2015, LDV Workshop, Moscow

LinuxTesting.org

Finding Several Bugs at once

Vitaly Mordan
mordan@ispras.ru

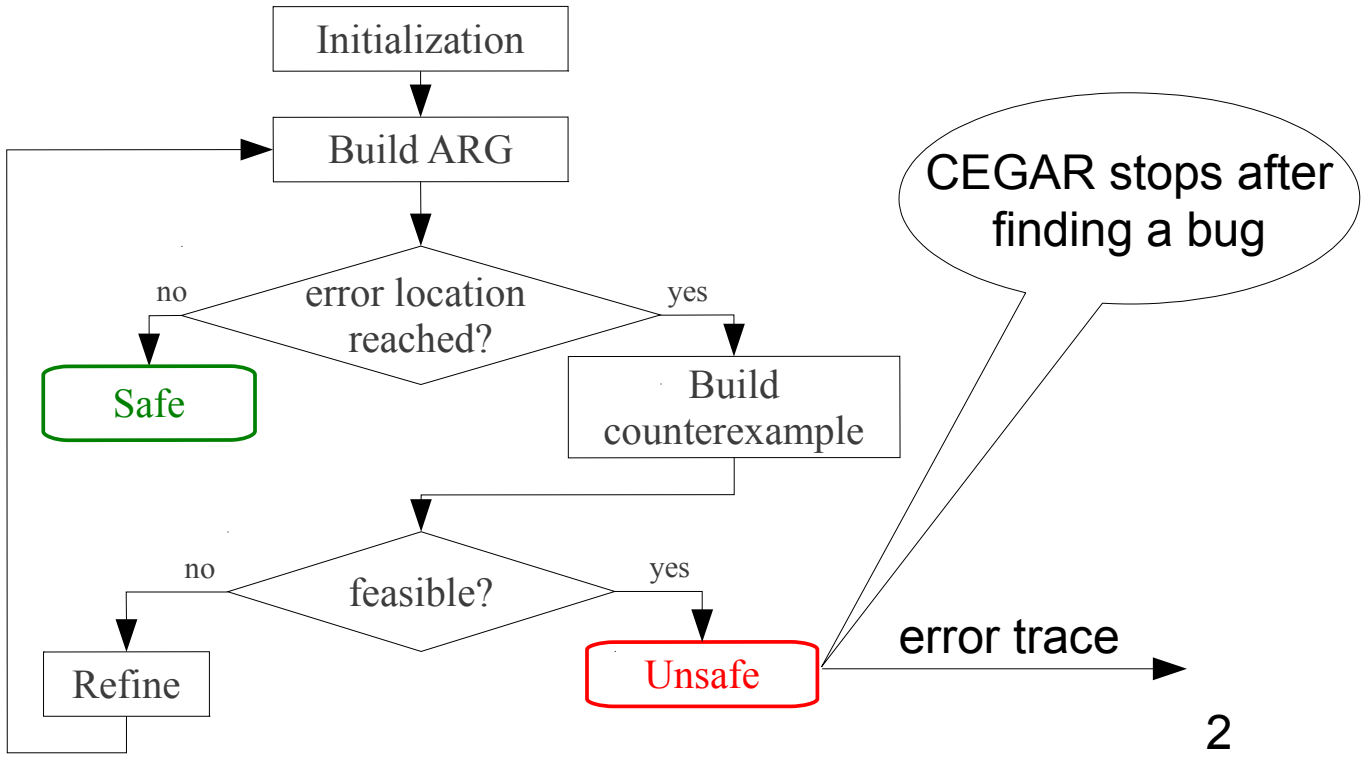
ISP RAS
Institute for System Programming of the Russian Academy of Sciences

The CEGAR Algorithm

- Checks 1 aspect

CMAV

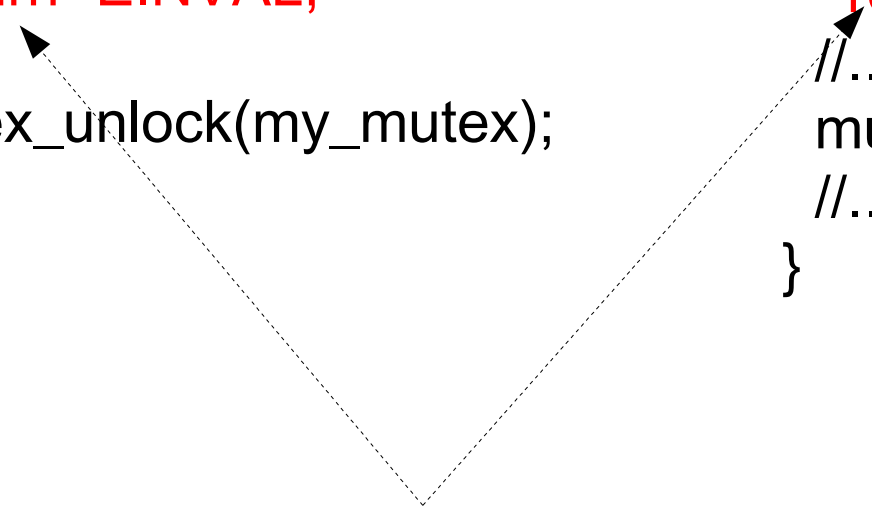
- Finds the first violation



Program with 2 Bugs

```
read (...) {
  //...
  mutex_lock(my_mutex);
  if (error(...))
    return -EINVAL;
  //...
  mutex_unlock(my_mutex);
  //...
}
```

```
write (...) {
  //...
  mutex_lock(my_mutex);
  if (error(...))
    return -EINVAL;
  //...
  mutex_unlock(my_mutex);
  //...
}
```



