Introduction to Formal Methods

Lecture 1

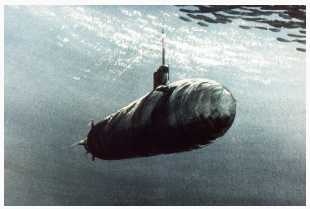
Course Overview

Hossein Hojjat & Fatemeh Ghassemi

September 23, 2018

Course Goals

What is this course about?

	Year	Project	Lines of code
	∼1960s	Apollo 11 mission	145K [John D. Cressler 2016]
	∼1970s	Safeguard Program (US Army anti-ballistic missile system)	2M [John Lamb 1985]
	∼1980s	IBM air traffic control systems	2M [Computerworld 1988]
	∼1990s	Seawolf Submarine	3.6M [Kevin Kelly 1995]
	∼1990s	Boeing 777	4M [Ron J.Pehrson 1996]

Android

$\sim$ **12M LOC**

[Geoff Varrall 2012]

Philips Healthcare MRI scanner

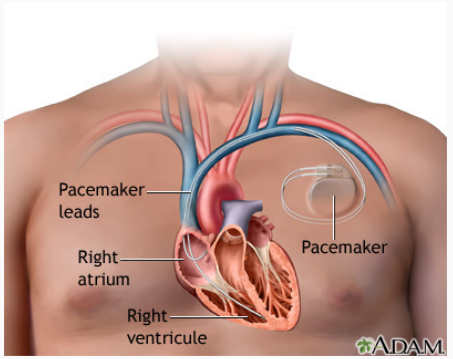
$\sim$ **10M LOC**

[Pierre Van de Laar, Teade Punter 2011]

Ford GT

$\sim$ **10M LOC**

[Jamal Hameedi 2015]



Pacemaker Device

$\sim$ **100K LOC**

[Dev Raheja 2015]

Appolo 11	Safeguard	Traffic Control	Seawolf Submarine	Boeing 777
145K LOC	2M LOC	2M LOC	3.6M LOC	4M LOC

Android

~ **12M LOC**

[Geoff Varrall 2012]

Philips Healthcare MRI scanner

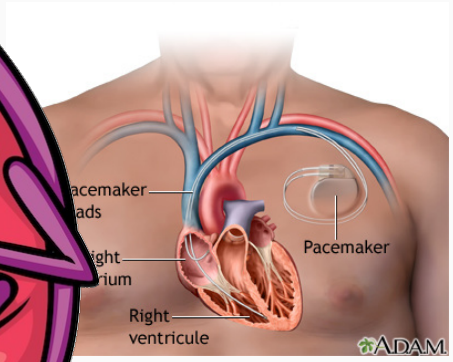
~ **10M LOC**

[Pierre Van de Laar, Teade Punter 2011]

Ford GT

~ **10M LOC**

[Jamal Hameedi 2015]



acemaker Device

~ **100K LOC**

[Dev Raheja 2015]

Appolo 11	Safeguard	Traffic Control	Seawolf Submarine	Boeing 777
145K LOC	2M LOC	2M LOC	3.6M LOC	4M LOC

News Headlines

Software glitch in signalling system led to Joo Koon train collision

YAHOO! NEWS

Dhany Osman
Editor
Yahoo News Singapore 15 November 2017

The New York Times | http://nyti.ms/1Or0eH0

STYLE

Nest Thermostat Glitch Leaves Users in the Cold

Disruptions

By NICK BILTON JAN. 13, 2016

The Nest Learning Thermostat is dead to me, literally. Last week, my once-beloved "smart" thermostat suffered from a mysterious software bug that drained its battery and sent our home into a chill in the middle of the night.

theguardian

Setting the date to 1 January 1970 will brick your iPhone, iPad or iPod touch

Date bug will prevent 64-bit iOS devices from booting up, rendering them inoperable even through fail-safe restore methods using iTunes

Samuel Gibbs
Friday 12 February 2016 08.23 EST

theguardian

Bug displays Chrome user's porn hours later on Apple computer

Student's incognito mode browsing reappeared after closing private window when he loaded video game Diablo III

Stuart Dredge
Thursday 14 January 2016 06.16 EST

2/21/2016 Volvo recalls 59,000 cars over software fault - BBC News

BBC | News | Sport | Weather | Shop | Earth | Travel

Volvo recalls 59,000 cars over software fault

20 February 2016 | Europe

Swedish carmaker Volvo is recalling 59,000 cars across 40 markets over a fault that can temporarily shut down the engine.

Forbes http://onforb.es/1JFAuKy

TAXES 1/27/2016 @ 2:48PM | 2,463 views

'Bug' Exposes Uber Driver's Tax Info, Including Name and Social Security Number

News Headlines

کد خبر : ۸۶۰۷۴ — شنبه / ۳ شهریور ۹۷ / ۰۶:۵۴

باگ در باک

مشکلات اینترنتی دلیل وقفه درسوخت‌رسانی جایگاه‌های استان و مشهد اعلام شد

در حالی که انبار شرکت نفت به اندازه کافی فراورده داشت، به دلیل اختلال در سایت سامانه فروش شرکت نفت این مشکل به وجود آمد ولی پس از رفع مشکل، روند خرید سوخت به روال عادی بازگشت.

۶ دی ۱۳۹۵ - ۰۹:۰۷

رئیس کانون توسعه دهندگان فضای مجازی:

۴۰درصد وبسایتهای دولتی باگ امنیتی دارند/ ضعف پرتالها در بروزرسانی

رئیس کانون توسعه دهندگان فضای مجازی گفت: مطابق بررسی های صورت گرفته بالغ بر ۴۰ درصد وبسایتها و پرتالهای دولتی باگ امنیتی دارند و هکرهای تازه کار می توانند به راحتی در این سایتها نفوذ کنند.

کد خبر: ۱۹۵۹۶۵ — تاریخ انتشار: ۲۲ : ۱۹ - ۰۵ فروردین ۱۳۹۴

صفحه نخست >> سیاست

کشف یک باگ امنیتی در سیستم اینترنتی بانک ملت

پایگاه خبری تحلیلی انتخاب (Entekhab.ir) :

در چند ساعت اخیر اینترنت فارسي پر شده از توييت‌هايي در مورد بانک ملت و يک اشکال امنيتي که در سيستم اين مجموعه کشف شده است.

ZarinPal
@zarinpal
Follow

دوستان و همراهان عزیز زرین‌پال به اطلاع می‌رسانیم که فردی با سو استفاده از باگی که در یکی از روترهای میکروتیک پیدا کرده موفق به بارگذاری یک سایت روی سرور دیگری شده است. لازم به ذکر است که هیچ دسترسی به داده‌های زرین پال وجود ندارد و این مشکل تا چند دقیقه دیگر برطرف می‌شود.

10:01 AM - 3 Aug 2018

97 Retweets 510 Likes

magiran.com > روزنامه ایران > شماره 6589 16/6/96 > صفحه 8 (فناوری) > متن

این صفحه PDF | فهرست مطالب | مطلب قبلی | مطلب بعدی

ایران

نگاه

رفع اختلال موقت سرویس های آنلاین تاکسی یاب

«یک باگ یا مشکل فني در بخش BACK END برنامه ios موجب قطع سرويس دهي اين اپليکيشن شده که در نهايت نيز بسرعت رفع شده است.»

از کاشف باگ سروش قدردانی شد

در حالیکه روز گذشته یکی از کارشناسان امنیت اطلاعات کشورمان باگ مهمی از پیام‌رسان سروش را کشف کرد که به فاصله چند ساعت از وی قدردانی شد.

۱۵ فروردین ۱۳۹۷ - ۱۰:۳۹ | اقتصادی | فناوری اطلاعات | اینترنت | موبایل | نظرات

Question

- We live in a world dominated by software: our lives depend on programs
- Is it possible to write bug-free software?

Reliable Software?

Tech.View: Cars and Software Bugs
May 16th 2010

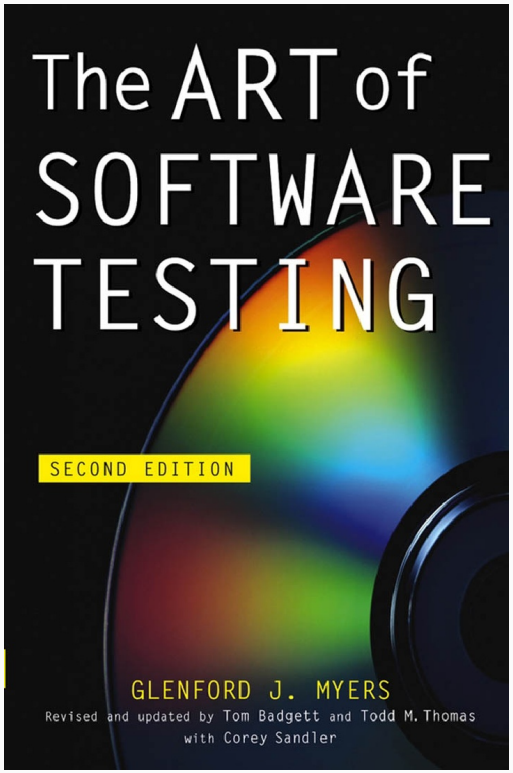
The Economist

"One thing computer programmers agree on is that there is no such thing as a bug-free piece of software. Yes, you can write a five-line "hello world" program and be reasonably confident it contains no errors. But **any piece of software** that does a **meaningful job** will contain hundreds, or even thousands, of **undetected bugs**."

[...]

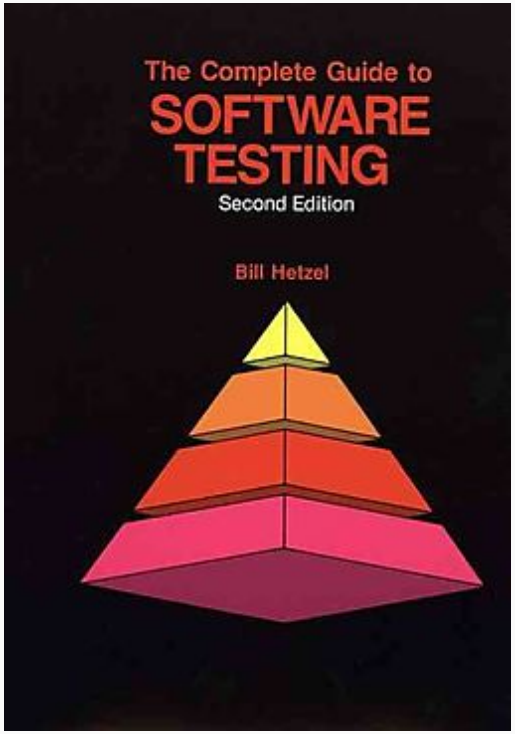
"Microsoft, for instance, reckons to find 10-20 defects per 1,000 lines of code during its in-house testing, and to whittle that down to 0.5 per 1,000 lines by the time the software is released to the public. Even so, a program like Microsoft's venerable **Windows XP** - which had 40m lines of code- would have contained at least **20,000** bugs when launched."

Reliable Software?



"Is it possible to test a program to find all of its errors? We will show you that the answer is negative, even for trivial programs. In general, it is impractical, often impossible, to find all the errors in a program."

Reliable Software?



"In short, we cannot achieve 100 percent confidence no matter how much time and energy we put into it!"

Reliable Software?

David Parnas

Pioneer of Software Engineering

"Software is released for use, not when it is known to be correct, but when the rate of discovering new errors slows down to one that management considers acceptable."

Reliable Software?

روزنامه شماره ۴۲۹۸ › بازار دیجیتال

شماره روزنامه: ۴۲۹۸ تاریخ چاپ: ۱۳۹۷/۰۱/۱۸ شماره خبر: ۳۳۷۲۴۹۹ بزرگنمایی متن خبر

دنیای اقتصاد

مرکز ملی فضای مجازی:

وجود باگ در پیام‌رسان‌ها طبیعی است

دنیای اقتصاد : انتشار مطالبی از ضعف امنیتی یکی از پیام‌رسان‌های داخلی متعلق به سازمان صدا و سیما اواخر هفته گذشته سروصدای زیادی ایجاد کرد. تازه‌ترین واکنش‌ها مربوط به مرکز ملی فضای مجازی است که با صدور اطلاعیه‌ای وجود ضعف امنیتی در ابتدای کار برای این برنامه‌ها را طبیعی دانسته که به مرور رفع می‌شود.

Are bugs a natural byproduct of software development that can't be avoided?

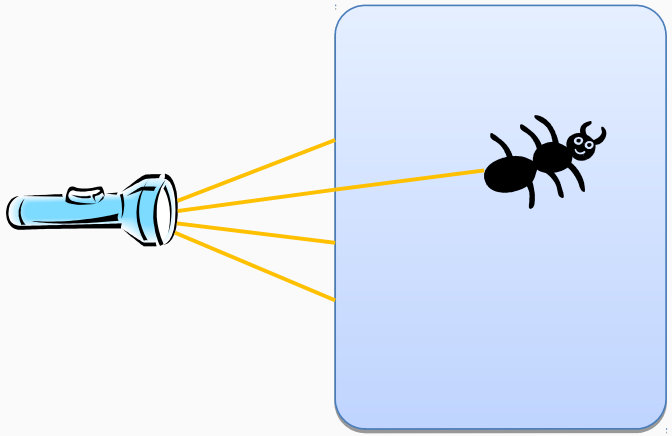
- Flight software for an Airbus A380 includes 120 million lines of code (Simon Bradley, Airbus Group)
- How do we trust such a huge piece of software?

