifted States

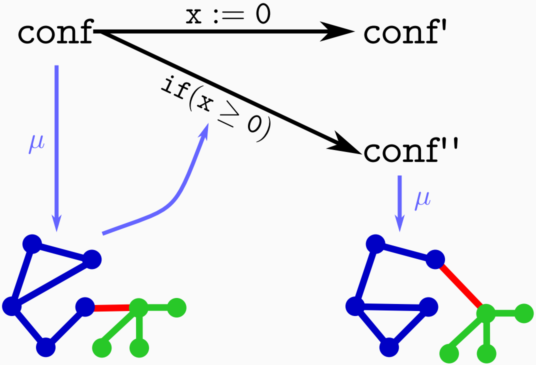
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



Programming and Debugging with Semantically Lifted States, Kamburjan et al. [ESWC'21]

Semantically Lifted States

A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



Programming and Debugging with Semantically Lifted States, Kamburjan et al. [ESWC'21]

Example

```
class C (Int i) Unit inc() this.i = this.i + 1; end end
main C c = new C(5);  Int i = c.inc(); end
```

Example

```
class C (Int i) Unit inc() this.i = this.i + 1; end end
main C c = new C(5);  Int i = c.inc(); end
```

```
prog:C a prog:class. prog:C prog:hasField prog:i.
run:obj1 a prog:C.   run:obj1 prog:i 5.
....
```

Example

```
class C (Int i) Unit inc() this.i = this.i + 1; end end
main C c = new C(5);  Int i = c.inc(); end
```

```
prog:C a prog:class. prog:C prog:hasField prog:i.
run:obj1 a prog:C.   run:obj1 prog:i 5.
....
```

A representation of (a) the full AST and (b) the full runtime state.

SMOL

Given the lifted state, we can use it for multiple operations.

- **Access it** to retrieve objects without traversing pointers.
- **Enrich it** with an ontology, perform logical reasoning and retrieve objects using a query *using the vocabulary of the domain*.
- **Combine it** with another knowledge graph and access external data based on information from the current program state.

Semantic Programming

```
class Platform(List<Server> serverList) ... end
class Server(List<Task> taskList) ... end
class Scheduler(List<Platform> platformList)
 Unit reschedule()
  List<Platform> l
    := access("SELECT ?x WHERE {?x a :Overloaded}");
  this.adaptPlatforms(l);
 end
end
```

Semantic Programming

```
class Platform(List<Server> serverList) ... end
class Server(List<Task> taskList) ... end
class Scheduler(List<Platform> platformList)
 Unit reschedule()
  List<Platform> l
    := access("SELECT ?x WHERE {?x a :Overloaded}");
  this.adaptPlatforms(l);
 end
end
```

```
:Overloaded
 owl:equivalentClass [
   owl:onProperty (:tasks, :length);
   owl:minValue 3;
 ].
```

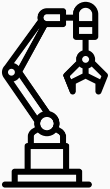
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.

Digital Twin

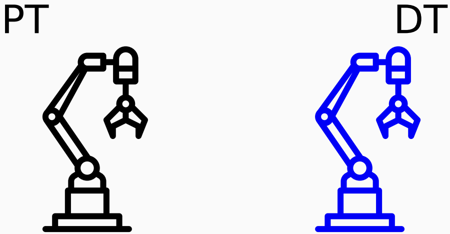
A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.

PT



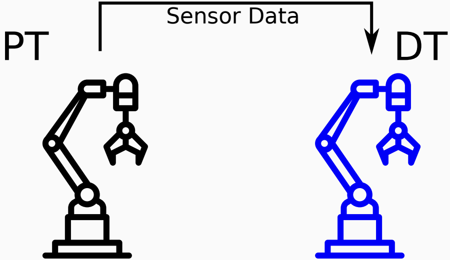
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



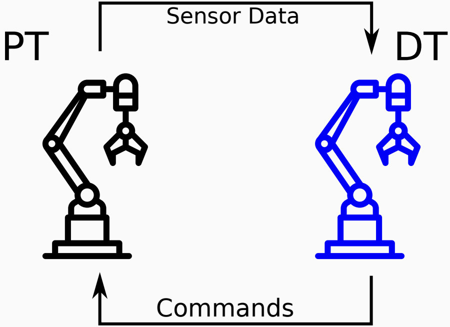
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



