Goblint & GobPie

Towards Usable Data Race Verification

Vesal Vojdani, CHESS Industry Day
with Karoliine Holter, Simmo Saan (Uni Tartu)
and Julian Erhard, Sarah Tilscher, Michael Schwarz, Helmut Seidl (TU Munich)

Multiple Access data race

Two threads simultaneously access same memory location...

$$
\begin{aligned}
T_1 :\ & \texttt{lock}(\&l_1); \\
& v = v + 1; \\
& \texttt{unlock}(\&l_1); \\
T_2 :\ & \texttt{lock}(\&l_1); \\
& v = v + 1; \\
& \texttt{unlock}(\&l_1);
\end{aligned}
$$

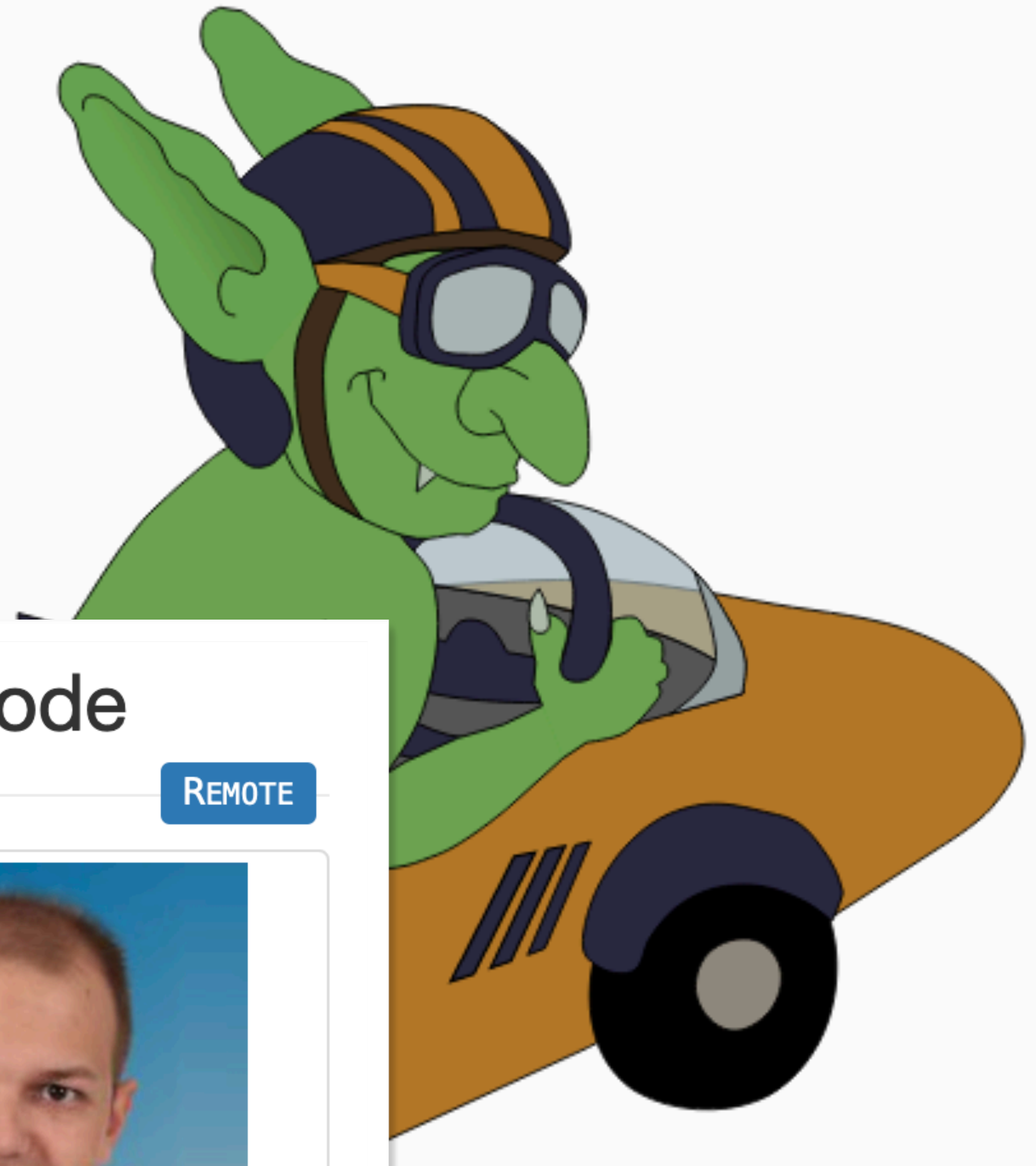
- List of accesses:
  $\langle v, \{l_1\}, \text{write}, \texttt{file.c} : 2 \rangle$
  $\langle v, \{l_1\}, \text{write}, \texttt{file.c} : 5 \rangle$
- $\mathtt{v}$ is protected by $\{l_1\}$.