"Good afternoon passengers. This is your captain speaking.
We are currently experiencing a software bug in our flight systems.
Please return to your seats, keep your seat belts fastened and prepare for crash."

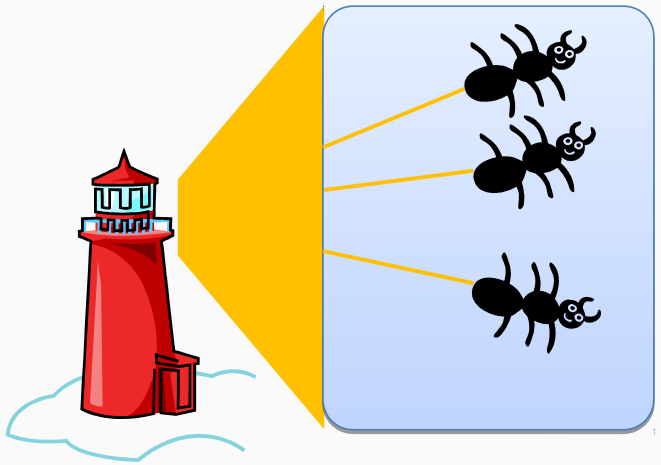
"Since 2001, Airbus has been integrating several tool supported **formal verification** techniques into the development process of avionics software products"

Jean Souyris et al., "Formal Verification of Avionics Software Product", FM 2009

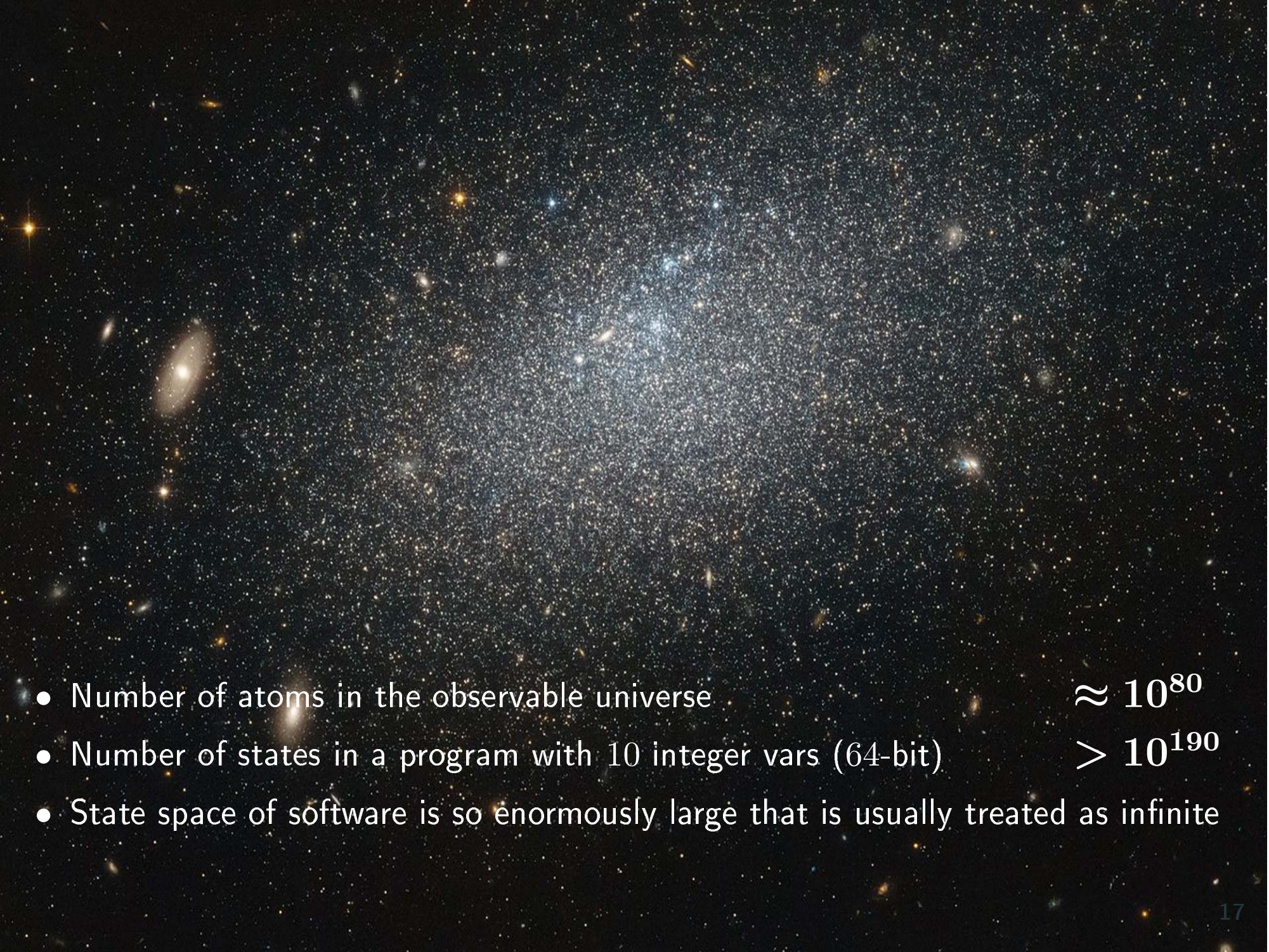
(Formal) Software Verification is the act of proving/disproving that a program is bug-free using mathematics



Testing and simulation can only check a few cases



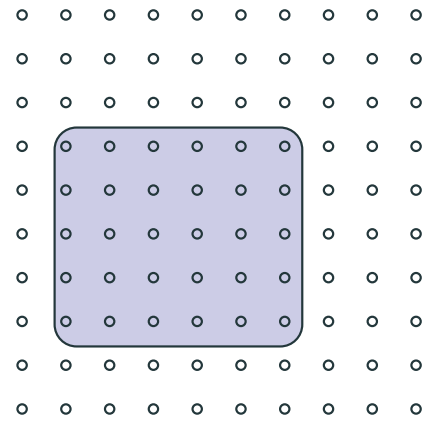
Software verification checks **all** possible behaviors



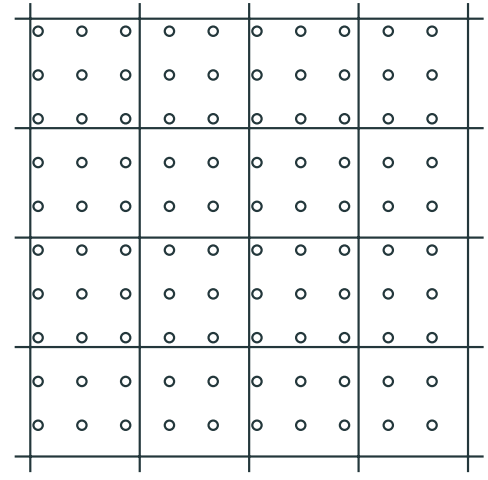
- Number of atoms in the observable universe $\approx 10^{80}$
- Number of states in a program with 10 integer vars (64-bit) $> 10^{190}$
- State space of software is so enormously large that is usually treated as infinite

Software Verification (Infinite Space)

- Automated
  - Check Finite Portion (Bounded Model Checking)
    - − no correctness guarantee
    - + short path to err



  - Check Finite Abstraction (Predicate Abstraction)
    - − incomplete
    - + correctness guarantee



- Semi-Automated (Interactive Theorem Prover)

GCD Example

```
    int x,y;
0:  assume(x≥0 ∧ y≥0);
1:  while(x≠y) {
2:   if (x>y) then
3:     x=x−y;
4:   else y=y−x; }
5:
```

GCD Example

```
    int x,y;
0:  assume(x≥0 ∧ y≥0);
1:  while(x≠y) {
2:   if (x>y) then
3:     x=x−y;
4:   else y=y−x; }
5:
```

After loop (line 5):

x = y = largest positive integer that divides x and y

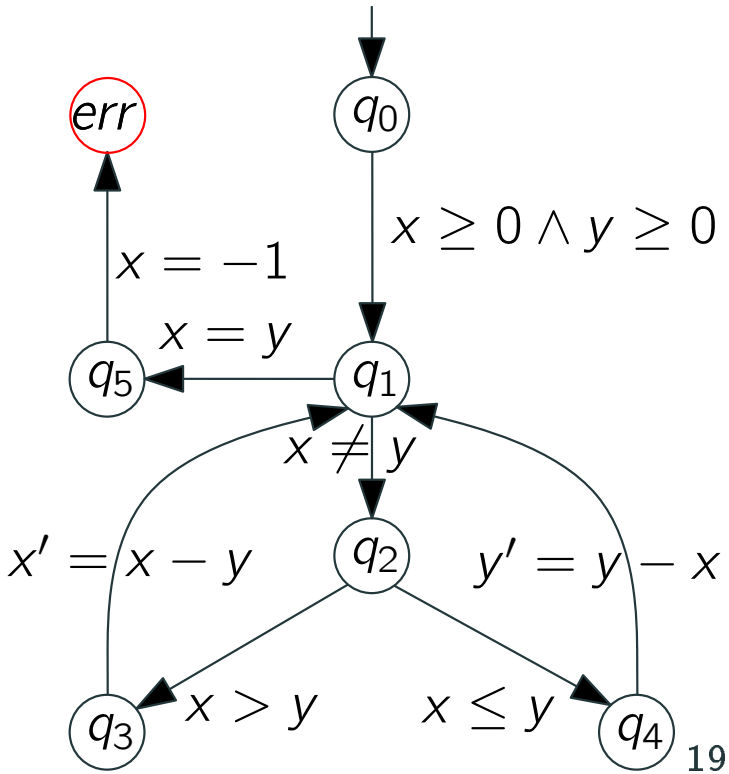
GCD Example

```
    int x,y;
0:  assume(x≥0 ∧ y≥0);
1:  while(x≠y) {
2:   if (x>y) then
3:     x=x−y;
4:   else y=y−x; }
5:  assert (x≠ −1);
```

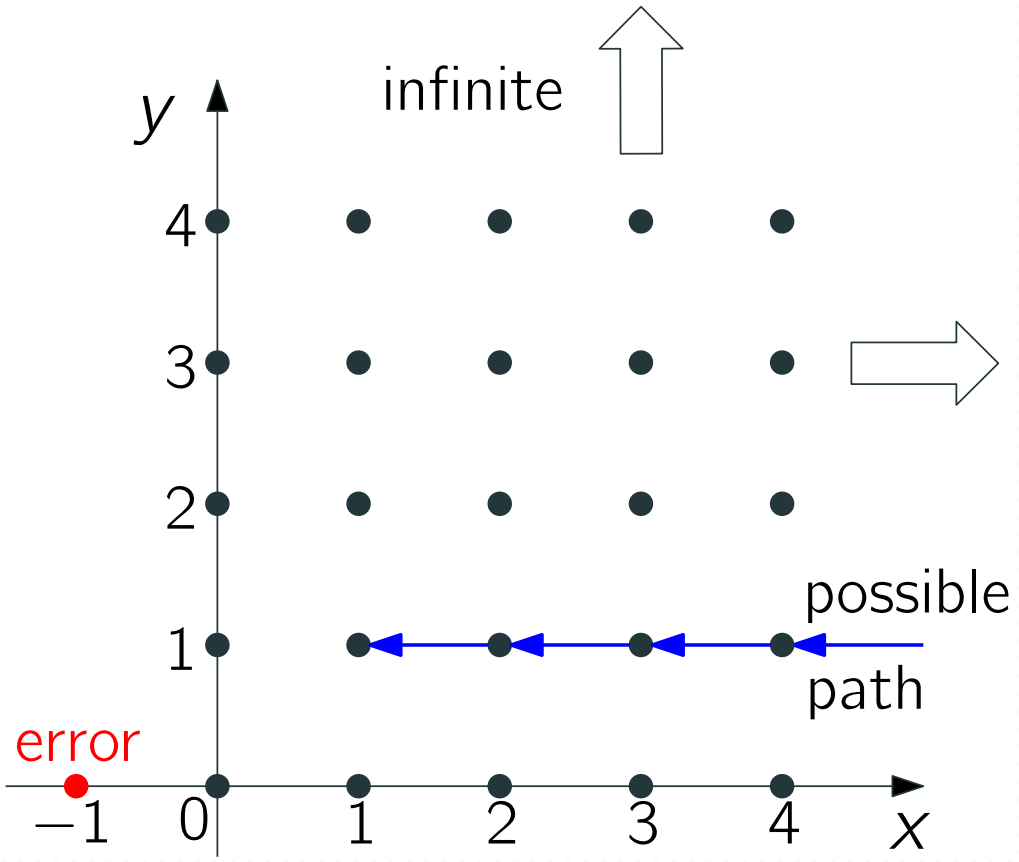
GCD Example

```
    int x,y;
0:  assume(x≥0 ∧ y≥0);
1:  while(x≠y) {
2:   if (x>y) then
3:     x=x−y;
4:   else y=y−x; }
5:  assert (x≠ −1);
```

