IA169 Model Checking

Introduction

Jan Strejček

Faculty of Informatics
Masaryk University

agenda

- a bit of motivation
- basic information about the course
- overview of model checking
- content of the course

Motivation

Motivation

- modern computer systems become more and more complicated
- humans and society become more and more dependent on them
- bugs can lead to huge economic losses or even loss of life

Motivation

- modern computer systems become more and more complicated
- humans and society become more and more dependent on them
- bugs can lead to huge economic losses or even loss of life
  - maiden flight of Ariane 5 - the rocket was destructed due to integer overflow induced by converting a 64bit floating point number to 16bit signed integer
  - Pentium FDIV bug - a bug in FPU of early Pentium processors
  - ...
- correctness cannot be guaranteed by simulation, testing, or code reviews

Motivation

- modern computer systems become more and more complicated
- humans and society become more and more dependent on them
- bugs can lead to huge economic losses or even loss of life
  - maiden flight of Ariane 5 - the rocket was destructed due to integer overflow induced by converting a 64bit floating point number to 16bit signed integer
  - Pentium FDIV bug - a bug in FPU of early Pentium processors
  - ...
- correctness cannot be guaranteed by simulation, testing, or code reviews

We need better ways to check that a system is correct!

Formal methods

Formal methods are a collection of notations and techniques for describing and analyzing systems. Methods are formal in the sense that they are based on some mathematical theories, such as logic, automata or graph theory.

Doron A. Peled, 2001

Formal methods

Formal methods are a collection of notations and techniques for describing and analyzing systems. Methods are formal in the sense that they are based on some mathematical theories, such as logic, automata or graph theory.

Doron A. Peled, 2001

Formal methods can be used for

- test generation, bug finding,
- verification, security analysis,
- equivalence checking,
- optimization, synthesis, ...

of various systems like software, hardware, protocols, etc.

Basic information about the course

Courses on formal methods at FI MU

situation before summer 2023

- IA169 System Verification and Assurance
  - introductory course on formal methods in general
  - presented by Jiří Barnat
- IA159 Formal Verification Methods
  - selected advanced topics
  - it had IA169 as prerequisite

Courses on formal methods at FI MU

situation before summer 2023

- IA169 System Verification and Assurance
  - introductory course on formal methods in general
  - presented by Jiří Barnat
- IA159 Formal Verification Methods
  - selected advanced topics
  - it had IA169 as prerequisite

current situation

- two independent courses
- IA169 Model Checking
  - basic formal methods for analysis of various systems
  - few of them are used for software (e.g. abstraction, CEGAR, PDR)
- IA159 Formal Methods for Software Analysis
  - methods designed primarily for analysis of software

Courses on formal methods at FI MU

content of IA159 Formal Methods for Software Analysis

- formal aspects of testing: coverage criteria
- automated test generation: greybox and whitebox fuzzing
- deductive verification
- static analysis and abstract interpretation
- shape analysis
- program slicing
- symbolic execution, bounded model checking, k-induction
- configurable program analysis (CPAchecker)
- verification via automata, symbolic execution and interpolation (Ultimate Automizer)
- verification witnesses

Courses on formal methods at FI MU

other courses related to formal methods

- IA085 Satisfiability and Automated Reasoning
- IV022 Principles of elegant programming
- IA072 Seminar on Verification
- IV120 Continuous and Hybrid Systems
- IA175 Algorithms for Quantitative Verification

Literature

- *E. M. Clarke, O. Grumberg, D. Kroening, D. Peled, and R. Bloem: Model Checking, Second Edition, MIT, 2018.*
- *Ch. Baier and J.-P. Katoen: Principles of Model Checking, MIT, 2008.*
- *E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem: Handbook of Model Checking, Springer, 2018.*
- some topics are not covered by these books; relevant sources will be referred and available in Study materials in IS

Lectures, seminars, and examination

- lectures every week (except 11 April 2024)
- seminar every other week starting on 7 March 2024
- no intrasemestral tests, no mandatory homeworks
- there will be an oral exam at the end

Overview of the model checking

Model checking

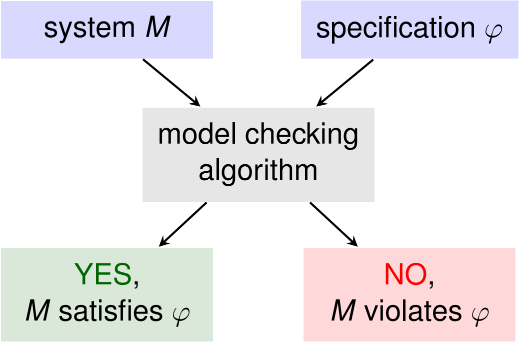
Goal of model checking

Decide whether a given system satisfies a given specification.

