IKOS: Sound Static Program Analysis

Maxime Arthaud
Motivation: V&V Cost Analysis

- **Requirement Engineering**
- **System/SW Design**
- **Software Development**
- **Unit Testing**
- **Integration Testing**
- **Acceptance Testing**

~80-90% of faults introduced in these phases
~96% of faults found in these phases

Relative cost to fix error

Phase in which error was detected and corrected

Code
Development Test
Acceptance Test
Operation

Requirements Design Code Development Test Acceptance Test Operation
IKOS

1. Fret
Safety requirements specification

2. IKOS
Static code analyzer for C/C++ with low false positive rate

3. Cocosim
Safety requirement verification on Simulink models

4. MARGInS
Validation testing to identify unusual behaviors
IKOS

IKOS performs a **compile-time** analysis of a C/C++ source code. It can **detect** or **prove the absence** of runtime errors.

```c
int tab[10];
for (int i = 0; i < 10; i++) {
    tab[i] = i * i;
}
```

IKOS is **NOT** a code style checker

IKOS is **NOT** a compiler: It can detect errors that compilers cannot catch

List of (possible) runtime errors:
- Buffer Overflows
- Null pointers
- Division by Zero
- Uninitialized Variables
- Assertion Prover
- Etc.
IKOS Design

C/C++ Code -> CLANG -> LLVM -> Fixpoint Engine
- Liveness Analysis
- Pointer Analysis
- Interval Domain
- Gauge Domain
- DBM Domain
- Polyhedra Domain

Verification Report
Verification Report

• **Safe**: The instruction is **proven free of runtime errors**

• **Error**: The instruction **always produces a runtime error**

• **Warning**:
  • The instruction **can produce an error** depending on the input
  • The instruction is **safe** but IKOS could **not prove it** (also called **false positive**)

The analysis discovers program properties: $0 \leq i \leq 9$

```c
int tab[10];

for (int i = 0; i < 10; i++) {
    tab[i] = i * i;
}
```
The analysis discovers program properties:

\[ 0 \leq i \leq 9 \]

The verification uses the properties discovered:

- Array-bound compliance
- Check that array \texttt{tab} has at least 10 elements

```c
int tab[10];

for (int i = 0; i < 10; i++) {
    tab[i] = i * i;
}
```

Access within bounds?
IKOS Checks

• Buffer overflow
• Division by zero
• Null pointer dereference
• Assertion prover
• Unaligned pointer
• Uninitialized variable
• Integer overflow (signed, unsigned)
• Invalid bit shift
• Invalid pointer comparison
• Invalid function pointer call
• Dead code
• Double free and Invalid lifetime
# IKOS Abstract Domains

<table>
<thead>
<tr>
<th>Domain</th>
<th>Constraints</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>$x \in [a, b]$</td>
<td>$n$</td>
</tr>
<tr>
<td>Congruence</td>
<td>$x \in aZ+b$</td>
<td>$n$</td>
</tr>
<tr>
<td>Gauge</td>
<td>$x \in [a^*i + b^*k + ..., a'^*i + b'^*k + ...]$</td>
<td>$K*n$</td>
</tr>
<tr>
<td>Difference Bound Matrices</td>
<td>$x - y \in [a, b]$</td>
<td>$n^3$</td>
</tr>
<tr>
<td>APRON Octagon</td>
<td>$x \pm y \in [a, b]$</td>
<td>$n^3$</td>
</tr>
<tr>
<td>APRON Polka Polyhedra</td>
<td>$a<em>x + b</em>y + ... + c \leq 0$</td>
<td>Exponential</td>
</tr>
<tr>
<td>APRON PPL Polyhedra</td>
<td>$a<em>x + b</em>y + ... + c \leq 0$</td>
<td>Exponential</td>
</tr>
<tr>
<td>Variable Packing of ...</td>
<td>...</td>
<td>$n$</td>
</tr>
</tbody>
</table>
Live demo
IKOS Installation

- Supported platforms:
  - Mac OS
  - Linux
  - Windows (using MinGW)
- Dependencies can be installed with a package manager (brew, apt-get, yum, ..)
- Installation instructions for each platform available in: doc/install/
- Bootstrap script for non-admin installations: downloads and compiles all missing dependencies (slow)
IKOS Usage

• Analyze a single file: `ikos file.c`
  • Runs the analysis
  • Prints the results
  • Generates an output database containing the analysis results: `output.db`

• Analyze a whole project:
  • `ikos-scan make`
  • `ikos program.bc`

• Generate a report from an output database: `ikos-report output.db`

• Examine the results in a graphical interface: `ikos-view output.db`
IKOS-SCAN

• Analyze a whole project with: `ikos-scan <command>`

• It compiles all executables to LLVM bitcode: `program.bc`

• It runs IKOS on the LLVM bitcode: `ikos program.bc`

• Works with most build systems: Make, CMake, Autoconf, etc...

