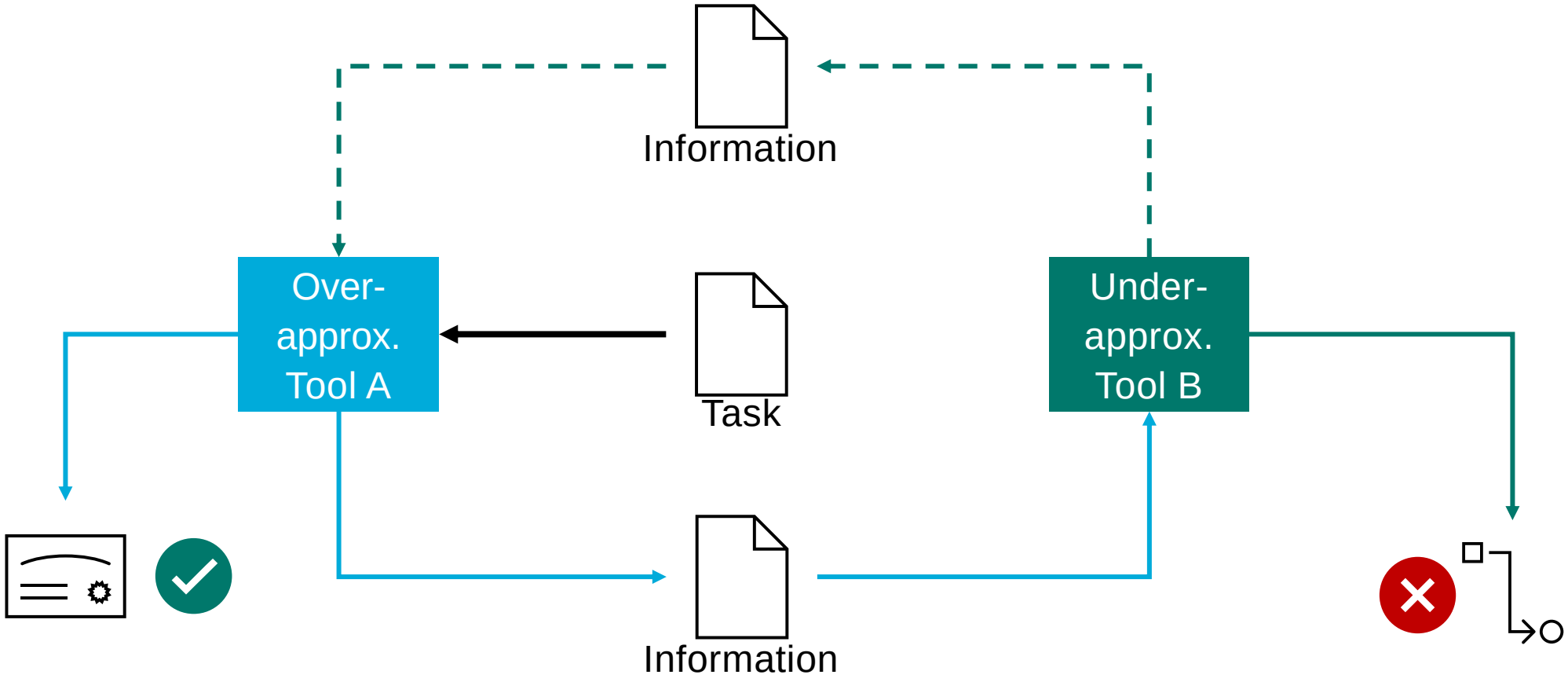
Carl von Ossietzky
Universität
Oldenburg

Unifying Cooperation of Over- and Under-Approximative Verification Techniques

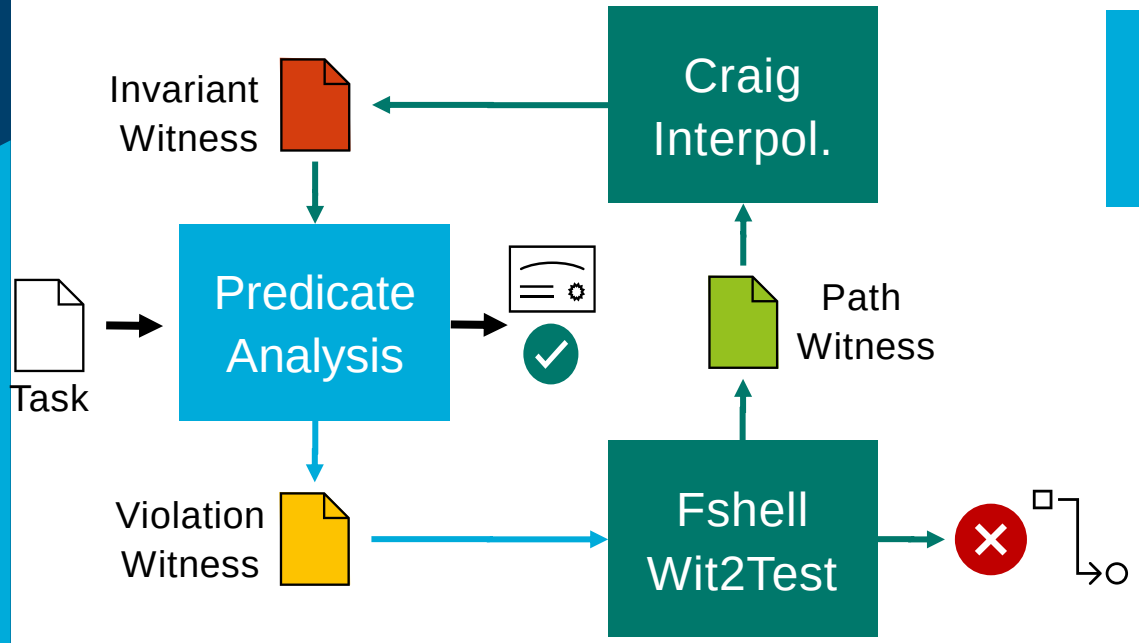
Jan Haltermann,
03.04.2022
3rd Coop Workshop

