A Hoare Logic for Domain Specification

Eduard Kamburjan[1]
Dilian Gurov[2]

Leiden, 04.03.24

[1]University of Oslo
[2]KTH Stockholm

Specification, huh, what is it good for?

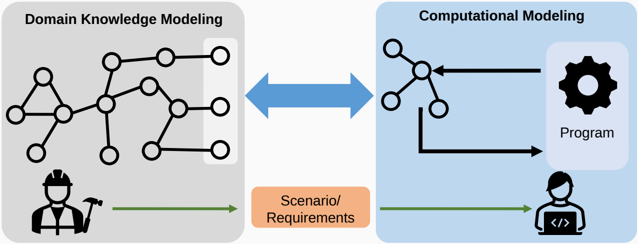
- Abstraction of computational details from other modules
- Intended computational behavior of the module itself

Specification, huh, what is it good for?

- Abstraction of computational details from other modules
- Intended computational behavior of the module itself
- Intended behavior w.r.t. business logic?

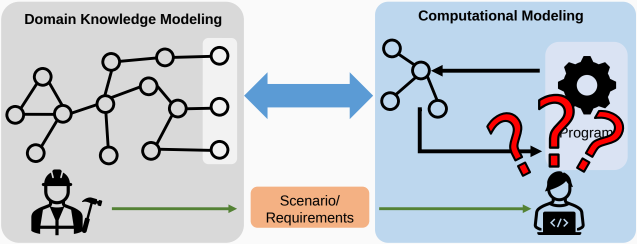
Specification, huh, what is it good for?

- Abstraction of computational details from other modules
- Intended computational behavior of the module itself
- Intended behavior w.r.t. business logic?



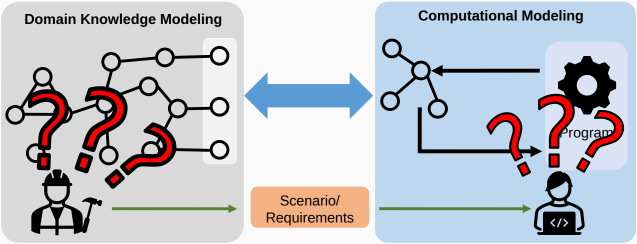
Specification, huh, what is it good for?

- Abstraction of computational details from other modules
- Intended computational behavior of the module itself
- Intended behavior w.r.t. business logic?



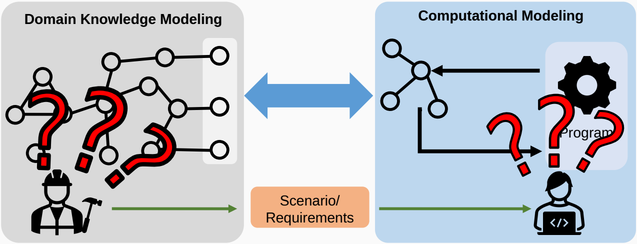
Specification, huh, what is it good for?

- Abstraction of computational details from other modules
- Intended computational behavior of the module itself
- Intended behavior w.r.t. business logic?



Specification, huh, what is it good for?

- Abstraction of computational details from other modules
- Intended computational behavior of the module itself
- Intended behavior w.r.t. business logic?



- Domain bugs are hard to find *and express*
- How can we use pragmatics of domain modeling tools inside a proof?
- How can we manage domain and computational specification during proofs?

What is a Car?

Suppose you model the assembly process of a car

```
procedure addWheels(p) nrWheels := p end
```

What is a Car?

Suppose you model the assembly process of a car

```
procedure addWheels(p) nrWheels := p end
```

Programmer

This procedure sets the number of wheels in a car to the value of p.

$$\{\top\}\mathit{addWheels}(p)\{\mathtt{nrWheels} \doteq \mathtt{p}\}$$

Subject Matter Expert

I want that in the end of this step, the car classifies as a small car.

$$\{\top\}\mathit{addWheels}(p)\{\mathtt{Small}(c)\}$$

What is a Car?

Suppose you model the assembly process of a car

```
procedure addWheels(p) nrWheels := p end
```

Programmer

This procedure sets the number of wheels in a car to the value of p.

$$\{\top\}\mathit{addWheels}(p)\{\texttt{nrWheels} \doteq \texttt{p}\}$$

Subject Matter Expert

I want that in the end of this step, the car classifies as a small car.

$$\{\top\}\mathit{addWheels}(p)\{\texttt{Small}(c)\}$$

How to enable both of them to specify their respective intent?

- SME does not know about how the car c is encoded
- Programmer does not know what it means for a car to be small.

