Model Checking and Code Analysis

Adam Streck

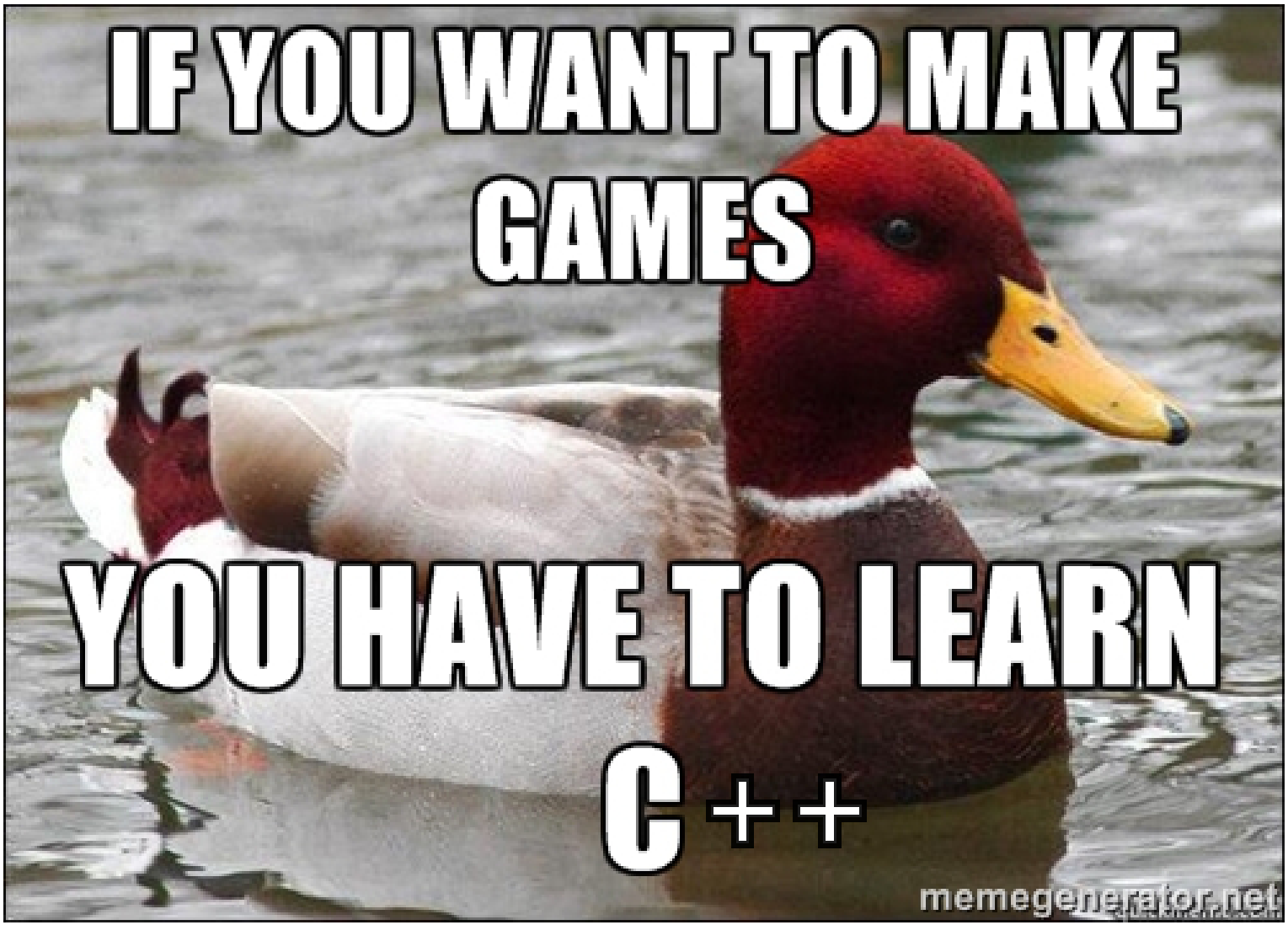
Matheon research center
Freie Universität Berlin