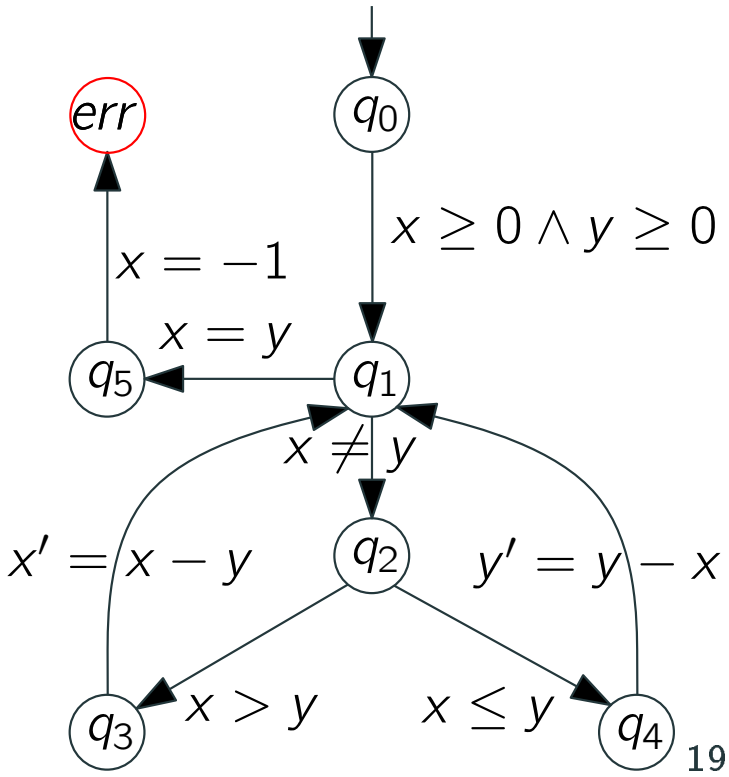
Control Flow Graph (CFG)



GCD Example

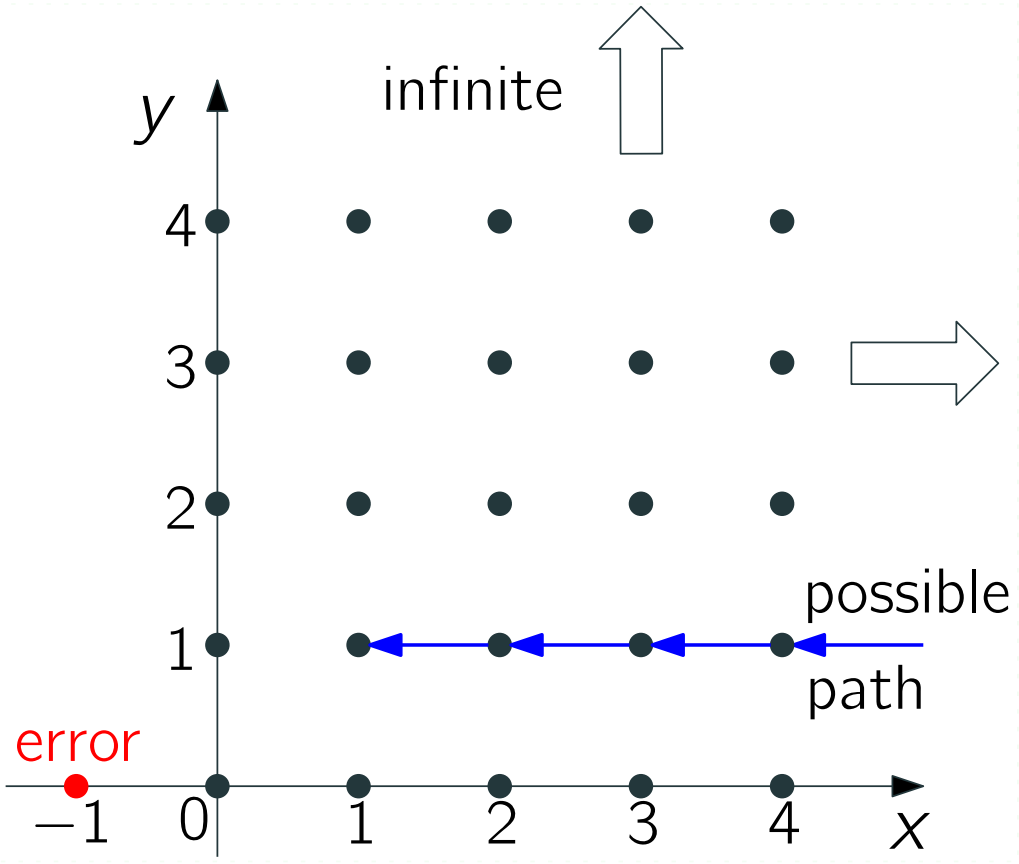


Control Flow Graph (CFG)

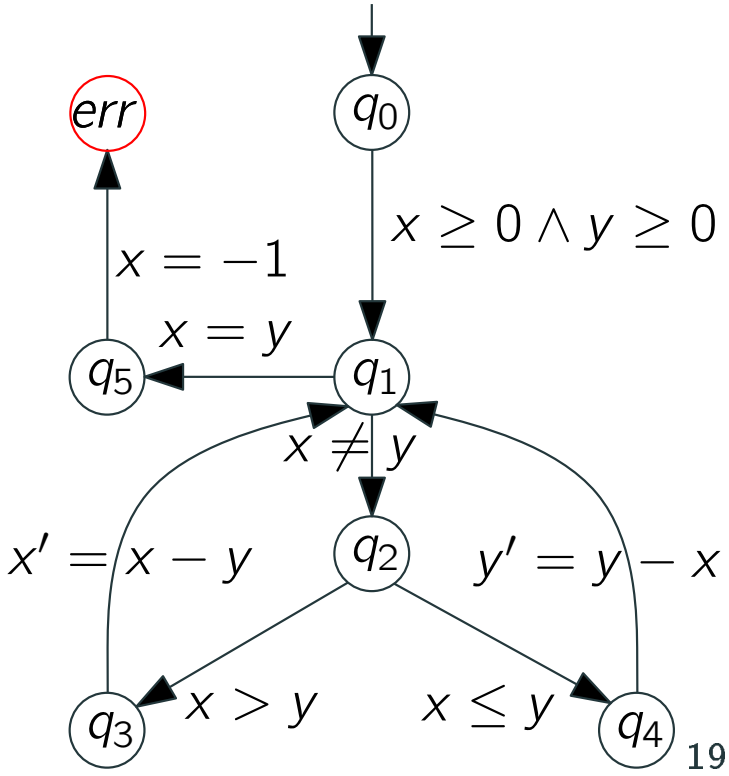


GCD Example

Invariant: A superset of reachable states of program
Prove Safety: Find suitable Invariant

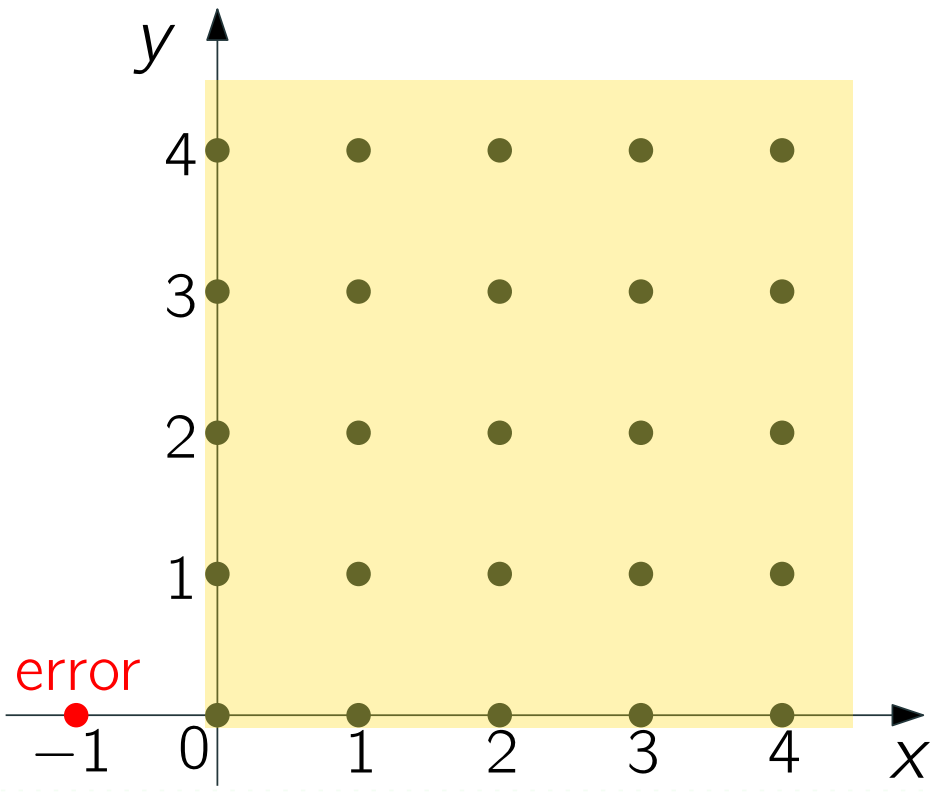


Control Flow Graph (CFG)



GCD Example

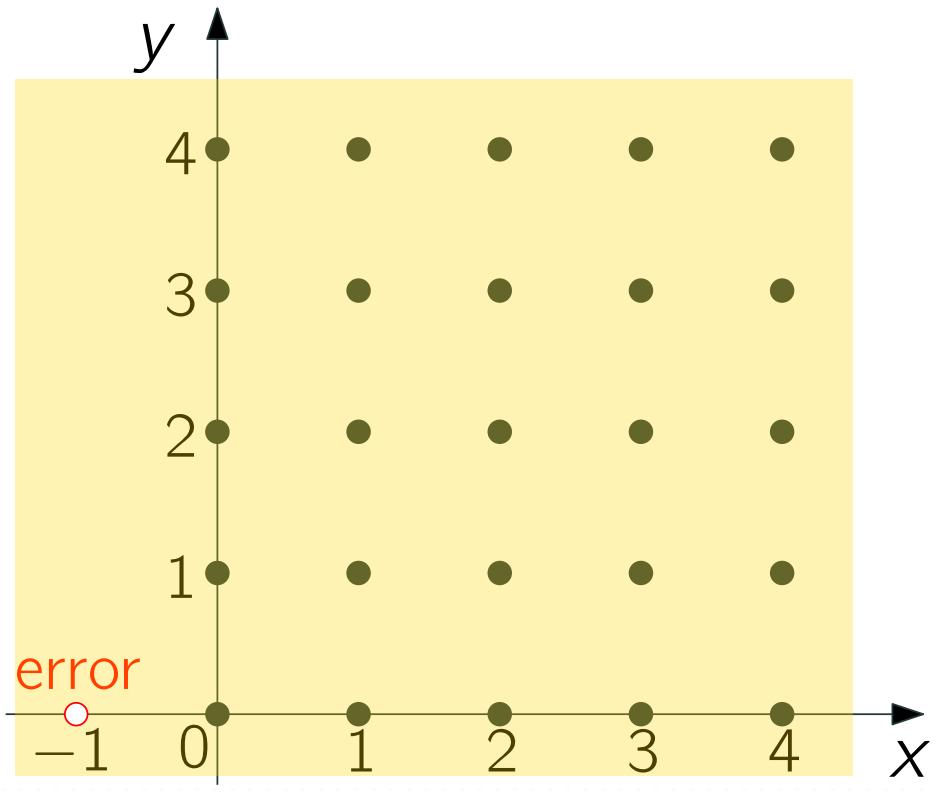
Invariant: A superset of reachable states of program
Prove Safety: Find suitable Invariant



$$x \geq 0 \wedge y \geq 0 \quad \checkmark$$

GCD Example

Invariant: A superset of reachable states of program
Prove Safety: Find suitable Invariant

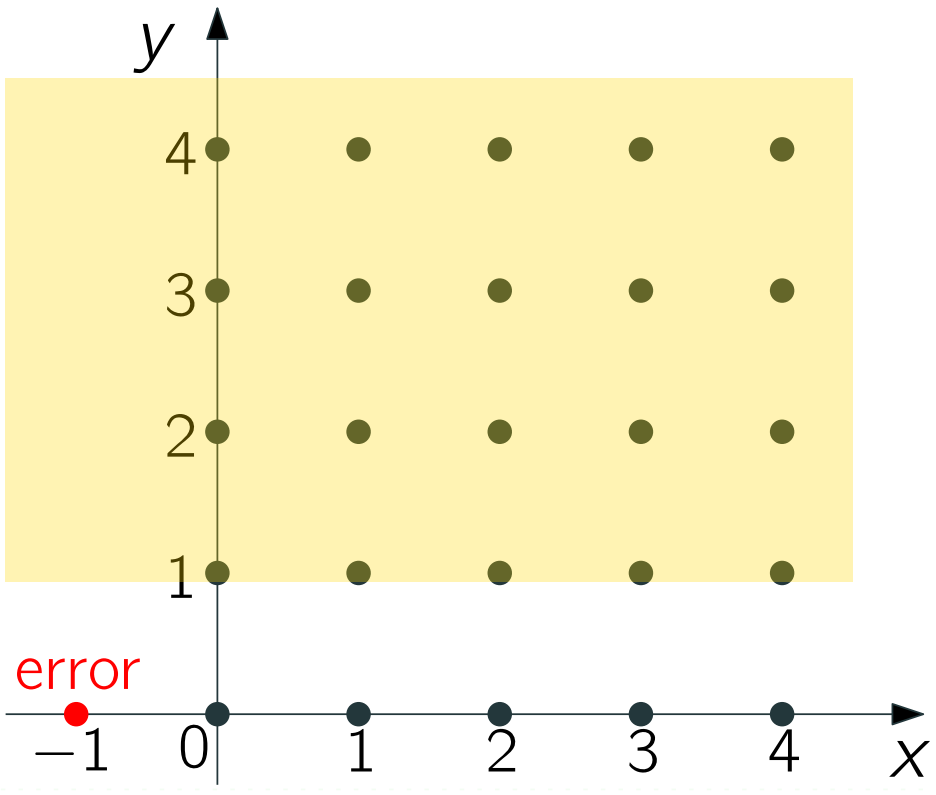


$$x \geq 0 \wedge y \geq 0 \quad \checkmark$$

$$x \neq -1 \quad \checkmark$$

GCD Example

Invariant: A superset of reachable states of program
Prove Safety: Find suitable Invariant



