

Stanse – Taking a Firm Stanse on Bugs

Jan Obdržálek, *Jiří Slabý*, Marek Trtík

ITI, Faculty of Informatics, Masaryk University, Brno

Nuremberg, March 3 2010

Presentation outline

- Background, the tool evolution.
- Features of the tool (configuration incl.).
- Found bugs (i.e. results).
- Work in progress and future work.
- Tool presentation & discussion.

Formal verification, analysis, and testing of software systems

- ITI/ANF Data Project (since 2006).
- Goal: automatically identify errors and bottlenecks in code.
- Aimed at real-life systems (large codebase etc.).

Stanse

- Developed at ITI, FI at the Masaryk University.
- First working version – Summer 2007.
- Evaluated at ANF Data, found some bugs.
- Recently completely rewritten.
- Public release under GPLv2 in September 2009.
- <http://stanse.fi.muni.cz/>

Formal verification, analysis, and testing of software systems

- ITI/ANF Data Project (since 2006).
- Goal: automatically identify errors and bottlenecks in code.
- Aimed at real-life systems (large codebase etc.).

Stanse

- Developed at ITI, FI at the Masaryk University.
- First working version – Summer 2007.
- Evaluated at ANF Data, found some bugs.
- Recently completely rewritten.
- Public release under GPLv2 in September 2009.
- <http://stanse.fi.muni.cz/>

Aim

- Error-finding tool based on *static analysis*.
- Influenced by Engler's METAL.
- Target language is C (ANSI C99), but extensible to C#/C++/Java.

Implementation

- *Full ANSI C99* support, including most GNU C extensions.
 - Able to parse Linux Kernel.
- Modular structure, easy extensibility, fast development.
- Easy to use *graphical interface* and error path inspection.
- Common bug tracking system independent of checkers.
- Intra- and inter-procedural analyses.
- Batch execution + makefile support.
- Web-based output support.

Aim

- Error-finding tool based on *static analysis*.
- Influenced by Engler's METAL.
- Target language is C (ANSI C99), but extensible to C#/C++/Java.

Implementation

- *Full ANSI C99* support, including most GNU C extensions.
 - Able to parse Linux Kernel.
- Modular structure, easy extensibility, fast development.
- Easy to use *graphical interface* and error path inspection.
- Common bug tracking system independent of checkers.
- Intra- and inter-procedural analyses.
- Batch execution + makefile support.
- Web-based output support.

Preprocessor

- Protect some functions against macro expansion in `cpp`.
 - Expanded function names vary depending on configuration.
- Code split to speed up parsing (later).
- Written in Perl.

C parser

- Output is AST in XML, CFG and Callgraph.
- Generated by `antlr3` (not so good).
- C and Java version: each parses different code parts.
 - The code split by preprocessor.
 - Speedup in the order of magnitude (3s → 100ms).

Internals Framework

- Lazy approach to support large code bases.
- Libraries for traversing the parser output.
- In Java (the same as the rest of the tool).

Preprocessor

- Protect some functions against macro expansion in `cpp`.
 - Expanded function names vary depending on configuration.
- Code split to speed up parsing (later).
- Written in Perl.

C parser

- Output is AST in XML, CFG and Callgraph.
- Generated by `antlr3` (not so good).
- C and Java version: each parses different code parts.
 - The code split by preprocessor.
 - Speedup in the order of magnitude (3s → 100ms).

Internals Framework

- Lazy approach to support large code bases.
- Libraries for traversing the parser output.
- In Java (the same as the rest of the tool).

Preprocessor

- Protect some functions against macro expansion in `cpp`.
 - Expanded function names vary depending on configuration.
- Code split to speed up parsing (later).
- Written in Perl.

C parser

- Output is AST in XML, CFG and Callgraph.
- Generated by `antlr3` (not so good).
- C and Java version: each parses different code parts.
 - The code split by preprocessor.
 - Speedup in the order of magnitude (3s → 100ms).

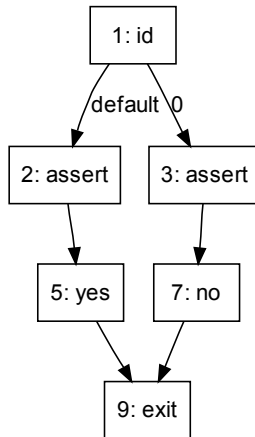
Internals Framework

- Lazy approach to support large code bases.
- Libraries for traversing the parser output.
- In Java (the same as the rest of the tool).

