TECHNISCHE UNIVERSITÄT DARMSTADT

TAMING BIG CATS

APM Workshop, Turin

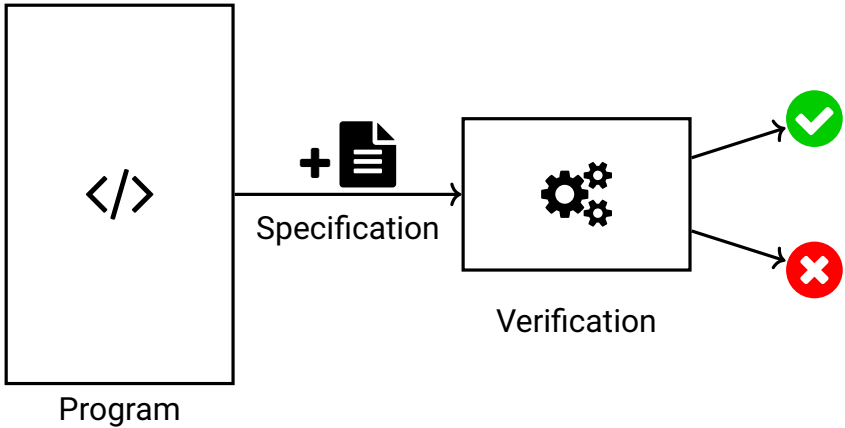
Marco Scaletta

17 April 2026

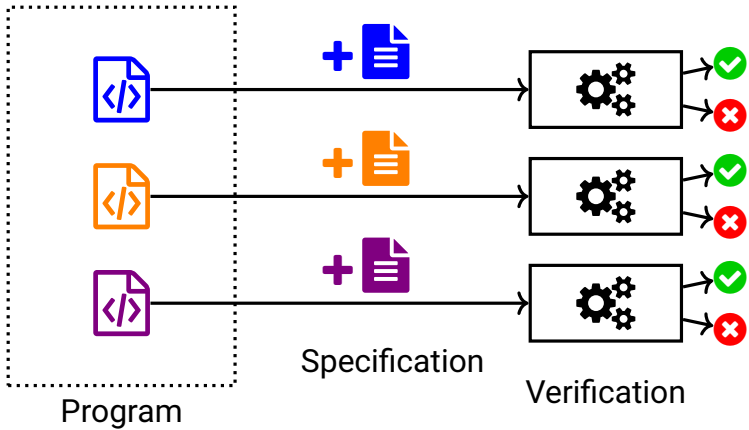
MOTIVATION

Section 1

MODULARITY: KEY TO SCALABILITY

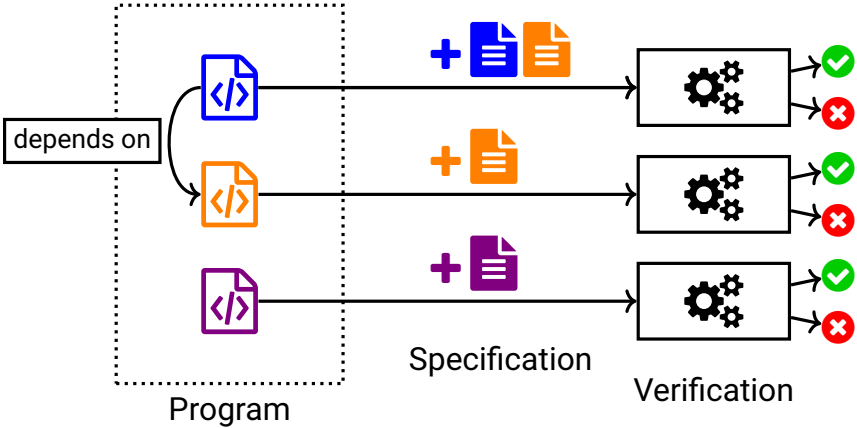


MODULARITY: KEY TO SCALABILITY



TECHNISCHE UNIVERSITÄT DARMSTADT

MODULARITY: KEY TO SCALABILITY



TECHNISCHE UNIVERSITÄT DARMSTADT

CONTRACTS AS MODULAR SPECIFICATION

Contracts

- Requirements
- Obligations

Hoare-style Contracts $\{P\}\mathtt{m}()\{Q\}$	**Responsibilities**
Precondition P	Caller
Postcondition Q	Callee