$x \geq 0 \wedge y \geq 0$ ✓

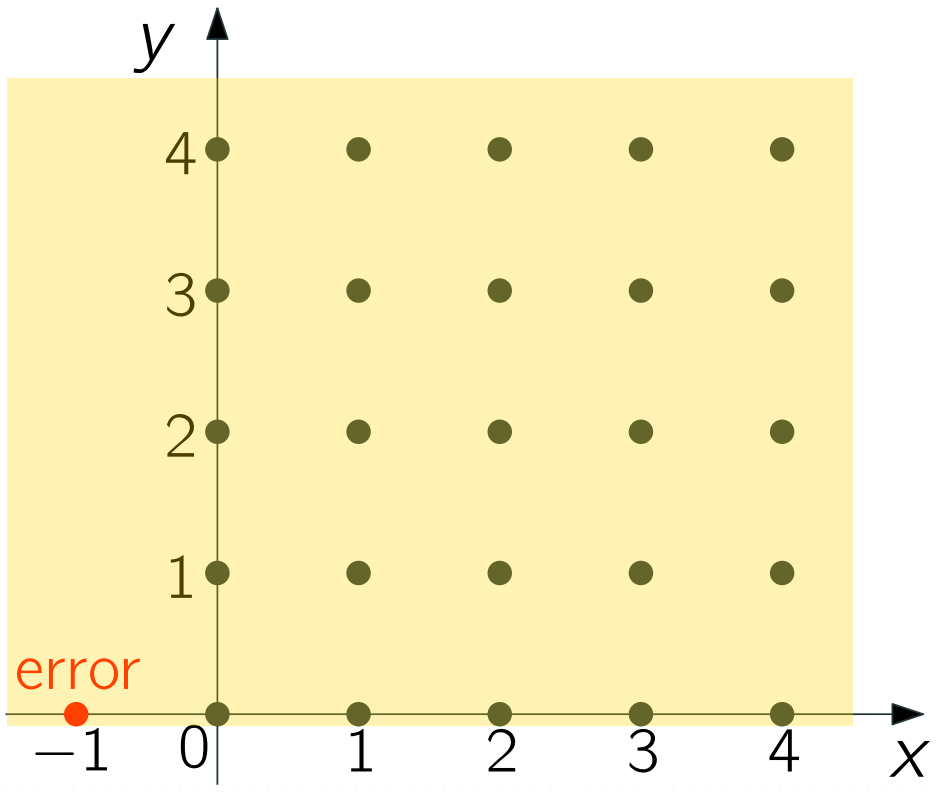
$x \neq -1$ ✓

$y \geq 1$ ×

(not superset)

GCD Example

Invariant: A superset of reachable states of program
Prove Safety: Find suitable Invariant



$x \geq 0 \wedge y \geq 0$ ✓

$x \neq -1$ ✓

$y \geq 1$ ×

(not superset)

$y \geq 0$ ×

(superset, unsuitable for safety proof)

GCD Example

Invariant: A superset of reachable states of program
Prove Safety: Find suitable Invariant

Challenge: Find program invariant automatically & efficiently

Questions of Interest

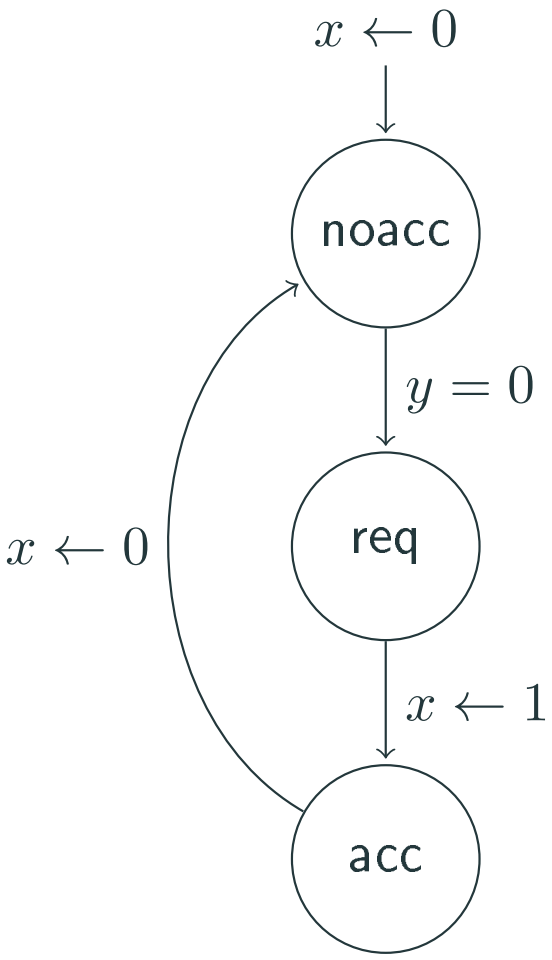
Example questions in program analysis and verification

- Will the program crash?
- Does it compute the correct result?
- Does it leak private information?
- How long does it take to run?
- How much power does it consume?

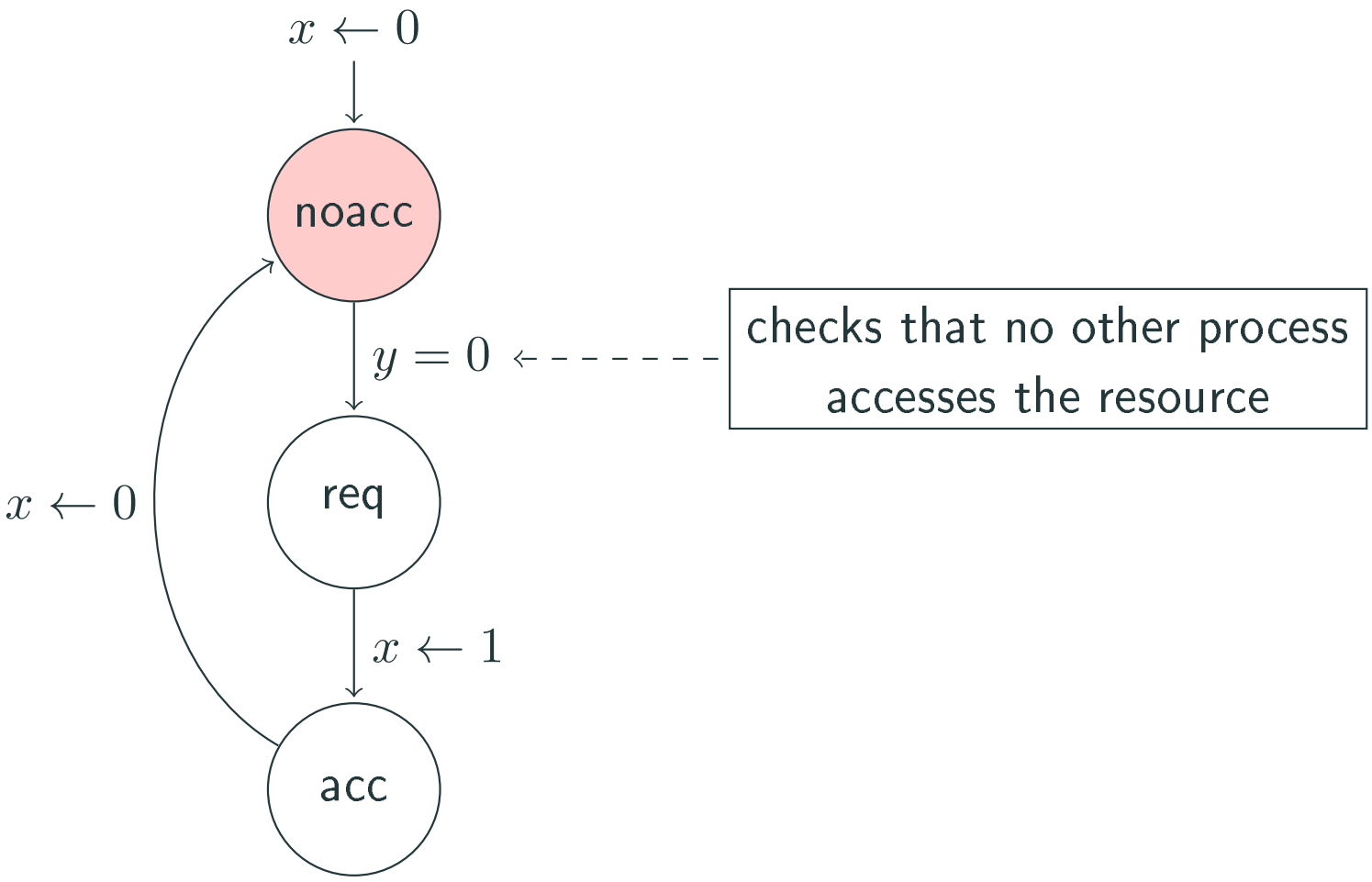
Model Checking

- Wide conceptual gap between the problem and the implementation domains in complex software
- Model Driven Engineering (MDE):
  use **models** to alleviate software complexity
- Model captures relevant aspects of system functionality
- In this course we are interested in **formal** models
  - (based on automata, graph theory, logic)
- Model checking [Clarke/Emerson; Queille/Sifakis 1981]:
  a technique to check if a property is valid in a model