2 bugs: mutex my_mutex is not released

Program with 2 Bugs

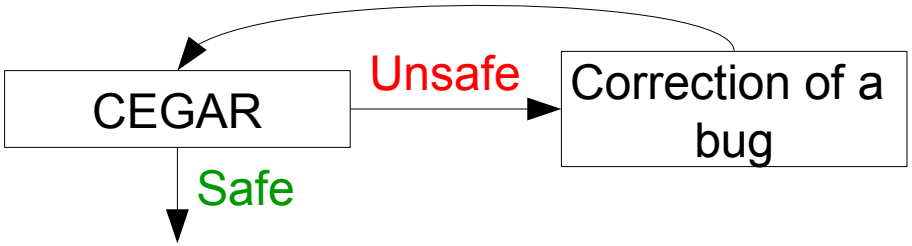
```
read (...) {
  //...
  mutex_lock(my_mutex);
  if (error(...))
    return -EINVAL;
  //...
  mutex_unlock(my_mutex);
  //...
}
```

Found by CEGAR

```
write (...) {
  //...
  mutex_lock(my_mutex);
  if (error(...))
    return -EINVAL;
  //...
  mutex_unlock(my_mutex);
  //...
}
```

CEGAR cannot find 2nd bug until correction of the 1st bug

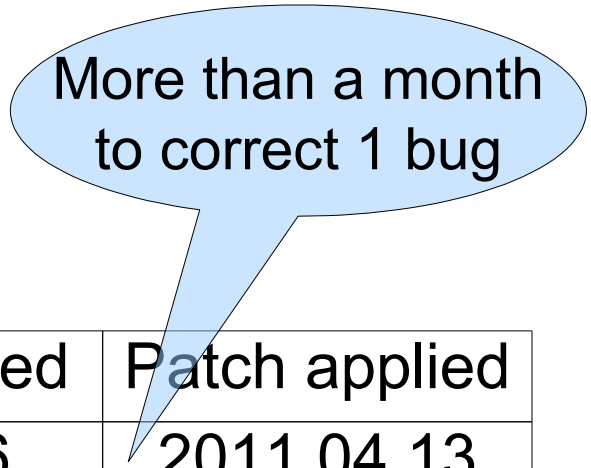
Iterative CEGAR



- Requires correction of a bug
  - Create bug reports
  - Prepare and apply of patch
  - Repeat verification
- **Time cannot be evaluated**

Example: Linux Kernel

- Bugs found by LDV Tools
- Same aspect, same module

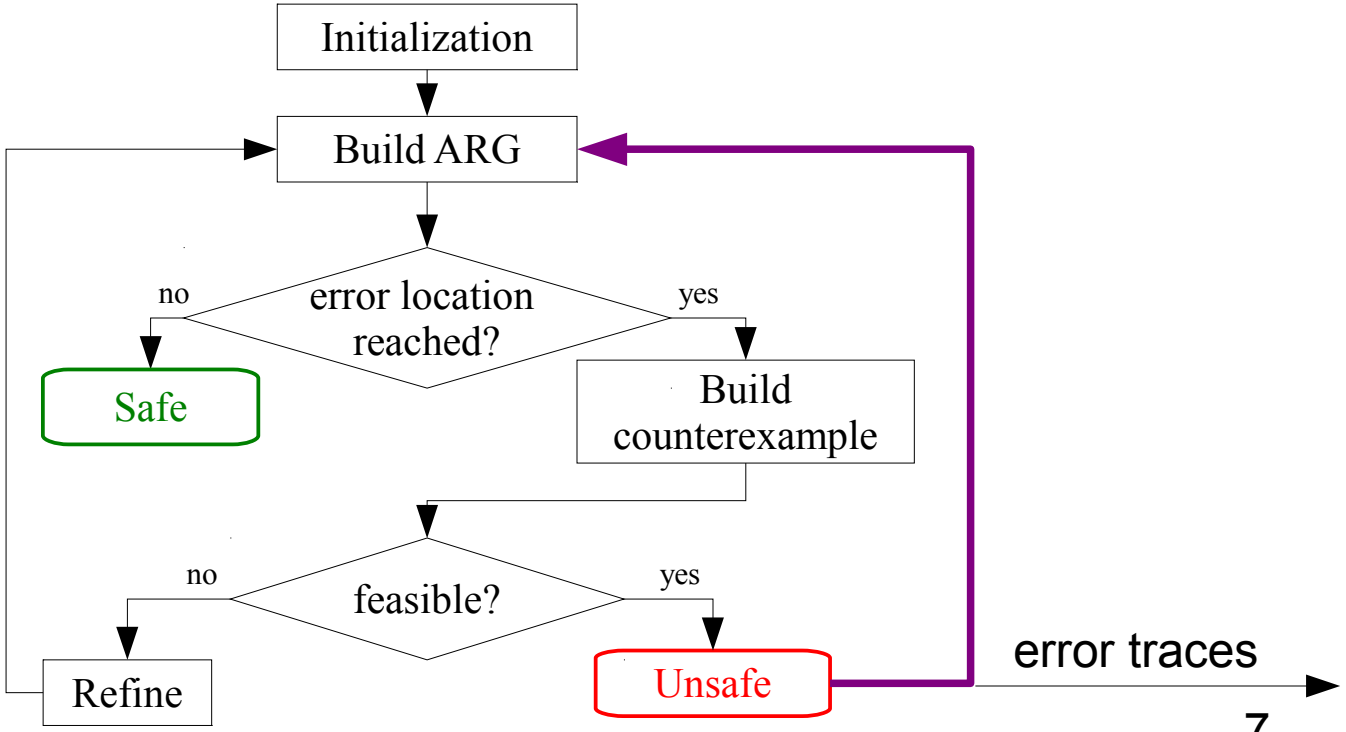


Bug report	Bug reported	Patch prepared	Patch applied
L0029	2011.03.09	2011.03.16	2011.04.13
L0034	2011.05.26	2011.05.27	2011.06.06

- ~3 month to correct just 2 similar bugs

Naive Solution

- Continue analysis after finding a bug
- Already was implemented in CPAchecker



Naive Solution Drawbacks

- Requires more time (expected)
- Analysis may terminate abnormally after finding some violations
- A lot of error traces correspond to the same bug
    - ~2000 error traces for ~20 bugs

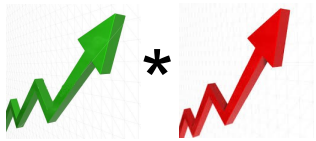
What is Manual Analysis?