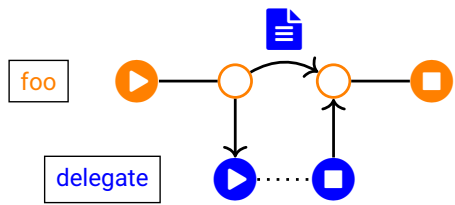
CONTRACTS AS MODULAR SPECIFICATION

Contracts	Hoare-style Contracts $\{P\}\texttt{delegate()}\{Q\}$	Responsibilities
▪ Requirements ▪ Obligations	**Precondition** P: $\{true\}$	Caller: foo
	Postcondition Q: $\{x = \text{old}(x) + 1\}$	Callee: delegate

Sequential Program

```
foo(){
    delegate();
    x=x+x;
    return;
}
```

$\{true\}delegate()\{x = \text{old}(x) + 1\}$



TECHNISCHE UNIVERSITÄT DARMSTADT

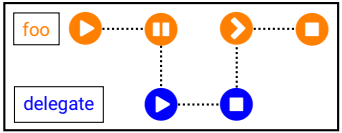
CONCURRENT PROGRAMS: CHALLENGES

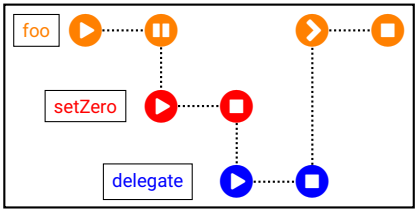
Concurrent Program
(Cooperative Scheduling)

```
foo(){
    f=!delegate();
    await(f);
    x=x+x;
    return;
}
setZero(){ x=0;return;}
```

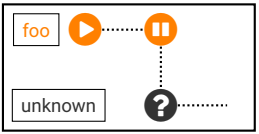
Asynchronous call

Suspension





Interference



Schedulability

Deadlock

Error

STATE OF THE ART: OVERVIEW

	Hoare Logic	Rely/Guarantee	Temporal Logic
Environment	✗	✓	✓
Procedure contracts	✓	✓	✗
Modular Verification	✓	✓/✗	✗
High Precision	✓	✗	✓
Internal behaviors	✗	✗	✓
Process-temporal behaviors	✗	✗	✗

A TRACE SEMANTICS FOR CONCURRENCY

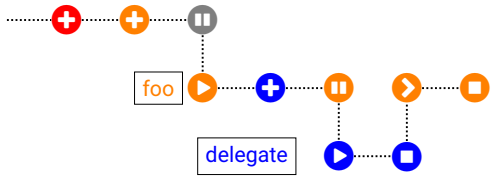
Section 2

TECHNISCHE UNIVERSITÄT DARMSTADT

TRACE SEMANTICS

Program execution: finite sequence of **states** $[\overline{x} \mapsto \overline{v}]$ and **events** (⊕,▶,⏸,⏵,■)

```
foo(){
  f=!delegate(); // ⊕ process creation
  await(f); // ⏸ suspension
  // ⏵ scheduling
  // ⏵ reactivation
  x=x+x; // state update
  return; // ■ termination
}
```



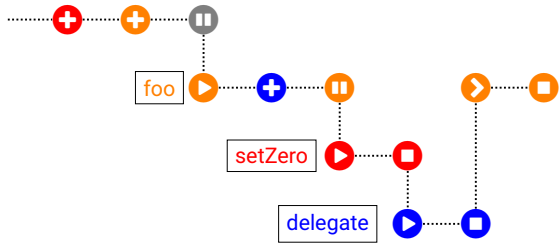
$\ldots \curvearrowright \text{invoc}(\texttt{setZero}) \curvearrowright \ldots \curvearrowright \text{invoc}(\texttt{foo}) \curvearrowright \ldots \curvearrowright \text{await}(\_)$
$\curvearrowright \text{start}(\texttt{foo}) \curvearrowright \ldots \curvearrowright \text{invoc}(\texttt{delegate}) \curvearrowright \ldots \curvearrowright \text{await}(\textit{foo}, \textit{delegate})$
$\curvearrowright \text{start}(\texttt{delegate}) \curvearrowright \ldots \curvearrowright \text{term}(\texttt{delegate}) \curvearrowright \text{react}(\textit{foo}, \textit{delegate}) \curvearrowright [x \mapsto v_x + v_x] \curvearrowright \text{term}(\textit{foo})$