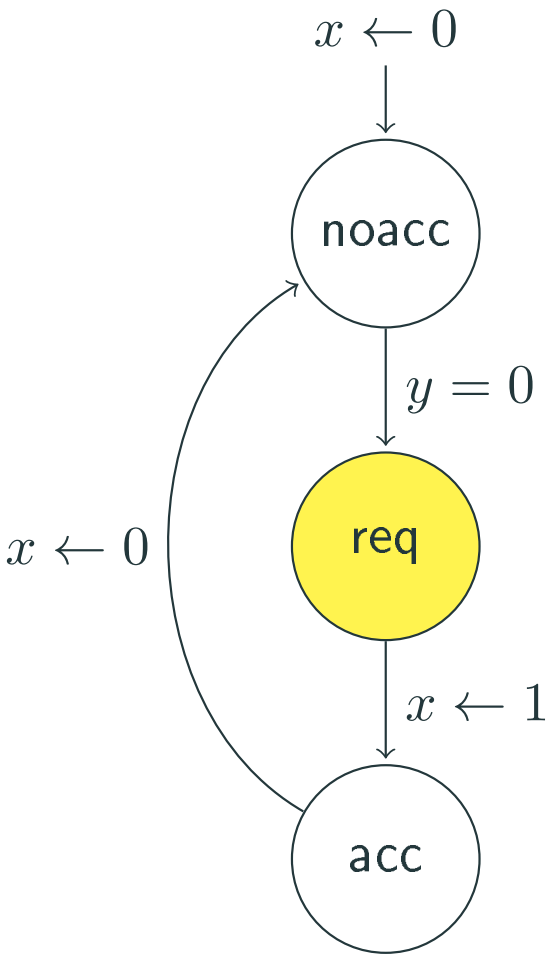
Access to a Shared Resource



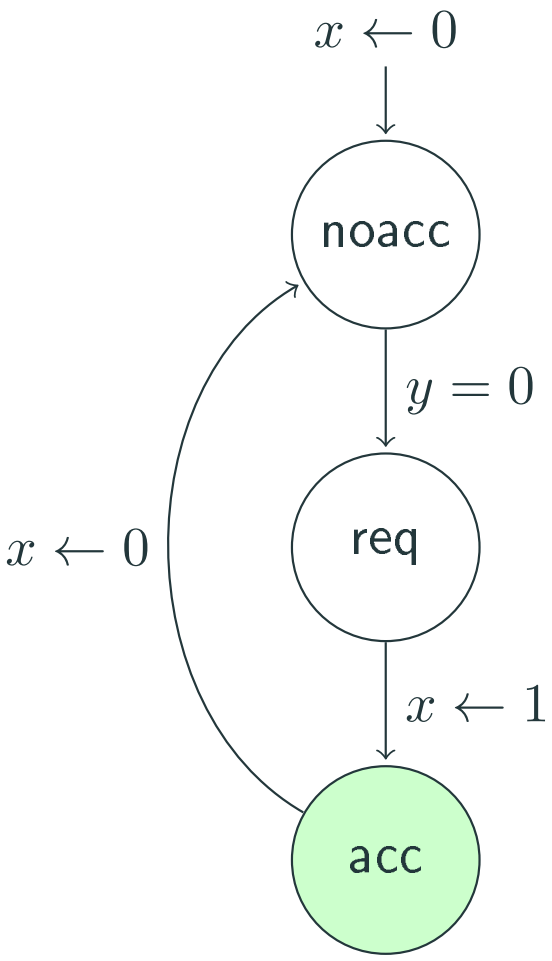
Access to a Shared Resource



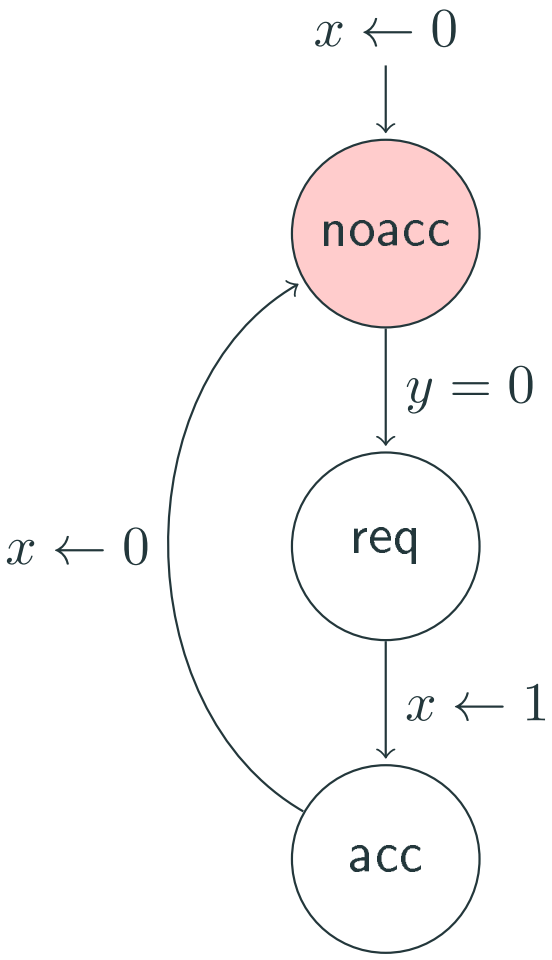
Access to a Shared Resource



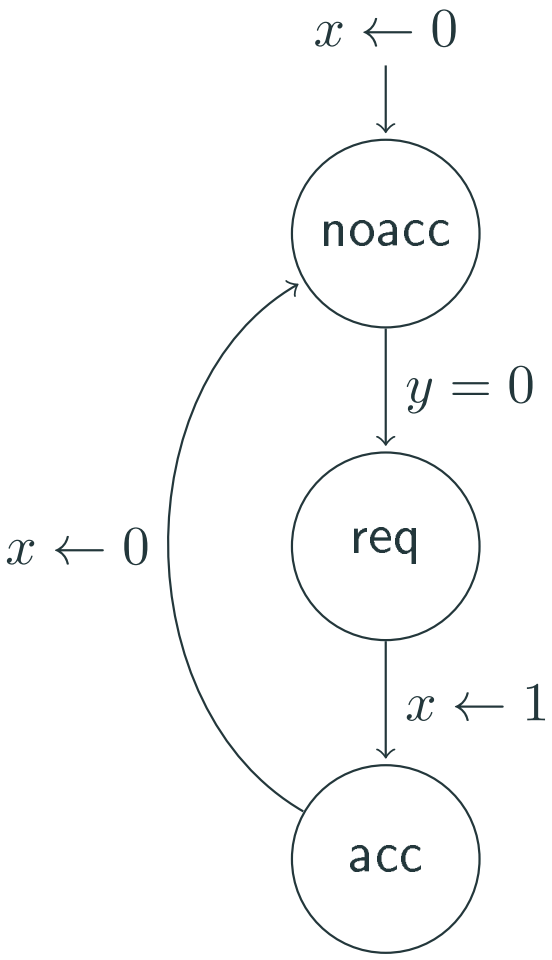
Access to a Shared Resource

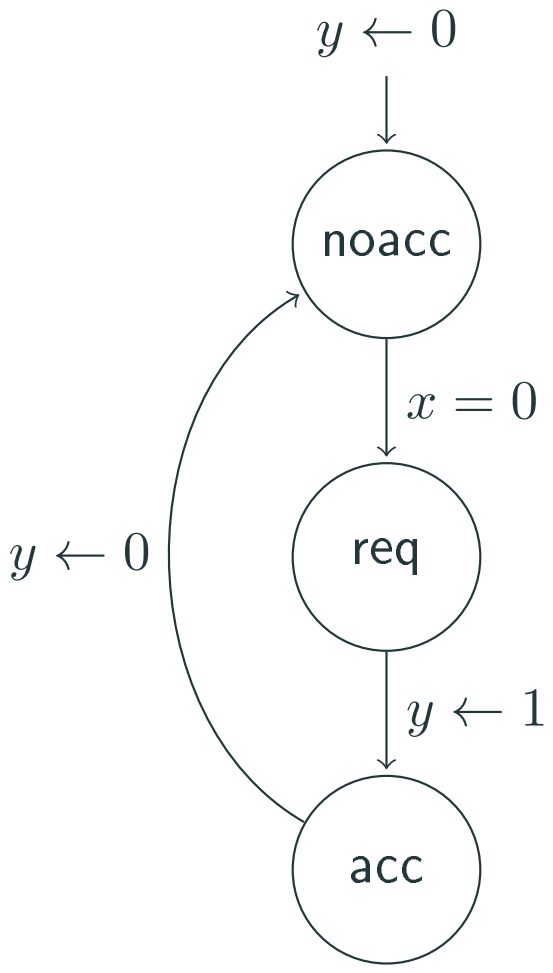


Access to a Shared Resource

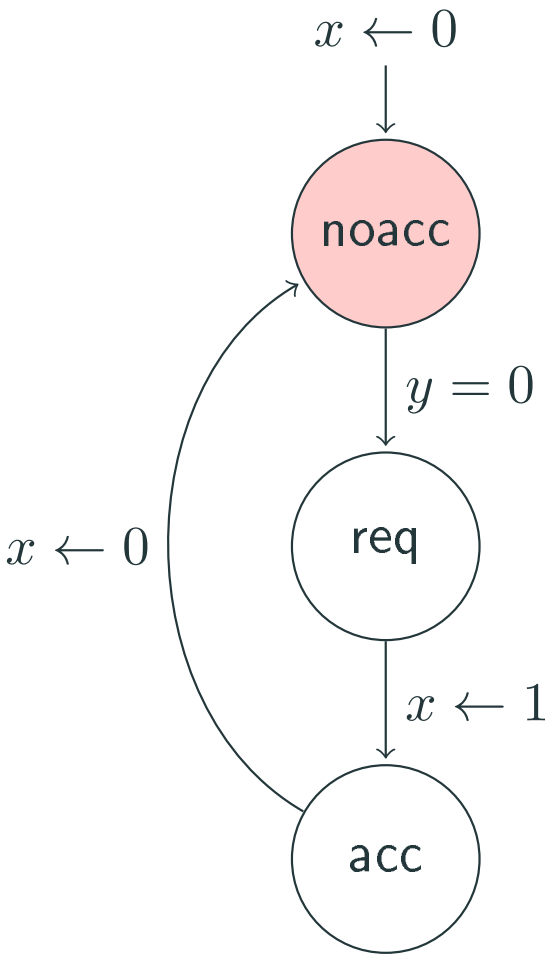


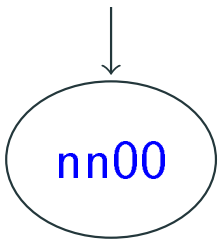
Access to a Shared Resource

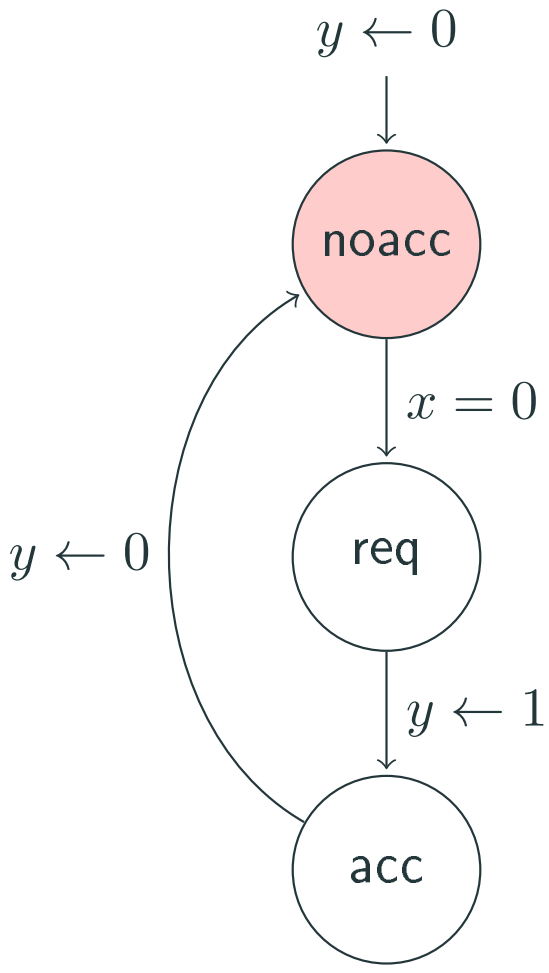




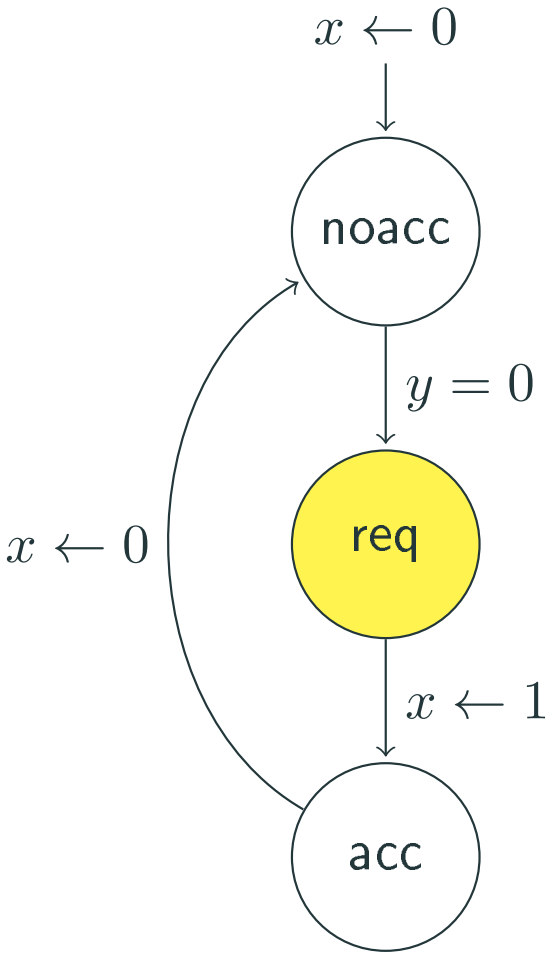