Imformation Exchange during Cooperation

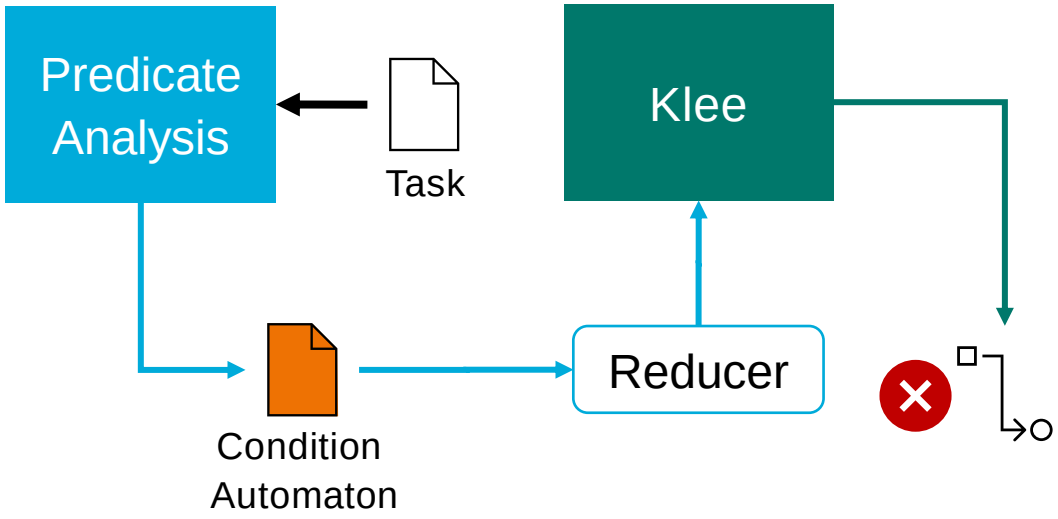


Two Exisiting Schema for Cooperation

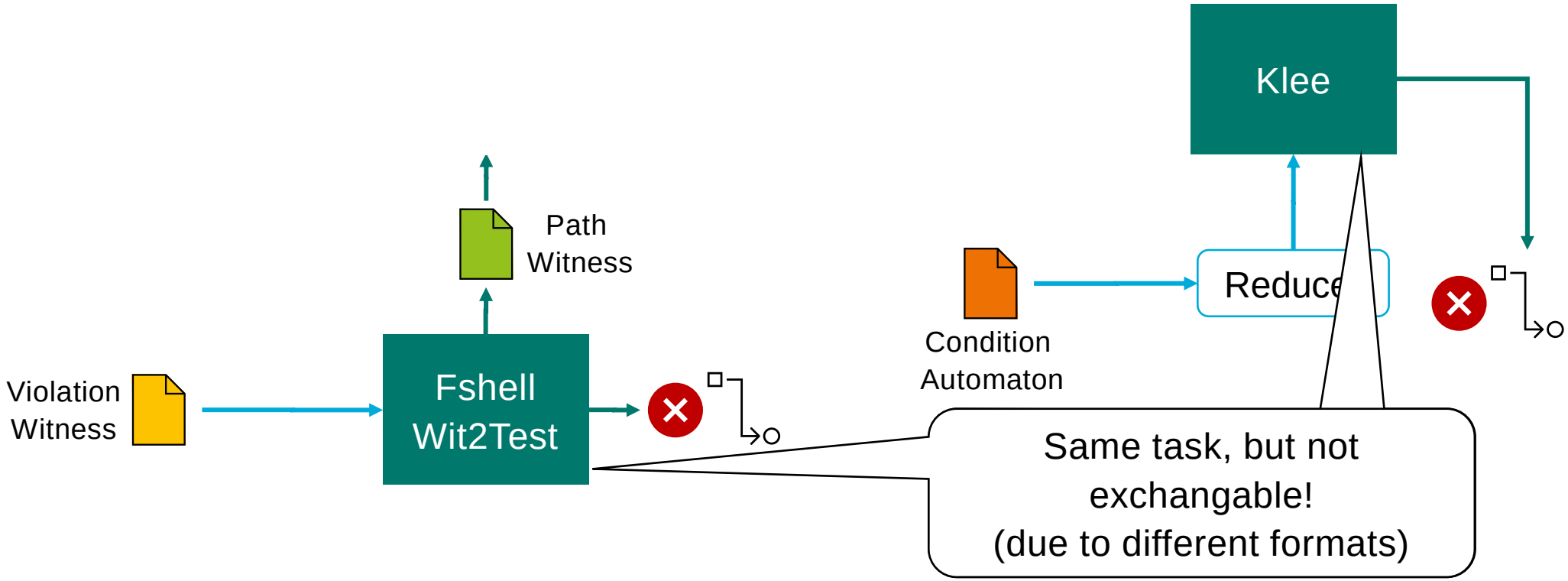
– Component-based CEGAR [BHLW22] (Instance, simplified)



– Reducer-Based Construction of Conditional Verifiers [BJLW18](Instance, simplified)