TECHNISCHE UNIVERSITÄT DARMSTADT

TRACE SEMANTICS

Program execution: finite sequence of **states** $[\overline{x} \mapsto \overline{v}]$ and **events** (⊕,▷,⏸,▷,⊖)

```
foo(){
  f=!delegate(); // ⊕ process creation
  await(f); // ⏸ suspension
  // ▷ scheduling
  // ▷ reactivation
  x=x+x; // state update
  return; // ⊖ termination
}
```



$\ldots \curvearrowright \text{invoc}(\texttt{setZero}) \curvearrowright \ldots \curvearrowright \text{invoc}(\texttt{foo}) \curvearrowright \ldots \curvearrowright \text{await}(\_)$
$\curvearrowright \text{start}(\texttt{foo}) \curvearrowright \ldots \curvearrowright \text{invoc}(\texttt{delegate}) \curvearrowright \ldots \curvearrowright \text{await}(\textit{foo}, \textit{delegate})$
$\curvearrowright \text{start}(\texttt{setZero}) \curvearrowright \ldots \curvearrowright \text{term}(\texttt{setZero})$
$\curvearrowright \text{start}(\texttt{delegate}) \curvearrowright \ldots \curvearrowright \text{term}(\texttt{delegate}) \curvearrowright \text{react}(\textit{foo}, \textit{delegate}) \curvearrowright [x \mapsto v_x + v_x] \curvearrowright \text{term}(\textit{foo})$

TRACE SEMANTICS

- Program execution: finite sequence of **states** $[\overline{x} \mapsto \overline{v}]$ and **events** (➕,▶,⏸,➡,⏹)
- Program behavior as **set of traces**
- Characterization of
  - deadlock
  - errors
  - interleaving
- Properties to avoid abnormal behaviors and interleaving

A TRACE LOGIC FOR CONCURRENCY

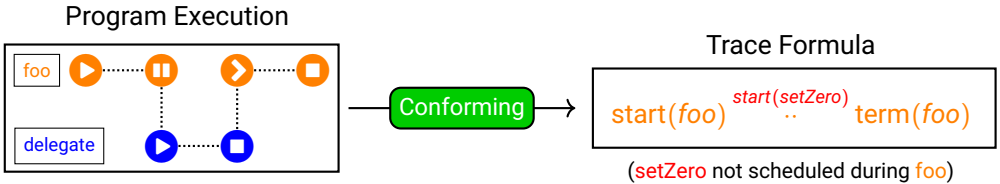
Section 3

TECHNISCHE UNIVERSITÄT DARMSTADT

TRACE LOGIC: OVERVIEW

Trace formulas (θ) represent **sets of finite program traces**

- Expressive and code independent (high reusability)
- $\overset{ev(\mathtt{m})}{\cdot\cdot}$ is set of finite traces with **no** event *ev* over `m`
- Can specify schedulability properties



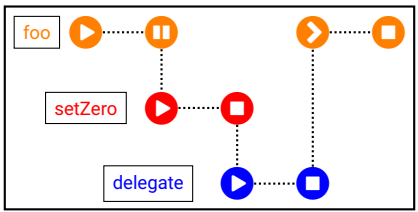
TECHNISCHE UNIVERSITÄT DARMSTADT

TRACE LOGIC: OVERVIEW

Trace formulas (θ) represent **sets of finite program traces**

- Expressive and code independent (high reusability)
- $\overset{ev(\mathtt{m})}{\cdot\cdot}$ is set of finite traces with **no** event *ev* over `m`
- Can specify schedulability properties

