C++ Support for Stanse

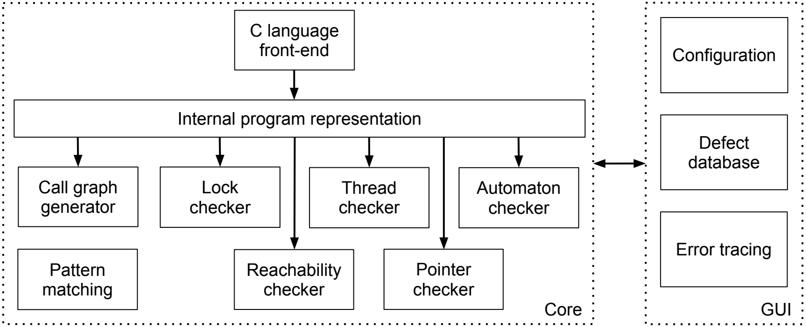
Martin Vejnár

June 29, 2011

Introduction

- Stanse is a bug-finding tool that is being developed at FI.
- Performs static analyses in the fashion similar to commercial tools like Coverity.
- Originally designed to support the C99 language, it is now used to periodically check Linux kernel sources.
- The goal of the thesis was to extend Stanse with the support for the C++ language.

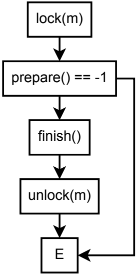
Stanse Architecture



- A language parser converts the source code to an internal representation.
- Checkers make use of the IR and the Stanse framework to detect defects.

Internal Program Representation

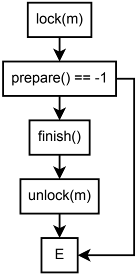
```
void perform_action()
{
    lock(m);
    if (prepare() == -1)
        return;
    finish();
    unlock(m);
}
```



- For each function a control-flow graph is constructed.
- Nodes of the CFG contain XML-encoded C language statements.
- Most checkers match CFG nodes against user-supplied patterns rather than interpreting them directly.

Internal Program Representation

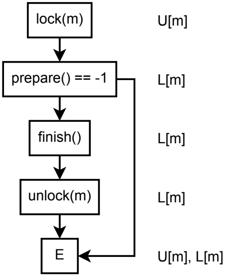
```
void perform_action()
{
    lock(m);
    if (prepare() == -1)
        return;
    finish();
    unlock(m);
}
```



- The function performs an atomic action consisting of two steps.
- If the preparation step fails, the mutex remains locked.

Example—Automaton Checker

```
void perform_action()
{
    lock(m);
    if (prepare() == -1)
        return;
    finish();
    unlock(m);
}
```



- The user provides two patterns, $\texttt{U}[\%1] \xrightarrow{\texttt{lock(\%1)}} \texttt{L}[\%1]$, and $\texttt{L}[\%1] \xrightarrow{\texttt{unlock(\%1)}} \texttt{U}[\%1]$
- The automaton checker then annotates the states and reports errors.

New Internal Representation

- Control flow within statements is not explicitly modeled (short circuit evaluation, ternary condition operator, etc.)
- Interprocedural navigation framework in Stanse can only handle one function call per CFG node.
- A new internal representation was needed if support for C++ programs was to be added.
- Stanse Internal Representation (SIR) was designed to have minimal impact on existing checkers.
- Only pattern-matching and intraprocedural navigation had to be updated.
- The old and the new representations can coexist.

Stanse Internal Representation

```
int fact(int x) {
    if (x)
        return x * fact(x - 1);
    else
        return 1;
}
```

\$1:	**value** *x* | 0 $\rightarrow_0$ \$7
\$2:	**sub** *x*, 1
\$3:	**call** `fact`, \$2 |$\rightarrow_1$ \$8
\$4:	**mul** *x*, \$3
\$5:	**phi** \$4, \$7
\$6	**exit** 0, \$5
\$7:	**value** 1 | $\rightarrow_0$ \$5
\$8:	**exit** 1

- Each CFG node contains an elementary instruction.
- At most one call per node.
- SIR units are transported between programs using JSON-encoding.
- Metadata about the source code is passed as well (source code positions, file names, etc.).

C++ frontend

- Clang (the LLVM C++ front-end) used to preprocess and parse C++ programs into ASTs.
- A CFG is generated for each function definition in the AST.
- This includes initialization and tracking of automatic and temporary variables, generation of destructor calls and exception paths.
- The tool is written in C++ and runs on Windows and Linux.
- Unit tests and diagnostic tools provided as well.

Conclusion

- SIR: syntax, formal semantics and JSON-encoding.
- Modifications to Stanse: call-graph generator, pattern matching, minor changes to the automaton checker.
- C++ frontend: a tool that translates C++ programs to SIR.

Extra: Late Binding

```
struct a {
    virtual int foo();
};
struct b : a {
    virtual int foo();
};

int bar(a & obj) {
    return obj.foo();
}
```

```
def  bar(obj):
     $1:  call v:a::foo, obj
     $2:  exit $1

def  v:a::foo(this):
     $1:  none | $\to_0$ $3
     $2:  call a::foo, this | $\to_0$ $4

     $3:  call b::foo, this
     $4:  phi $2, $3
     $5:  exit $4
```

- For each virtual function a dispatch function is created.
- The dispatch function determines the type of the implicit *this* parameter and calls the appropriate function.
- Currently, the call is dispatched to one of the functions non-deterministically.