Two Exisiting Schema for Cooperation



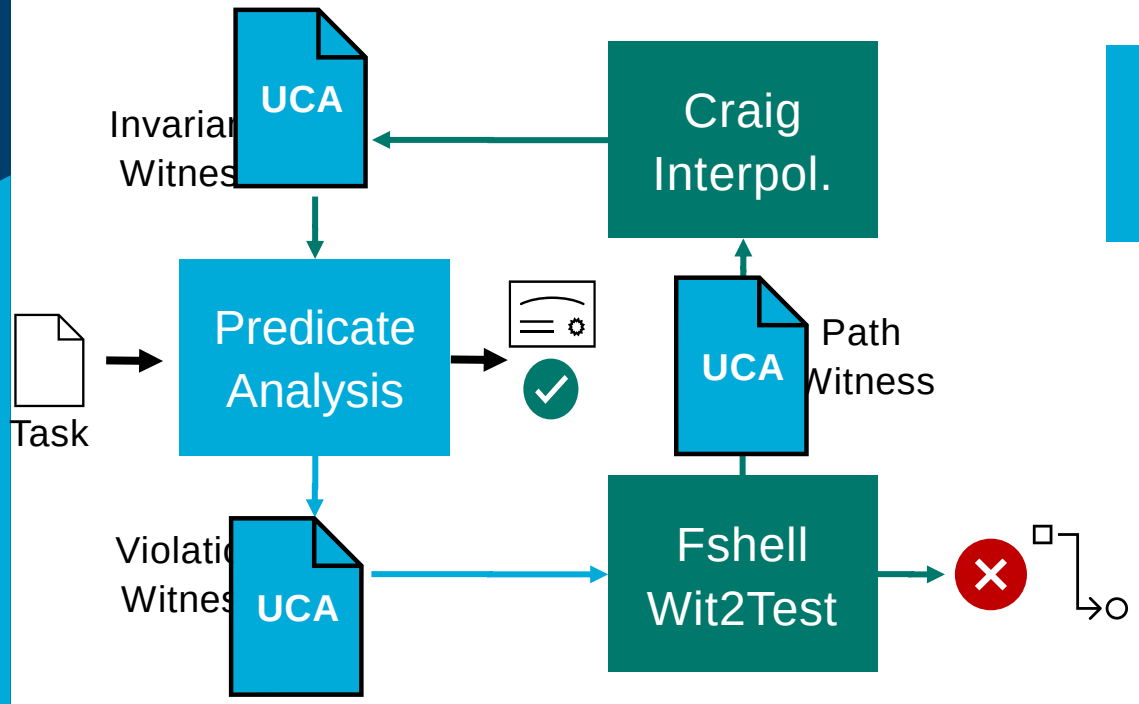
Format for information exchange

- Popular choice: existing format fitting context
  - standardized or internal
- General requeirements:
  - I
  - Precise
  - Easy to generate
  - Supported by many tools
  - …

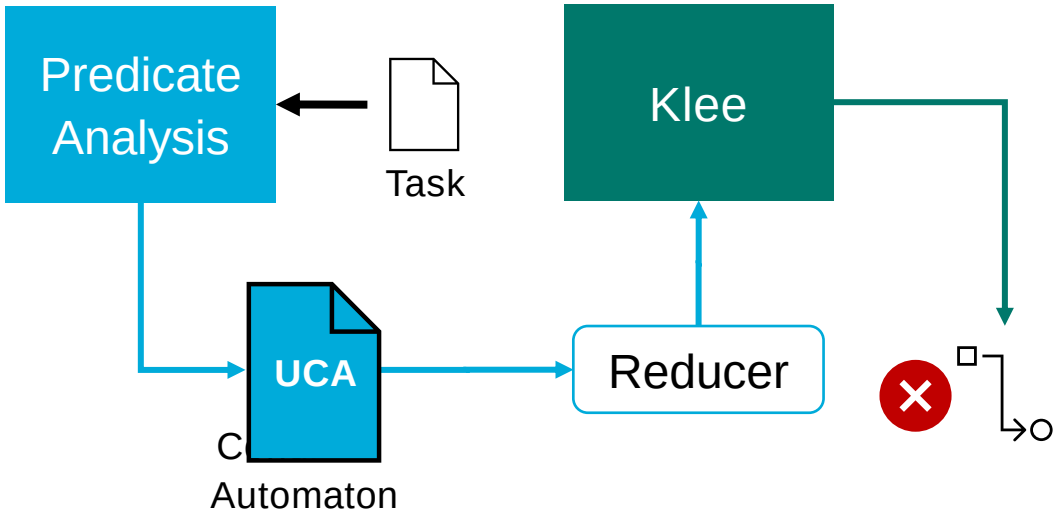
Goal: Find format that is universally applicable!