Program Execution



Non Conforming

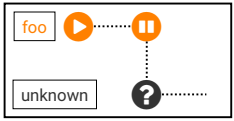
Trace Formula

$$start(foo) \overset{start(setZero)}{\cdot\cdot} term(foo)$$

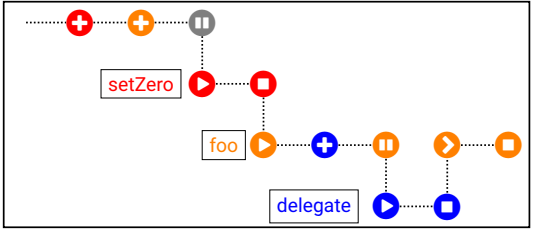
(setZero not scheduled during foo)

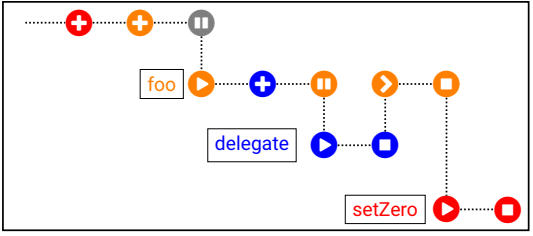
TECHNISCHE UNIVERSITÄT DARMSTADT

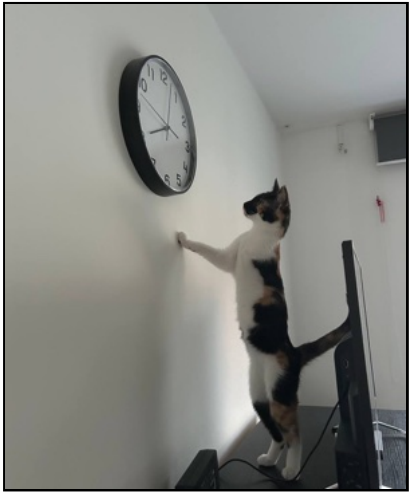
NEED FOR CONTEXT-AWARENESS



Schedulability



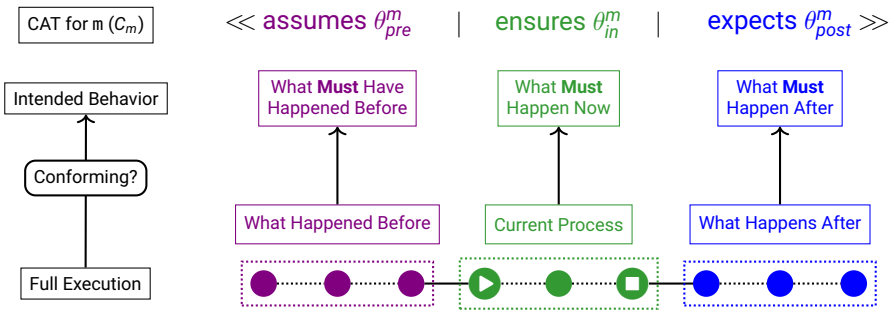




IT'S TIME FOR CATS*!

*CONTEXT-AWARE TRACE CONTRACTS

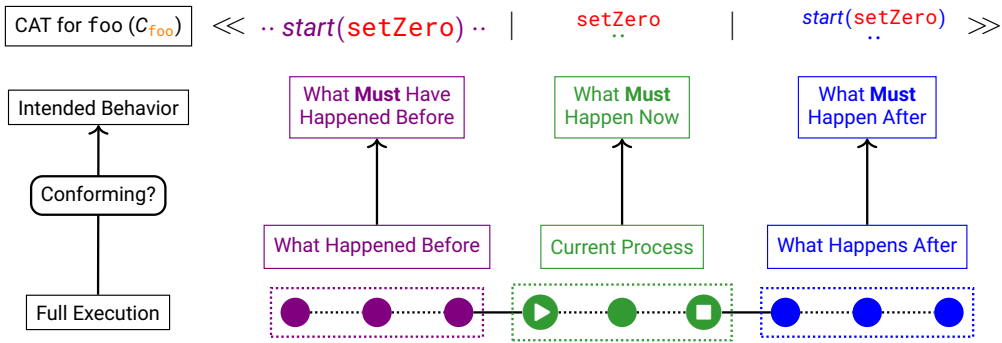
CONTEXT-AWARE TRACE CONTRACTS (CATS)



A CAT specifies global behavior with procedure-centric view.