- Determine if it is false alarm
  - No actual bug
- Determine reason of aspect violation
- Prepare bug report for developers
- Cannot be automatized

Similar Error Traces

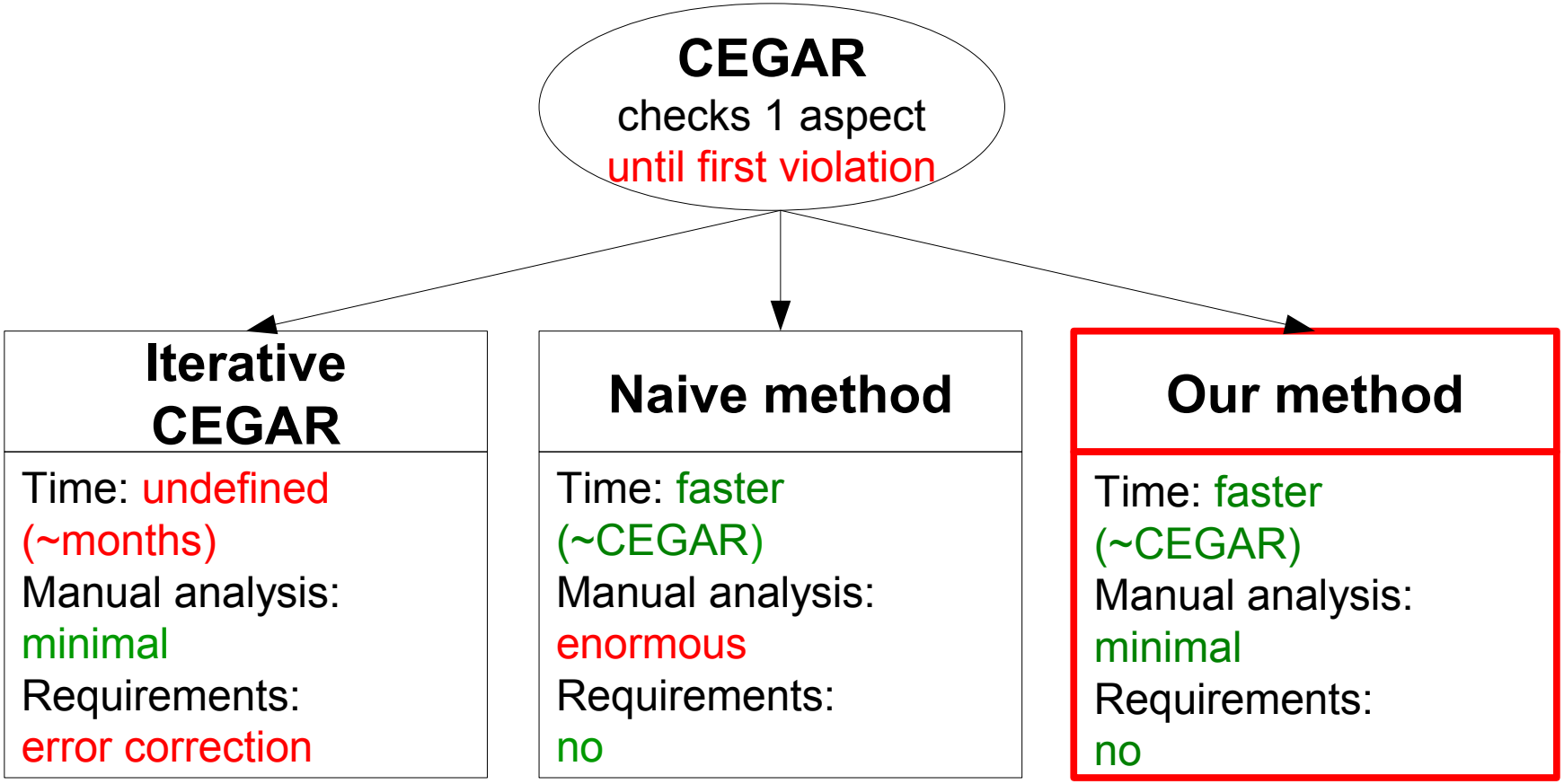
- Number of programs
- Number of traces

time



- Provides only few useful results
- **Large amount** of manual analysis time are wasted
  - Method is not used

Finding Several Bugs



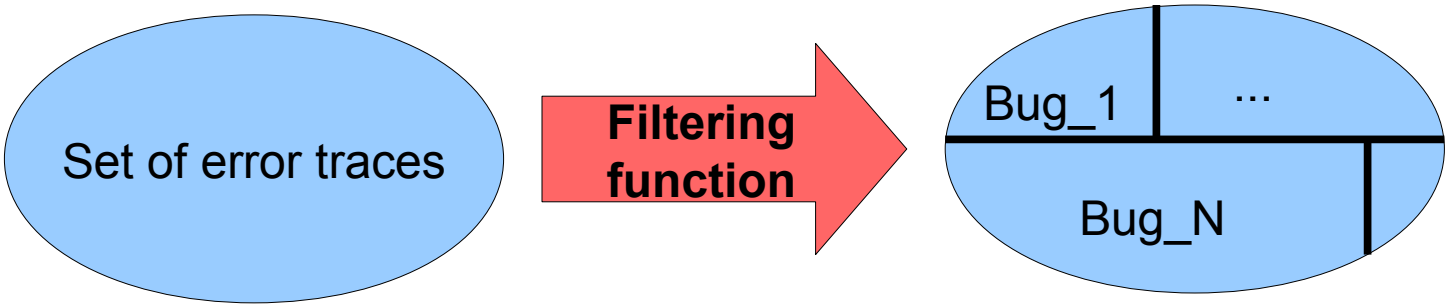
Multiple Error Analysis (MEA)

- We propose
  - Continue analysis after finding a bug
  - Process results with minimal manual analysis
- Main idea
  - Filter error traces

Formal Definitions

- Aspect
  - Formal representation of requirement
- Error trace
  - Sequence of operations in source code
- Bug (error)
  - Reason of aspect violation
- Equivalent error traces
  - Correspond to the same bug