Two Exisiting Schema for Cooperation

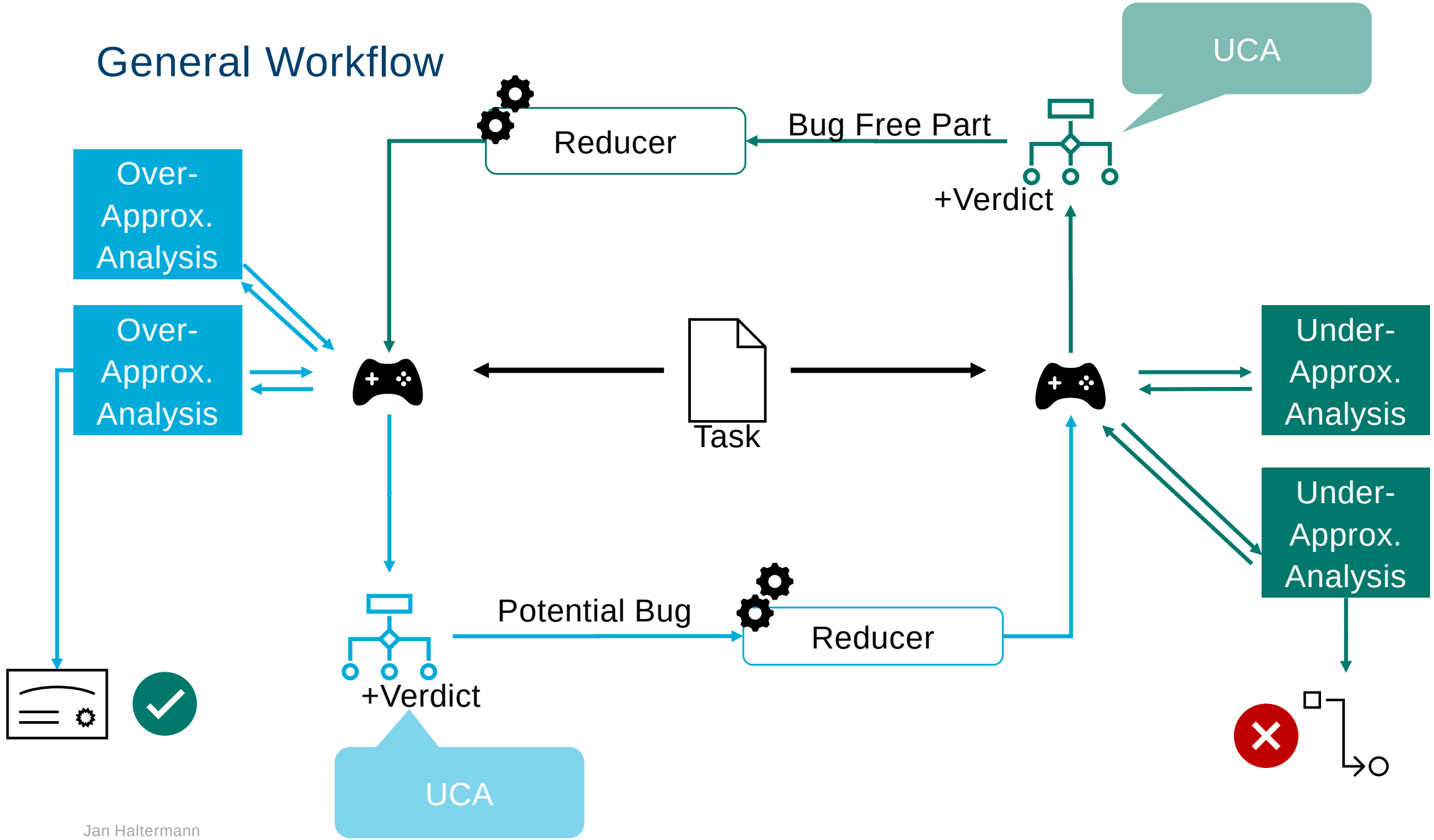
– Component-based CEGAR [BHLW22] (Instance, simplified)



– Reducer-Based Construction of Conditional Verifiers [BJLW18](Instance, simplified)



General Workflow



Idea: Universal Condition Automaton

Extension of Condition Automaton [BJLW18]:

A universal condition automaton (UCA) A = (Q, Σ,δ,q_0, F) is a NFA consisting of

- a finite set $Q \subseteq \Omega \times \Phi$ of pairs of states and invariants
  - four additional states $\{(q_{err}, true), (q_t, true), (q_n, true), (q_f, true)\}$
- an initial state $q_0 \in Q$,
- an alphabet $\Sigma \subseteq 2^G \times \Phi$ a transition relation $\delta \subseteq Q \times \Sigma \times Q$, and
- a set $F \subseteq Q$ of final states.

– Semantics: accepting path = Safe

Advantage: Expressiveness & Applicability

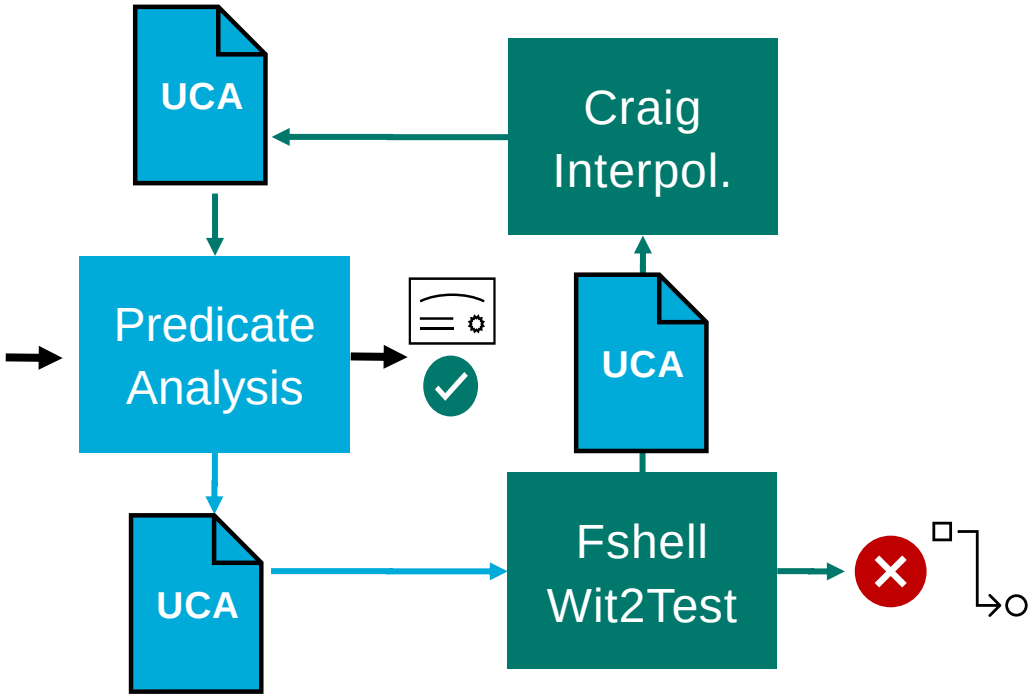
Encode existing formats as UCA without information loss:

1. Violation Witnesses (or counter-example in general)
2. Correctness Witnesses (yaml + graphml)
3. Precision Increment (Invariant Witness)
4. Condition Automaton
5. Test Input (e.g. TEST-COMP Test Suite)
6. Explored Paths
7. Invariants
8. [ARG]*

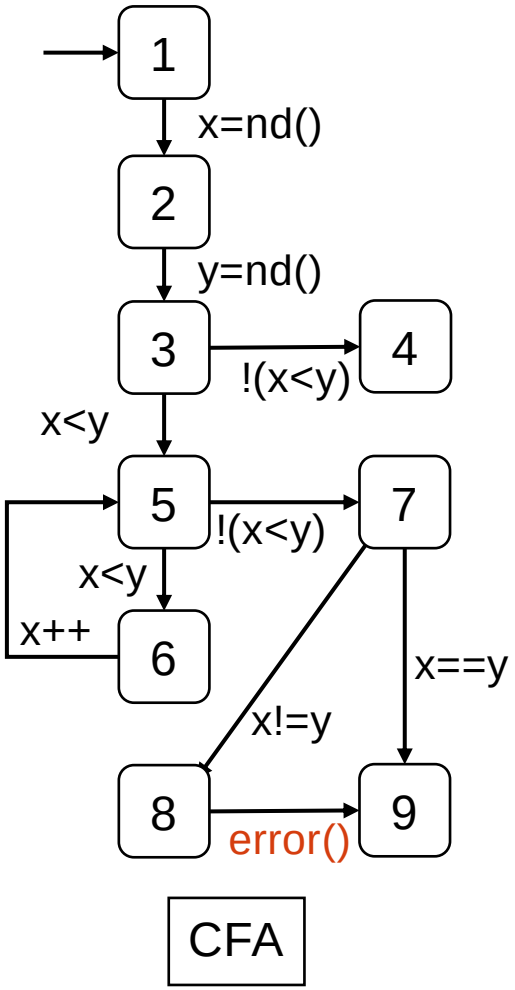
*only for certain information

Example C-CEGAR (simplified)

```
int main() {
  int x = nondet_int();
  int y = nondet_int();
  if (!(x<y)) return 0;
  while (x<y) {
    x=x+1;
  }
  assert(x==y);
  return 0;
}
```

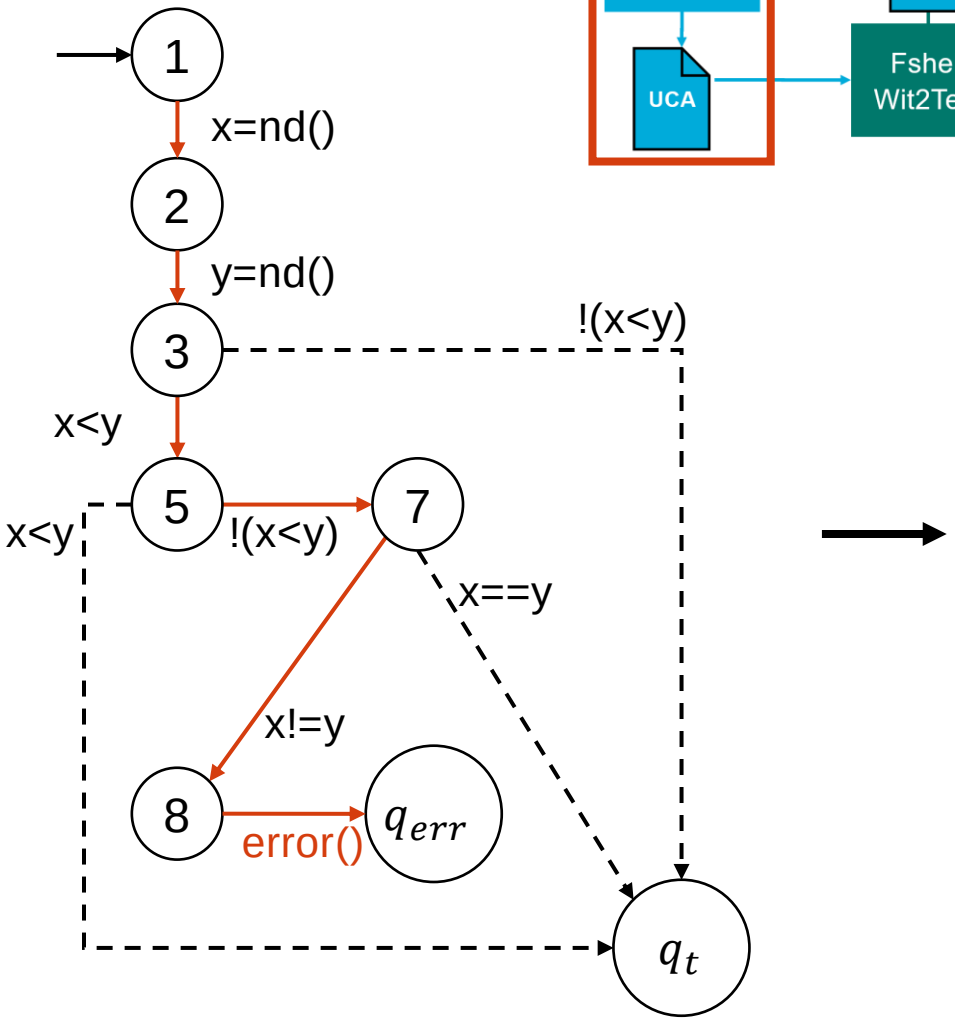


Example C-CEGAR (simplified)

