A Hoare Logic for Domain Specification

Eduard Kamburjan[1]
Dilian Gurov[2]

DL Workshop, Bergen, 19.06.24

[1]University of Oslo
[2]KTH Stockholm

Description Logics in Program Specification

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

```
/*@
requires
  \forall i. 0 <= i < arr.size -> arr[i] > 0

ensures \result > 0 @*/
int sumArray(int[] arr) { ... }
```

Description Logics in Program Specification

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

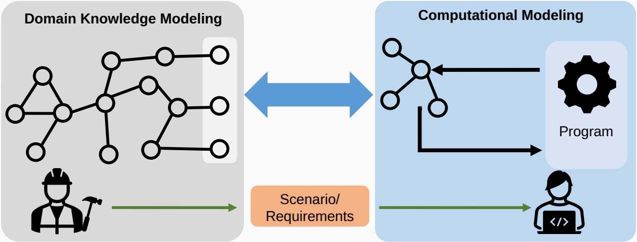
```
/*@
requires
  \forall i. 0 <= i < arr.size -> arr[i] > 0
requires arr != null
ensures \result > 0 @*/
int sumArray(int[] arr) { ... }
```

Description Logics in Program Specification

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

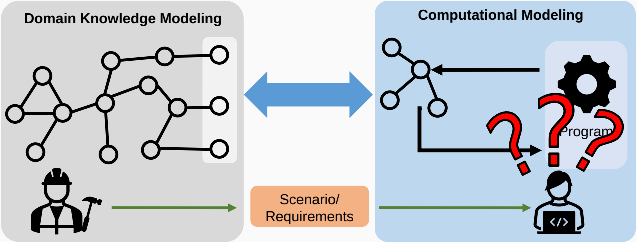
Description Logics in Program Specification

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



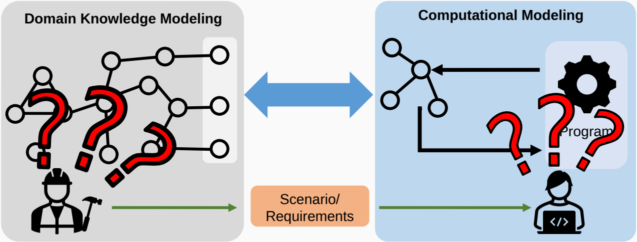
Description Logics in Program Specification

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



Description Logics in Program Specification

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



- Domain bugs are hard to find *and express*
- How can we use pragmatics of DL/OWL tools inside a proof?
- How can we manage domain and computational specification during proofs?
- **How to use description logics for program specification?**

Program Specification

Hoare Triples

Specifies programs in terms of their precondition and postcondition.

$$\{pre\}s\{post\}$$

- If *pre* holds in the state before *s* is executed, and
- the execution of *s* terminates,
- then *post* holds in the final state
- Usual notion of validity

$$\{\mathtt{i} \geq 0\}\mathtt{i} \ := \ \mathtt{i} + 1;\{\mathtt{i} > 0\}$$

Program Specification

State Logic

- Formulas *pre*, *post* defined in state logic
- FO logic extended with program variables as terms
- Models are transition systems where program variables are interpreted per states
- Important: state logic is closed under term substitution

$$
\begin{aligned}
(\mathtt{i} \geq 0)[\mathtt{i} \setminus \mathtt{i}+1] &= \mathtt{i}+1 > 0 \\
(\mathtt{i} \geq 0)[\mathtt{i} \setminus 0] &= 0 > 0
\end{aligned}
$$

Hoare Triple

- Used since 70s, in certain variants main approach to program verification
- Weakest Precondition (wp) is backwards reasoning: generate *pre* from *post*
- Rules for contracts, loops, recursion, . . .

Example Axiomatic Semantics

$$\frac{}{\vdash \{\phi[v \setminus e]\}\texttt{v := e}\{\phi\}}$$

Hoare Triple

- Used since 70s, in certain variants main approach to program verification
- Weakest Precondition (wp) is backwards reasoning: generate *pre* from *post*
- Rules for contracts, loops, recursion, ...