Clustering Task

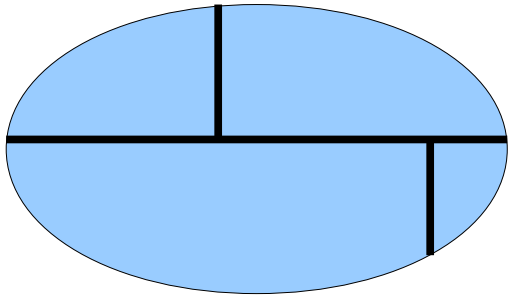


- Equivalence classes for error traces (bugs)
  - All error traces corresponding to the same bug
  - Without missing any bugs
- Manually analyze only 1 error trace for each equivalence class
- Undecidable task in general case

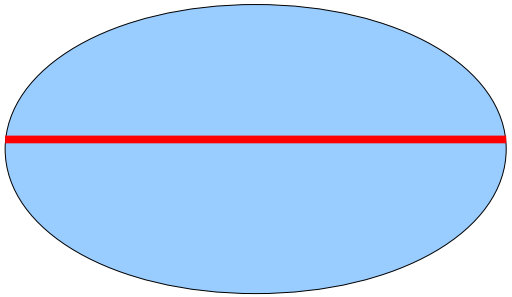
Suggested Solution

- **Approximate** filtering function
- Some equivalence classes may be the same
- Main requirements
  - Do not miss any bug
    - Equivalence class contains only 1 bug
  - Minimize number of the same found bugs

Filtering function result

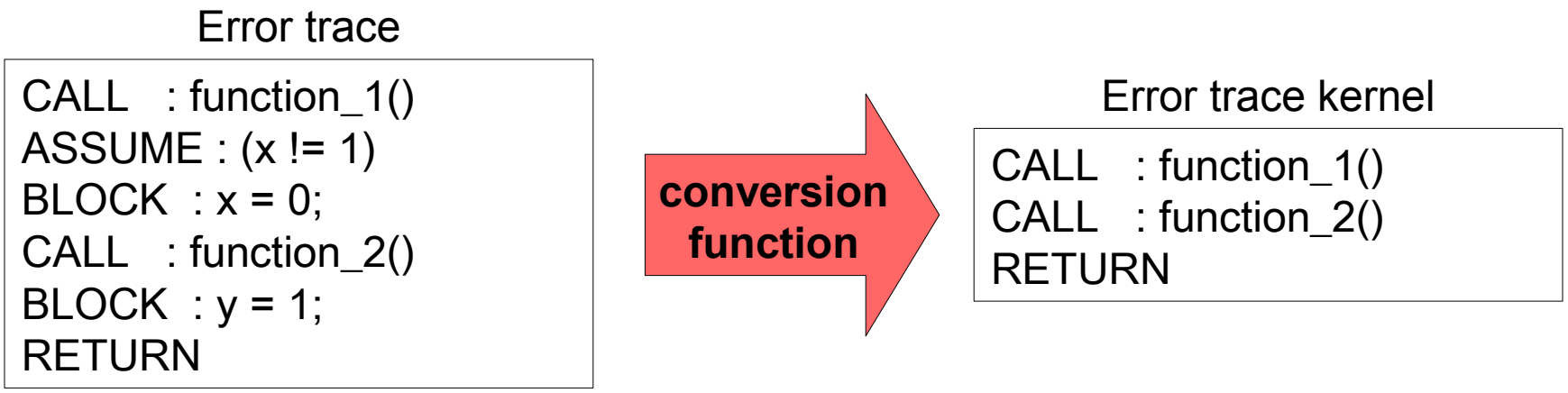


Real bugs distribution



Filtering Functions

- Find error trace kernel (**conversion function**)
  - Relevant parts of error trace
- Compare error trace kernels

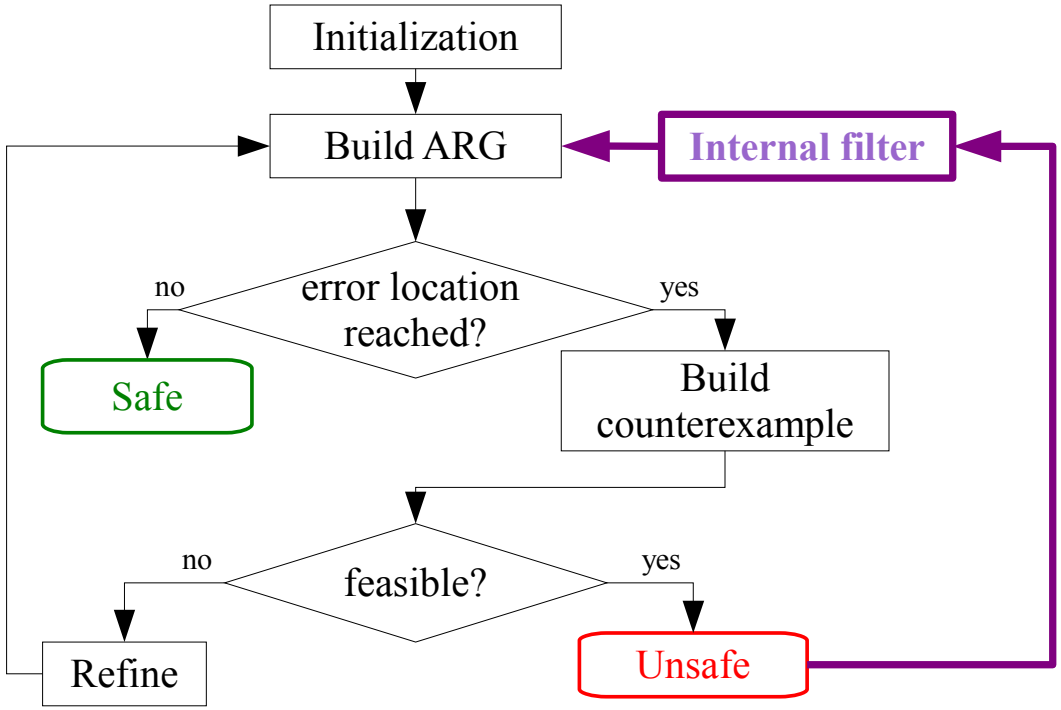


Multi-Level Filtering

1) Internal filtering
   - Inside MEA
2) External filtering
   - Outside MEA
3) Manual filtering

