

Thread-Modular Abstract Interpretation for Multi-Threaded Code



Michael Schwarz Simmo Saan Helmut Seidl Julian Erhard Vesal Vojdani

Department of Informatics, Technical University of Munich

Setting

Non-relational, thread-modular analysis of the values of global variables in C programs

Approaches from literature

Miné's style (e.g. [2])

Vojdani's style (e.g. [4, 5])

- Propagate values from unlocks of a mutex to its locks, provided appropriate side-conditions are met
- Compute set of *protecting mutexes* always held when global is accessed
- Publish value on unlock of last protecting mutex

```
main:
lock(b); g = 0; unlock(b);
y = create(t1);
lock(a);
lock(b);
x = g;
...

t1:
lock(a);
lock(b);
g = 42;
unlock(a);
g = 17;
unlock(b);
```

- Miné: $x \mapsto \{0, 17, 42\}$
- Vojdani: $x \mapsto \{0, 17\}$
- Generally, **incomparable!**

Contributions

- Formulation of both styles in a common framework
- Comparison
- Principled soundness proofs for both styles of analyses
- Identify weaknesses and propose improved versions

Side-Effecting Equation Systems[1]

Accumulate flow-insensitive information for globals during flow-sensitive analysis of locals



$$\llbracket u, \text{unlock}(b) \rrbracket^\# \eta = \text{Let } \sigma' = \dots \text{ In } \underbrace{\{[g] \mapsto (\eta[u])g \mid g \in \dots\}}_{\text{Side-Effects}}, \sigma' \underbrace{\sigma'}_{\text{Contribution to } [v]}$$

Ingredients for More Precise Analyses

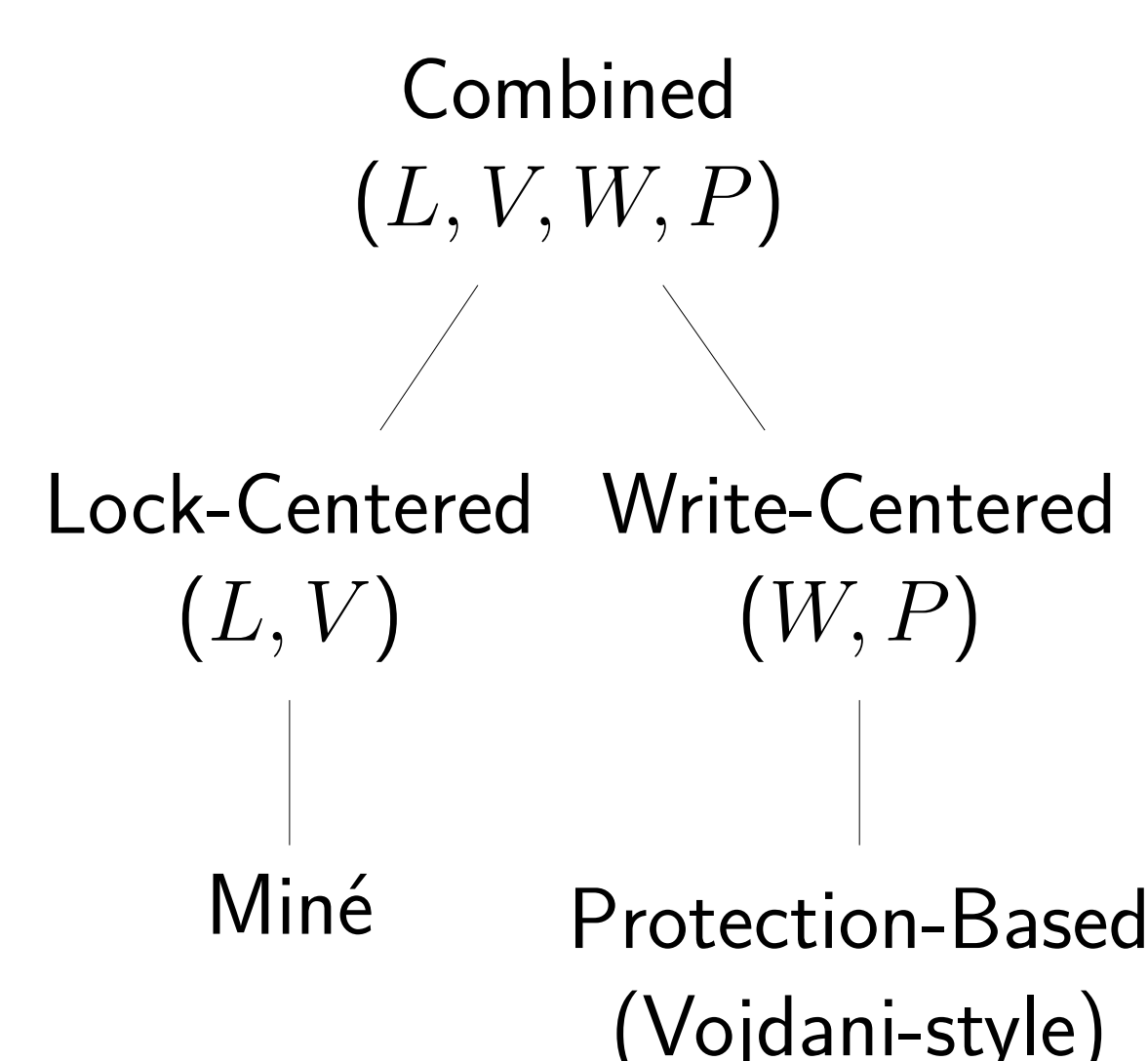
Consider further finite abstraction to exclude more reads

For each global g

- Wg : Set of locksets held *when* last writing to g
- Pg : Set of locksets held *since* last writing to g

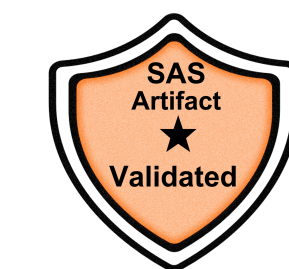
For each mutex a

- La : Set of locksets held when last acquiring a
- Va : Set of globals that must have been written locally since last acquiring a



Experimental Evaluation

All analyses implemented within the static analysis framework for multi-threaded C programs GOBLINT[5].