Code Example

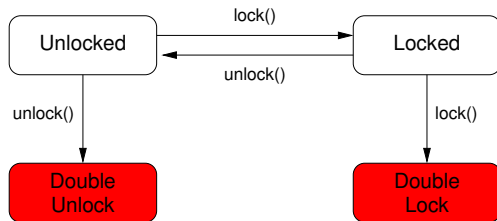
```
if (cond)
  yes();
else
  no();
```

```
<ifElseStatement>
  <id>cond</id>          # cond
  <expressionStatement> # true
    <functionCall>
      <id>yes</id>
    </functionCall>
  </expressionStatement>
  <expressionStatement> # false
    <functionCall>
      <id>no</id>
    </functionCall>
  </expressionStatement>
</ifElseStatement>
```



Automaton Checker

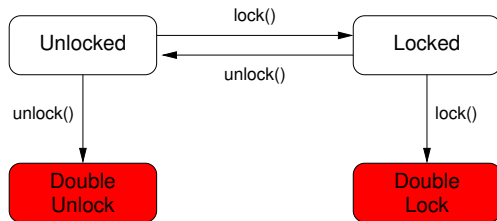
- Uses *state automata* to describe erroneous behavior.



- Finds locking, memory, reference count and other errors.
 - Whatever can be described by SM can be also checked.
 - E.g. lock imbalances, nested locking, AB-BA deadlocks, NULL dereferences, sleep inside atomic, resource leaks etc.
- Highly variable thanks to XML configuration.
 - Basic userspace configurations provided.
 - Kernel ones available too (differ in function names defined).

Automaton Checker

- Uses *state automata* to describe erroneous behavior.



- Finds locking, memory, reference count and other errors.
 - Whatever can be described by SM can be also checked.
 - E.g. lock imbalances, nested locking, AB-BA deadlocks, NULL dereferences, sleep inside atomic, resource leaks etc.
- Highly variable thanks to XML configuration.
 - Basic userspace configurations provided.
 - Kernel ones available too (differ in function names defined).

Thread Checker

- Finds threads and simultaneously follows them.
- Watches locks and unlocks (interprocedurally).
- Builds Resource Allocation Graphs.
 - To track lock dependencies.
- Interleaves the graphs of threads and finds cycles.
- Finds non-trivial locking problems across threads.

Reachability Checker

- Reports unreachable (dead) code.
- Only 200 lines long.
- It serves only as a framework demonstration.
 - But found few bugs as well: `if (error); return;`

Thread Checker

- Finds threads and simultaneously follows them.
- Watches locks and unlocks (interprocedurally).
- Builds Resource Allocation Graphs.
 - To track lock dependencies.
- Interleaves the graphs of threads and finds cycles.
- Finds non-trivial locking problems across threads.

Reachability Checker

- Reports unreachable (dead) code.
- Only 200 lines long.
- It serves only as a framework demonstration.
 - But found few bugs as well: `if (error) return;`

Automaton Checker configuration example

```
<automaton>
  <start state="U" />

  <transition from="U[A]" by="lock[A]" to="L[A]" />
  <transition from="L[A]" by="unlock[A]" to="U[A]" />

  <error from="U[A]" by="unlock[A]" desc="double unlock" ... />
  <error from="L[A]" by="lock[A]" desc="double lock" ... />

  <pattern name="lock">
    <functionCall>
      <id>__st_mutex_lock_st__</id>
      <var name="A" />
    </functionCall>
  </pattern>
  <pattern name="unlock">
    <functionCall>
      <id>__st_mutex_unlock_st__</id>
      <var name="A" />
    </functionCall>
  </pattern>
</automaton>
```

Worked Example

Having the configuration above and a project:

| Source (my.c) | Makefile |
|---|---|
| <pre>void bad_locking(struct mutex *my_lock, int cond) { mutex_lock(my_lock); if (cond) mutex_lock(my_lock); }</pre> | <pre>CFLAGS=-Wall -O2 CC=gcc all: my.o</pre> |