February 18, 2014

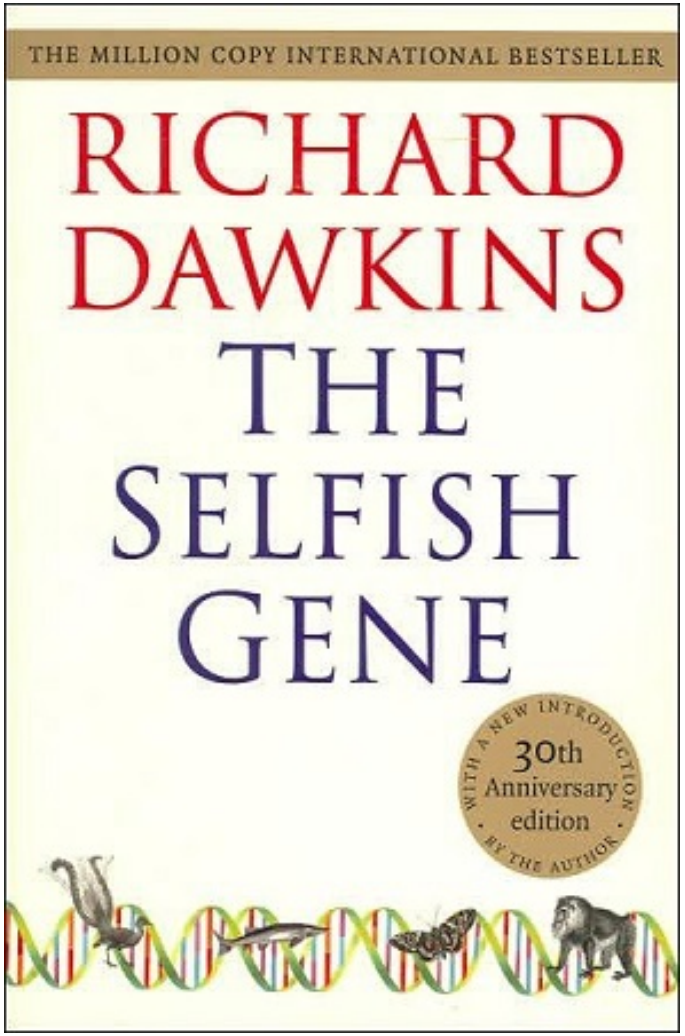
Overview

- Me and why I won't know answers to your questions.
- Where no one has tested before.
- What the smeg is model checking?
- Bisimulations, hybrid systems, timed automata: not swears.

My way to C++


IF YOU WANT TO MAKE GAMES
YOU HAVE TO LEARN C++
memegenerator.net

My way to biomathematics

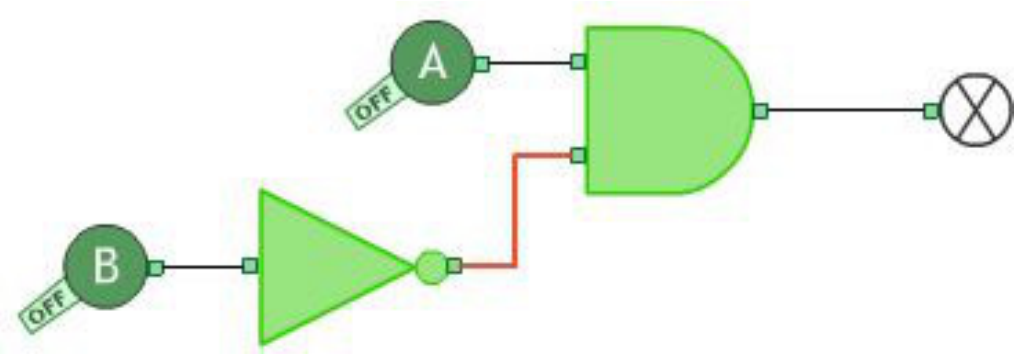


My way to science

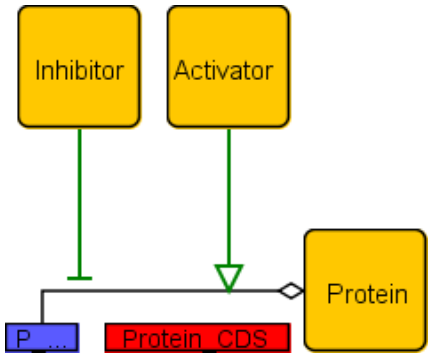
DiViNe

BioDiViNE

Circuits and ... circuits?