Access to a Shared Resource

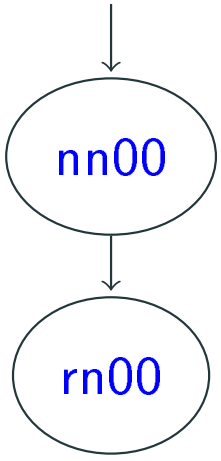


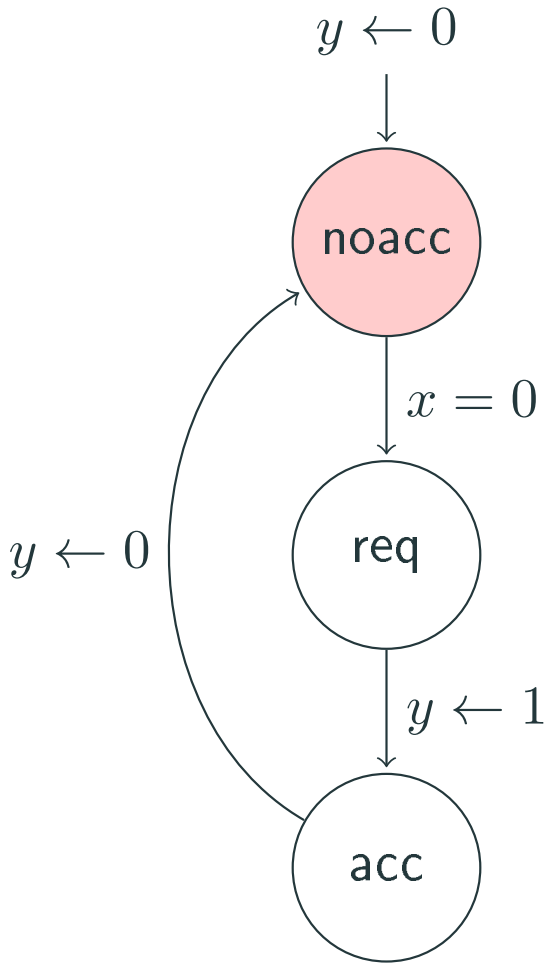




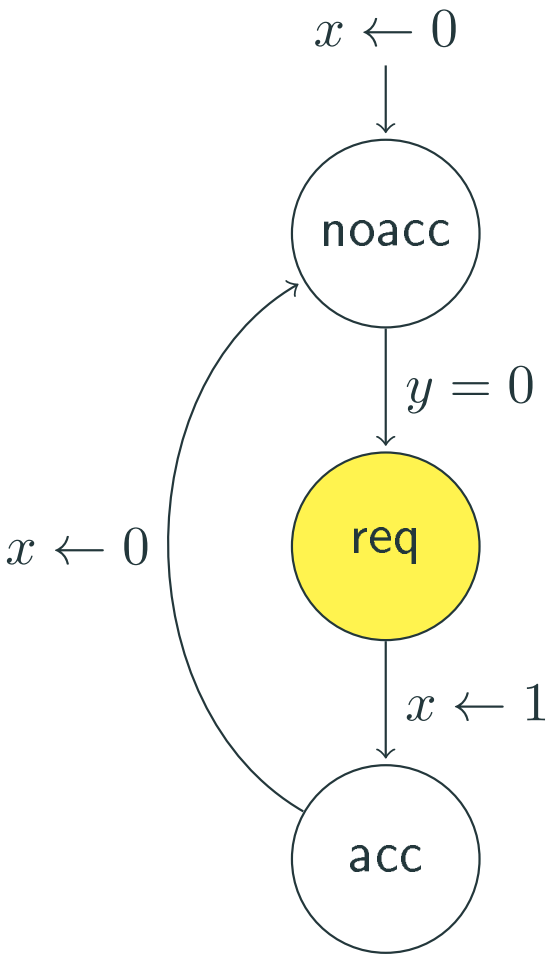
Access to a Shared Resource

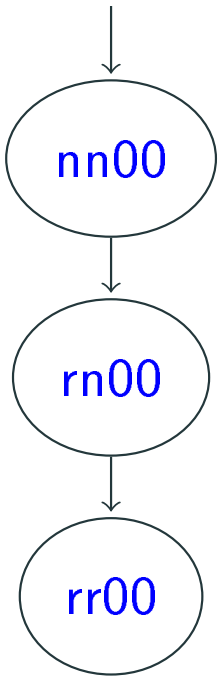


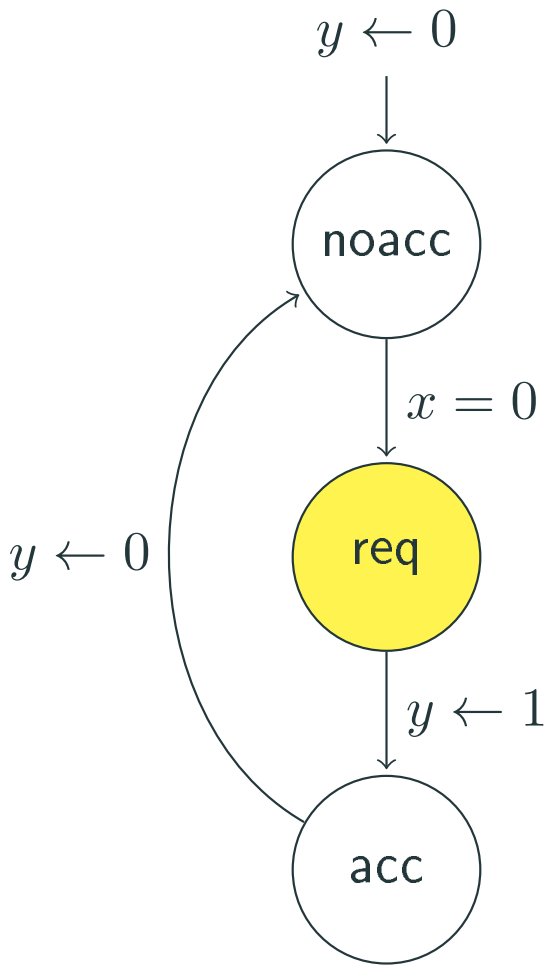




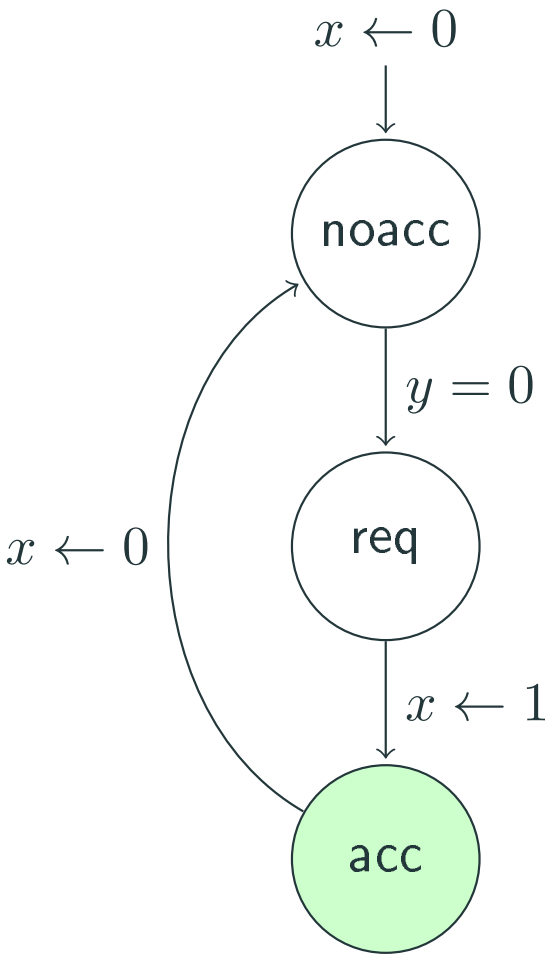
Access to a Shared Resource

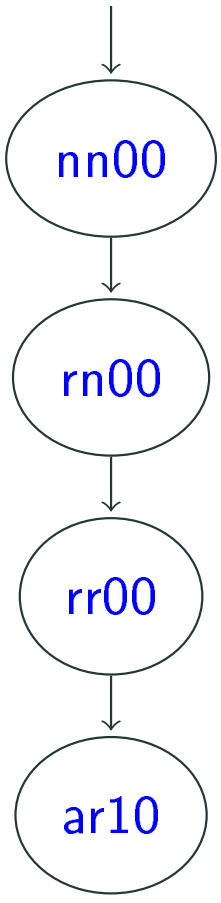


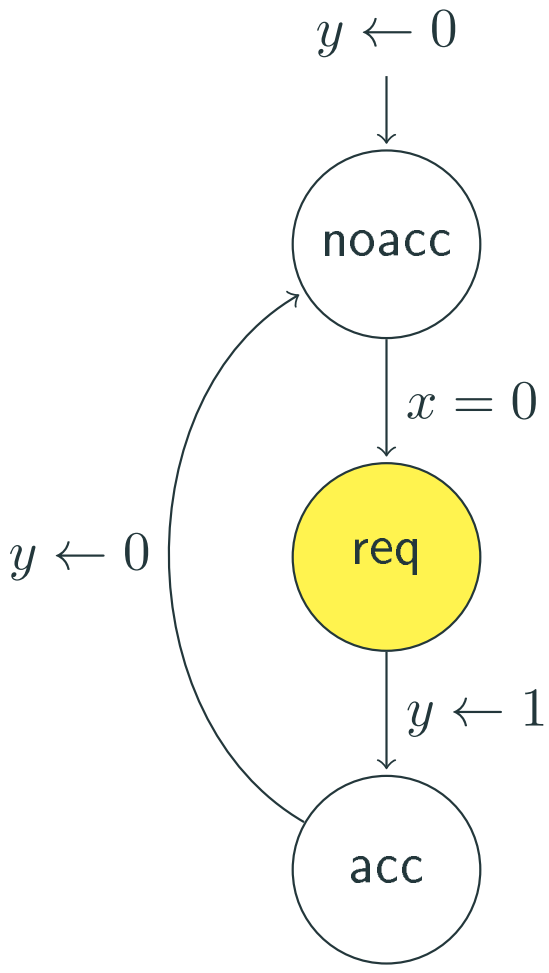




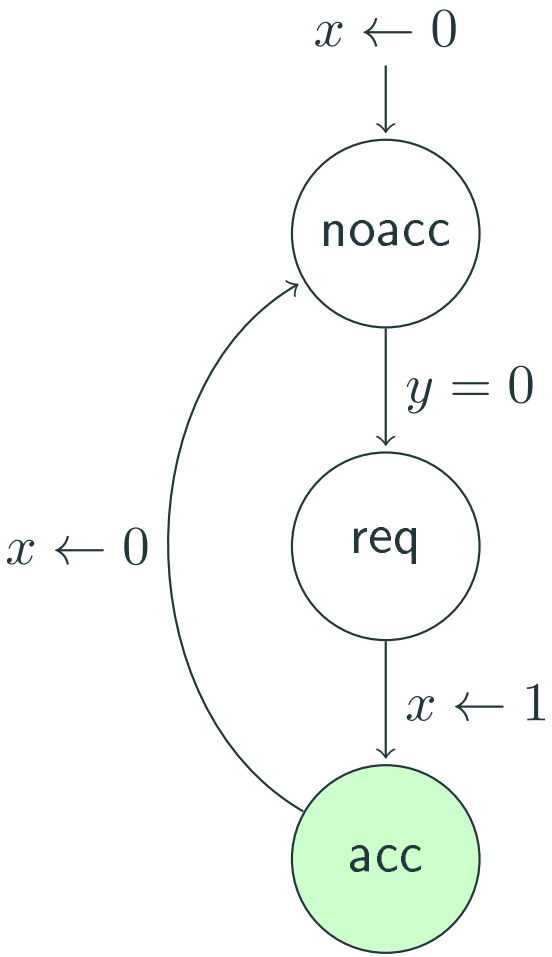
Access to a Shared Resource

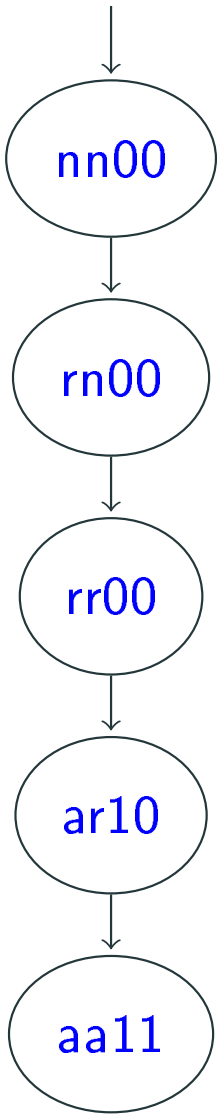