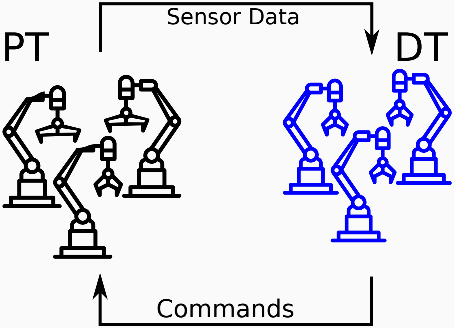
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



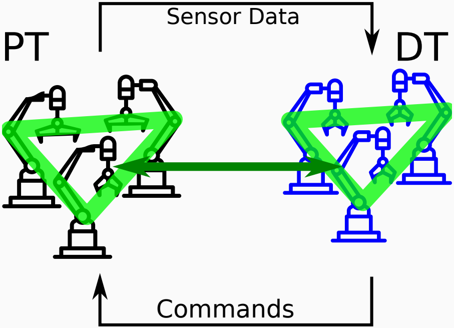
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



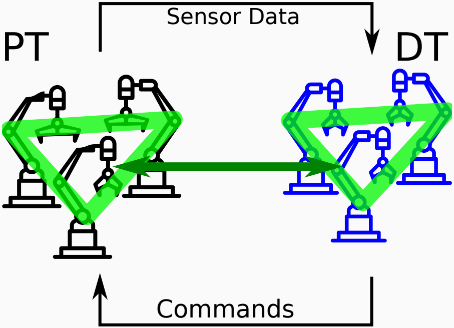
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



- Common data representation
- Data view on both twins: Twinning as a data property

Knowledge Graphs and Asset Models

Asset Model

An asset model is an organized, digital description of the composition and properties of a physical asset.

Our Asset Model

A knowledge graph describing the structure of the physical twin.

Knowledge Graphs and Asset Models

Asset Model

An asset model is an organized, digital description of the composition and properties of a physical asset.

Our Asset Model

A knowledge graph describing the structure of the physical twin.

```
ast:heater1 a ast:Heater. ast:heater1 ast:in ast:room1.
ast:heater2 a ast:Heater. ast:heater2 ast:in ast:room2.
ast:heater1 ast:id 13. ast:heater2 ast:id 12.
ast:room1 ast:leftOf ast:room2.
```

Knowledge Graphs and Asset Models

Asset Model

An asset model is an organized, digital description of the composition and properties of a physical asset.

Our Asset Model

A knowledge graph describing the structure of the physical twin.

```
ast:heater1 a ast:Heater. ast:heater1 ast:in ast:room1.
ast:heater2 a ast:Heater. ast:heater2 ast:in ast:room2.
ast:heater1 ast:id 13. ast:heater2 ast:id 12.
ast:room1 ast:leftOf ast:room2.
```

`htLeftOf` **subPropertyOf** `ast:in` **o** `ast:leftOf` **o** **inverse**(`ast:in`)

Checking the Twinning Property

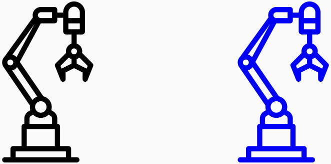
Combining the Knowledge

- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge

Checking the Twinning Property

Combining the Knowledge

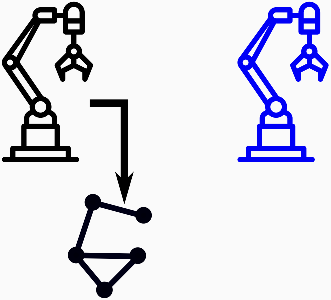
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

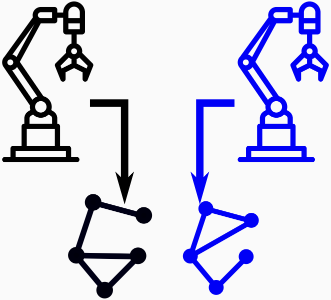
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

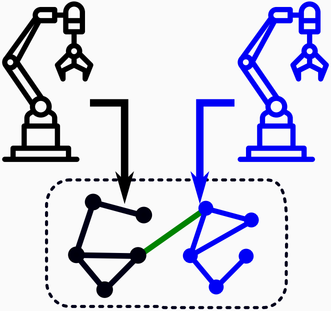
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge

Possible Constraints

- Constraint on asset model
  "Is the asset model consistent?"
- Constraint on program
  "Is this a sensible simulation structure?"
- Constraints on twinning
  "Does the program have the same structure as the asset?"

SMOL and FMI

Functional Mock-Up Interface (FMI)

Standard for (co-)simulation units, called function mock-up units (FMUs). Can also serve as interface to sensors and actuators.