• Works by overriding environment variables: CC, CXX, LD
IKOS-SCAN

Live demo
Analyzing a library

• The analysis needs an **entry point** (i.e, `main`)

• **Workaround**: create a small program that uses the library

• Extract the LLVM bitcode from an object file: `ikos-scan-extract file.o`

• Analyze a program with a specific entry point: `ikos file.bc -e=MyMain`
IKOS-VIEW

• **Graphical interface** to examine the analysis results

• Starts a **web server** in the terminal, opens the default **browser**

• **ikos-view output.db**
IKOS-VIEW

Live demo
IKOS Abstract Domains Guidelines

• Start with fast but imprecise domain
• Go towards slow but precise domain
• Stop when the analysis is too slow for your use case
• Recommended order:
  • Interval: -d=interval
  • Gauge + Interval + Congruence: -d=gauge-interval-congruence
  • Variable Packing DBM: -d=var-pack-dbm
  • Variable Packing Polyhedra: -d=var-pack-apron-ppl-polyhedra
IKOS Assumptions

• The source code is compiled with Clang for the **host architecture**
• Clang defines:
  • The data model (size of types)
  • The memory layout (alignments)
  • The endianness
  • The semantic of floating points
  • Etc...
IKOS Assumptions

• The program is **single-threaded**
• The program does **not** receive **signals** or **interrupts**
• **Unknown extern functions:**
  • Do not update global variables
  • Can write on their pointer arguments
  • Do not call user-defined functions (no callbacks)
• **Assembly code** is treated as a call to an unknown extern function
• **Recursive functions** can update any value in memory
False positives

- **False positive**: invalid warning

- **Objective**: low rate of **false positives**

- **Common source of false positives**:
  - Unknown library functions
  - “Bad” code patterns
  - Imprecision of the analysis
Modeling library functions

• The analyzer does **not** require the **libraries** used by your program

• Unknown library functions will trigger a warning ("ignored call side effect“ in **ikos-view**)

• Modeling library functions can **reduce the number of warnings**

• Write “stubs”: fake implementations of library functions
Include <ikos/analyzer/intrinsic.h>

char* fgets(char* restrict str,
           int size,
           FILE* restrict stream) {
  __ikos_assert(size >= 0);
  __ikos_forget_mem(stream, sizeof(FILE));
  __ikos_abstract_mem(str, size);
  errno = __ikos_nondet_int();
  return __ikos_nondet_int() ? str : NULL;
}
IKOS Annotations

- Annotating your source code can **reduce the number of warnings**
- List of intrinsic functions: `analyzer/include/ikos/analyzer/intrinsic.h`
  - `__ikos_assert(condition)`
  - `__ikos_assume(condition)`
  - `__ikos_nondet_int()`
  - `__ikos_check_mem_access(ptr, size)`
  - `__ikos_assume_mem_size(ptr, size)`
  - `__ikos_forget_mem(ptr, size)`
  - `__ikos_abstract_mem(ptr, size)`
  - `__ikos_print_values(description, var)`
IKOS Annotations

```c
ret = talg->parse_algoid_params(buf, param_len, param);
```
int (*fun)(const u8*, u16, alg_param*) =
    talg->parse_algoid_params;
__ikos_assume(fun == parse_algoid_params_generic ||
    fun == parse_algoid_params_ecdsa_with ||
    fun == parse_algoid_params_ecPublicKey ||
    fun == parse_algoid_params_rsa);
ret = fun(buf, param_len, param);
ret = talg->parse_algoid_params(buf, param_len, param);
Bad code pattern (1)

```c
CommandResult = XXX();
if (CommandResult == TRUE) {
    FilenameState = YYY();
    if (FilenameState == FM_NAME_IS_INVALID) {
        CommandResult = FALSE;
    }
}
if (CommandResult == TRUE) {
    CommandResult = ZZZ();
}
if (CommandResult == TRUE) {
    // ...
}
return CommandResult;
```
Bad code pattern (1)

- Bad readability
- Prone to errors
- Hard for static analyzers
- Please use “early return on errors”
Bad code pattern (1)

CommandResult = XXX();
if (!CommandResult) {
    return FALSE;
}
CommandResult = YYY();
if (CommandResult == FM_NAME_IS_INVALID) {
    return FALSE;
}
CommandResult = ZZZ();
if (!CommandResult) {
    return FALSE;
}
// ...
Bad code pattern (2)