TECHNISCHE UNIVERSITÄT DARMSTADT

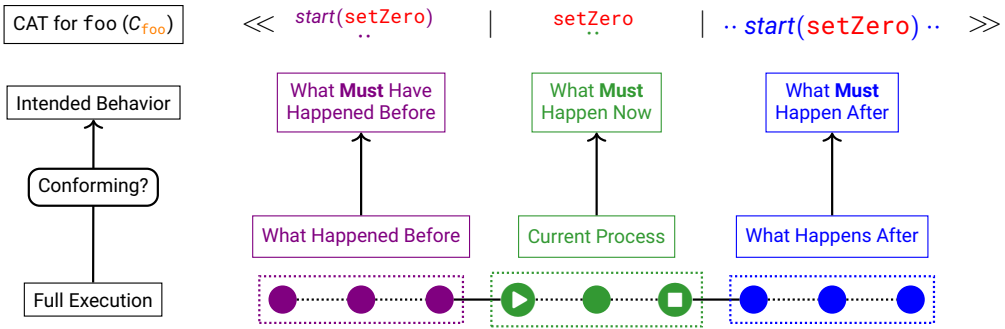
CONTEXT-AWARE TRACE CONTRACTS (CATS)



Context: setZero **before** foo

TECHNISCHE UNIVERSITÄT DARMSTADT

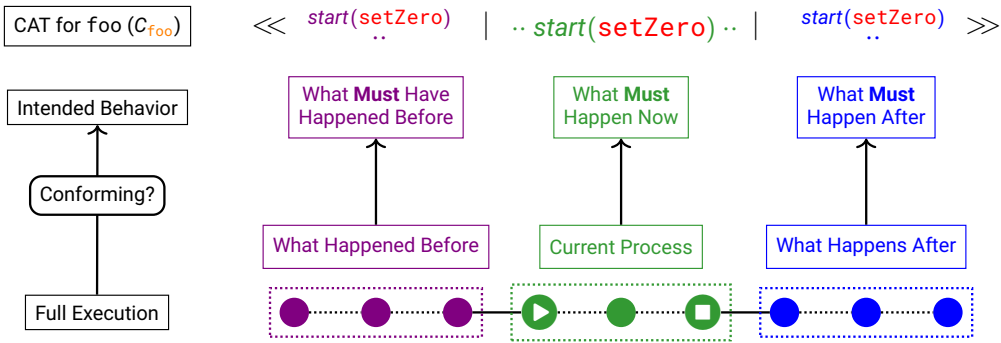
CONTEXT-AWARE TRACE CONTRACTS (CATS)



Context: setZero **after** foo

TECHNISCHE UNIVERSITÄT DARMSTADT

CONTEXT-AWARE TRACE CONTRACTS (CATS)



Context: `setZero` **not before/after** `foo`

CONTRACT VALIDITY

$$C_m = \ll \textit{assumes}\ \theta^m_{pre} \,|\, \textit{ensures}\ \theta^m_{in} \,|\, \textit{expects}\ \theta^m_{post} \gg$$

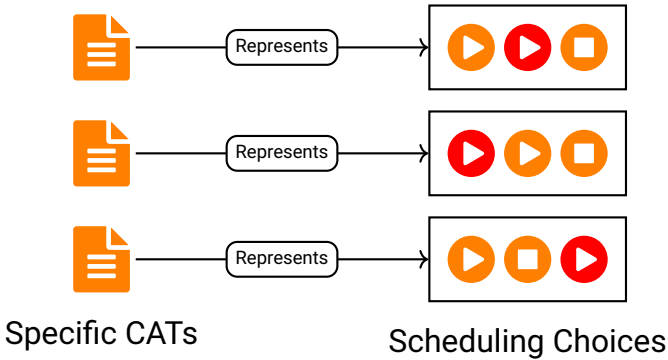
General Validity: $m \models C_m$

(Context fulfills $\theta^m_{pre}, \theta^m_{post}$) **AND** (Internal behavior fulfills θ^m_{in})