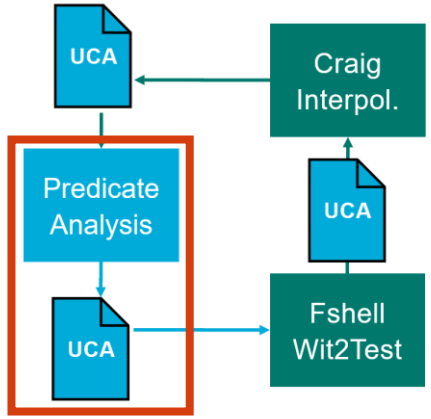


CFA

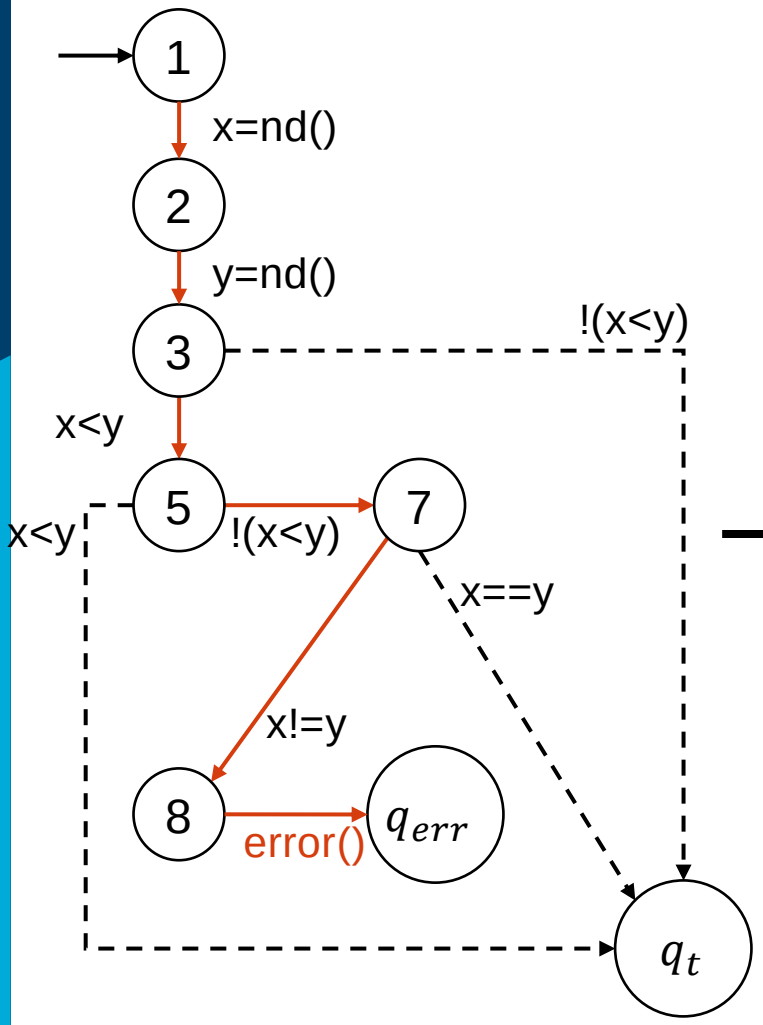
Predicate Analysis



UCA (Cex)