$$
\begin{aligned}
T_1 :\ & \texttt{lock}(\&l_1); \\
& v = v + 1; \\
& \texttt{unlock}(\&l_1); \\
T_2 :\ & \texttt{lock}(\&l_2); \\
& v = v + 1; \\
& \texttt{unlock}(\&l_2);
\end{aligned}
$$

- List of accesses:
  $\langle v, \{l_1\}, \text{write}, \texttt{file.c} : 2 \rangle$
  $\langle v, \{l_2\}, \text{write}, \texttt{file.c} : 5 \rangle$
- No common lock!

Goblint: Race Detection World Champion

Potentially somewhat useful as well …



1. **Goblint (1304)**
2. Deagle (1211)
3. Dartagnan (768)
4. UAutomizer (756)
5. UGemCutter (732)
6. UTaipan (612)
7. CPAchecker (400)
8. Locksmith (226)
9. Theta (205)
10. ...

Targeted Static Analysis for OCaml C Stubs: Eliminating gremlins from the code

REMOTE

Track OCaml 2023

When **Sat 9 Sep 2023 10:07 - 10:30 at Grand Crescent** - Session 1 Chair(s): Benoît Montagu

Abstract Migration to OCaml 5 requires updating a lot of C bindings due to the removal of naked pointer support. Writing OCaml user-defined primitives in C is a necessity, but is unsafe and error-prone. It does not benefit from either OCaml's or C's type checking, and existing C static analysers are not aware of the OCaml GC safety rules, and cannot infer them from existing macros alone. The alternative is automatically generating C stubs, which requires correctly managing value lifetimes. Having a static analyser for OCaml to C interfaces is useful outside the OCaml 5 porting effort too.

After some motivating examples of real bugs in C bindings a static analyser is presented that finds these known classes of bugs. The tool works on the OCaml abstract parse and typed trees, and generates a header file and a caller model. Together with a simplified model of the OCaml runtime this is used as input to a static analysis framework, Goblint. An analysis is developed that tracks dereferences of OCaml values, and together with the existing framework reports incorrect dereferences. An example is shown how to extend the analysis to cover more safety properties.

The tools and runtime models are generic and could be reused with other static analysis tools.