STANSE can find bugs:

```
$ stanse -c AutomatonChecker:conf.xml --makefile 'Makefile'
Error(s) found:
my.o.preproc, line 6: {AutomatonChecker of conf.xml} in function
'bad_locking' double lock [traces: 1]
trace [locations: 4]
start[file: my.o.preproc, line: 4]
middle[file: my.o.preproc, line: 5]
middle[file: my.o.preproc, line: 5]
end[my_lock][file: my.o.preproc, line: 6]
```

Worked Example

Having the configuration above and a project:

| Source (my.c) | Makefile |
|---|---|
| <pre>void bad_locking(struct mutex *my_lock, int cond) { mutex_lock(my_lock); if (cond) mutex_lock(my_lock); }</pre> | <pre>CFLAGS=-Wall -O2 CC=gcc all: my.o</pre> |

STANSE can find bugs:

```
$ stanse -c AutomatonChecker:conf.xml --makefile 'Makefile'
```

```
Error(s) found:
```

```
my.o.preproc, line 6: {AutomatonChecker of conf.xml} in function
'bad_locking' double lock [traces: 1]
```

```
trace [locations: 4]
```

```
start[file: my.o.preproc, line: 4]
```

```
middle[file: my.o.preproc, line: 5]
```

```
middle[file: my.o.preproc, line: 5]
```

```
end[my_lock][file: my.o.preproc, line: 6]
```

Worked Example

Having the configuration above and a project:

| Source (my.c) | Makefile |
|---|---|
| <pre>void bad_locking(struct mutex *my_lock, int cond) { mutex_lock(my_lock); if (cond) mutex_lock(my_lock); }</pre> | <pre>CFLAGS=-Wall -O2 CC=gcc all: my.o</pre> |

STANSE can find bugs:

```
$ stanse -c AutomatonChecker:conf.xml --makefile 'Makefile'
```

Error(s) found:

```
my.o.preproc, line 6: {AutomatonChecker of conf.xml} in function
'bad_locking' double lock [traces: 1]
trace [locations: 4]
start[file: my.o.preproc, line: 4]
middle[file: my.o.preproc, line: 5]
middle[file: my.o.preproc, line: 5]
end[my_lock][file: my.o.preproc, line: 6]
```

There are many tools available, some of them are compared here

| Tool name | Open-source | Configurable | Batch support | Inter-proc. | Languages |
|-----------|-------------|--------------|---------------|-------------|-------------------|
| COVERITY | no | yes | yes | yes | GC, C++, C#, Java |
| KLOCWORK | no | yes | yes | yes | GC, C++, C#, Java |
| SPLINT | yes | no | no | no | C |
| SPARSE | yes | no | yes | no | GC |
| CLANG | yes | no | no | no | GC*, C++, Obj-C |
| UNO | yes | yes | no | yes | C [†] |
| STANSE | yes | yes | yes | yes | GC |
| FINDBUGS | yes | no | yes | no | Java |

GC stands for GNU C.

*CLANG is still under heavy development, its GNU C support is incomplete.

[†]C support in UNO does not even implement full C99.

What bugs have we found? I.

ANF Data

- `malloc()` not tested against NULL.
- Memory leaks.

Linux kernel

Over 70 of the following kind of errors (reported and fixed):

- Pointer usage violations
- Locking discipline errors
 - I.e. potential real-world deadlocks (an example follows).
- Function pairing
 - E.g. `get_cpu/put_cpu`, `interrupt disable/enable`, etc.
- Unreachable code

Other code

- CESNET (Czech backbone) project developing HW accelerators.
- Small libraries/tools (`acl`, `attr`).

What bugs have we found? I.

ANF Data

- `malloc()` not tested against NULL.
- Memory leaks.

Linux kernel

Over 70 of the following kind of errors (reported and fixed):

- Pointer usage violations
- Locking discipline errors
 - I.e. potential real-world deadlocks (an example follows).
- Function pairing
 - E.g. `get_cpu/put_cpu`, `interrupt disable/enable`, etc.
- Unreachable code

Other code

- CESNET (Czech backbone) project developing HW accelerators.
- Small libraries/tools (`acl`, `attr`).

What bugs have we found? I.

ANF Data

- `malloc()` not tested against NULL.
- Memory leaks.

Linux kernel

Over 70 of the following kind of errors (reported and fixed):

- Pointer usage violations
- Locking discipline errors
 - I.e. potential real-world deadlocks (an example follows).
- Function pairing
 - E.g. `get_cpu/put_cpu`, `interrupt disable/enable`, etc.
- Unreachable code

Other code

- CESNET (Czech backbone) project developing HW accelerators.
- Small libraries/tools (`acl`, `attr`).