SMOL and FMI

Functional Mock-Up Interface (FMI)

Standard for (co-)simulation units, called function mock-up units (FMUs). Can also serve as interface to sensors and actuators.

```
//simplified shadow
class Monitor(Cont[out Double val] sys,
              Cont[out Double val] shadow)
  Unit run(Double threshold)
    while shadow != null do
      sys.doStep(1.0);  shadow.doStep(1.0);
      if(sys.val - shadow.val >= threshold) then ... end
    end ...
```

Constraints on Digital Twins

SMOL and FMI

SMOL with FMOs

FMOs are objects, so they are part of the knowledge graph.

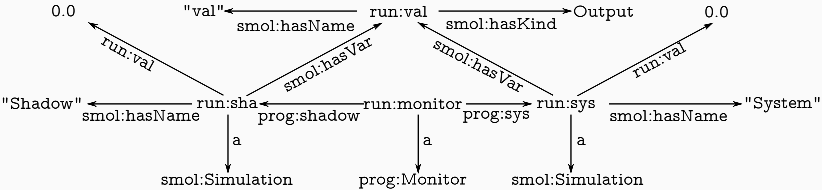
```
class Monitor(Cont[out Double val] sys,
                 Cont[out Double val] shadow)
```

SMOL and FMI

SMOL with FMOs

FMOs are objects, so they are part of the knowledge graph.

```
class Monitor(Cont[out Double val] sys,
              Cont[out Double val] shadow)
```



Knowledge Structures over Simulation Units, Kamburjan and Johnsen. [ANNSIM'22]

Semantically Lifting the Digital Twin

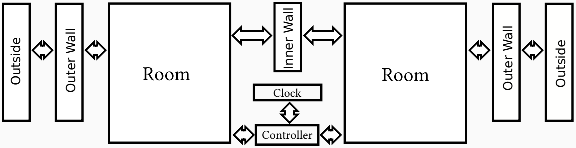
SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twins.

Semantically Lifting the Digital Twin

SPARQL

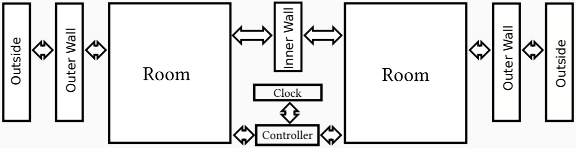
Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twin.



Semantically Lifting the Digital Twin

SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twin.



```
class Room(Cont[...] f,
           Wall inner, Wall outer, Controller ctrl,
           Int id) end
class Controller(Cont[...] f,
                 Room left, Room right, Int id) end
class InnerWall(Cont[...] f, Room left, Room right) end
```

Semantically Lifting the Digital Twin

SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twin.

Query to detect non-sensical setups:

```
SELECT ?room WHERE { ?ctrl a prog:Controller.
                   ?ctrl prog:left ?room.
                   ?ctrl prog:right ?room }
```

Semantically Lifting the Digital Twin

SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twin.

Query to check structural consistency for heaters:

```
SELECT * WHERE { ?o1 prog:id ?id1. ?h1 ast:id ?id1.
                 ?o2 prog:id ?id2. ?h2 ast:id ?id2.
                 ?h1 htLeftOf ?h2.
                 ?c a prog:Controller.
                 ?c prog:left ?o1. ?c prog:right ?o2.}
```

On-Going Work: Repairing your Twin

Semantic Reflection

One can use the knowledge graph *within* the program to detect structural drift: Formulate query to retrieve all mismatching parts

```
....
List<Repairs> repairs =
 construct("SELECT ?room ?wallLeft ?wallRight WHERE
   {?x ast:id ?room.
    ?x ast:right [ast:id ?wallRight].
    ?x ast:left [ast:id ?wallLeft].
    FILTER NOT EXISTS {?y a prog:Room; prog:id ?room.}}");
```

Repair function must restore structure.

Digital Twin Reconfiguration Using Asset Models, Kamburjan et al. [ISoLA'22]

Program Analysis and Optimization

Challenges

Optimization and Static Analysis of SMOL

- Every program optimization is unsound for SMOL, because the whole AST can be accessed through semantic reflection.
- Similarly, Garbage Collection is not possible, because every objects can be accessed even if no pointers to it exist.
- Static Analysis requires to analyze possible results of queries

Tools from Description Logic

First results that two notions from Description Logics can help with garbage collection and typing: *ontology modules* and *query subsumption*.

Ontology Modules

We must have a notion of encapsulation for knowledge graphs!