Model checking

Goal of model checking

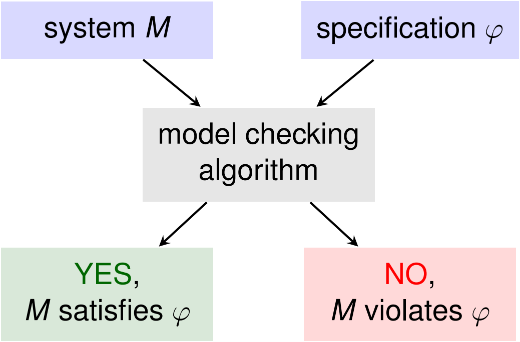
Decide whether a given system satisfies a given specification.



Model checking

Goal of model checking

Decide whether a given system satisfies a given specification.

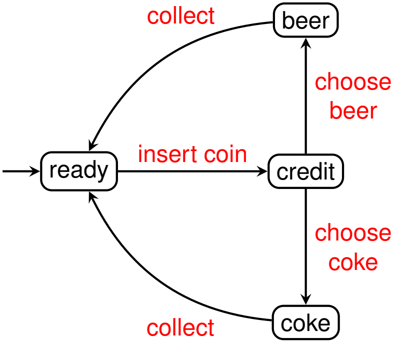


We are checking whether M is a model of φ, hence the name.

Model checking: actions versus states

action-based model checking

- system exhibits actions and specification talks about actions
- basic formalism for system description is labeled transition system

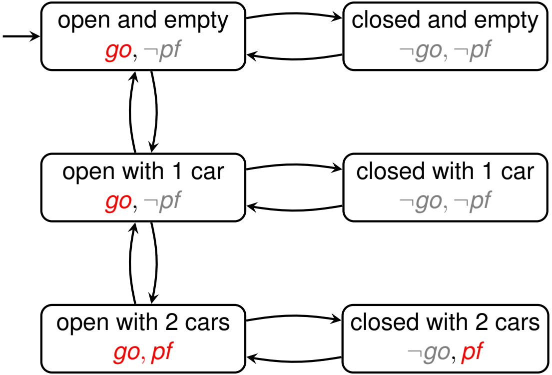


Model checking: actions versus states

state-based model checking

- there is a set of atomic propositions (AP)
- in each state of the system, each atomic proposition either valid or not
- specification talks about atomic propositions
- basic formalism for system description is Kripke structure

go = gate open
pf = parking full



Model checking: specification

safety property

- all reachable states/actions are safe (no error state/action is reachable)
- examples
  - at every moment, at most one process is in a critical section
  - the system cannot reach any deadlock state
- we talk about reachability analysis

Model checking: specification

safety property

- all reachable states/actions are safe (no error state/action is reachable)
- examples
  - at every moment, at most one process is in a critical section
  - the system cannot reach any deadlock state
- we talk about reachability analysis

linear time property

- each run of the system (a linear sequence of actions or states) has to satisfy a given property
- examples
  - each request is eventually processed
  - a system runs forever unless it receives a termination signal
- property can be specified by an ω-automaton (e.g. Büchi automaton) or a formula of linear temporal logic (LTL) or other linear time logic

Model checking: specification

branching time property

- the behavior of the system (a single tree of actions or states) has to satisfy a given property
- examples
  - there exists a sequence of inputs leading to certain action
  - in each moment of system execution, the execution can be terminated
- property can be specified by an formula of computation tree logic (CTL) or CTL* or other branching time logic

Model checking: specification

branching time property

- the behavior of the system (a single tree of actions or states) has to satisfy a given property
- examples
  - there exists a sequence of inputs leading to certain action
  - in each moment of system execution, the execution can be terminated
- property can be specified by an formula of computation tree logic (CTL) or CTL* or other branching time logic

equivalence with another system

- the system is equivalent to another given system up to a given equivalence, for example strong or weak bisimulation equivalence
- we talk about equivalence checking
- presented in IA006 Selected Topics on Automata Theory aka FJA II