Lifted Specification

Ontologies and Description Logics

For domain modeling and specification a rich body of methodologies and tools exist.

$$\texttt{HasFourWheels} \sqsubseteq \texttt{Small} \qquad \exists\texttt{wheels}.\exists\texttt{hasValue}.4 \equiv \texttt{HasFourWheels}$$

Can be used to give a program state a meaning in the domain, called *lifting* [ESWC'21]

Lifted Specification

Ontologies and Description Logics

For domain modeling and specification a rich body of methodologies and tools exist.

$$\texttt{HasFourWheels} \sqsubseteq \texttt{Small} \qquad \exists\texttt{wheels}.\exists\texttt{hasValue}.4 \equiv \texttt{HasFourWheels}$$

Can be used to give a program state a meaning in the domain, called *lifting* [ESWC'21]

$$\begin{Bmatrix} - \\ \texttt{p} \doteq 4 \end{Bmatrix} \texttt{addWheels}(\texttt{p}) \begin{Bmatrix} \texttt{Small}(c) \\ - \end{Bmatrix}$$

- Upper component specifies the state as interpreted in the domain
- Lower component specifies non-lifted state

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

$$\left\{ \qquad\qquad\qquad \right\}$$

$$\texttt{nrWheels} := \texttt{p}$$

$$\left\{ \texttt{Small}(c) \qquad\qquad\qquad\qquad\qquad\qquad \right\}$$

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\left\{ \qquad\qquad\qquad \right\}$$

$$\texttt{nrWheels} := \texttt{p}$$

$$\left\{ \texttt{Small}(c), \texttt{HasFourWheels}(c), \texttt{hasValue}(\texttt{wheelsVar}, 4) \right\}$$

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\left\{ \qquad\qquad\qquad \right\}$$

$$\mathtt{nrWheels} := \mathtt{p}$$

$$\left\{ \begin{array}{c} \mathtt{Small}(c), \mathtt{HasFourWheels}(c), \mathtt{hasValue}(\mathtt{wheelsVar}, 4) \\ \mathtt{nrWheels} \doteq 4 \end{array} \right\}$$

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\left\{ \begin{array}{c} \\ \mathtt{p} \doteq 4 \end{array} \right\}$$

$$\mathtt{nrWheels} := \mathtt{p}$$

$$\left\{ \begin{array}{c} \mathtt{Small}(c), \mathtt{HasFourWheels}(c), \mathtt{hasValue}(\mathtt{wheelsVar}, 4) \\ \mathtt{nrWheels} \doteq 4 \end{array} \right\}$$

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\left\{\begin{matrix}\mathtt{hasValue}(\mathtt{pVar}, 4) \\ \mathtt{p} \doteq 4\end{matrix}\right\}$$

$$\mathtt{nrWheels} := \mathtt{p}$$

$$\left\{\begin{matrix}\mathtt{Small}(c), \mathtt{HasFourWheels}(c), \mathtt{hasValue}(\mathtt{wheelsVar}, 4) \\ \mathtt{nrWheels} \doteq 4\end{matrix}\right\}$$

Keeping State and Lifted State Connected

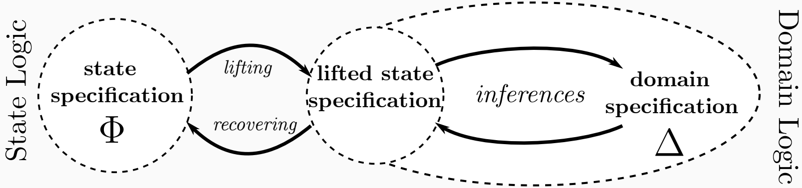
State Lifting

Function μ from runtime states to knowledge graphs.

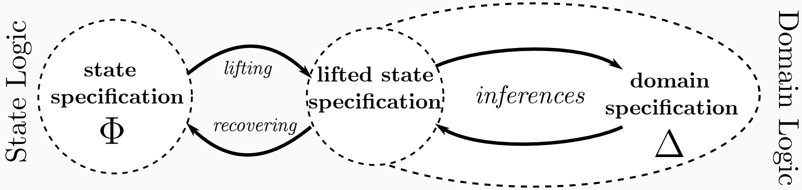
Specification Lifting

Function $\widehat{\mu}$ from program assertions to axioms. Must be compatible to state lifting:

$$\sigma \models \phi \rightarrow \mu(\sigma) \models \widehat{\mu}(\phi)$$



A Signature Perspective



Kernel and Generator

Let Σ be the signature of the domain specification.

- The kernel of $\widehat{\mu}$ is a signature **ker** $\widehat{\mu} \subseteq \Sigma$.
- A core generator α maps axioms Δ to axioms $\alpha(\Delta)$ with $\alpha(\Delta) \models \Delta$

- Kernel generator can either implement deduction, or abduction
- In case of abduction: ABox abduction with signature abducibles

Some Rules

- First you generate the kernel
- Additional premise trivial if α is deductive

$$\text{pre-core}\ \frac{\Delta_2 \models^{\mathbf{K}} \alpha(\Delta_2) \qquad \mathbf{K} \vdash \left\{{\Delta_1 \atop \Phi_1}\right\} s \left\{{\Delta_2, \alpha(\Delta_2) \atop \Phi_2}\right\}}{\mathbf{K} \vdash \left\{{\Delta_1 \atop \Phi_1}\right\} s \left\{{\Delta_2 \atop \Phi_2}\right\}}$$