Edwin Török

XenServer, Cloud Software Group

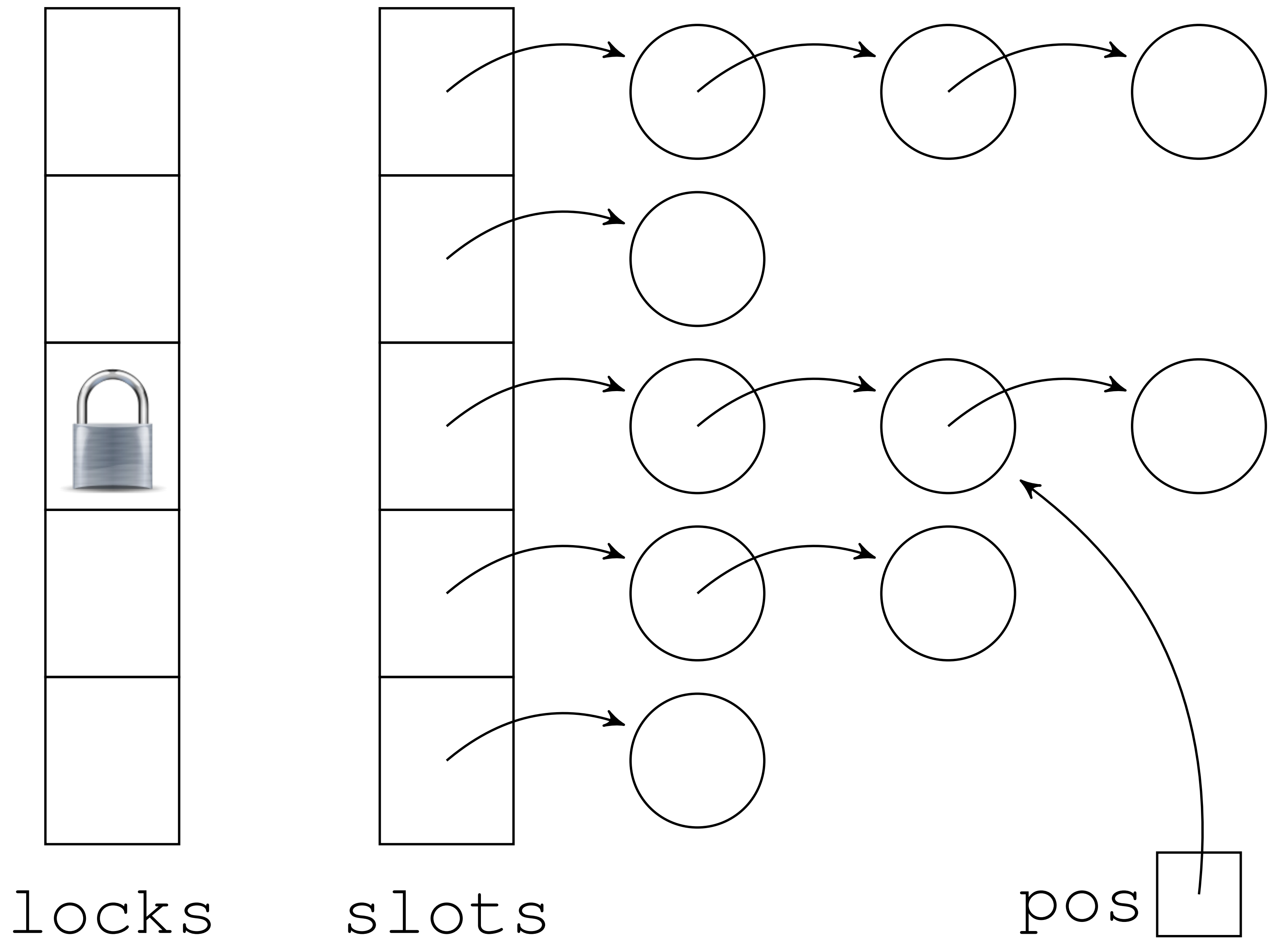
Why just "somewhat useful"

- When running our SV-COMP configuration on top Github repos.
- Inconclusive results in 2h.
- One clearly needs precision adjustment…
- And targeted abstractions!

Goblint 2023-11-10 16:13:46 UTC goblint.svcomp.Concrat						
status	cputime (s)	walltime (s)	memory (MB)	safe	vulnerable	unsafe
Show all	Min:Max	Min:Max	Min:Max	Min	Min:Max	Min:Max
TIMEOUT	7200	7200	13800			
unknown	16.3	16.3	49.7	50	1	32
TIMEOUT	7200	7200	14200			
unknown	48.6	48.6	360	29	0	9
unknown	3480	3480	5730	118	0	6
TIMEOUT	7200	7200	9950			
TIMEOUT	7200	7200	979			
true	.186	.187	26.6	2	0	0
EXCEPTION (Stack overflow)	90.1	90.1	384			
EXCEPTION (Failure)	.173	.173	30.9			
unknown	2320	2320	11400	233	1	18
EXCEPTION (Stack overflow)	1120	1120	3160			
TIMEOUT	7200	7200	21600			
EXCEPTION (Stack overflow)	939	939	3240			
EXCEPTION (Stack overflow)	34.6	34.6	191			
EXCEPTION (Stack overflow)	1120	1120	7660			
SEGMENTATION FAULT	2320	2320	3880			
true	4.86	4.86	42.4	83	0	0

Strength: Locks

- Infer the locked addresses
  $\texttt{locks}[i]$
- Information about pointers
  $\texttt{pos} \in \texttt{slot}[i]$
- Disjointness information
  $\texttt{slot}[i] \cap \texttt{slot}[j] = \emptyset$



Failings…

Non-Locking Concurrency

- Thread-creation and joining
- Data segmentation
- When the number of threads is unbounded, this is hard…
- These “real-world” race challenges were submitted to SV-COMP!
- https://github.com/goblint/bench/blob/master/concrat/race-challenges/README.md

```c
int *datas;

void *thread(void *arg) {
  int i = (int)arg;
  datas[i] = …; // No locking needed
  return NULL;
}

int main() {
  int threads_total = __VERIFIER_nondet_int();

  pthread_t *tids = malloc(threads_total * sizeof(…));
  datas = malloc(threads_total * sizeof(int));

  // create threads
  for (int i = 0; i < threads_total; i++) {
    pthread_create(&tids[i], NULL, &thread, (void*)i);
  }

  // join threads
  for (int i = 0; i < threads_total; i++) {
    pthread_join(tids[i], NULL);
  }

  // compute with data — no locking needed

}
```