• Single global variable containing everything

```c
AppData_t g;

typedef struct {
    PipeId_t CmdPipeId;
    uint16 usCmdPipeDepth;
    char cCmdPipeName[OS_MAX_API_NAME];
    int32 ulfd;
    uint32 uiRunStatus;
    // ...
    uint8 lastCmdBchErrorStatus;
} AppData_t;
```
Bad code pattern (2)

• Makes the buffer overflow analysis harder

• Please split it into different global variables
Bad code pattern (3)

• Small integers for loop counters

```c
void f(uint16_t n) {
    for (uint16_t i = 0; i < n; i++) {
        // ...
    }
}
```
Bad code pattern (3)

• Small integers for loop counters

```c
void f(uint16_t n) {
    for (uint16_t i = 0; i < n; i++) {
        // ...
    }
}
```

• Integer promotion rules of C

```c
void f(uint16_t n) {
    for (uint16_t i = 0;
         (unsigned int)i < (unsigned int)n;
         i = (uint16_t)((unsigned int)i + 1)) {
        // ...
    }
}
```
Bad code pattern (3)

• Creates temporary variables in the LLVM bitcode

• Leads to imprecision of the analysis

• Please use `size_t` (or `int`) for loop indexes
Imprecision

• Initialization functions returning an error code

```c
int Init(void) {
    int status = Register();
    if (status != SUCCESS) {
        return status;
    }

    status = InitEvent();
    if (status != SUCCESS) {
        return status;
    }

    // ...
```
Imprecision

- Imprecision due to the abstract union in the analysis

- Temporary workaround: add `exit(0)` for each failure branch (ugly!)

- Proper fix in the next update – using partitioning
Success Story: BioSentinel

• Space biology mission
• CubeSat spacecraft
• Developed at NASA Ames, in collaboration with JPL, JSC, MSFC
• Flight software built on top of CFS
Success Story: BioSentinel
Success Story: BioSentinel

• Each application was analyzed with IKOS

• The CFE framework was modeled to improve the analysis (~ 1200 LOC)

• Low rate of warnings: 1.31% in average

• Found ~ 17 real bugs
## Success Story: BioSentinel

<table>
<thead>
<tr>
<th>Application</th>
<th>Abstract Domain</th>
<th>Time</th>
<th>Errors</th>
<th>Warnings</th>
<th>Warnings%</th>
<th>Checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>adio</td>
<td>var-pack-dbm</td>
<td>1 min 6.92 sec</td>
<td>0</td>
<td>1</td>
<td>0.07%</td>
<td>1334</td>
</tr>
<tr>
<td>brdio</td>
<td>var-pack-dbm</td>
<td>8.02 sec</td>
<td>0</td>
<td>8</td>
<td>0.97%</td>
<td>818</td>
</tr>
<tr>
<td>ci</td>
<td>var-pack-dbm</td>
<td>19.98 sec</td>
<td>0</td>
<td>6</td>
<td>0.65%</td>
<td>923</td>
</tr>
<tr>
<td>comio</td>
<td>var-pack-dbm</td>
<td>1 min 4.83 sec</td>
<td>0</td>
<td>4</td>
<td>0.26%</td>
<td>1494</td>
</tr>
<tr>
<td>epsio</td>
<td>var-pack-dbm</td>
<td>30.64 sec</td>
<td>0</td>
<td>5</td>
<td>0.42%</td>
<td>1181</td>
</tr>
<tr>
<td>letio</td>
<td>var-pack-dbm</td>
<td>24.33 sec</td>
<td>0</td>
<td>18</td>
<td>1.64%</td>
<td>1095</td>
</tr>
<tr>
<td>ms</td>
<td>interval</td>
<td>0.16 sec</td>
<td>0</td>
<td>0</td>
<td>0%</td>
<td>444</td>
</tr>
<tr>
<td>saio</td>
<td>var-pack-dbm</td>
<td>22.35 sec</td>
<td>0</td>
<td>8</td>
<td>0.64%</td>
<td>1246</td>
</tr>
<tr>
<td>sensio</td>
<td>var-pack-dbm</td>
<td>4.67 sec</td>
<td>0</td>
<td>79</td>
<td>9.56%</td>
<td>826</td>
</tr>
<tr>
<td>spe</td>
<td>interval</td>
<td>0.16 sec</td>
<td>0</td>
<td>0</td>
<td>0%</td>
<td>445</td>
</tr>
<tr>
<td>thrio</td>
<td>var-pack-dbm</td>
<td>19.33 sec</td>
<td>0</td>
<td>4</td>
<td>0.38%</td>
<td>1043</td>
</tr>
<tr>
<td>to</td>
<td>var-pack-dbm</td>
<td>2 min 18.32 sec</td>
<td>0</td>
<td>33</td>
<td>1.98%</td>
<td>1666</td>
</tr>
<tr>
<td>xactio</td>
<td>var-pack-dbm</td>
<td>22.18 sec</td>
<td>0</td>
<td>6</td>
<td>0.51%</td>
<td>1165</td>
</tr>
</tbody>
</table>
**BioSentinel Bug (1)**

```c
ssize_t numbytes = read(fd, &FrameBufferRaw[0], 4096);

if (numbytes < 0) {
    return ERROR;
}

// ...

numbytes -= 9; // Integer overflow

memcpy(/*dst*/ FrameBuffer, /*src*/ FrameBufferRaw, /*size*/ numbytes);
```

