Static Analysis with Goanna

Model checking for large code bases

Ansgar Fehnker
Mistakes are made

- Even good programmers make mistakes
- Bugs cost an average company US$50k per programmer per year (IDC)
- Finding bugs when testing up to 80 times more expensive than finding them when coding (IDC)

What We Do

Goanna Static Analysis for C/C++

*Inspects code automatically for*

- memory corruption and leaks
- software quality issues
- security vulnerabilities
- API rule violation
- coding standards violations
- identifies >100 types of serious defects

Does not execute, but investigate code.
Use Model Checking for Static Analysis of real code.

Possible through the use of very coarse abstractions.

Semantics added (only) if necessary

Summaries for inter-procedural checking

Able to find real bugs
Under The Hood

Syntactic Model Checking

- Discover syntactical structure of program by analysis of AST
- Map the syntactical structure of a program to a finite state model (Kripke Structure)
- Use temporal logic model checking to check for potential bugs and deficiencies in the code
Syntactical Model Checking

Source Code
```c
int main(void) {
    int i,a=0;
    int *p = (int *) malloc(sizeof(int));
    for (i=1000; i > 0; i--){
        a = *p + i;
        i = i*2;
    }
}
```

What happens?

When does it happen?
**Syntactical Model Checking**

**Source Code**
```c
int main(void) {
    int i, a=0;
    int *p = (int *) malloc(sizeof(int));
    for (i=1000; i > 0; i--){
        a = *p + i;
        i = i*2;
    }
    ... 
```

**Automatic Translation**

**Syntactic Pattern**

**Model**

```
AG decl => A
use W write
```

**Temporal Pattern**

**Model Checker**
Syntactical Model Checking

Source Code
```c
int main(void) {
    int i, a = 0;
    int *p = (int *) malloc(sizeof(int));
    for (i = 1000; i > 0; i--){
        a = *p + i;
        i = i*2;
    }
    ...
}
```

Syntactic Pattern
- **Automatic Translation**
- **Temporal Pattern**

Model
- AG decl => A
- use W write

Warnings
1. Goanna – Pointer p used a
2. Goanna – Uninitialised variable
3. Goanna – Dead Code found

Trace
- Line 1 Decl
- Line 2 Decl *
- Line 3 For-loop
- Line 4 Exp *

Model Checker

Example: Uninitialized Variable
```c
int foo(int n) {
    int x = 0, y = 1, q, i = 0;
    do {
        int oldy = y;
        y = x;
        q = x + oldy;
        x = q;
        i++;
    } while (i < n);
    return q;
}
```
Example: Uninitialized Variable

```c
int foo(int n) {
    int x = 0, y = 1, q, i = 0;
    do {
        int oldy = y;
        y = x;
        q = x + oldy;
        x = q;
        i++;
    } while(i < n);
    return q;
}
```

Temporal Specification

Forall var Never read Before write
Example: Uninitialized Variable

```c
int foo(int n) {
    int x = 0, y = 1, q, i = 0;
    do {
        int oldy = y;
        y = x;
        q = x + oldy;
        x = q;
        i++;
    } while(i < n);
    return q;
}
```

**Temporal Specification**

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Temporal Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>var_q</td>
<td>Forall var Never read Before write</td>
</tr>
</tbody>
</table>

**Output**

Goanna - analyzing file
Number of functions: 1
Total runtime : 0.01 second

**Note**

Completely Automatic Analysis
Syntactic Model Checking

- Uses a very coarse abstraction.
- Adds syntactic information as labels in Kripke structure
- Translates static analysis problems to CTL
- Uses model checking to analyse resulting model

Advantage: Very flexible
Challenge: Inter-procedural checking

Goanna Architecture

Input: Check Queries (Language)

User Defined Checks/Queries
Languages & Compilers
- C/C++
- ARM Assembly
- gcc 4.4
- MS Visual Studio

False Path Elimination
Interprocedural Analysis

Model Generation
Interval Constraint Solving

Model Checking

IDE & Tools
- Visual Studio 10
- Visual Studio 08
- Visual Studio 05
- Eclipse

Output: Warnings & Traces
Warning Manager & Metrics

Demo
Ongoing Work

Inter-procedural Model Checking

- Labels are distributed over functions
- Inlining not an option
  - Large models
  - Problem with recursion
  - Monolithic
- Example: double free