What bugs have we found? II.

One simple kernel example

```
static int set_lock_status(struct hotplug_slot *hotplug_slot,
                          u8 status)
{
    struct slot *slot = hotplug_slot->private;
    ...
    mutex_lock(&slot->ctrl->crit_sect);

    /* has it been >1 sec since our last toggle? */
    if ((get_seconds() - slot->last_emi_toggle) < 1)
        return -EINVAL;
    ...
    mutex_unlock(&slot->ctrl->crit_sect);
    return 0;
}
```

Source: `linux-2.6.28/drivers/pci/hotplug/pciehp_core.c`

Trigger: `echo '1' > /sys/bus/pci/slots/.../lock` **twice in 1s**

Full list at <http://stanse.fi.muni.cz/bugs.html>.

What bugs have we found? II.

One simple kernel example

```
static int set_lock_status(struct hotplug_slot *hotplug_slot,
                          u8 status)
{
    struct slot *slot = hotplug_slot->private;
    ...
    mutex_lock(&slot->ctrl->crit_sect);

    /* has it been >1 sec since our last toggle? */
    if ((get_seconds() - slot->last_emi_toggle) < 1)
        return -EINVAL;
    ...
    mutex_unlock(&slot->ctrl->crit_sect);
    return 0;
}
```

Source: `linux-2.6.28/drivers/pci/hotplug/pciehp_core.c`

Trigger: `echo '1' > /sys/bus/pci/slots/.../lock` **twice in 1s**

Full list at <http://stanse.fi.muni.cz/bugs.html>.

False Positives

Oscillating between 30 and 70% – depending on checker and source codes, but quite high.

Fighting FP

- Improving configuration.
 - The quality is important and alters directly the FP-to-bugs ratio.
 - Too naive automata results in high FP ratio.
 - Prune bugs in the configuration proper.
 - Tune-up based on previous results.
- FP detectors: a Java code finding certain patterns.
 - E.g. leaks of memory pointer by globals are mostly a FP.
- Better or new analyses/checkers.
 - Inappropriate (non-sophisticated) techniques.
 - E.g. no bugs of such type present in the code.
 - Symbolic execution or simple simulation to ensure the paths are reachable.

False Positives

Oscillating between 30 and 70% – depending on checker and source codes, but quite high.

Fighting FP

- Improving configuration.
 - The quality is important and alters directly the FP-to-bugs ratio.
 - Too naive automata results in high FP ratio.
 - Prune bugs in the configuration proper.
 - Tune-up based on previous results.
- FP detectors: a Java code finding certain patterns.
 - E.g. leaks of memory pointer by globals are mostly a FP.
- Better or new analyses/checkers.
 - Inappropriate (non-sophisticated) techniques.
 - E.g. no bugs of such type present in the code.
 - Symbolic execution or simple simulation to ensure the paths are reachable.

False Positives

Oscillating between 30 and 70% – depending on checker and source codes, but quite high.

Fighting FP

- Improving configuration.
 - The quality is important and alters directly the FP-to-bugs ratio.
 - Too naive automata results in high FP ratio.
 - Prune bugs in the configuration proper.
 - Tune-up based on previous results.
- FP detectors: a Java code finding certain patterns.
 - E.g. leaks of memory pointer by globals are mostly a FP.
- Better or new analyses/checkers.
 - Inappropriate (non-sophisticated) techniques.
 - E.g. no bugs of such type present in the code.
 - Symbolic execution or simple simulation to ensure the paths are reachable.

False Positives

Oscillating between 30 and 70% – depending on checker and source codes, but quite high.

Fighting FP

- Improving configuration.
 - The quality is important and alters directly the FP-to-bugs ratio.
 - Too naive automata results in high FP ratio.
 - Prune bugs in the configuration proper.
 - Tune-up based on previous results.
- FP detectors: a Java code finding certain patterns.
 - E.g. leaks of memory pointer by globals are mostly a FP.
- Better or new analyses/checkers.
 - Inappropriate (non-sophisticated) techniques.
 - E.g. no bugs of such type present in the code.
 - Symbolic execution or simple simulation to ensure the paths are reachable.