---

**warning:** Possible buffer overflow, pointer '&FrameBuffer[0]' with offset 0 bytes points to global variable 'FrameBuffer' of size 4096 bytes
BioSentinel Bug (2)

**warning:** Possible buffer overflow, pointer `&cmd[n + 2]` accesses 1 bytes at offset between 8 and 16 bytes of local variable `cmd` of size 16 bytes

```c
uint8_t cmd[16];
uint8_t n;
// ...
switch(cmd_request) {
    case CMD_OPEN:
        n = CMD_OUT + CMD_OPEN; // 6 + 8 = 14
        break;
    // ...
}
// ...
cmd[n + 2] = 0; // 14 + 2 = 16
```
Guidelines

• Use a lightweight static analyzer first: cppcheck, clang-tidy, pvs-studio, etc.
• Use ikos-scan to generate the llvm bitcode (.bc): ikos-scan make
• Use ikos on the llvm bitcode (.bc): ikos program.bc
• Try different abstract domains: ikos –d=var-pack-dbm program.bc
• Use ikos-view to examine the results: ikos-view output.db
• (Optional) Model key library functions
• (Optional) Annotate the code
• (Optional) Avoid “bad” patterns
• (Optional) Add ikos in your continuous build system?
Comparison: Frama-C EVA

<table>
<thead>
<tr>
<th>Category</th>
<th>IKOS</th>
<th>Frama-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td>Commercial Support</td>
<td>No</td>
<td>TrustInSoft</td>
</tr>
<tr>
<td>Rate of false positives</td>
<td>Similar</td>
<td></td>
</tr>
<tr>
<td>Execution time</td>
<td>Similar</td>
<td></td>
</tr>
<tr>
<td>Warning messages</td>
<td>Good</td>
<td>Simple</td>
</tr>
<tr>
<td>Analysis features</td>
<td>Few</td>
<td>Many</td>
</tr>
<tr>
<td>Graphical Interface</td>
<td>Good</td>
<td>Simple</td>
</tr>
<tr>
<td>Analyze a whole project</td>
<td>Easy</td>
<td>Complicated</td>
</tr>
</tbody>
</table>
## Comparison: PolySpace

<table>
<thead>
<tr>
<th>Category</th>
<th>IKOS</th>
<th>PolySpace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>Free</td>
<td>Expensive</td>
</tr>
<tr>
<td>Commercial Support</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Rate of false positives</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Execution time</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Warning messages</td>
<td>Similar</td>
<td></td>
</tr>
<tr>
<td>Analysis features</td>
<td>Few</td>
<td>Many</td>
</tr>
<tr>
<td>Graphical Interface</td>
<td>Good</td>
<td>Very good</td>
</tr>
<tr>
<td>Analyze a whole project</td>
<td>Easy</td>
<td>???</td>
</tr>
</tbody>
</table>
## Comparison: Astrée

<table>
<thead>
<tr>
<th>Category</th>
<th>IKOS</th>
<th>Astrée</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td><strong>Free</strong></td>
<td>Expensive</td>
</tr>
<tr>
<td>Commercial Support</td>
<td>No</td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td>Rate of false positives</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Execution time</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Warning messages</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Analysis features</td>
<td>Few</td>
<td>???</td>
</tr>
<tr>
<td>Graphical Interface</td>
<td>Good</td>
<td>???</td>
</tr>
<tr>
<td>Analyze a whole project</td>
<td>Easy</td>
<td>???</td>
</tr>
</tbody>
</table>
IKOS at a glance

- IKOS is a static analyzer for C/C++ targeting safety critical software

- IKOS is open source: https://github.com/NASA-SW-VnV/ikos

- Contact: ikos@lists.nasa.gov
Thank you.

Questions?