Example Axiomatic Semantics

$$\frac{}{\vdash \{\phi[v \setminus e]\}\texttt{v := e}\{\phi\}} \quad \frac{\vdash pre \rightarrow pre' \qquad \vdash \{pre'\}s\{post'\} \qquad \vdash post' \rightarrow post}{\vdash \{pre\}s\{post\}}$$

Hoare Triple

- Used since 70s, in certain variants main approach to program verification
- Weakest Precondition (wp) is backwards reasoning: generate *pre* from *post*
- Rules for contracts, loops, recursion, ...

Example Axiomatic Semantics

$$\frac{}{\vdash \{\phi[v \setminus e]\}\texttt{v := e}\{\phi\}} \qquad \frac{\vdash pre \rightarrow pre' \qquad \vdash \{pre'\}s\{post'\} \qquad \vdash post' \rightarrow post}{\vdash \{pre\}s\{post\}}$$

Example

$$\frac{\dfrac{}{\vdash \texttt{i} \geq 0 \rightarrow \texttt{i} + 1 > 0} \qquad \dfrac{}{\vdash \{\texttt{i} + 1 > 0\}\texttt{i := i} + 1\{\texttt{i} > 0\}} \qquad \dfrac{}{\vdash \texttt{i} > 0 \rightarrow \texttt{i} > 0}}{\vdash \{\texttt{i} \geq 0\}\texttt{i := i} + 1\{\texttt{i} > 0\}}$$

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.

$$\{-\}\texttt{addWheels}(\texttt{p})\{\texttt{nrWheels} \doteq \texttt{p}\}$$

Subject Matter Expert

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

$$\{-\}\texttt{addWheels}(\texttt{p})\{\texttt{Small}(c)\}$$

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.

$$\{-\}\texttt{addWheels}(\texttt{p})\{\texttt{nrWheels} \doteq \texttt{p}\}$$

Subject Matter Expert

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

$$\{-\}\texttt{addWheels}(\texttt{p})\{\texttt{Small}(c)\}$$

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.

Giving Meaning to States

Do we really want to use *DL* to specify state? Do we need to use DL to specify *state*?

Semantic Lifting

Semantic lifting is a technique to interpret a program state as a knowledge graph.

- Formally: function μ from runtime states to knowledge graphs.

Examples

$$\mu(\langle \mathtt{i} \mapsto 5\rangle) = \{\mathtt{hasValue}(\mathtt{iVar}, 5), \dots\}$$

- Lifting μ may add some knowledge $\mathbf{K} = \{\mathtt{wheels}(c, \mathtt{nrWheelsVar}\}$

$$\mu(\langle \mathtt{nrWheels} \mapsto 4\rangle) = \{\mathtt{hasValue}(\mathtt{nrWheelsVar}, 4), \mathtt{wheels}(c, \mathtt{nrWheelsVar}) \dots\}$$

Useful for highly domain specific software when combined with reflection

[Programming and Debugging with Semantically Lifted States, ESWC'21]

Lifted Specification

Ontologies and Description Logics

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

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

Lifted Specification

Ontologies and Description Logics

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

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

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

- 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.

$$\texttt{wheels}(c, \texttt{wheelsVar}) \vdash \left\{ \qquad \right\}$$

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

$$\left\{ \begin{array}{l} \texttt{Small}(c) \\ \\ \end{array} \right\}$$

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

$$\mathtt{wheels}(c, \mathtt{wheelsVar}) \vdash \left\{ \quad \right\}$$

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

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

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

$$\mathtt{wheels}(c, \mathtt{wheelsVar}) \vdash \left\{ \quad \right\}$$

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

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

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

$$\texttt{wheels}(c, \texttt{wheelsVar}) \vdash \left\{ \begin{array}{c} \\ \texttt{p} \doteq 4 \end{array} \right\}$$

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

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

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

$$\mathtt{wheels}(c, \mathtt{wheelsVar}) \vdash \left\{ \begin{array}{c} \mathtt{hasValue}(\mathtt{pVar}, 4) \\ \mathtt{p} \doteq 4 \end{array} \right\}$$