AG \{(\text{free} \Rightarrow \neg (\exists x \text{ not } \text{alloc U free}))\}
Solution

- Model programs as Recursive State Machine
- Use 3-valued logic model checking to capture partial information
- Use summaries to avoid inlining

Advantages

- Local analysis
- Partial analysis
- Supports incremental analysis

Related Work

- Analysis of Recursive State Machines
  - By Rajeev Alur, Kousha Etessami, Mihalis Yannakakis.
- Model Checking Partial State Spaces with 3-Valued Temporal Logics
  - By Glenn Bruns and Patrice Godefroid
- Abstraction refinement for 3-valued-logic analysis
  - By Alexey Loginov, Thomas Reps, Mooly Sagiv
- Multi-valued symbolic model-checking
  - Marsha Chechik, Benet Devereux, Steve Easterbrook, Arie Gurfinkel
Recursive State Machine

- A collection of state machines
- Each state machine may contain states and "boxes"
- States are labelled.
- Each state machine has entry and exit states
- A box points to another state machine
- Can be recursive
- Semantics given by Kripke structure
3-valued Kleene Logic

- Extends binary logic with 3rd value: $M$
- Negation: not $T = F$, not $F = T$, not $M = M$
- $x \lor y =$
  - $T$ if $x = T$ or $y = T$,
  - $F$ if $x = F$ and $y = F$
  - $M$ otherwise
- $x \land y =$
  - $T$ if $x = T$ and $y = T$,
  - $F$ if $x = F$ and $y = F$
  - $M$ otherwise

Summaries

- Given a property $\Phi$ of the form $EG \phi_1$ or $E \phi_1 U \phi_2$
- External assumption
  - The external assumption refers to the caller of a sub-system
  - Says whether $\Phi$ is assumed to be $T$, $F$, or $M$ when system returns
- A summary consists of
  - (internal) assumptions
  - (internal) guarantees
Summaries

- Internal assumption maps
  - External assumption to a mapping from boxes to T,F,M
- Internal guarantee maps
  - External assumption to a mapping from states to T,F,M
- Assumptions label boxes, guarantees label states
- An assumption refers to guarantees given by callees
- The guarantee to what a caller can assume about the system
- They state whether $\Phi$ is T, F, or M
Example: $\Phi = E$ not alloc $U$ free

Assumption A1

- if $\text{extA} \models \Phi$ then $A2 \models \Phi$
- if $\text{extA} \not\models \Phi$ then $A2 \models \Phi$

Guarantee A1

- if $\text{extA} \models \Phi$ then $S0 \models \Phi$
- if $\text{extA} \not\models \Phi$ then $S1 \models \Phi$
- if $\text{extA} \models \Phi$ then $S2 \models \Phi$

Coherence

- A summary is coherent if the guarantees match the assumptions.
- Matching means that for any external assumption $\text{extA}$
  - a state $s$ is labelled $T$ if $s \models \Phi$ given the assumptions
  - a state $s$ is labelled $F$ if $s \not\models \Phi$ given the assumptions
  - and is labelled $M$ otherwise
Coherence

- A summary is coherent if the guarantees match the assumptions.
- Matching means that for any external assumption extA
  - a state s is labelled T if $s \models \Phi$ given the assumptions
  - a state s is labelled F if $s \not\models \Phi$ given the assumptions
  - and is labelled M otherwise
- This means for $\Phi = E \phi_1 \cup \phi_2$ or $\Phi = EG \phi_1$
  - s is labelled T if $s \models \Phi$ or $E \phi_1 \cup T$

An assumption that $\Phi$ is true

An assumption that $\Phi$ might be true
Coherence

- A summary is coherent if the guarantees match the assumptions.
- Matching means that for any external assumption extA:
  - A state $s$ is labelled $T$ if $s \models \Phi$ given the assumptions.
  - A state $s$ is labelled $F$ if $s \not\models \Phi$ given the assumptions.
  - and is labelled $M$ otherwise.
- This means for $\Phi = E\phi_1 U \phi_2$ or $\Phi = EG\phi_1$:
  - $s$ is labelled $T$ if $s \models \Phi$ or $E\phi_1 U T$.
  - $s$ is labelled $F$ if $s \not\models \Phi$ and $E\phi_1 U T$ and not $E\phi_1 U M$.
  - and is labelled $M$ otherwise.