Model checking: finite and infinite system

finite systems

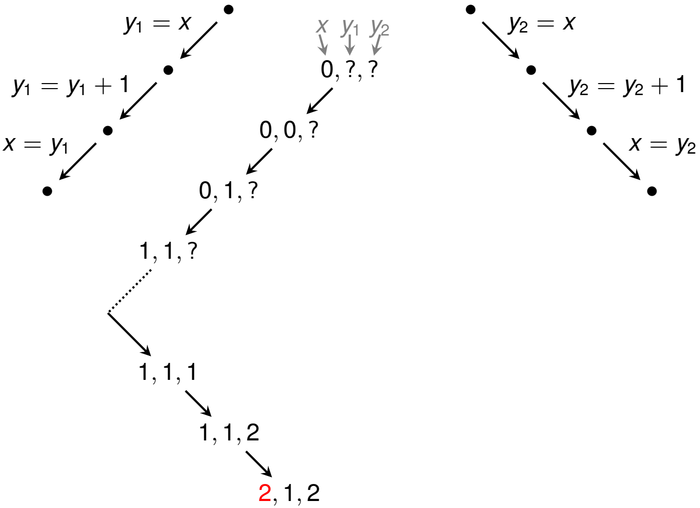
- systems with finitely many states
- basically every model checking problem is decidable by exhaustive (explicit of implicit) enumeration of its reachable states
- in practice, systems are not described by a labeled transition system or a Kripke structure, but by some implicit description in a programming language, VHDL, some modelling language (e.g. Promela in SPIN), etc.
- the state space of the system can be extremely large even if its description is small due to large data domains, parallelism, etc.
- known as state explosion problem, it can make model checking infeasible
- the state explosion problem can be mitigated by
  - partial order reduction
  - symbolic algoirthms where sets of states are represented by formulae or BDDs
  - abstraction applied to the original system

Example

- two parallel processes, each increases x by 1
- what is the result of the computation starting with $x = 0$?

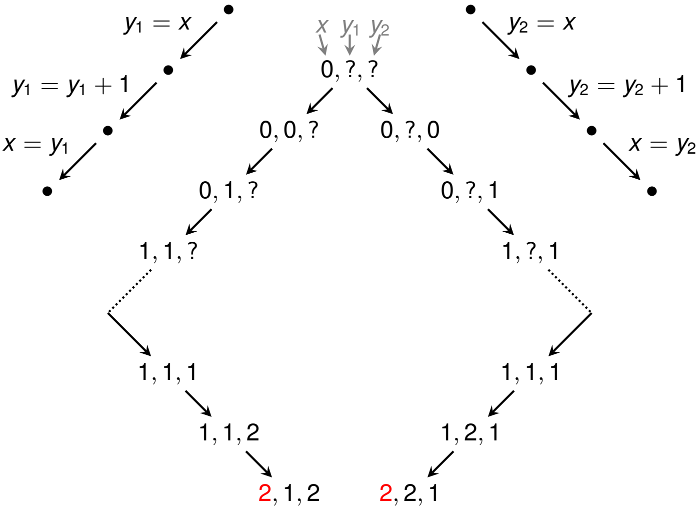
Example

- two parallel processes, each increases x by 1
- what is the result of the computation starting with $x = 0$?



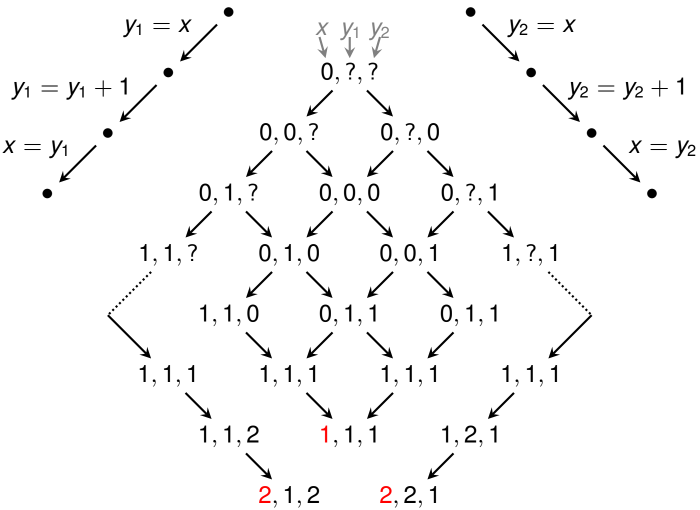
Example

- two parallel processes, each increases x by 1
- what is the result of the computation starting with $x = 0$?