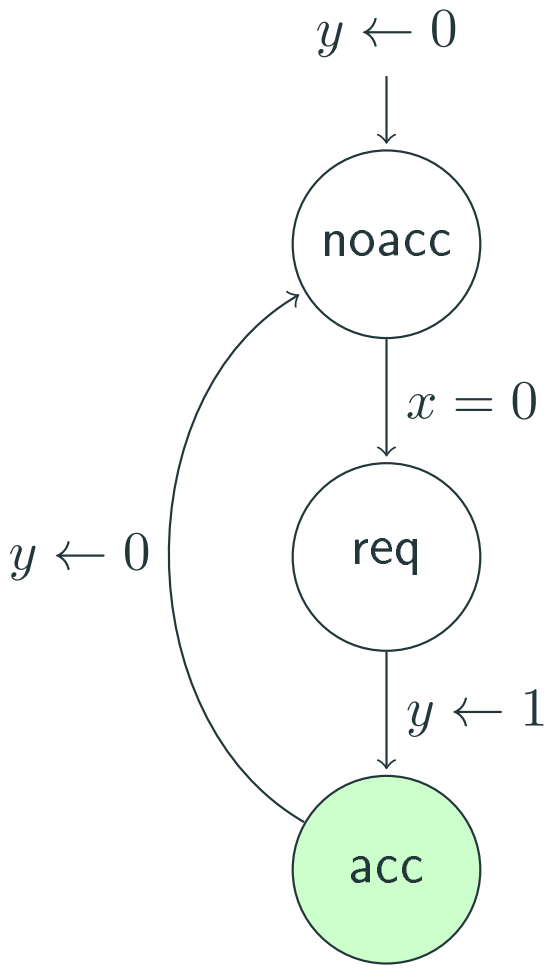




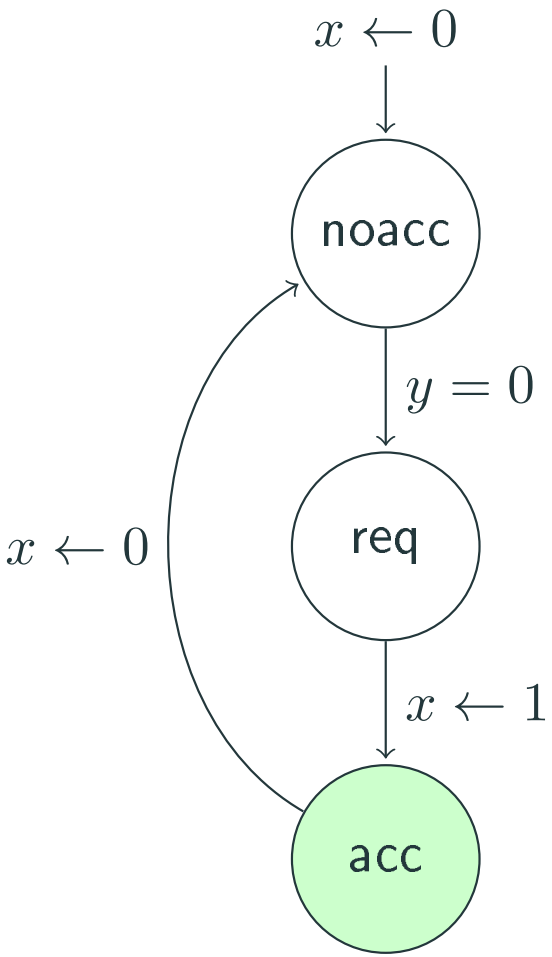
Access to a Shared Resource

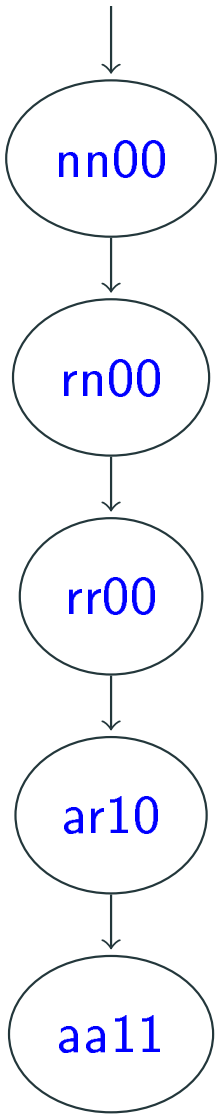


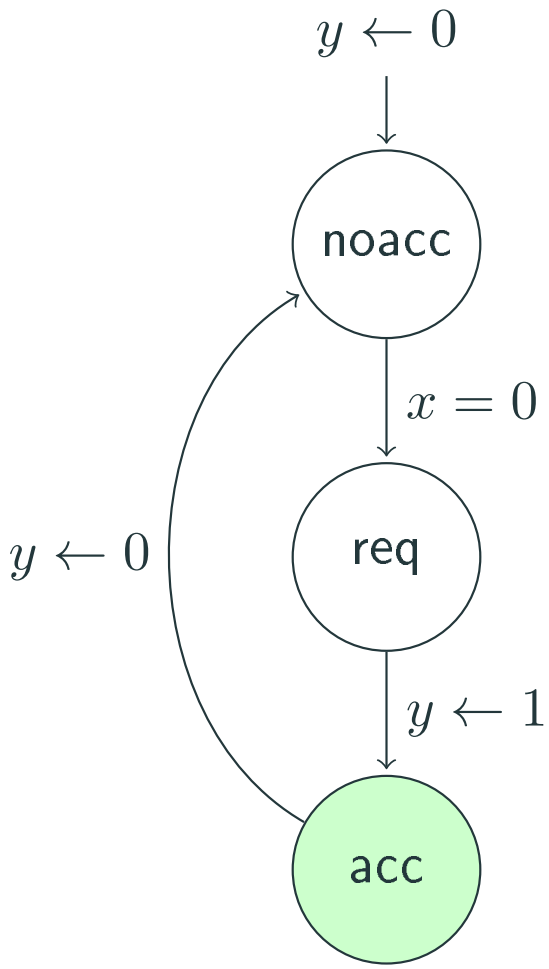




Access to a Shared Resource

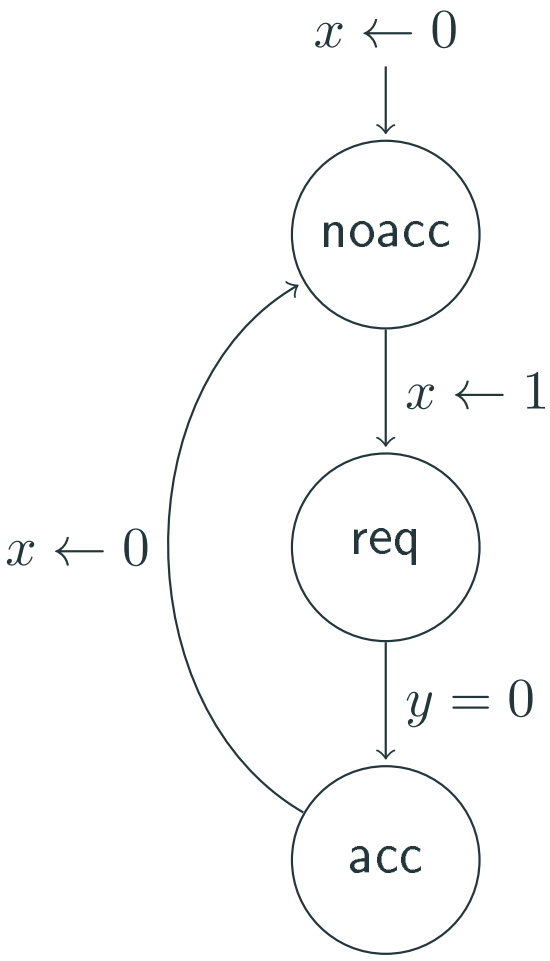




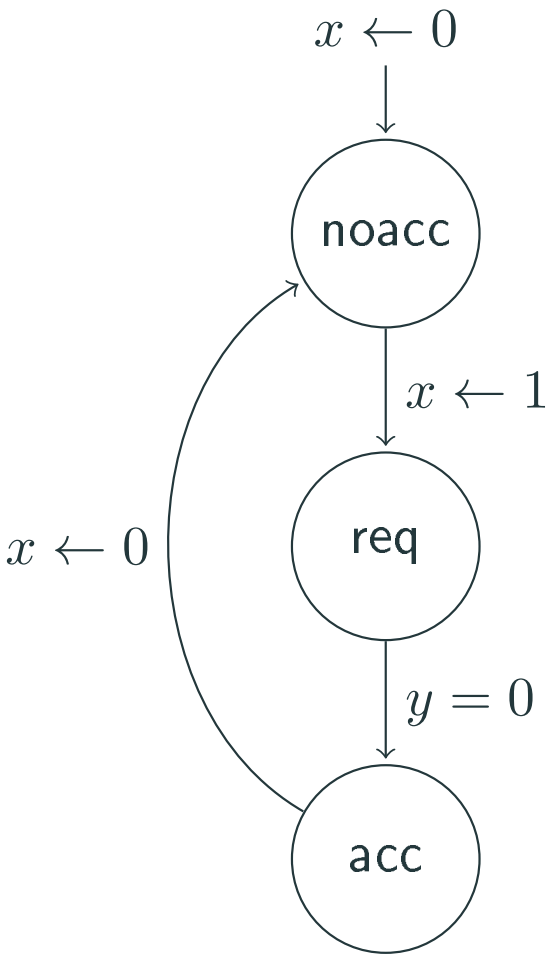


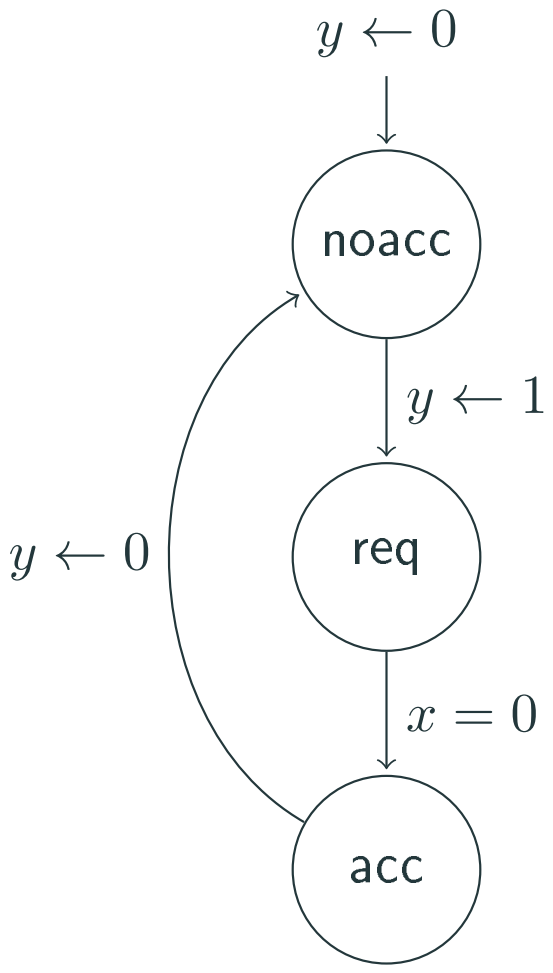
Safety Violation

Second Attempt

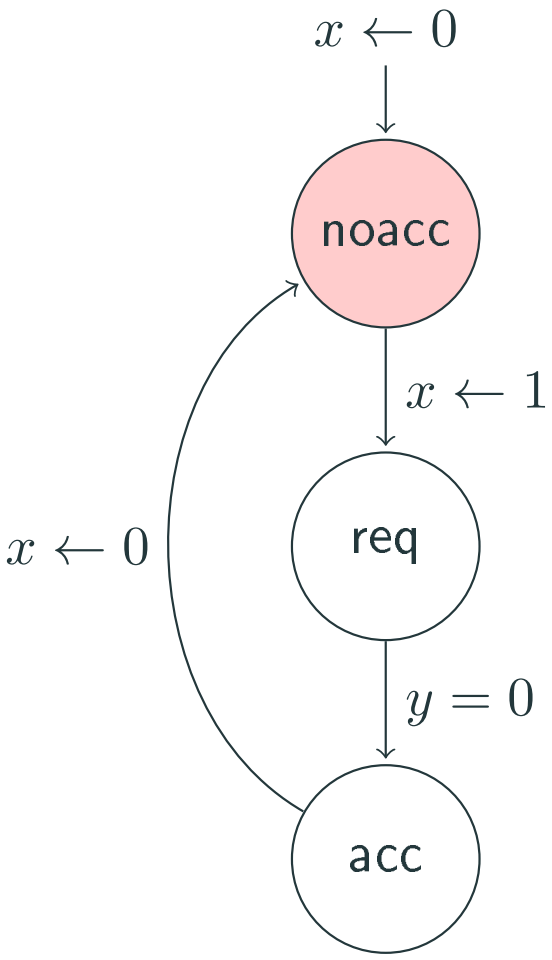


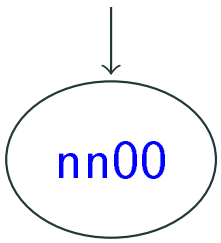
Second Attempt

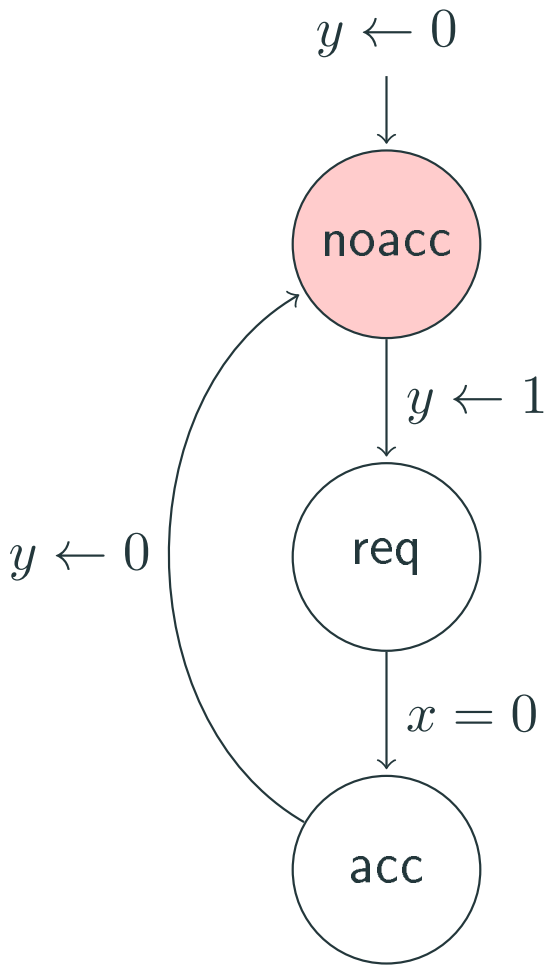