Benchmarks: 13 not-too-small multi-threaded POSIX programs

Runtime: Increases with sophistication

Protection-Based Miné ([2]) Lock-Centered Write-Centered Combined

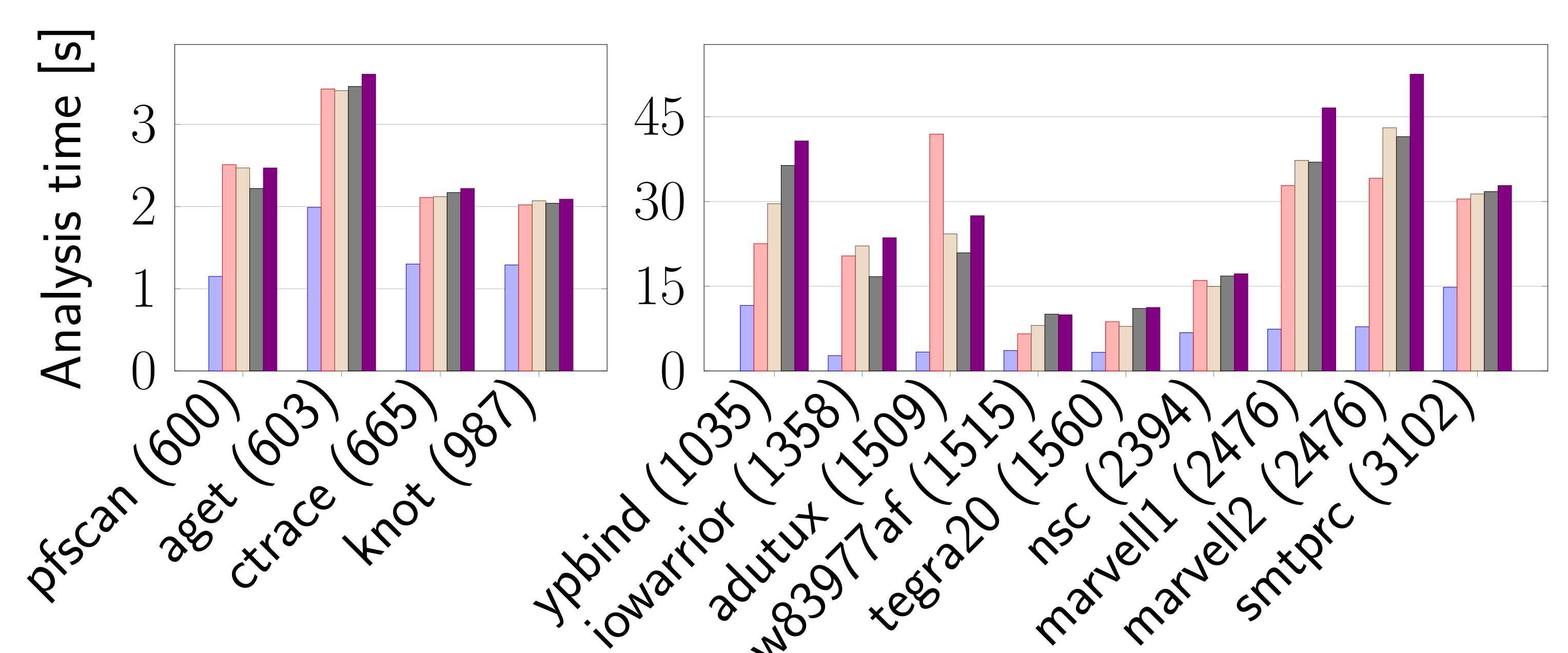


Figure 1: Analysis times per benchmark program (logical LoC in parentheses).

Precision: (as measured by abstract values of globals read)

- Equally precise for 11/13 benchmarks
- For pfscan and ypbind: [2] less precise for 6% resp. 16% of globals

Experimental Conclusions

Protection-Based Analysis sufficiently precise at low cost.

Future Work

- Take thread *ids* into account
- Extend to handle relational analyses and Weak Memory effects
- Further refinement based on abstraction of local traces

References

- Kalmer Apinis, Helmut Seidl, and Vesal Vojdani. Side-effecting constraint systems: a swiss army knife for program analysis. In *APLAS '12*, pages 157–172. Springer, 2012.
- Antoine Miné. Static analysis of run-time errors in embedded real-time parallel C programs. *Logical Methods in Computer Science*, 8(1):1–63, mar 2012.
- Michael Schwarz, Simmo Saan, Helmut Seidl, Kalmer Apinis, Julian Erhard, and Vesal Vojdani. Improving thread-modular abstract interpretation. In *SAS '21*, volume 12913 LNCS, pages 359–383. Springer, 2021.
- Vesal Vojdani. *Static Data Race Analysis of Heap-Manipulating C Programs*. PhD thesis, University of Tartu., December 2010.
- Vesal Vojdani, Kalmer Apinis, Vootele Rõtov, Helmut Seidl, Varmo Vene, and Ralf Vogler. Static Race Detection for Device Drivers: The Goblint Approach. In *ASE '16*, pages 391–402. ACM, 2016.