Ontology Modules

We must have a notion of encapsulation for knowledge graphs!

Ontology Module

Given a KB $\mathcal{K}$ and a signature Σ, the module $\mathcal{M}_{\mathcal{K}}^{\Sigma}$ of $\mathcal{K}$ w.r.t. Σ is a sub-KB that gives the same answers w.r.t. Σ.

- $\mathcal{M}_{\mathcal{K}}^{\Sigma} \subseteq \mathcal{K}$
- $\forall q.\ (\mathbf{sig}(q) \subseteq \Sigma) \rightarrow \mathsf{ans}(\mathcal{K}, q) = \mathsf{ans}(\mathcal{M}_{\mathcal{K}}^{\Sigma}, q)$

Ontology Modules

We must have a notion of encapsulation for knowledge graphs!

Ontology Module

Given a KB $\mathcal{K}$ and a signature Σ, the module $\mathcal{M}^{\Sigma}_{\mathcal{K}}$ of $\mathcal{K}$ w.r.t. Σ is a sub-KB that gives the same answers w.r.t. Σ.

- $\mathcal{M}^{\Sigma}_{\mathcal{K}} \subseteq \mathcal{K}$
- $\forall q.\ (\mathbf{sig}(q) \subseteq \Sigma) \rightarrow \mathsf{ans}(\mathcal{K}, q) = \mathsf{ans}(\mathcal{M}^{\Sigma}_{\mathcal{K}}, q)$

- Modules may expand signature $\mathbf{sig}(\mathcal{M}^{\Sigma}_{\mathcal{K}}) \supseteq \Sigma$
- Multiple notions of modules available
- Beware: ontology modules are *extracted*

Ontology Modules: Example

$$\mathcal{K} = \{\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},$$
$$\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists \texttt{servers.List}, \alpha_3 = \texttt{Task} \sqsubseteq \texttt{Object},$$
$$\texttt{Platform}(\mathsf{a}), \texttt{List}(\mathsf{b}), \texttt{Task}(\mathsf{c}), \texttt{servers}(\mathsf{a}, \mathsf{b})\}$$

,

Optimizing Semantically Lifted Programs through Ontology Modularity, Kamburjan and Chen [NWPT'21]

Ontology Modules: Example

$$\mathcal{K} = \{\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},$$

$$\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists \texttt{servers}.\texttt{List}, \alpha_3 = \texttt{Task} \sqsubseteq \texttt{Object},$$

$$\texttt{Platform}(\mathsf{a}), \texttt{List}(\mathsf{b}), \texttt{Task}(\mathsf{c}), \texttt{servers}(\mathsf{a}, \mathsf{b})\}$$

$$\Sigma = \{\texttt{Busy}\}$$

,

Optimizing Semantically Lifted Programs through Ontology Modularity, Kamburjan and Chen [NWPT'21]

Ontology Modules: Example

$$\mathcal{K} = \{\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},$$
$$\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists\texttt{servers.List}, \alpha_3 = \texttt{Task} \sqsubseteq \texttt{Object},$$
$$\texttt{Platform}(\mathsf{a}), \texttt{List}(\mathsf{b}), \texttt{Task}(\mathsf{c}), \texttt{servers}(\mathsf{a}, \mathsf{b})\}$$

$$\Sigma = \{\texttt{Busy}\}$$

$$\mathcal{M}^{\Sigma}_{\mathcal{K}} = \{\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},$$
$$\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists\texttt{servers.List},$$

Optimizing Semantically Lifted Programs through Ontology Modularity, Kamburjan and Chen [NWPT'21]

Ontology Modules: Example

$$\begin{aligned}\mathcal{K} = \{&\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},\\ &\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists\texttt{servers.List}, \alpha_3 = \texttt{Task} \sqsubseteq \texttt{Object},\\ &\texttt{Platform}(\mathsf{a}), \texttt{List}(\mathsf{b}), \texttt{Task}(\mathsf{c}), \texttt{servers}(\mathsf{a}, \mathsf{b})\}\end{aligned}$$

$$\Sigma = \{\texttt{Busy}\}$$

$$\begin{aligned}\mathcal{M}_{\mathcal{K}}^{\Sigma} = \{&\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},\\ &\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists\texttt{servers.List},\\ &\texttt{Platform}(\mathsf{a}), \texttt{List}(\mathsf{b}), \texttt{servers}(\mathsf{a}, \mathsf{b})\}\end{aligned}$$

Optimizing Semantically Lifted Programs through Ontology Modularity, Kamburjan and Chen [NWPT'21]