Internal Filtering

- “Lightweight” filtering
- Internal representation of error traces
- Efficient and simple
- Conversion function
  - Applied to any programs
- Main goals
  - Optimize external filtering

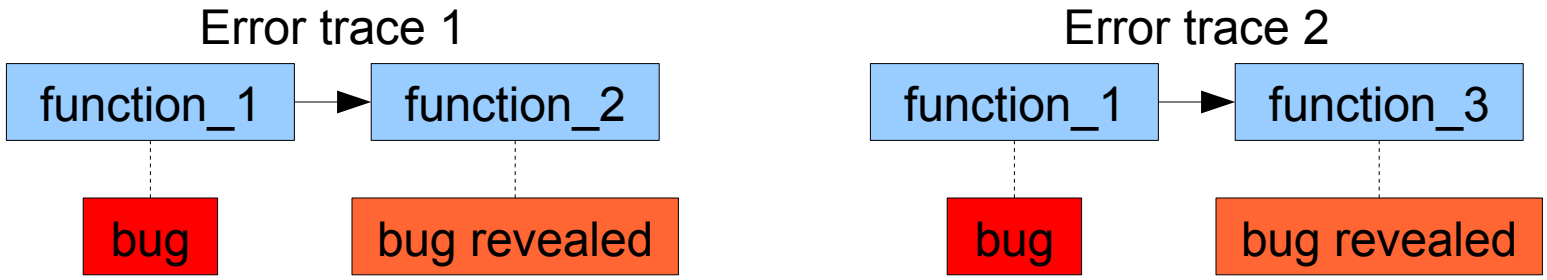


External Filtering

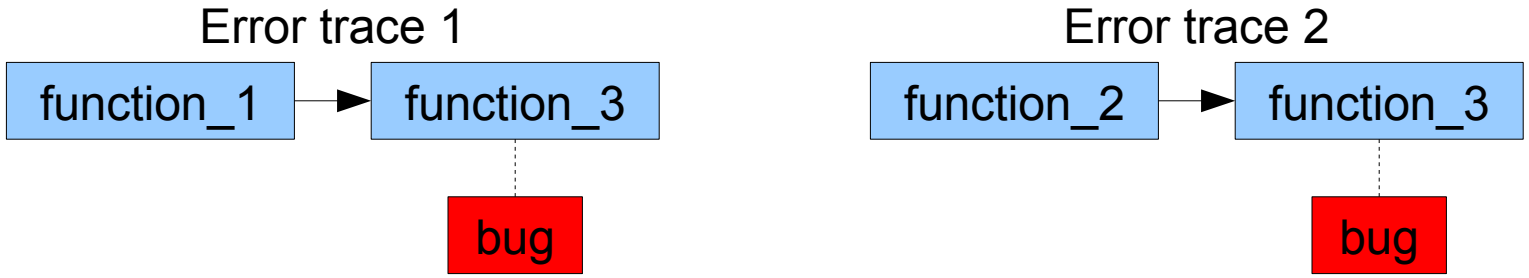
- “Heavyweight” filtering
- Internal representation of aspects
- May not be efficient
- Conversion function
  - Applied only to specified aspects
- Main goals
  - Provide only “undecidable” cases for manual filtering

Examples of Undecidable Cases

- One bug, revealed later in many places



- Different paths to the bug



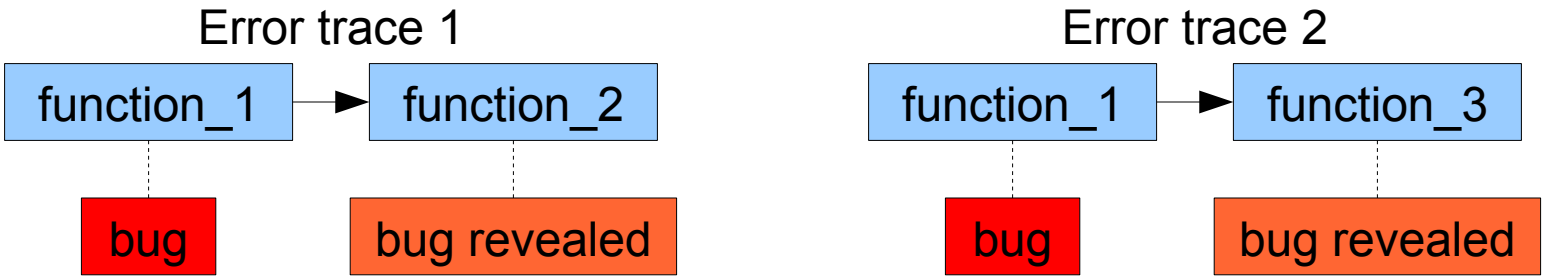
User Part in Manual Filtering

- User specify
  - Bounds of the bug
    - Operations in error traces
  - Conversion function
    - Same as in external filtering
  - **Comparison function**
    - How to compare error trace kernels
    - Full equivalence, inclusion, etc.
- Result depends on specified parameters

Algorithm Part in Manual Filtering

- Algorithm
  - Extracts new trace representing marked bug
  - Applies conversion function
  - Applies comparison function
    - Marks up equivalent error traces
- User may relaunch algorithm
  - In case of wrong parameters

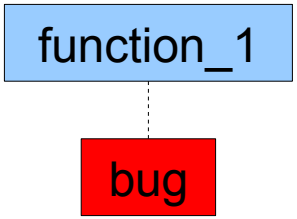
Manual Filtering Example



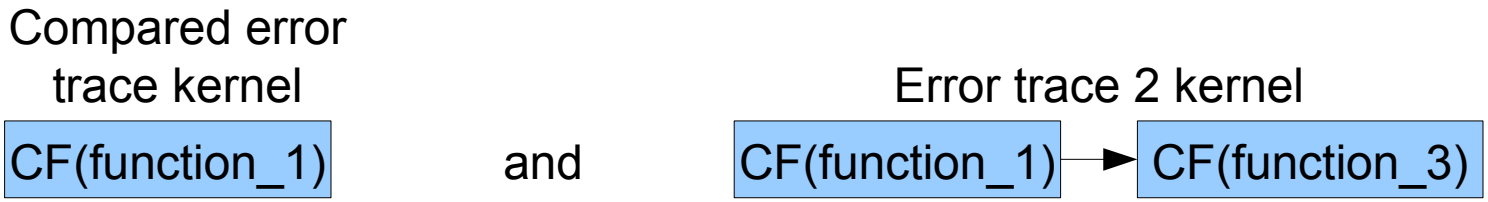
- User (examines error trace 1)
  - Bounds of bug: function_1
  - Conversion function: default (CF)
  - Comparison function: inclusion