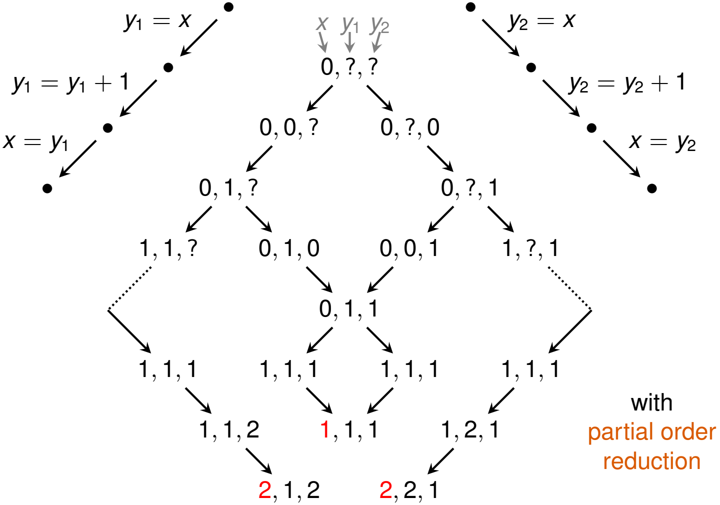
Example

- two parallel processes, each increases x by 1
- what is the result of the computation starting with $x = 0$?



Example

- two parallel processes, each increases x by 1
- what is the result of the computation starting with $x = 0$?



Model checking: finite and infinite systems

infinite systems

- must be described in a finite way using modelling language, pseudocode, etc.
- there has to be some regularity in the state space
- undecidable in general
- decidable for some classes of systems and properties

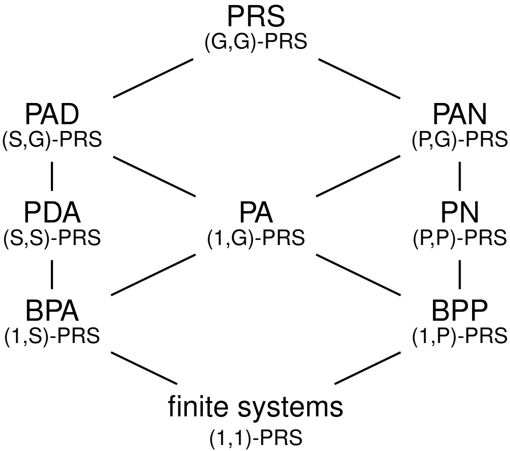
Model checking: finite and infinite system

```
y1=0;
y2=0;
while (y2 != x2) {
   y1 = y1 + x1;
   y2++;
}
```

- verification of algorithms vs. verification of programs
- if all variables are of finite type (e.g. `int`), the system is finite and problems are decidable
- if variables are (unbounded) integers, the state space is infinite
- decidability of the halting problem for algorithms and programs

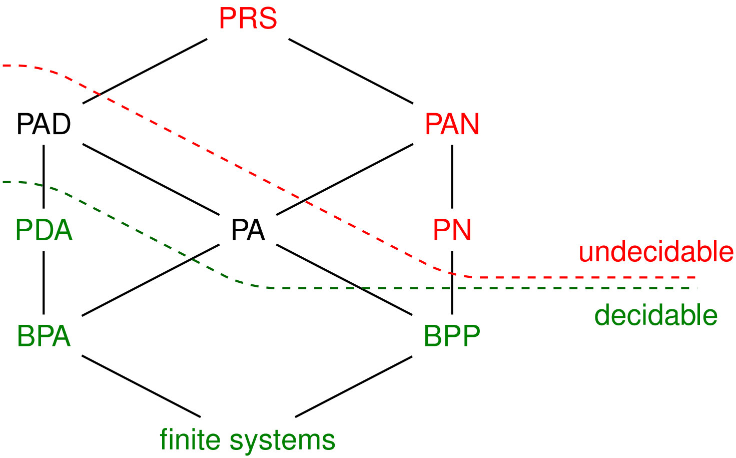
PRS-hierarchy of infinite systems

- the hierarchy of process rewrite systems (PRS) presented in IA006 Selected Topics on Automata Theory aka FJA II
- contains many known classes of infinite systems including BPA, BPP, PA, Petri nets (PN), and pushdown processes (PDA)