Ontology Modules: Example

$$\begin{aligned}\mathcal{K} = \{&\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},\\ &\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists \texttt{servers}.\texttt{List}, \alpha_3 = \texttt{Task} \sqsubseteq \texttt{Object},\\ &\texttt{Platform}(\mathsf{a}), \texttt{List}(\mathsf{b}), \texttt{Task}(\mathsf{c}), \texttt{servers}(\mathsf{a}, \mathsf{b})\}\\ \Sigma = \{&\texttt{Busy}\}\\ \mathcal{M}^{\Sigma}_{\mathcal{K}} = \{&\alpha_1 = \texttt{Busy} \sqsubseteq \texttt{Platform} \sqcap \texttt{NonEmpty},\\ &\alpha_2 = \texttt{NonEmpty} \sqsubseteq \exists \texttt{servers}.\texttt{List},\\ &\texttt{Platform}(\mathsf{a}), \texttt{List}(\mathsf{b}), \texttt{servers}(\mathsf{a}, \mathsf{b})\}\end{aligned}$$

Application

- Use $\mathcal{M}^{\textbf{sig}(\texttt{Q})}_{\mathcal{K}}$ to approximate dependencies of `access(Q)`.
- E.g., garbage collection: remove object if it is in no module.

Optimizing Semantically Lifted Programs through Ontology Modularity, Kamburjan and Chen [NWPT'21]

Type System

Challenge

Does **List**`<C> l := access(...);` indeed return a list of `C` objects?

KGs are untyped, deriving concepts requires reasoning....

Type System

Challenge

Does **List**`<C> l := access(...);` indeed return a list of `C` objects?

KGs are untyped, deriving concepts requires reasoning....

Query Containment under Entailment Regimes

A query Q is contained in Q' under some entailment regime for KG $\mathcal{K}$, ($Q \subseteq^{\mathcal{K}}_{\mathsf{er}} Q'$) if all answers to Q are also answers to Q'.

Type System

Challenge

Does **List**`<C> l := access(...);` indeed return a list of `C` objects?
KGs are untyped, deriving concepts requires reasoning....

Query Containment under Entailment Regimes

A query Q is contained in Q' under some entailment regime for KG $\mathcal{K}$, ($Q \subseteq^{\mathcal{K}}_{\text{er}} Q'$) if all answers to Q are also answers to Q'.

$$\textbf{(acc-type)}\ \frac{\Gamma \vdash \texttt{l} : \textbf{List}\texttt{<C>} \qquad \texttt{SELECT ?x } \{\texttt{P}\} \ \subseteq^{\mathcal{K}}_{\text{er}} \ \texttt{SELECT ?x } \{\texttt{?x a prog:C}\}}{\Gamma \vdash^{\mathcal{K}}_{\text{er}} \texttt{l:=access("SELECT ?x } \{\texttt{P}\}\texttt{")}}$$

Where $\mathcal{K}$ does *not* contain the lifted state, but only the ontology.

Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes, Kamburjan and Kostylev [DL'21]

Conclusion

Conclusion

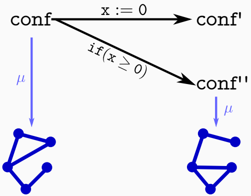
Semantically Lifted Programs and Semantic Reflection

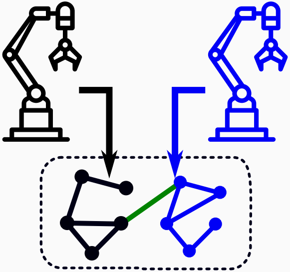
- Combining knowledge representation and programming
- Fully formal setting for digital twins
- Future work: static analysis, concurrency

Conclusion

Semantically Lifted Programs and Semantic Reflection

- Combining knowledge representation and programming
- Fully formal setting for digital twins
- Future work: static analysis, concurrency

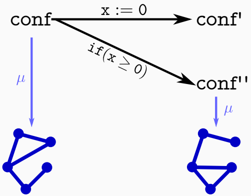


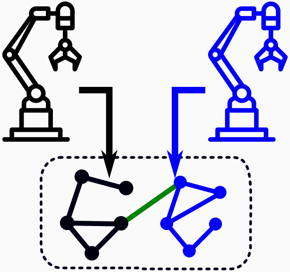


Conclusion

Semantically Lifted Programs and Semantic Reflection

- Combining knowledge representation and programming
- Fully formal setting for digital twins
- Future work: static analysis, concurrency





Thank you for your attention