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

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

Keeping State and Lifted State Connected

Specification Lifting

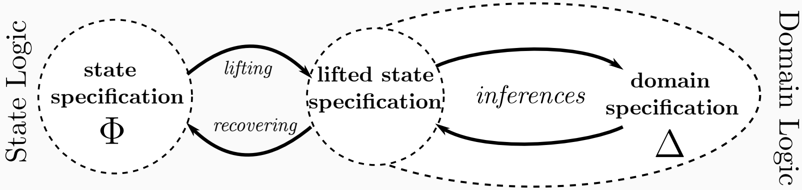
Function $\widehat{\mu}$ from program assertions to axioms. Must be compatible to state lifting:

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

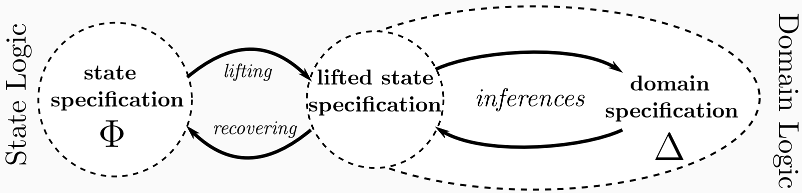
$$\begin{aligned} \widehat{\mu}(\mathtt{v} \doteq \mathtt{1}) &= \{\mathtt{hasValue}(\mathtt{vVar}, \mathtt{1})\} \\ \widehat{\mu}^{-1}(\{\mathtt{hasValue}(\mathtt{vVar}, \mathtt{1})\}) &= \mathtt{v} \doteq \mathtt{1} \end{aligned}$$

- Inverse lifting also allows to derive conditions in the state logic
- State lifting can be defined for language, specification lifting is per application due to loss of expressive power
- Not refinement! Lifting gives a different perspective, not a less abstract one

A Signature Perspective



A Signature Perspective

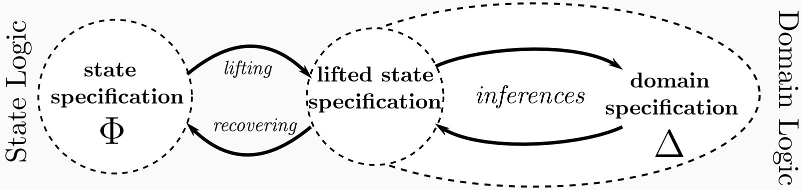


Kernel and Generator

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 $\alpha(\Delta) \models \Delta$

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

Validity and Judgement

Validity

Given a compatible pair $\mu, \widehat{\mu}$, a set of contracts $\mathbf{C}$ and a set of axioms $\mathbf{K}$.

$$\begin{Bmatrix} \Delta_1 \\ \Phi_1 \end{Bmatrix} s \begin{Bmatrix} \Delta_2 \\ \Phi_2 \end{Bmatrix}$$

is valid if $\forall(\sigma, \sigma') \in [\![s]\!]_{\mathbf{C},\mathbf{K}} \cdot \left(\sigma \models_{\mathbf{K}} \left\{\begin{smallmatrix} \Delta_1 \\ \Phi_1 \end{smallmatrix}\right\} \rightarrow \sigma' \models_{\mathbf{K}} \left\{\begin{smallmatrix} \Delta_2 \\ \Phi_2 \end{smallmatrix}\right\}\right)$

Next: how to design a sound calculus to prove validity?

$$\mathbf{C}, \mathbf{K} \vdash \begin{Bmatrix} \Delta_1 \\ \Phi_1 \end{Bmatrix} s \begin{Bmatrix} \Delta_2 \\ \Phi_2 \end{Bmatrix}$$