Some Rules

- First you generate the kernel
- Additional premise trivial if α is deductive

$$\text{pre-core}\ \frac{\Delta_2 \models^{\mathbf{K}} \alpha(\Delta_2) \qquad \mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta_2, \alpha(\Delta_2)\\ \Phi_2\end{smallmatrix}\right\}}{\mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta_2\\ \Phi_2\end{smallmatrix}\right\}}$$

- Second you generate state assertions from the kernel axioms

$$\text{post-inv}\ \frac{\mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta, \Delta_2\\ \Phi_2 \wedge \widehat{\mu}^{-1}(\Delta_2)\end{smallmatrix}\right\}}{\mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta, \Delta_2\\ \Phi_2\end{smallmatrix}\right\}}\ \mathrm{sig}(\Delta_2) \subseteq \mathbf{ker}\ \widehat{\mu}$$

Some Rules

- First you generate the kernel
- Additional premise trivial if α is deductive

$$\text{pre-core}\ \frac{\Delta_2 \models^{\mathbf{K}} \alpha(\Delta_2) \qquad \mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta_2,\alpha(\Delta_2)\\ \Phi_2\end{smallmatrix}\right\}}{\mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta_2\\ \Phi_2\end{smallmatrix}\right\}}$$

- Same for precondition
- On state assertions, we can now use standard Hoare rules

- Second you generate state assertions from the kernel axioms

$$\text{post-inv}\ \frac{\mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta,\Delta_2\\ \Phi_2 \wedge \widehat{\mu}^{-1}(\Delta_2)\end{smallmatrix}\right\}}{\mathbf{K} \vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta,\Delta_2\\ \Phi_2\end{smallmatrix}\right\}}\ \mathrm{sig}(\Delta_2) \subseteq \mathbf{ker}\ \widehat{\mu}$$

A Car is a `Car`

- Standard Hoare calculus rules must check that specifications are consistent, and
- remove all domain knowledge, as it may have changed

$$\text{var}\ \frac{\widehat{\mu}(\Phi) \models^{\mathbf{K}} \Delta}{\mathbf{K} \vdash \left\{{\emptyset \atop \Phi[\mathtt{v}\backslash\mathtt{expr}]}\right\}\mathtt{v\ :=\ expr}\left\{{\Delta \atop \Phi}\right\}} \qquad \text{skip}\ \frac{}{\mathbf{K} \vdash \left\{{\Delta \atop \Phi}\right\}\text{skip}\left\{{\Delta \atop \Phi}\right\}}$$

A Car is a `Car`

- Standard Hoare calculus rules must check that specifications are consistent, and
- remove all domain knowledge, as it may have changed

$$\text{var}\ \frac{\widehat{\mu}(\Phi) \models^{\mathbf{K}} \Delta}{\mathbf{K} \vdash \left\{{\emptyset \atop \Phi[\mathtt{v}\backslash\mathtt{expr}]}\right\}\mathtt{v}\ :=\ \mathtt{expr}\left\{{\Delta \atop \Phi}\right\}} \qquad \text{skip}\ \frac{}{\mathbf{K} \vdash \left\{{\Delta \atop \Phi}\right\}\mathsf{skip}\left\{{\Delta \atop \Phi}\right\}}$$

But now, we can prove that our program does the right thing:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\mathtt{hasValue}(\mathtt{wheelsVar}, 4) \models^{\mathbf{K}} \mathtt{HasFourWheels}(c), \mathtt{hasValue}(\mathtt{wheelsVar}, 4)}}{\mathbf{K} \vdash \left\{{- \atop \mathtt{p} \doteq 4}\right\}\mathtt{nrWheels} := \mathtt{p}\left\{{\mathtt{HasFourWheels}(c), \mathtt{hasValue}(\mathtt{wheelsVar}, 4) \atop \mathtt{nrWheels} \doteq 4}\right\}}}{\mathbf{K} \vdash \left\{{- \atop \mathtt{p} \doteq 4}\right\}\mathtt{nrWheels} := \mathtt{p}\left\{{\mathtt{HasFourWheels}(c), \mathtt{hasValue}(\mathtt{wheelsVar}, 4) \atop -}\right\}}}{\mathbf{K} \vdash \left\{{- \atop \mathtt{p} \doteq 4}\right\}\mathtt{nrWheels} := \mathtt{p}\left\{{\mathtt{HasFourWheels}(c) \atop -}\right\}}$$

Conclusion

Summary

- Managing description logic axioms in program verification
- No integration, retains separation of concerns
- A domain interpretation of contracts without refinement

Conclusion

Summary

- Managing description logic axioms in program verification
- No integration, retains separation of concerns
- A domain interpretation of contracts without refinement

Full details on arxiv

Eduard Kamburjan, Dilian Gurov:

A Hoare Logic for Domain Specification

`https://doi.org/10.48550/arXiv.2402.00452`

Conclusion

Summary

- Managing description logic axioms in program verification
- No integration, retains separation of concerns
- A domain interpretation of contracts without refinement

Full details on arxiv

Eduard Kamburjan, Dilian Gurov:

A Hoare Logic for Domain Specification

`https://doi.org/10.48550/arXiv.2402.00452`

Thank you for your attention