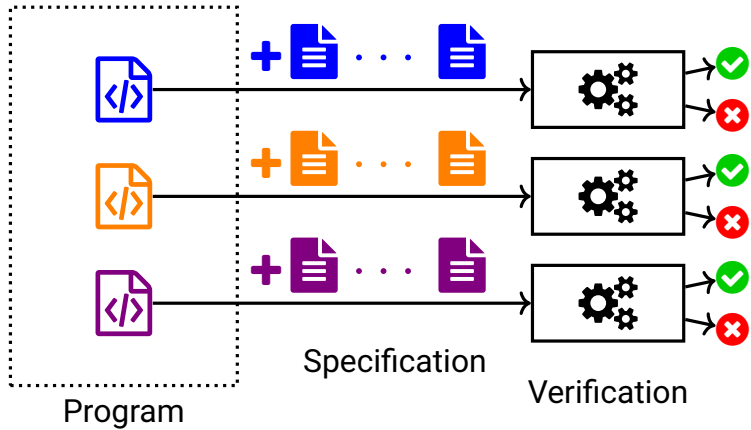
Context-dependent Validity: $m \models^{\text{ctx}} C_m$

(Context fulfills $\theta^m_{pre}, \theta^m_{post}$) $\implies$ (Internal behavior fulfills θ^m_{in})

CONTEXT-DEPENDENT CONTRACTS

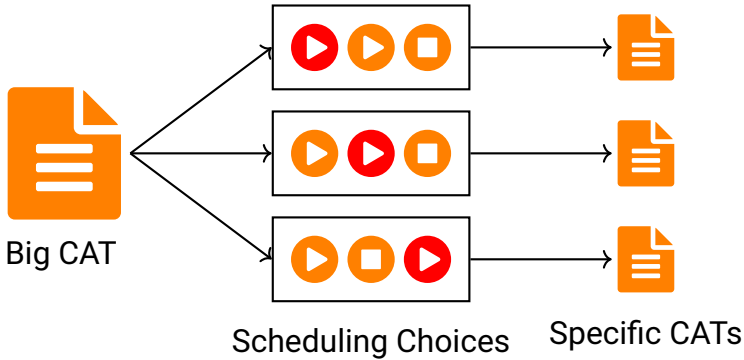


CONTEXT-DEPENDENT CONTRACTS



TECHNISCHE UNIVERSITÄT DARMSTADT

BIG CATS: OVERVIEW



TECHNISCHE UNIVERSITÄT DARMSTADT

BIG CATS: AN EXAMPLE

Same structure as a normal CAT: $C_{\texttt{foo}} = \ll \overbrace{\cdot\cdot\, invoc(\texttt{setZero})\, \cdot\cdot}^{\Phi} | \cdot\cdot | \cdot\cdot \gg$

Commitments: scheduling choices (▶) for `setZero`

- ({▶}, ∅, ∅): `setZero` **before** `foo`
- (∅,{▶}, ∅): `setZero` **during** `foo`
- (∅, ∅, {▶}): `setZero` **after** `foo`

Generating **specific CATs**:

- $C_{\texttt{foo}} + (\{▶\}, \emptyset, \emptyset) = \ll \Phi \wedge \cdot\cdot\, start(\texttt{setZero})\, \cdot\cdot \,|\, \overset{start(\texttt{setZero})}{\cdot\cdot} \,|\, \overset{start(\texttt{setZero})}{\cdot\cdot} \gg$
- $C_{\texttt{foo}} + (\emptyset, \{▶\}, \emptyset) = \ll \Phi \wedge \overset{start(\texttt{setZero})}{\cdot\cdot} \,|\, \cdot\cdot\, start(\texttt{setZero})\, \cdot\cdot \,|\, \overset{start(\texttt{setZero})}{\cdot\cdot} \gg$
- $C_{\texttt{foo}} + (\emptyset, \emptyset, \{▶\}) = \ll \Phi \wedge \overset{start(\texttt{setZero})}{\cdot\cdot} \,|\, \overset{start(\texttt{setZero})}{\cdot\cdot} \,|\, \cdot\cdot\, start(\texttt{setZero})\, \cdot\cdot \gg$