Manual Filtering Example

- Compared error trace

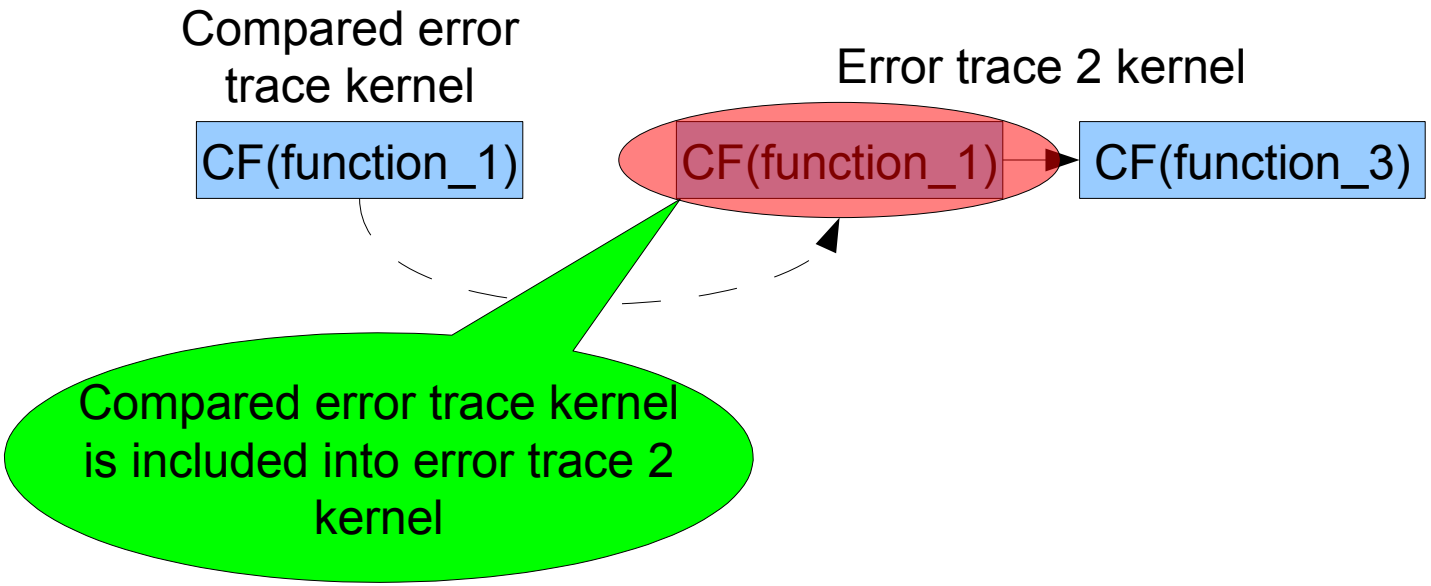


- Conversion function (CF)



Manual Filtering Example

- Comparison function (inclusion)



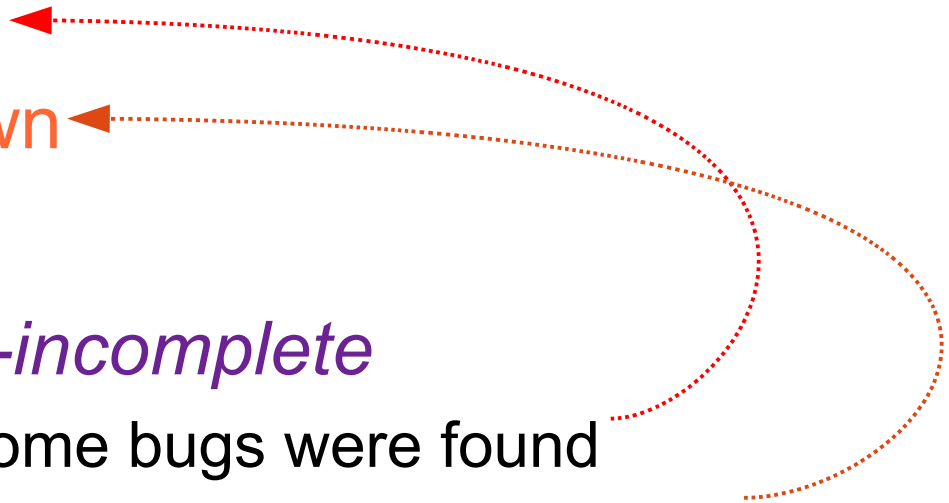
- Result: error trace 2 is equivalent to error trace 1

Manual Filtering Advantages

- User analyses only 1 error trace for each bug
  - Manual analysis time ~ CEGAR
- Equivalence marks can be reassigned
- User can add new conversion or comparison functions

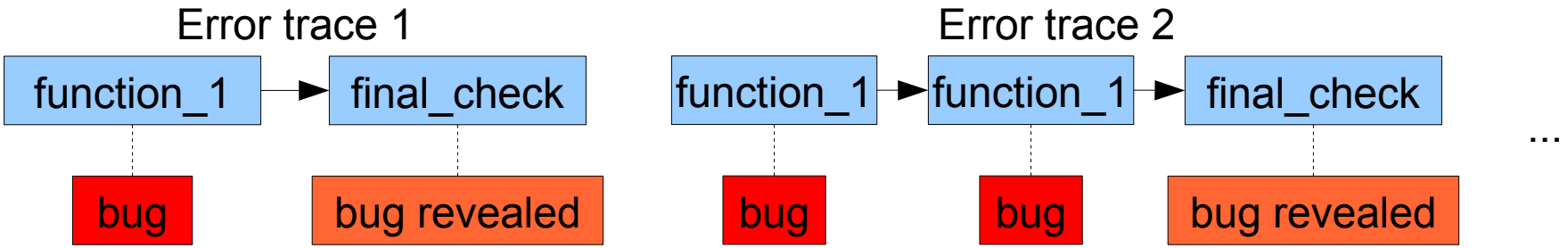
New Verdict

- CEGAR verdicts
  - Safe
  - Unsafe
  - Unknown
- MEA
  - *Unsafe-incomplete*
    - Some bugs were found
    - Analysis was not completed (e.g., time exhausted)
    - There may be more bugs



Main Reason of Unsafe-Incomplete

- Bug may generate infinite number of error traces
- May require manual filtering or heuristic external filtering



MEA Implementation

1) Internal filtering
   - Path equality filter
   - ABE filter
   - Function call tree filter
   - ...