Some Rules

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

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

Some Rules

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

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

- Second you generate state assertions from the kernel axioms

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

Some Rules

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

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

- Second you generate state assertions from the kernel axioms

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

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

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

$$\textbf{(var)}\ \frac{\widehat{\mu}(\Phi) \models^{\mathbf{K}} \Delta}{\mathbf{C}, \mathbf{K} \vdash \left\{\begin{matrix}\emptyset \\ \Phi[\mathtt{v} \backslash \mathtt{expr}]\end{matrix}\right\} \mathtt{v\ :=\ expr} \left\{\begin{matrix}\Delta \\ \Phi\end{matrix}\right\}}$$

$$\textbf{(skip)}\ \frac{}{\mathbf{C}, \mathbf{K} \vdash \left\{\begin{matrix}\Delta \\ \Phi\end{matrix}\right\} \mathsf{skip} \left\{\begin{matrix}\Delta \\ \Phi\end{matrix}\right\}}$$

A Car is a $\texttt{Car}$

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

$$\textbf{(var)}\ \frac{\widehat{\mu}(\Phi) \models^{\mathbf{K}} \Delta}{\mathbf{C}, \mathbf{K} \vdash \{{}^{\emptyset}_{\Phi[\mathtt{v} \backslash \mathtt{expr}]}\}\mathtt{v} \ := \ \mathtt{expr}\{{}^{\Delta}_{\Phi}\}} \qquad \textbf{(skip)}\ \frac{}{\mathbf{C}, \mathbf{K} \vdash \{{}^{\Delta}_{\Phi}\}\mathsf{skip}\{{}^{\Delta}_{\Phi}\}}$$

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

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

Perspectives

Why not just inverse-lift post-condition before proving anything?

Perspectives

Why not just inverse-lift post-condition before proving anything?

Justification

- Rules allow domain view at every point during the proof attempt
- Use justifications etc. to have a domain interpretation of failed proofs!
- “Variable $\mathtt{p}$ has wrong value” vs. “5 wheels do not make a small car”

$$\left\{\begin{matrix} - \\ \mathtt{p} \doteq 4 \end{matrix}\right\}\mathtt{nrWheels}\ :=\ \mathtt{p} + 1\left\{\begin{matrix} \mathtt{Small}(c) \\ - \end{matrix}\right\}$$

- Possibly more: derive what conditions you would need to derive post-condition

Perspectives

Why not just inverse-lift post-condition before proving anything?

Contracts

- Use DL reasoners whenever possible: consequence and contracts

$$\textbf{(contract)}\ \frac{}{\mathbf{C},\mathbf{K}\vdash \mathrm{Pre}(\mathbf{C},\mathtt{p},\mathtt{e})\ \mathtt{p}(\mathtt{e})\ \mathrm{Post}(\mathbf{C},\mathtt{p},\mathtt{e})}$$

$$\textbf{(cons)}\ \frac{\mathbf{C},\mathbf{K}\vdash \left\{\begin{smallmatrix}\Delta_1'\\ \Phi_1'\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta_2'\\ \Phi_2'\end{smallmatrix}\right\} \qquad \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} \rightarrow_{\mathbf{K}} \left\{\begin{smallmatrix}\Delta_1'\\ \Phi_1'\end{smallmatrix}\right\} \qquad \left\{\begin{smallmatrix}\Delta_2'\\ \Phi_2'\end{smallmatrix}\right\} \rightarrow_{\mathbf{K}} \left\{\begin{smallmatrix}\Delta_2\\ \Phi_2\end{smallmatrix}\right\}}{\mathbf{C},\mathbf{K}\vdash \left\{\begin{smallmatrix}\Delta_1\\ \Phi_1\end{smallmatrix}\right\} s \left\{\begin{smallmatrix}\Delta_2\\ \Phi_2\end{smallmatrix}\right\}}$$

Conclusion

Main Theoretical Result

- Sound Lifted Hoare calculus for a while language with loops and procedures.
- Rules for all statements, plus rules for all steps in the lifting procedure.

Conclusion

Summary

- Using description logics in program verification
- A domain interpretation of contracts without refinement: managing perspectives

Conclusion

Summary

- Using description logics in program verification
- A domain interpretation of contracts without refinement: managing perspectives

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