Points-to analysis (Master's thesis)

- Improves knowledge about values in pointers for more exact analyses.
- Memory checker knows which pointers point to the same objects.

Statistical approach (Master's thesis)

- Intended for the locking checker.
- Tracks which variables are altered under which locks.
- Based on a technique used in METAL.

Plugin for IDE (Bachelor's thesis)

- Plugin for Eclipse, Netbeans.
- For easier deployment of the tool (immediate response).

Symbolic execution

- Part of Marek's and my dissertation.
- Needn't be part of STANSE.
 - Not decided whether it will be "STANSE 2.0" or a stand-alone tool.

Points-to analysis (Master's thesis)

- Improves knowledge about values in pointers for more exact analyses.
- Memory checker knows which pointers point to the same objects.

Statistical approach (Master's thesis)

- Intended for the locking checker.
- Tracks which variables are altered under which locks.
- Based on a technique used in METAL.

Plugin for IDE (Bachelor's thesis)

- Plugin for Eclipse, Netbeans.
- For easier deployment of the tool (immediate response).

Symbolic execution

- Part of Marek's and my dissertation.
- Needn't be part of STANSE.
 - Not decided whether it will be "STANSE 2.0" or a stand-alone tool.

Points-to analysis (Master's thesis)

- Improves knowledge about values in pointers for more exact analyses.
- Memory checker knows which pointers point to the same objects.

Statistical approach (Master's thesis)

- Intended for the locking checker.
- Tracks which variables are altered under which locks.
- Based on a technique used in METAL.

Plugin for IDE (Bachelor's thesis)

- Plugin for Eclipse, Netbeans.
- For easier deployment of the tool (immediate response).

Symbolic execution

- Part of Marek's and my dissertation.
- Needn't be part of STANSE.
 - Not decided whether it will be "STANSE 2.0" or a stand-alone tool.

Points-to analysis (Master's thesis)

- Improves knowledge about values in pointers for more exact analyses.
- Memory checker knows which pointers point to the same objects.

Statistical approach (Master's thesis)

- Intended for the locking checker.
- Tracks which variables are altered under which locks.
- Based on a technique used in METAL.

Plugin for IDE (Bachelor's thesis)

- Plugin for Eclipse, Netbeans.
- For easier deployment of the tool (immediate response).

Symbolic execution

- Part of Marek's and my dissertation.
- Needn't be part of STANSE.
 - Not decided whether it will be "STANSE 2.0" or a stand-alone tool.

Get rid of XML

- XML is a beast in memory consumption.
- Switch to standard Java objects.
 - The ones returned from parser? Language dependency!

Java to C switch

- Generated parser is slow.
 - It is yet to be optimized by `antlr3` authors.
- Method invocation is not the fastest operation in Java.

Better preprocessor

- Currently we have none.
- Nothing binds preprocessed line numbers to original ones.

Get rid of XML

- XML is a beast in memory consumption.
- Switch to standard Java objects.
 - The ones returned from parser? Language dependency!

Java to C switch

- Generated parser is slow.
 - It is yet to be optimized by `antlr3` authors.
- Method invocation is not the fastest operation in Java.

Better preprocessor

- Currently we have none.
- Nothing binds preprocessed line numbers to original ones.

Get rid of XML

- XML is a beast in memory consumption.
- Switch to standard Java objects.
 - The ones returned from parser? Language dependency!

Java to C switch

- Generated parser is slow.
 - It is yet to be optimized by `antlr3` authors.
- Method invocation is not the fastest operation in Java.

Better preprocessor

- Currently we have none.
- Nothing binds preprocessed line numbers to original ones.

Conclusion

- STANSE is still in the *early phase* (read: still much work to do).
 - There is work in progress.
- It is not only a checker, it is rather a *platform/framework*.
- However it *found many bugs* in real-world large code base.
- Visit <http://stanse.fi.muni.cz/> for more info.

*We want to hear about your problems! **

* Applies only to errors in your (or somebody else's) code.

Conclusion

- STANSE is still in the *early phase* (read: still much work to do).
 - There is work in progress.
- It is not only a checker, it is rather a *platform/framework*.
- However it *found many bugs* in real-world large code base.
- Visit <http://stanse.fi.muni.cz/> for more info.

*We want to hear about your problems! **

* Applies only to errors in your (or somebody else's) code.

Thank you for your attention!

Questions?