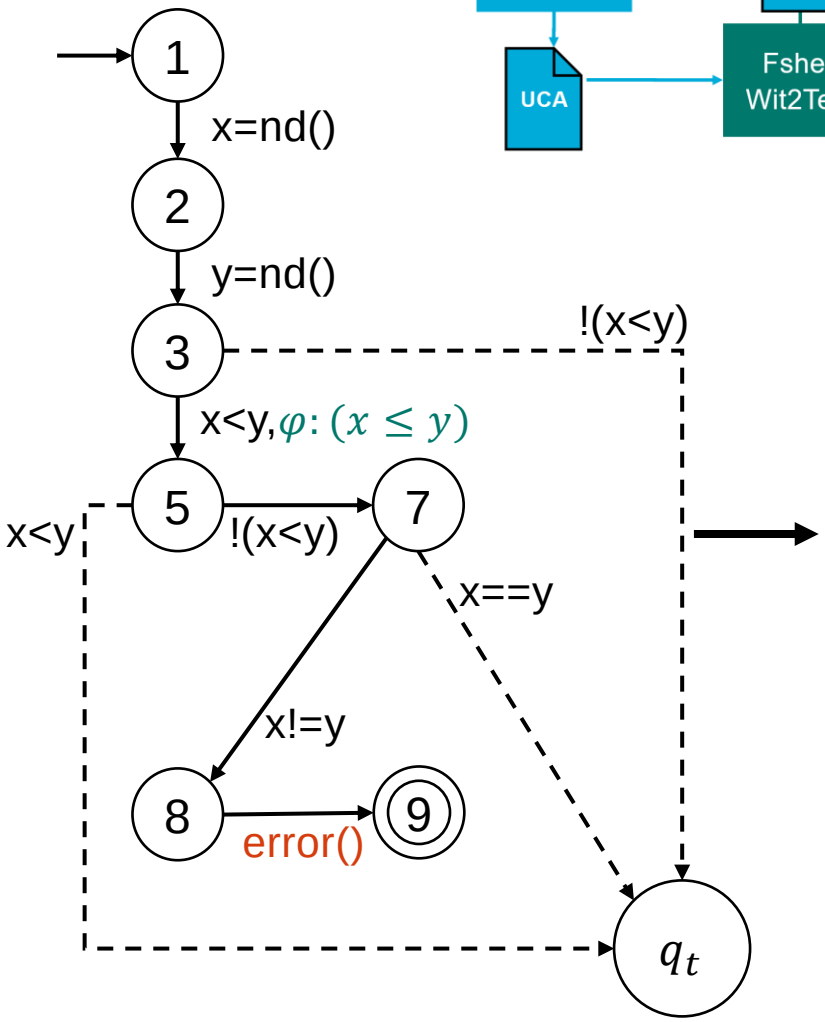
Example C-CEGAR



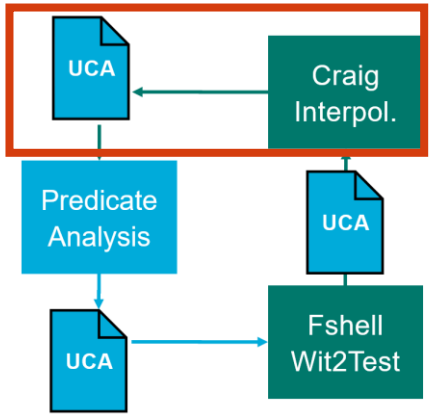
UCA (Cex)

Craig Interpol.

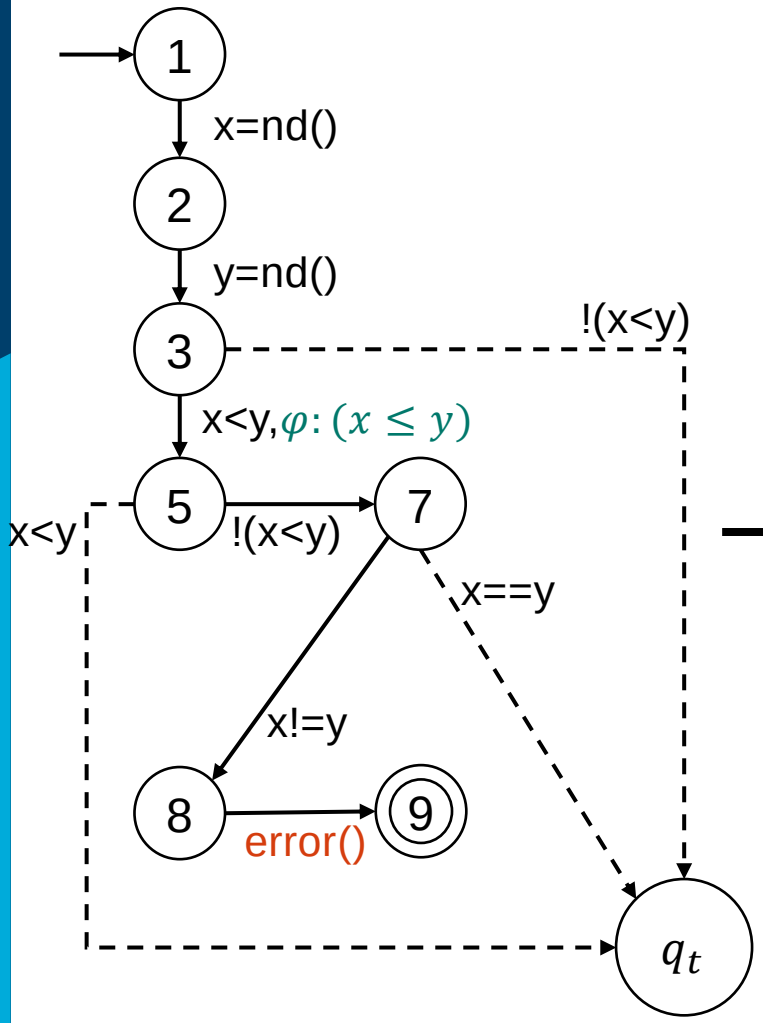
[simplified]



UCA (Prec. Incr.)

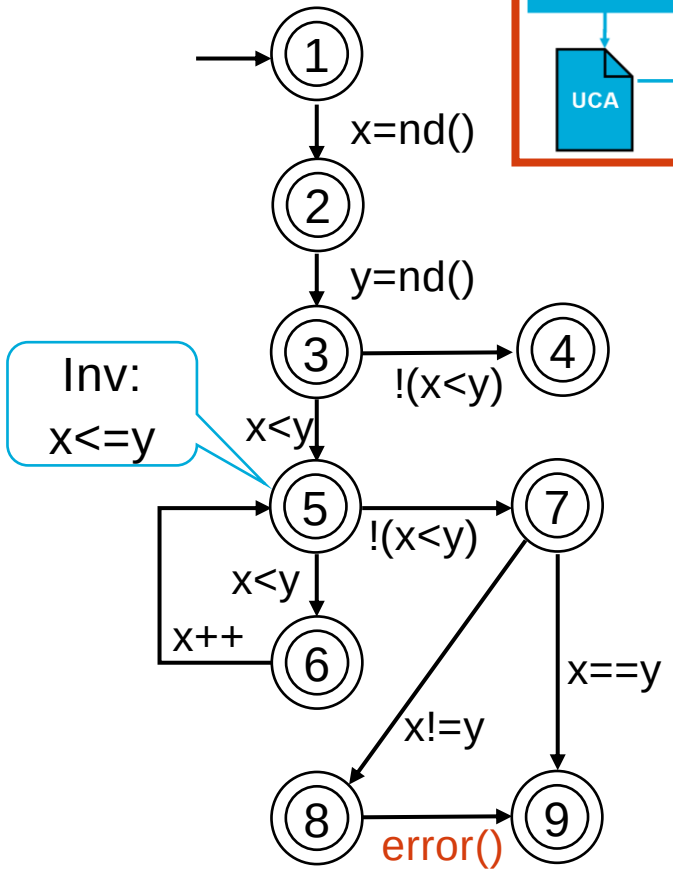


Example C-CEGAR (simplified)

