Scaling Up Automated Verification:

A Case Study and A Formalization IDE for Building High Integrity Software

Daniel Welch—Clemson University, USA (Advisor: Murali Sitaraman)

Problem

Goal: Automated and scalable verification of component-based software

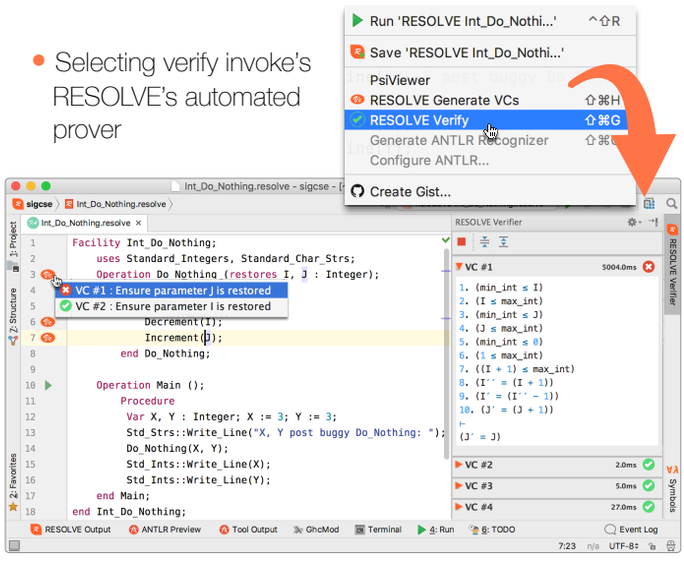
- Language must support formal contracts
- Not possible in existing popular programming languages without additions: Dafny, Eiffel, KeY
- **Research Contribution #1:** An IDE that lets you specify and verify components
- **Research Contribution #2:** A case study involving components using a variety of mathematical theories to show scalability

A Formalization IDE

An IDE built on the JetBrains [4] platform that is integrated with the RESOLVE verifying compiler

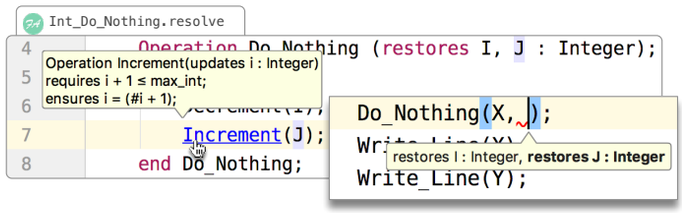
Automated Verification

- Selecting verify invoke's RESOLVE's automated prover



Software Engineering Education

- Design by contract assistance

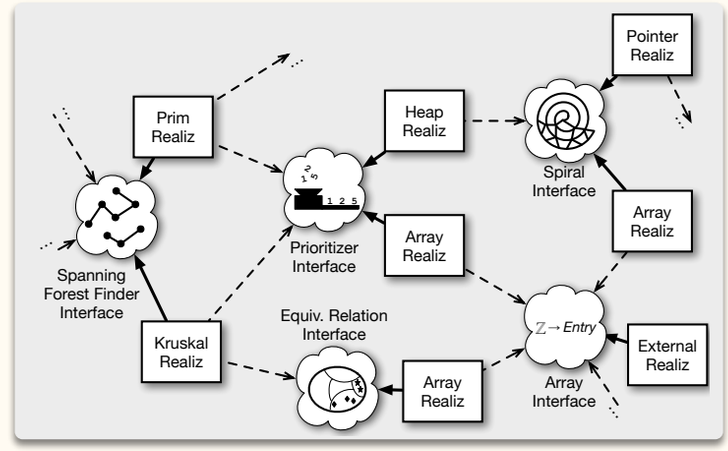


- Parameter mode awareness
- Code generation

- Reference completion

A Case Study: Spanning Forests

Component-based implementation using formally specified and verified components in RESOLVE



Mathematical Modeling

- Each component is equipped with a mathematical, conceptual specification:

Prioritizer.resolve

```
12  Type family Entry_Keeper is modeled by Cart_Prod
13          Entry_Tally : FMSet(Entry);
14          Is_Accepting : ℙ;
15      end;
16      exemplar K;
17      constraints ||K.Entry_Tally|| ≤ Max_Capacity;
18      initialization
19          ensures K.Is_Accepting ∧ K.Entry_Tally = Φ;
```

Formal, Extensible Contracts

- Specifications are made extensible via user-defined mathematical theory modules that simply contracts:

Multiset_Theory.resolve

```
30  // Union Plus
31  Def (M : MSet) ⊎ (N : MSet) : MSet; /*...*/
32    Corollary UP_1: ∀ M, N : MSet, M ⊎ N = N ⊎ M;
      Corollary UP_2: ∀ L, M, N : MSet, L ⊎ (M ⊎ N) =
```

Prioritizer.resolve

```
22  Operation Add_Entry(clears x : Entry;
23                      updates K : Entry_Keeper);
24  requires ||K.Entry_Tally|| + 1 ≤ Max_Capacity ∧
25           K.Is_Accepting;
26  ensures K.Entry_Tally = #K.Entry_Tally ⊎ \#x/ ∧
27           K.Is_Accepting;
```

Future and Ongoing Work

- Analysis and verification of proof obligations arising from case-study component realizations
- Evaluation of the IDE and its features in the SE curriculum at Clemson University

References

[1] D. Welch and C. T. Cook, Y. Sun and M. Sitaraman. A web integrated verifying compiler for RESOLVE: a research perspective. In: D. Janakiram, K. Sen, and V. Kulkarni (eds). ISEC 2014. ACM.
[2] M. Kabbani, D. Welch et al. Formal Reasoning Using an Iterative Approach with an Integrated Web IDE. F-IDE 2015: 56-71.
[3] D. Welch and M. Sitaraman. Engineering and Employing Reusable Software Components for Modular Verification. In: Werner C., Botterweck G. (eds) ICSR 2017. LNCS. (to appear)
[4] JetBrains: IDEs. Software product line, available on https://www.jetbrains.com/

This research is funded in part by NSF grants CCF-0811748, CCF-1161916, and DUE-1022941