TECHNISCHE UNIVERSITÄT DARMSTADT

BIG CATS: AN EXAMPLE

Same structure as a normal CAT: $C_{\texttt{foo}} = \ll \overbrace{\cdot\cdot\, invoc(\texttt{setZero})\, \cdot\cdot}^{\Phi} \,|\, \cdot\cdot \,|\, \overset{\texttt{setZero}}{\cdot\cdot} \gg$

Commitments: scheduling choices (▶) for `setZero`

- ({▶}, ∅, ∅): `setZero` **before** `foo`
- (∅,{▶}, ∅): `setZero` **during** `foo`
- (∅, ∅, {▶}): `setZero` **after** `foo` (not applicable)

Generating **specific CATs**:

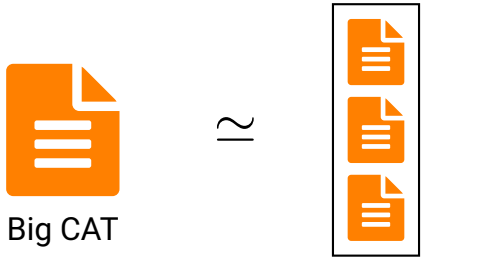
- $C_{\texttt{foo}} + (\{▶\}, \emptyset, \emptyset) = \ll \Phi \wedge \cdot\cdot\, start(\texttt{setZero})\, \cdot\cdot \,|\, \overset{start(\texttt{setZero})}{\cdot\cdot} \,|\, \overset{start(\texttt{setZero})}{\cdot\cdot} \gg$
- $C_{\texttt{foo}} + (\emptyset, \{▶\}, \emptyset) = \ll \Phi \wedge \overset{start(\texttt{setZero})}{\cdot\cdot} \,|\, \cdot\cdot\, start(\texttt{setZero})\, \cdot\cdot \,|\, \overset{start(\texttt{setZero})}{\cdot\cdot} \gg$
- $C_{\texttt{foo}} + (\emptyset, \emptyset, \{▶\}) =$ **contradiction!**

TECHNISCHE UNIVERSITÄT DARMSTADT

VALIDITY OF BIG CATS

Theorem

A Big CAT is **valid** iff all its committed CATs are **valid w.r.t. execution context** (if no interleaving)



WHY BIG CATS?

- ✔ Reduce specification effort
- ✔ Enable reusability
- ✔ Formal derivation of committed CATs
- ✔ Ensure intended coverage
- ✔ Avoid redundancy

MODULAR VERIFICATION: OVERVIEW

- **Judgments** $s : \theta$ (s is conforming to θ)
- **Proof Calculus**

$$\frac{\text{premise}_1 \quad \cdots \quad \text{premise}_n}{\text{conclusion}}$$

- **Symbolic execution**

$$\textit{(assign)}\ \frac{\Gamma \vdash \mathcal{U}\{x := e\}s : \theta}{\Gamma \vdash \mathcal{U}\ x = e; s : \theta}$$

$$\textit{(react)}\ \frac{\text{“f terminated”} \qquad \Gamma \vdash \mathcal{U}\ \{\texttt{react}(f)\}s : \theta}{\Gamma \vdash \mathcal{U}\ \textbf{\texttt{await}}(f); s : \theta}$$

$$\textit{(sched)}\ \frac{\text{“f exists”} \qquad \text{“apply contract for each schedulable process”}}{\Gamma \vdash \mathcal{U}\ \textbf{\texttt{await}}(f); s : \theta}$$

CONCLUSION

Section 4

SUMMARY

- **Trace Semantics**: characterization of concurrent behaviors
- **Trace Logic**: expressive, code independent
- **CATs**: assumptions over environment
- **Big CATs**: paradigm to specify concurrent behaviors
- **Sound Calculus**: to verify Big CATs

	Hoare Logic	Rely/Guarantee	Temporal Logic	(Big) CATs
Environment	✗	✓	✓	✓
Procedure contracts	✓	✓	✗	✓
Modular Verification	✓	✓/✗	✗	✓
High Precision	✓	✗	✓	✓
Internal behaviors	✗	✗	✓	✓
Process-temporal behaviors	✗	✗	✗	✓