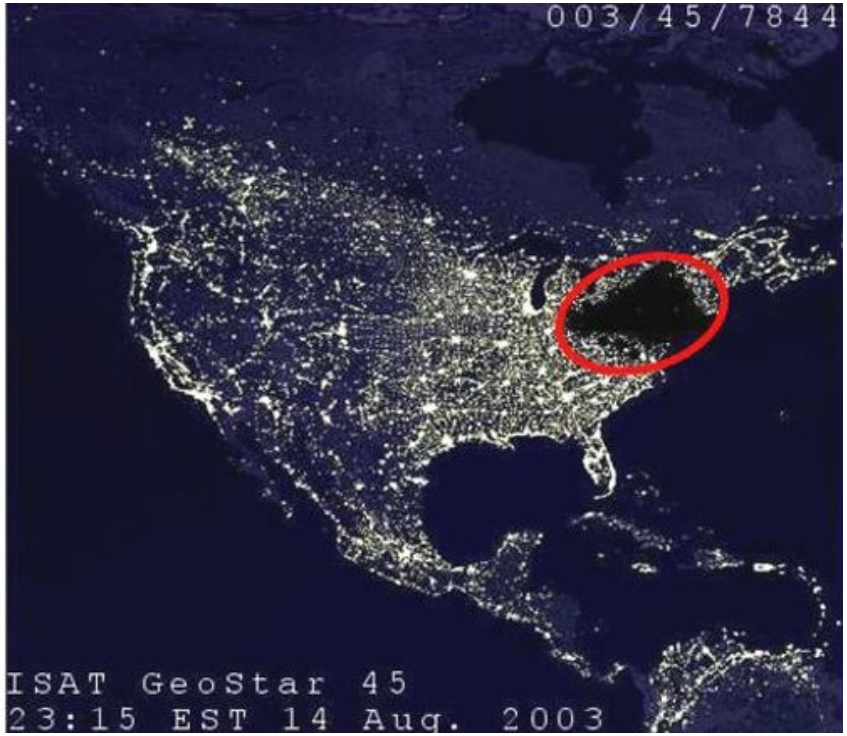


≈



Now for something completely different

Bad things happen [1]



[1] left: http://www.magikus.com/, right: http://www.ptd.siemens.de/

We do not want bad things to happen

Testing:

- Dumb "It worked on *my* computer"
- Smart Unit test, script testing, mocks, input coverage.
- Automated Valgrind, Clang Sanitizer

Is it enough?

DiViNe

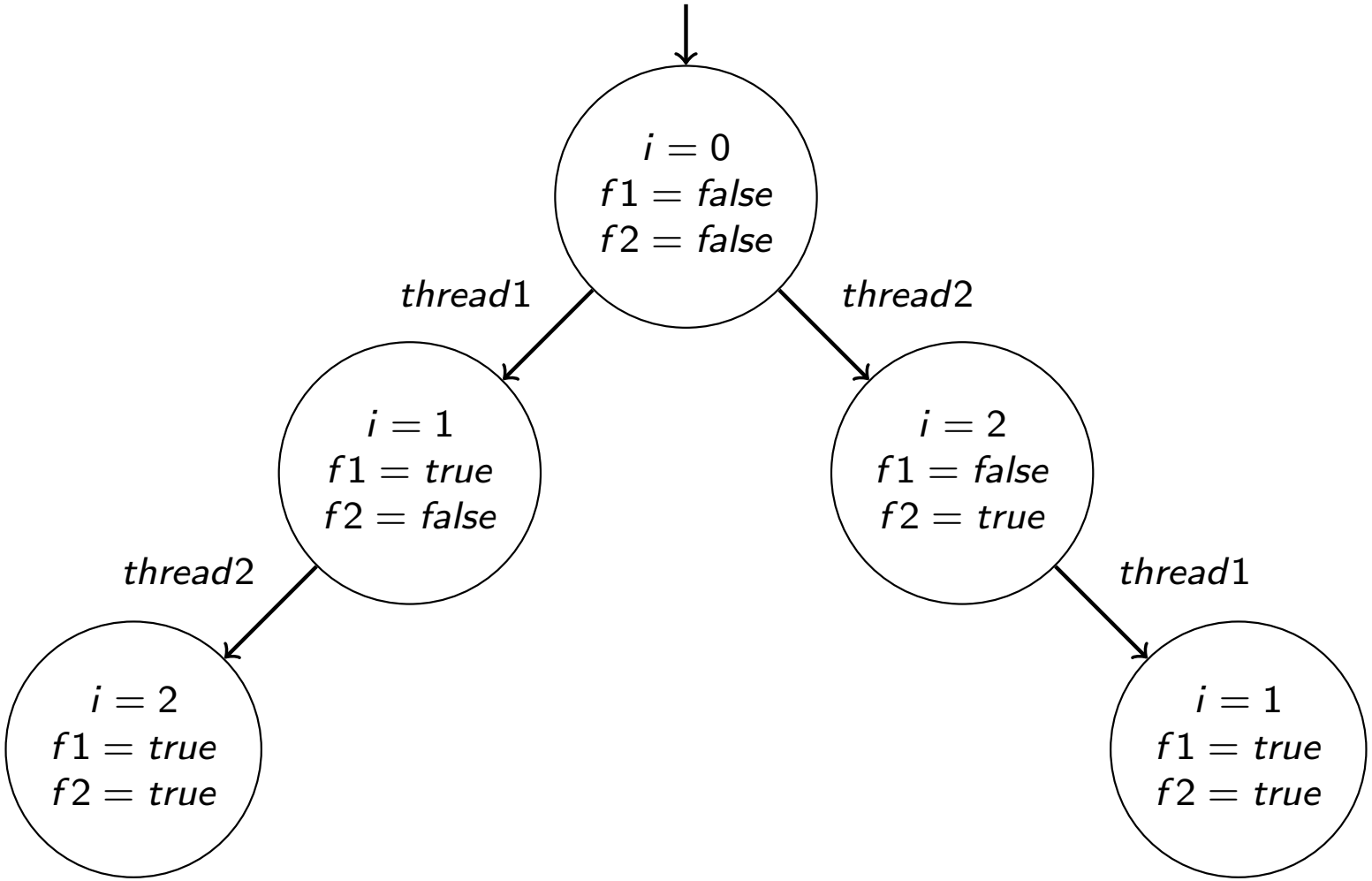
Race condition example
C++ code in DiViNe