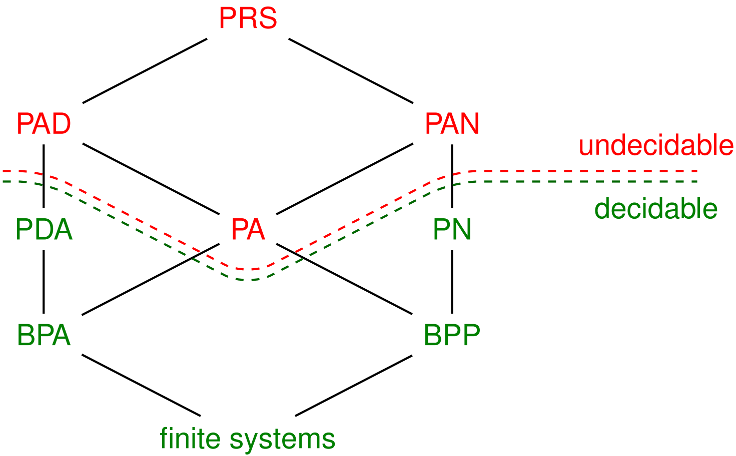
Decidability of equivalence checking

the decidability boundary of strong bisimulation in the PRS-hierarchy



Decidability of model checking

the decidability boundary of the action-based LTL model checking in the PRS-hierarchy

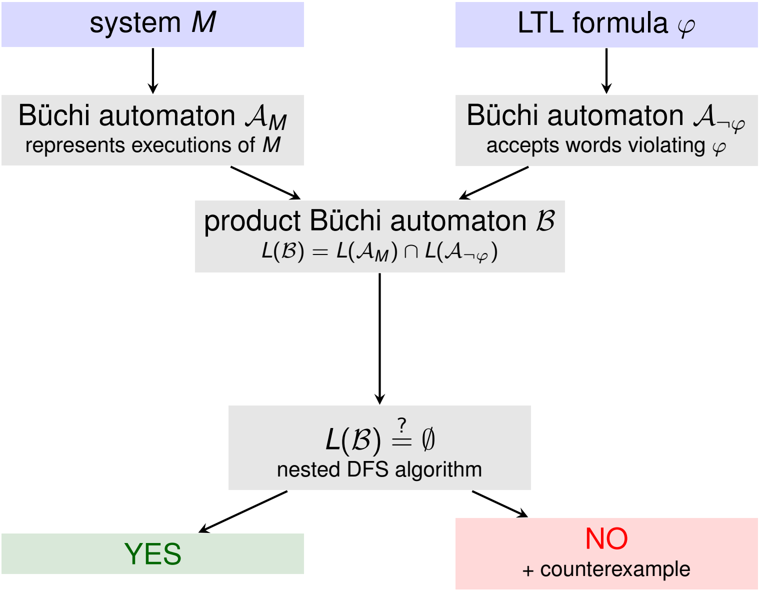


Content of the course

Actual topics of the course

- introduction to model checking - 1 lecture
- LTL model checking of finite systems - 3 lectures
- CTL model checking of finite systems - 2 lectures
- bounded model checking and k-induction - 1 lecture
- reachability in pushdown systems - 1 lecture
- abstraction and CEGAR - 2 lectures
- property-driven reachability (PDR)- 1 lecture
- the remaining lecture can be about
  - partial order reduction
  - LTL model checking of pushdown systems
  - hyperproperties

Automata-based LTL model checking of finite systems



Automata-based LTL model checking of finite systems

system M

LTL formula φ

Büchi automaton $\mathcal{A}_M$
represents executions of M

Büchi automaton $\mathcal{A}_{\neg\varphi}$
accepts words violating φ

product Büchi automaton $\mathcal{B}$
$L(\mathcal{B}) = L(\mathcal{A}_M) \cap L(\mathcal{A}_{\neg\varphi})$