2) External filtering
   - Aspects model functions filter

3) Manual filtering
   - Analytics Center

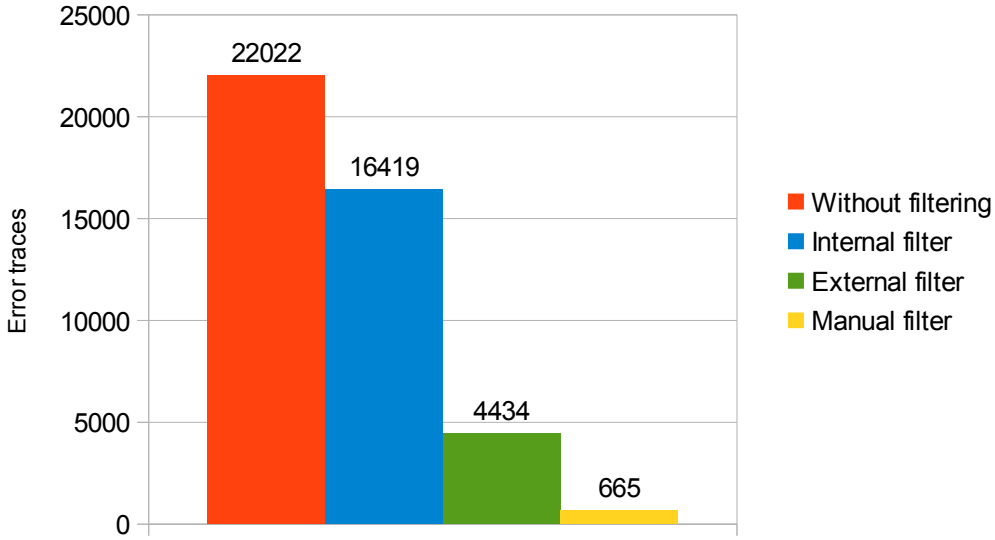
LDV Tools

Experiments

- Benchmark
  - 440 verification tasks (Unsafes in CEGAR)
  - Linux kernel 3.12-rc1
  - CPAchecker 1.3.4
- Time: ~2 times more than in CEGAR
- Verdicts
  - 212 Unsafes (~48%)
  - 228 Unsafe-incompletes (~52%)