Example: $\Phi = E$ not alloc $U$ free

<table>
<thead>
<tr>
<th>Assumption A1</th>
<th>Guarantee A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>extA $\models \Phi$</td>
<td>$s_0 \models \Phi$</td>
</tr>
<tr>
<td>extA $\models \phi_1$</td>
<td>$s_1 \models \phi_1$</td>
</tr>
<tr>
<td>extA $\models \phi_2$</td>
<td>$s_2 \models \phi_2$</td>
</tr>
</tbody>
</table>

If $A2 \models \Phi$ then $S0 \models \Phi$
Example: $\Phi = E$ not alloc U free

<table>
<thead>
<tr>
<th></th>
<th>Assumption A1</th>
<th>Guarantee A1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if extA $\models \Phi$ T M F</td>
<td>if extA $\models \Phi$ T M F</td>
</tr>
<tr>
<td></td>
<td>then A2 $\models \Phi$ T M F</td>
<td>then S0 $\models \Phi$ M M M</td>
</tr>
<tr>
<td></td>
<td>then S1 $\models \Phi$ T M M</td>
<td>then S1 $\models \Phi$ T M M</td>
</tr>
<tr>
<td></td>
<td>then S2 $\models \Phi$ T M M</td>
<td>then S2 $\models \Phi$ T M M</td>
</tr>
</tbody>
</table>

If ext A $\not\models \Phi$ then S2 $\not\models \Phi$
Example: $\Phi = E \text{ not alloc U free}$

### Assumption A1

<table>
<thead>
<tr>
<th>if $\text{extA} \models \Phi$</th>
<th>T</th>
<th>M</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>then $S0 \models \Phi$</td>
<td>T</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

### Guarantee A1

<table>
<thead>
<tr>
<th>if $\text{extA} \not\models \Phi$</th>
<th>T</th>
<th>M</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>then $S0 \not\models \Phi$</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>then $S2 \models \Phi$</td>
<td>T</td>
<td>M</td>
<td>F</td>
</tr>
</tbody>
</table>

Given assumptions, use CTL model checking to make guarantees coherent.

**Consistency**

- The summaries for all sub-structures are **consistent** if the assumptions match the guarantees.

- Matching means that
  - given the labelling in the immediate successors of a “box”,
  - the labelling of the box (assumption) coincides with that of the guarantee of the callee.
Example: $\Phi = E$ not alloc U free

**Summary A1**
- if $\text{extA} \models \Phi$ then $S0 \models \Phi$
- if $\text{A2} \models \Phi$ then $S1 \models \Phi$
- if $\text{S1} \models \Phi$ then $S2 \models \Phi$
- if $\text{S2} \models \Phi$ then $S3 \models \Phi$

**Guarantee A2**
- if $\text{extA} \models \Phi$ then $S0 \models \Phi$
- if $\text{A2} \models \Phi$ then $S1 \models \Phi$
- if $\text{S1} \models \Phi$ then $S2 \models \Phi$

Example:
- $\Phi = E$ not alloc U free
Iterate

- Initialise all summaries to M
- Make all guarantees consistent

Coherent?

- Make all assumptions coherent
- Decide remaining M

Consistent?

Deciding Remaining M

Theorem

If all summaries are coherent and consistent then

Case: $\Phi = \text{EG} \phi_1$

- If a state $s$ is labelled M then $s\models \Phi$

Case: $\Phi = \text{E} \phi_1 \cup \phi_2$

- If a state $s$ is labelled M then $s\not\models \Phi$
Iteration over sub-formulae

- Structural induction over CTL in ENF
  
  \[ \Phi = p \mid \text{not } \phi_1 \mid \phi_1 \text{ or } \phi_2 \mid \text{EX } \phi_1 \mid \text{EG } \phi_1 \mid \text{E } \phi_1 \text{ U } \phi_2 \]