reduced automaton $\mathcal{B}'$
by partial order reduction

$L(\mathcal{B}') \stackrel{?}{=} \emptyset$
nested DFS algorithm

YES

NO
+ counterexample

Automata-based LTL model checking of finite systems

system M

LTL formula φ

abstraction $\longrightarrow$

Büchi automaton $\mathcal{A}_M$
represents executions of M

Büchi automaton $\mathcal{A}_{\neg\varphi}$
accepts words violating φ

product Büchi automaton $\mathcal{B}$
$L(\mathcal{B}) = L(\mathcal{A}_M) \cap L(\mathcal{A}_{\neg\varphi})$

reduced automaton $\mathcal{B}'$
by partial order reduction

$L(\mathcal{B}') \stackrel{?}{=} \emptyset$
nested DFS algorithm

YES

NO
+ counterexample

CTL model checking, bounded model checking, reachability in PDA

CTL model checking of finite systems

- definition of CTL
- basic algorithm
- binary decision diagrams (BDD)
- symbolic algorithm based on BDDs
- definition of CTL*

CTL model checking, bounded model checking, reachability in PDA

CTL model checking of finite systems

- definition of CTL
- basic algorithm
- binary decision diagrams (BDD)
- symbolic algorithm based on BDDs
- definition of CTL*

bounded model checking and k-induction

- finite systems represented by propositional formulae
- SAT-based algorithm for bounded model checking
- extension with k-induction

CTL model checking, bounded model checking, reachability in PDA

CTL model checking of finite systems

- definition of CTL
- basic algorithm
- binary decision diagrams (BDD)
- symbolic algorithm based on BDDs
- definition of CTL*

bounded model checking and k-induction

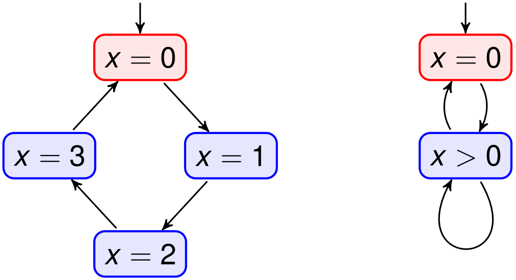
- finite systems represented by propositional formulae
- SAT-based algorithm for bounded model checking
- extension with k-induction

reachability in pushdown systems

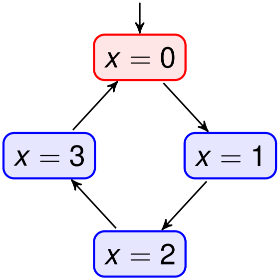
- formalism of pushdown systems (PDA) for description of infinite systems
- algorithm for reachability in PDA
- (algorithm for state-based LTL model checking of PDA)

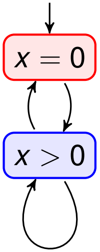
Abstraction and CEGAR

- reduces the size of systems to be analyzed
- can transform an infinite system into a finite one
- the set of system behaviours is usually increased (source of false alarms)



- reduces the size of systems to be analyzed
- can transform an infinite system into a finite one
- the set of system behaviours is usually increased (source of false alarms)





- counterexample-guided abstraction refinement (CEGAR)

Property-driven reachability (PDR), hyperproperties

property-driven reachability (PDR)

- recent approach to reachability analysis based on SAT solving
- implemented in IC3
- very successful in particular in hardware verification

Property-driven reachability (PDR), hyperproperties

property-driven reachability (PDR)

- recent approach to reachability analysis based on SAT solving
- implemented in IC3
- very successful in particular in hardware verification

hyperproperties

- properties talking about several behaviours at once
- applications in security
- for example, it can say that some information is not revealed
- can be presented as an extension of LTL

Model checking in practice

- model checking, in particular reachability analysis, is used in practice, especially for hardware systems
- successful tools (like NuSMV) combine abstraction and symbolic algorithms

Model checking in practice

- model checking, in particular reachability analysis, is used in practice, especially for hardware systems
- successful tools (like NuSMV) combine abstraction and symbolic algorithms

success story from Intel

- in hardware development, the main debugging methods are testing or simulation
- in development of execution cluster of Core i7 (2008), formal verification has been used as a primary validation vehicle
  - simulation has been dropped
  - only 3 bugs escaped to silicon (2 other bugs were detected during the pre-silicon stage by full chip testing)
  - this number is usually about 40, the previous minimum is 11
- more information in Kaivola et al: *Replacing Testing with Formal Verification in Intel Core i7 Processor execution Engine Validation*, CAV 2009, LNCS 5643, Springer, 2009.