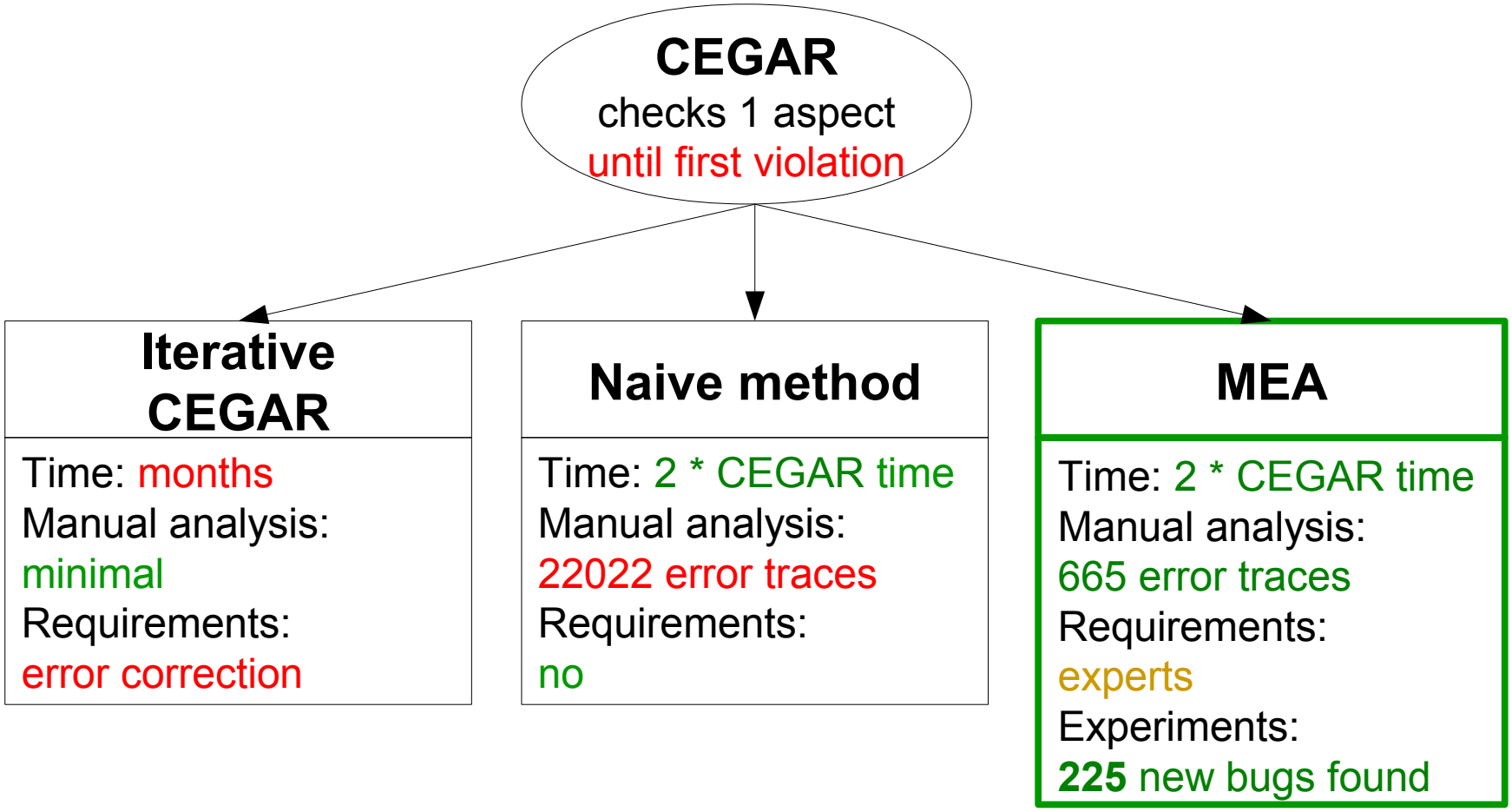
Filtering Results

Filter	Error traces
No filtering	22022
Internal filtering (ABE filter)	16419 (75%)
External filter (aspects model functions filter)	4434 (20%)
Manual filter	665 (3%)
CEGAR	440



225 new potential bugs were found!

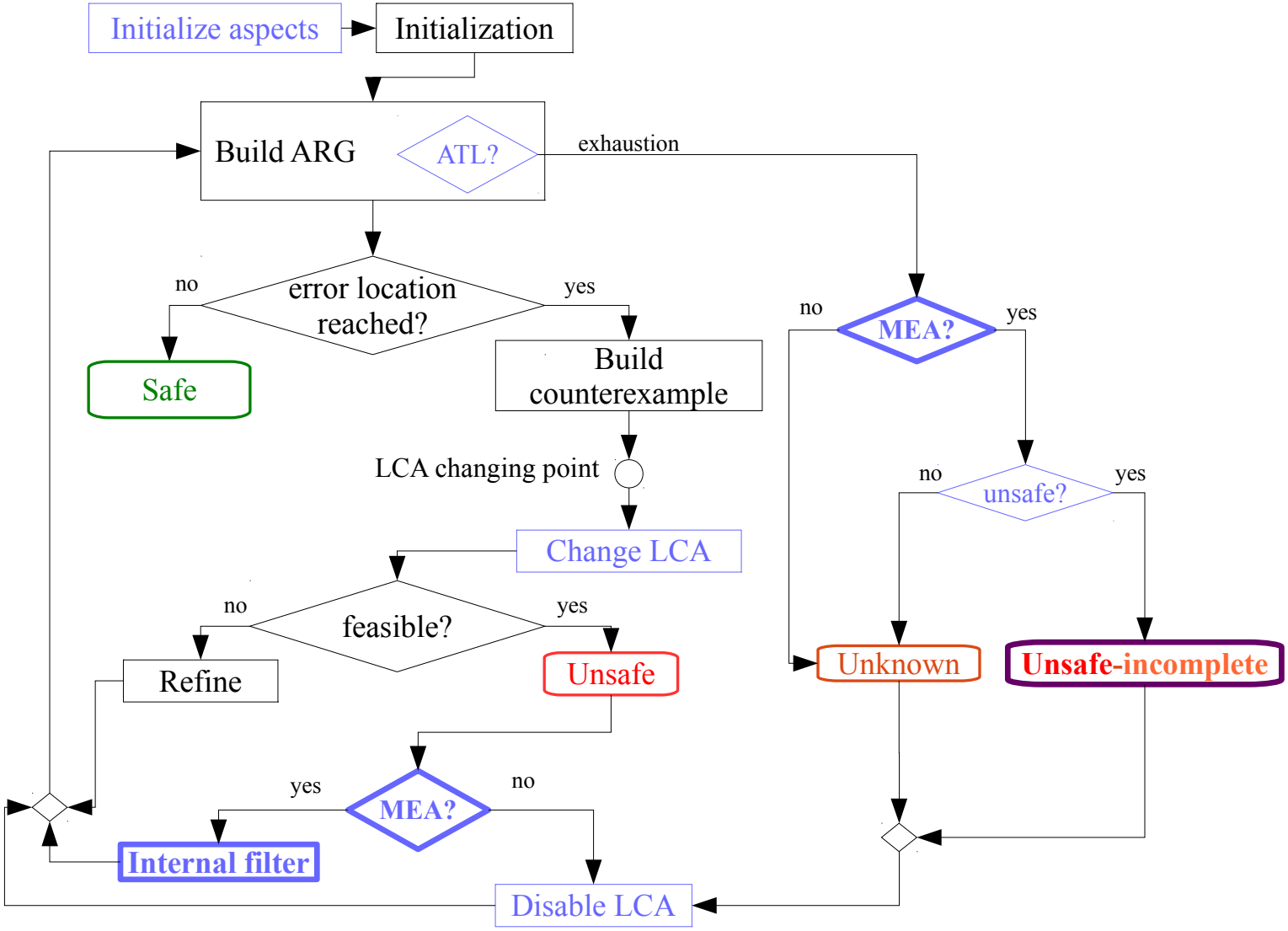
Conclusion


CEGAR
checks 1 aspect
until first violation
Iterative CEGAR
Time: months
Manual analysis:
minimal
Requirements:
error correction
Naive method
Time: 2 * CEGAR time
Manual analysis:
22022 error traces
Requirements:
no
MEA
Time: 2 * CEGAR time
Manual analysis:
665 error traces
Requirements:
experts
Experiments:
225 new bugs found

MAV with MEA

- Integration with Multi-Aspect Verification
- Find all violations of all aspects
- MAV
  - Internal filter support
  - *Unsafe-incomplete* verdicts support
- MEA
  - External filters extension

MAV with MEA Algorithm



MAV with MEA First Experiment

610 Unsafes in CMAV

- Time: 210 hours (2 times more than in CMAV)
- Verdicts
  - 217 Unsafe (37%)
  - 370 Unsafe-incomplete (63%)
  - 23 missed (“hidden errors”)

Filter	Error traces
No filtering	539 602
Internal filtering (CPAchecker, ABE filter)	29 401 (5%)
External filter (LDV Tools, extended aspects model functions filter)	3 323 (**0.6%**)

LinuxTesting.org

Thank you

Mordan Vitaly
mordan@ispras.ru

ISP RAS
Institute for System Programming of the Russian Academy of Sciences