What kind of sorcery is that?!?

NuSMV

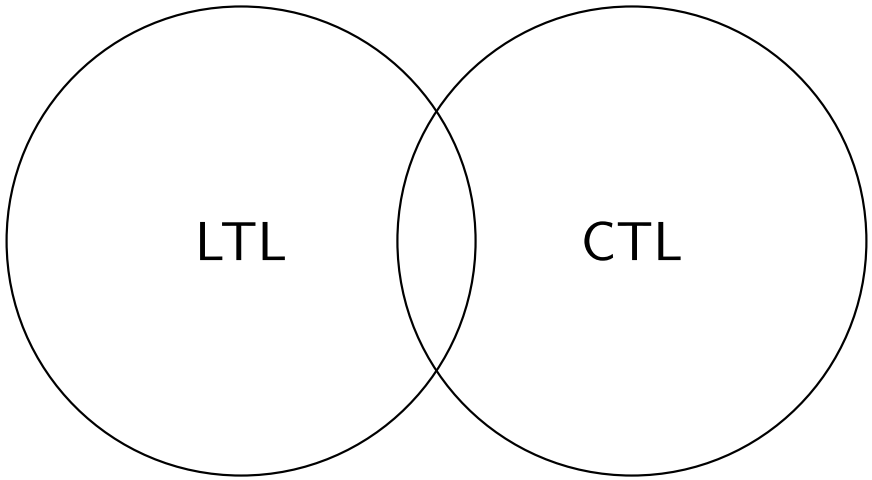
Race condition example
SMV model for NuSMV

State space of our race condition



Property representation

A rigorous, simple, yet expressive language is needed.



die Speisekarte

Wikipedia

Yahoda

Other C++ guys

- CBMC - a bounded MC, provides Visual Studio (2010, 2012), Eclipse integration
- SatAbs - relies on NuSMV, boolean abstraction
- Wolverine - HOARE style correctness, lazy abstraction
- LLBMC - competitor to DiViNe, German

And... does it work?

Brute force description: a 64KB program possibly has $\sim 10^{150}$ states with $\sim 10^{300}$ transition.
Exact interpretation: an array of 100 integers has $2^{32^{100}}$ values.
Reduction: cone of influence, attainable values (our i has only $0, 1, 2$.)

What about this guy?

```
int f(unsigned int i) {
    int x = -256;
    while (x < 0) {
        x = x + i; // + 1;
    }
    return x;
}
```

Non-precise methods

Bounded check: only finite unwinding of loops is conducted.
Abstraction: e.g. $int \to \{-, 0, +\}$. Predicate abstraction, boolean abstraction. CEGAR.
Non-MC methods: Abstract interpretation (Parasoft C++ test), Symbolic execution (KLOVER)

Kind-of-demo [2]

CMBC and bounded check demo

[2] CBMC, SatAbs, Wolverine did not work with MVC++2010 even thought they should. Wolverine did not compile on various versions of GCC. CBMC did not compile on Cygwin. LLBMC did does not detect explicit overflow error. Clang Sanitizer has at least three known open bugs that prevented its execution on Debian, NiXos and Cygwin

Extensions and variants

- Timed: UppAll
- Hybrid: HyTech
- Probabilistic: PRISM
- Verilog: EBMC
- Other languages: Python, Java, Haskell

Summary

- Expert tools (require knowledge, skill, time and resources).
- Strong guarantees (maybe less error-prone than pure proofs).
- Theory for more than half a century. Practical use in the last decade.

> *...For the recent Intel® CoreTM i7 design we took a step further and used formal verification as the primary validation vehicle for the core execution cluster, the component responsible for the functional behaviour of all microinstructions...*[3]

[3] Kaivola et. al., Replacing Testing with Formal Verification in Intel® CoreTM i7 Processor Execution Engine Validation, Proceeding CAV '09, 2009

The end

Thank you for your attention