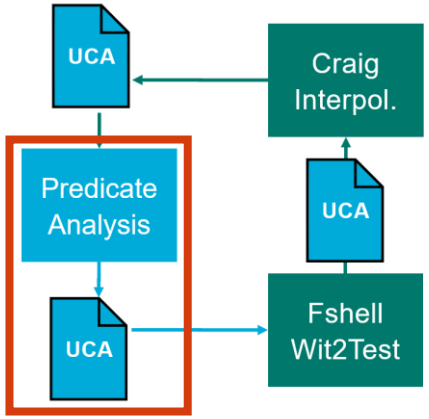


UCA (Prec. Incr.)

Predicate Analysis



UCA (CorWit)*



*Otherwise-Edges removed for presentation

Evaluation (ongoing)

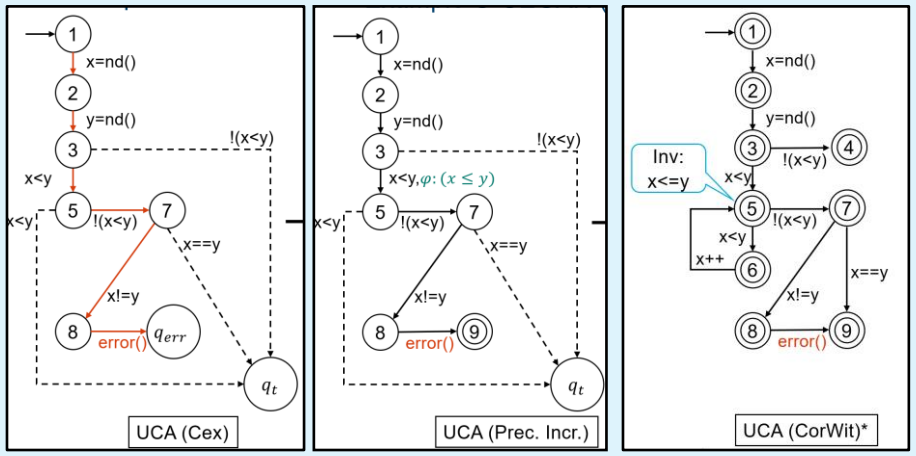
- RQ1: Are UCAs a suitable exchange format?

- RQ2: Is it beneficial to use UCAs ?

Summary

Existing Artifacts:

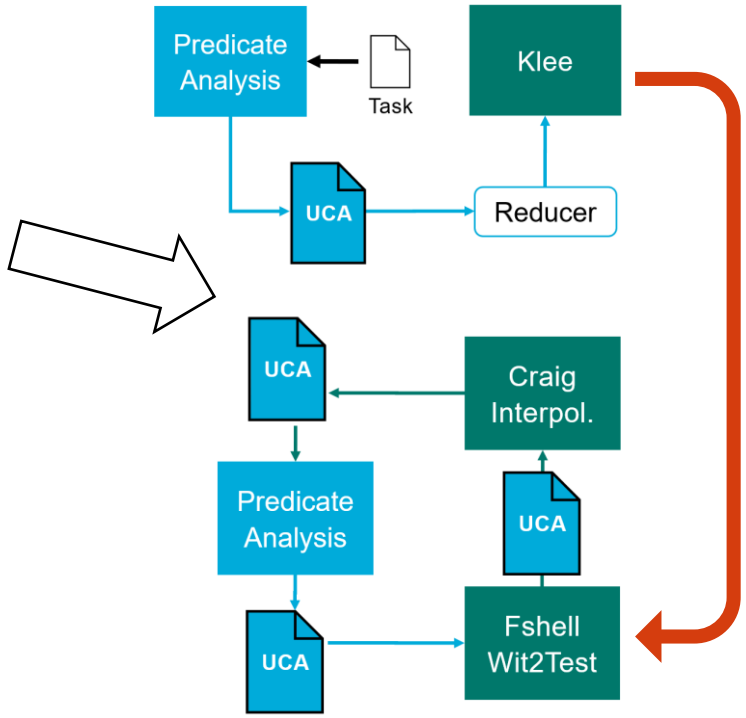
1. Violation Witnesses
2. Correctness Witnesses
3. Precision Increment
4. Condition Automaton
5. Test Input
6. Explored Paths
7. Invariants
8. [ARG]*



A universal condition automaton (UCA) A = (Q, Σ,δ,q_0, F) is a NFA:

- a finite set $Q \subseteq \Omega \times \Phi$ of pairs of states and invariants with four additional states $\{(q_{err}, true), (q_t, true), (q_n, true), (q_f, true)\}$
- an initial state $q_0 \in Q$,
- an alphabet $\Sigma \subseteq 2^G \times \Phi$ a transition relation $\delta \subseteq Q \times \Sigma \times Q$, and
- a set $F \subseteq Q$ of final states.

Universal Condition Automaton



References

[BHLW22] Beyer, D., Haltermann, J., Lemberger, T. Wehrheim, H.: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. In: ICSE 2022

[BJLW18] Beyer, D., Jakobs M.-C., Lemberger, T. Wehrheim, H.: Reducer-based construction of conditional verifiers. In ICSE 2018

[DGH16] Daca P., Gupta A., Henzinger T.A. Abstraction-driven Concolic Testing. In: VMCAI 2016