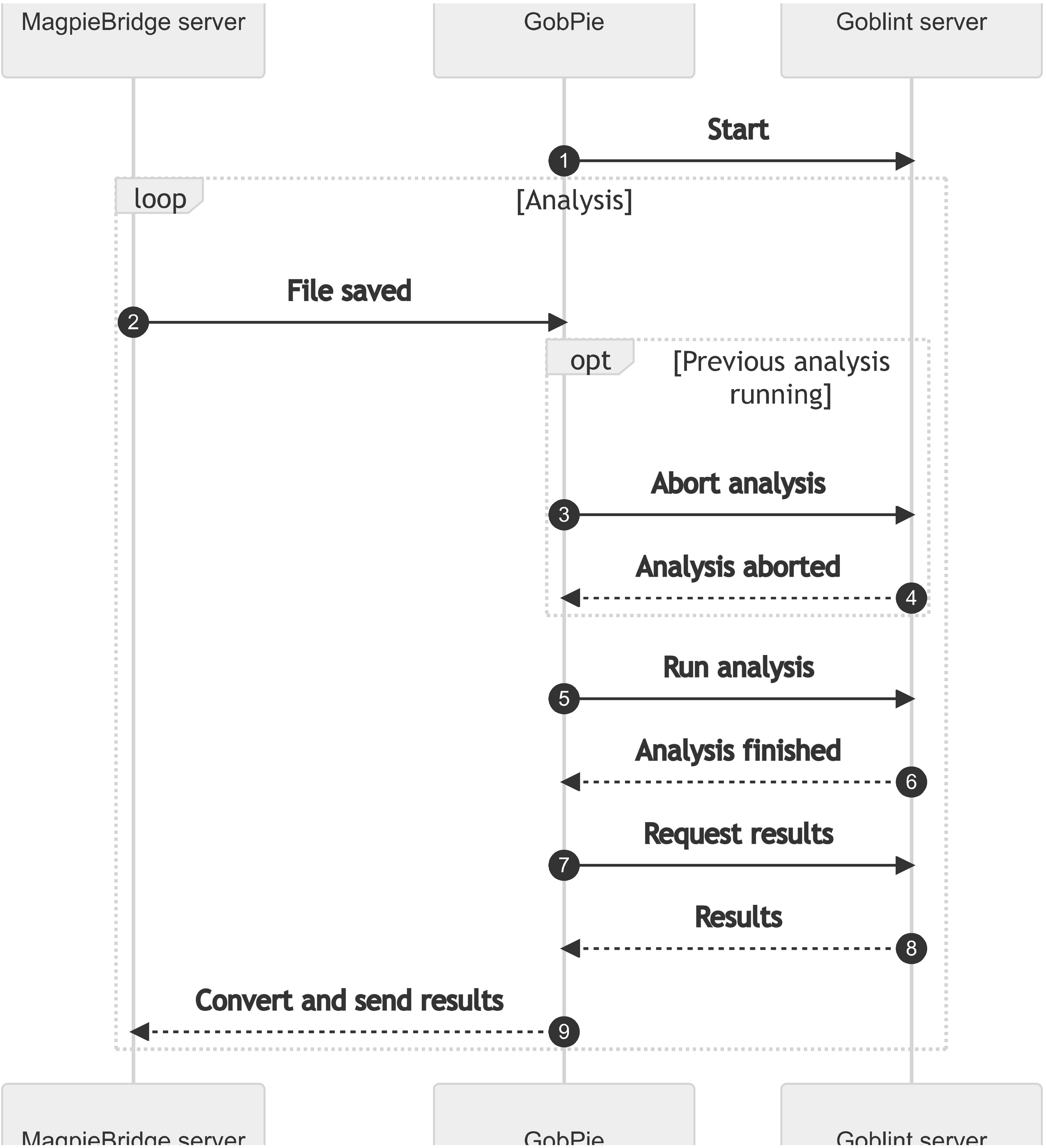
Towards usable analysis

Feedback from one industrial partner…

- False alarm can still be inisightful and useful part of code review, but…
- “Why do you produce so many warnings about the same issue?”
- We (and perhaps others too) have not paid that much attention to explaining verification outcomes.
- Heuristic analyzers are much nicer and give actionable feedback.

Interactive analysis

- Incremental abstract interpretation
- GUI integration via MagPie Bridge
- Server mode for Goblint
- Astronomical speedup
  (for superficial analysis)
- Modest speedup for SV-COMP quality analyses



Abstract Debugging

More familiar experince of analysis results… (??)