- \( \Phi \) holds for the system if the initial state of the initial sub-system is labelled \( \Phi \)

- Labelling with temporal operators EX, EG, EG might require to create a copy of a sub-system

Example: \( \Phi = \text{E not alloc U free} \)
Example: $\Phi = E \text{ not alloc} \ U \ free$

Summary

- CTL model checking algorithm for recursive state machines
- Linear for each sub-formulae
- Exponential in the number of sub-formulae
- Uses sub-system summaries
  - Sub-systems are checked one-by-one
  - 3 valued summaries allow for partial evaluation
  - Coherence and consistency allow for incremental evaluation
Back to Reality

Goanna Results in Practice

Some SATE Results

Static Analysis Tool Exposition (NIST)
- NIST selected 5 code bases for analysis
- NIST selected known CVEs to be found
- NIST selected random warnings for manual evaluation

Goanna SATE participation
- We used the default checks (55 checks)
- Geared towards quality issues, omitted checks for “cosmetic issues”
- “Sanity” assumption
Some SATE Results

<table>
<thead>
<tr>
<th>Number of Warnings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome 375.54</td>
<td>1079</td>
</tr>
<tr>
<td>Chrome 375.70</td>
<td>1173</td>
</tr>
<tr>
<td>Dovecot</td>
<td>180</td>
</tr>
<tr>
<td>Wireshark 2.0</td>
<td>534</td>
</tr>
<tr>
<td>Wireshark 2.9</td>
<td>532</td>
</tr>
</tbody>
</table>

Top10

<table>
<thead>
<tr>
<th>Warning Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTR: null-pos-assign</td>
<td>952</td>
</tr>
<tr>
<td>RED: cmp-never</td>
<td>788</td>
</tr>
<tr>
<td>RED: cmp-always</td>
<td>448</td>
</tr>
<tr>
<td>PTR: null-cmp-aff</td>
<td>328</td>
</tr>
<tr>
<td>SPC: uninit-var-some</td>
<td>189</td>
</tr>
<tr>
<td>PTR: null-assign-fun-pos</td>
<td>144</td>
</tr>
<tr>
<td>RED: unused-val-ptr</td>
<td>124</td>
</tr>
<tr>
<td>PTR: param-unchi-some</td>
<td>94</td>
</tr>
<tr>
<td>RED: unused-var-all</td>
<td>67</td>
</tr>
<tr>
<td>RED: case-reach</td>
<td>46</td>
</tr>
</tbody>
</table>

PTR: Pointer misuse
RED: Redundant code
SPC: Unspecified behavior

---

SEM-const-call

```
unichar_t uni_ucs4_to_titlecase(unichar_t chr)
{ (...)
  if ( !uint16_find(titlecase16_keys, 
    N_ELEMENTS(titlecase16_keys), chr, &idx))
    return chr; (...)

Semantic attributes are a GNU language extension
uni_ucs4_to_titlecase has __attribute__((const))
(see unichar.h)
uint16_find has not
GNU says: "(...) a function that calls a non-const function usually must not be const"
```
**RED-cmp-never**

`director_args_parse_ip_port()` only returns TRUE or FALSE. `director_args_parse_ip_port()<0` never true. `ip` and `port` might not be assigned, but this failure is not detected.

```c
if (str_array_length(args) != 2 ||
    director_args_parse_ip_port(conn, args, &ip, &port) < 0) {  
    i_error("director(%s): Invalid CONNECT args", conn->name); 
    return FALSE;  
}
```

**PTR-null-assign-fun-pos**

`director_reconnect_timeout(struct director *dir)`

```c
struct director_host *cur_host, 
    *preferred_host = director_get_preferred_right_host(dir); 
(...)  
if (cur_host != preferred_host) 
    (void)director_connect_host(dir, preferred_host);  
else {(...)  
}
```

- `director_get_preferred_right_host` might NULL
- `director_connect_host` dereferences `preferred_host`
- Potential NULL deref
Goanna is a static analysis solution for C/C++
Desktop and server version available at redlizards.com
It uses a combination of model checking and static analysis to find serious bugs in real code
It finds serious bugs in real code
It is named after a bug-eating lizard