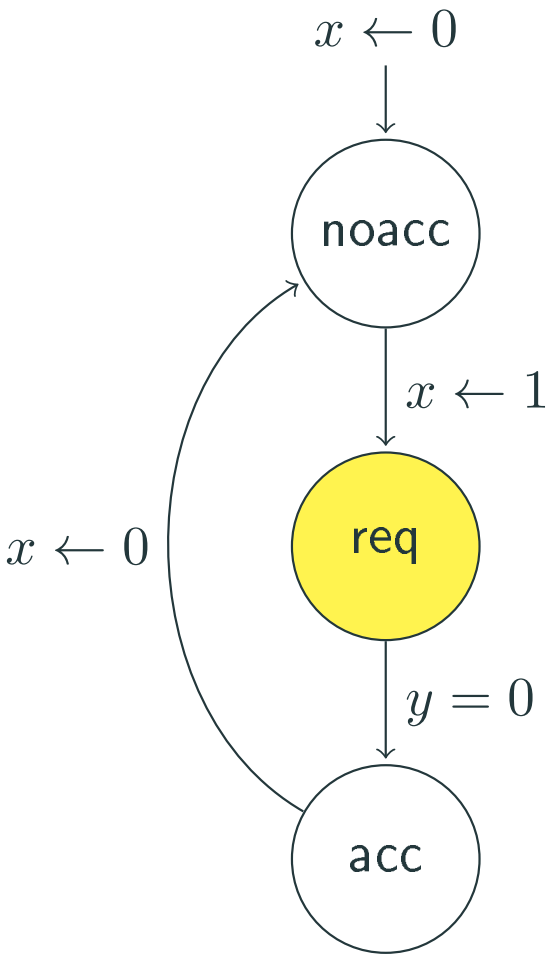
Second Attempt

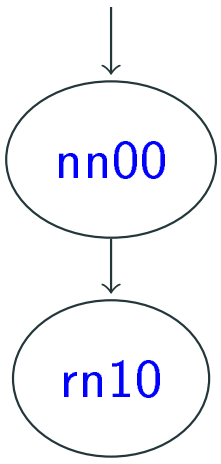


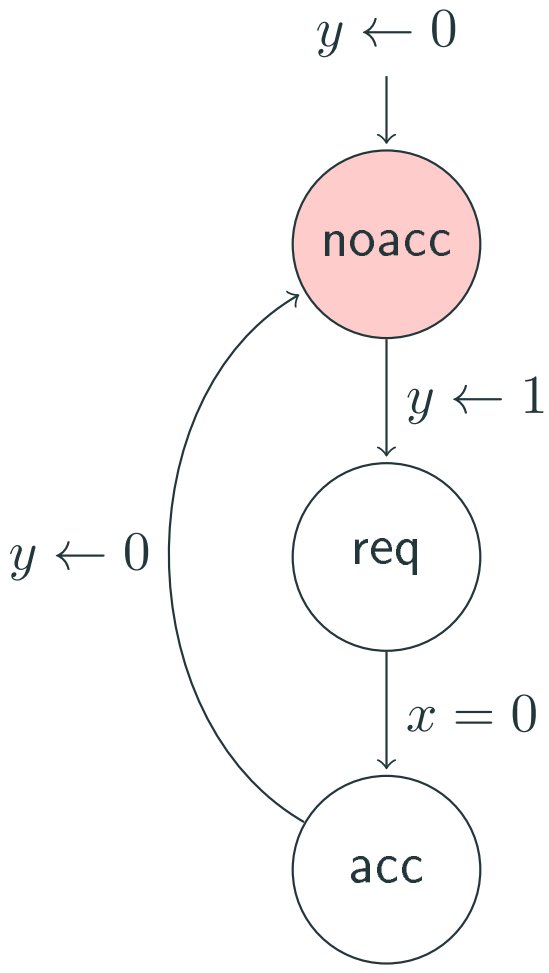




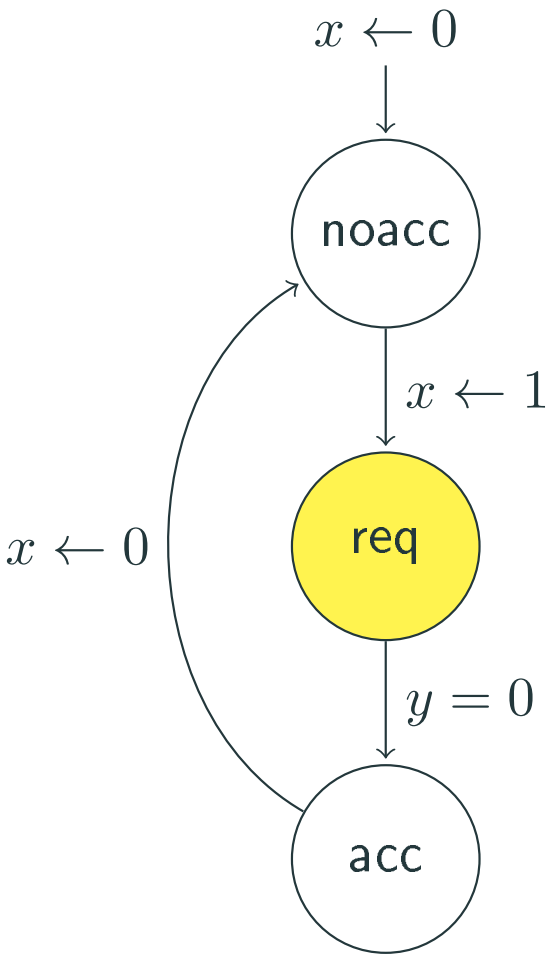
Second Attempt

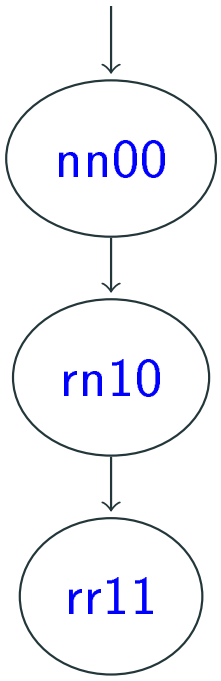


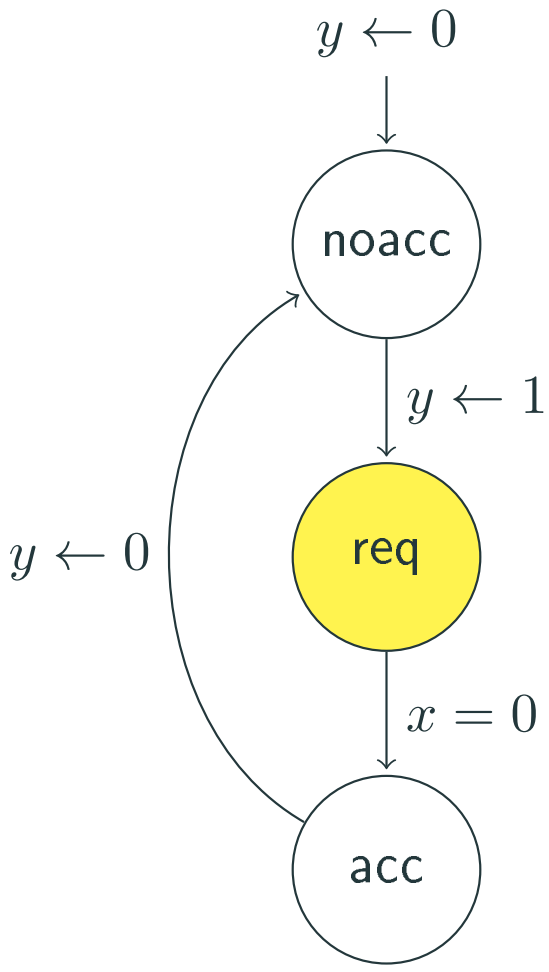




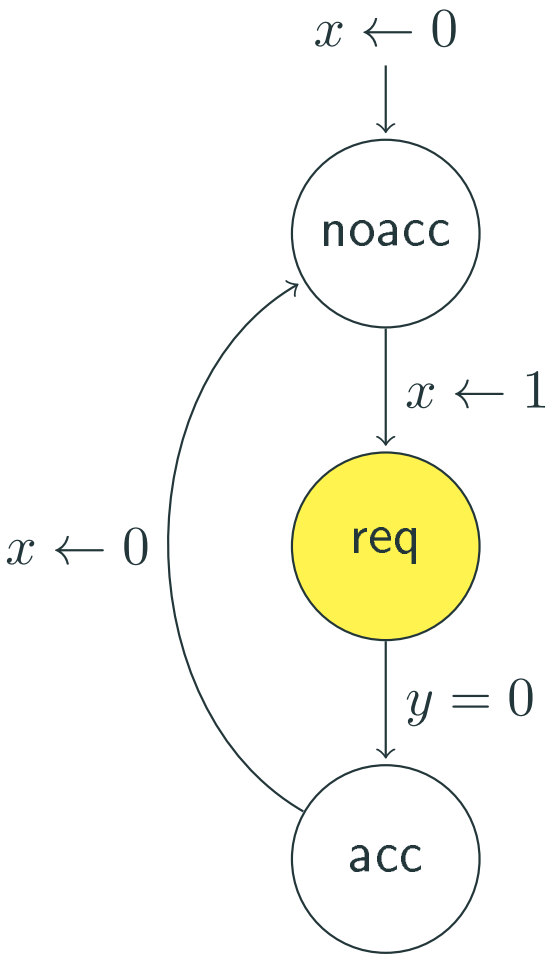
Second Attempt

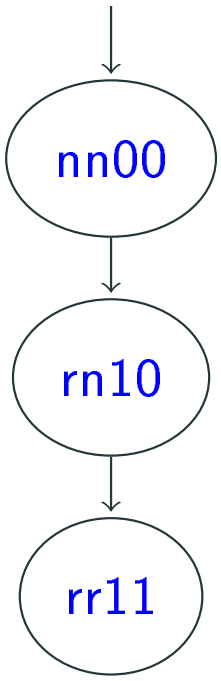




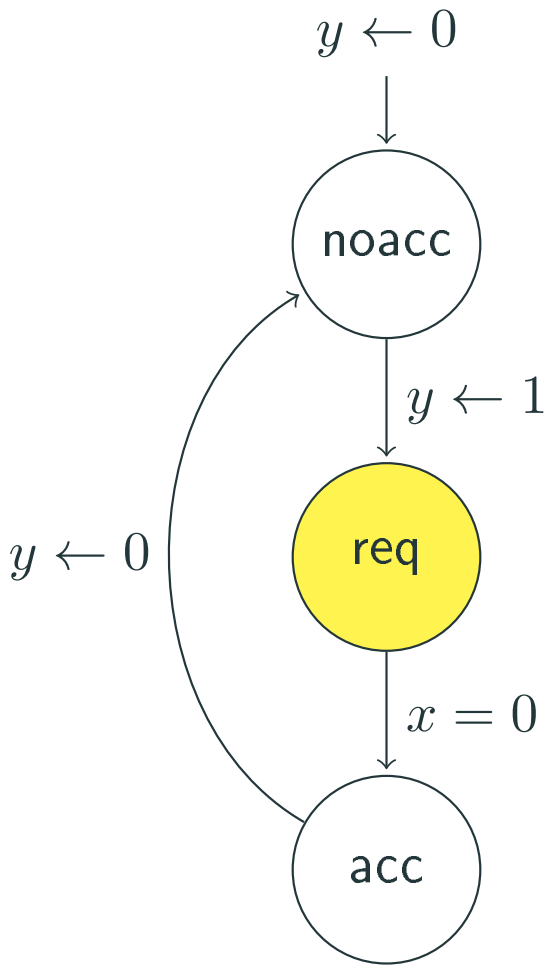


Second Attempt

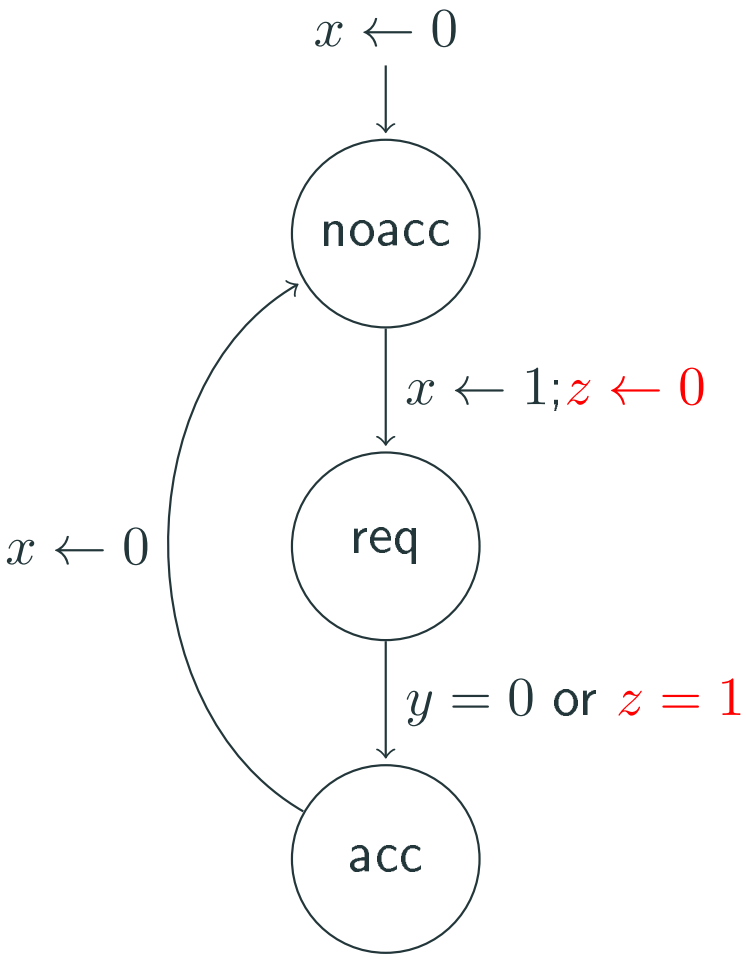


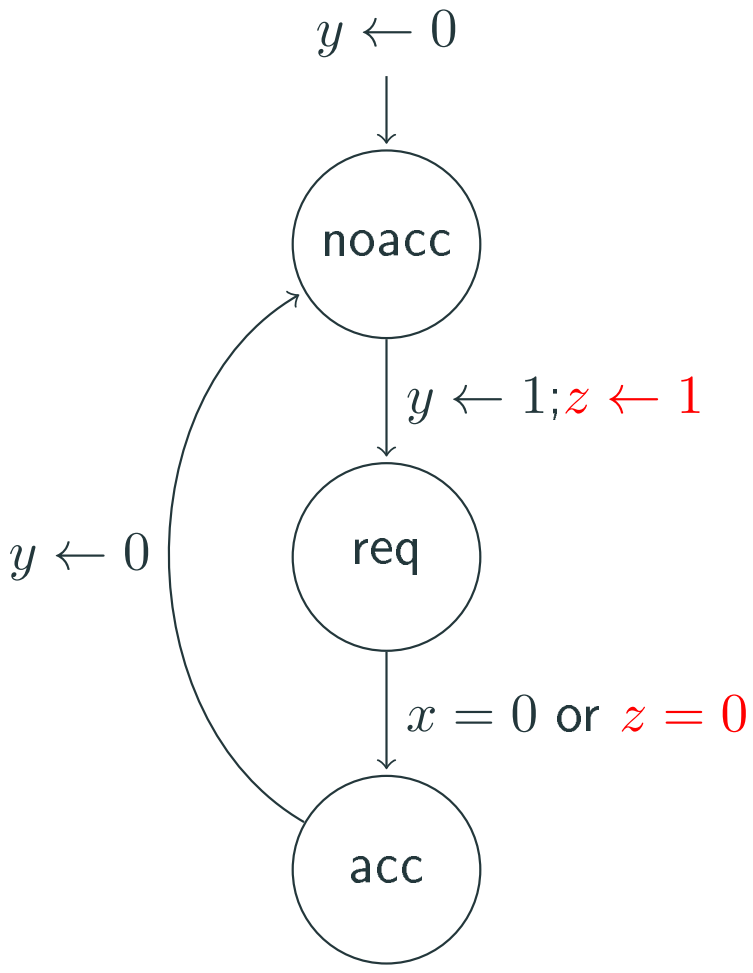


Deadlock

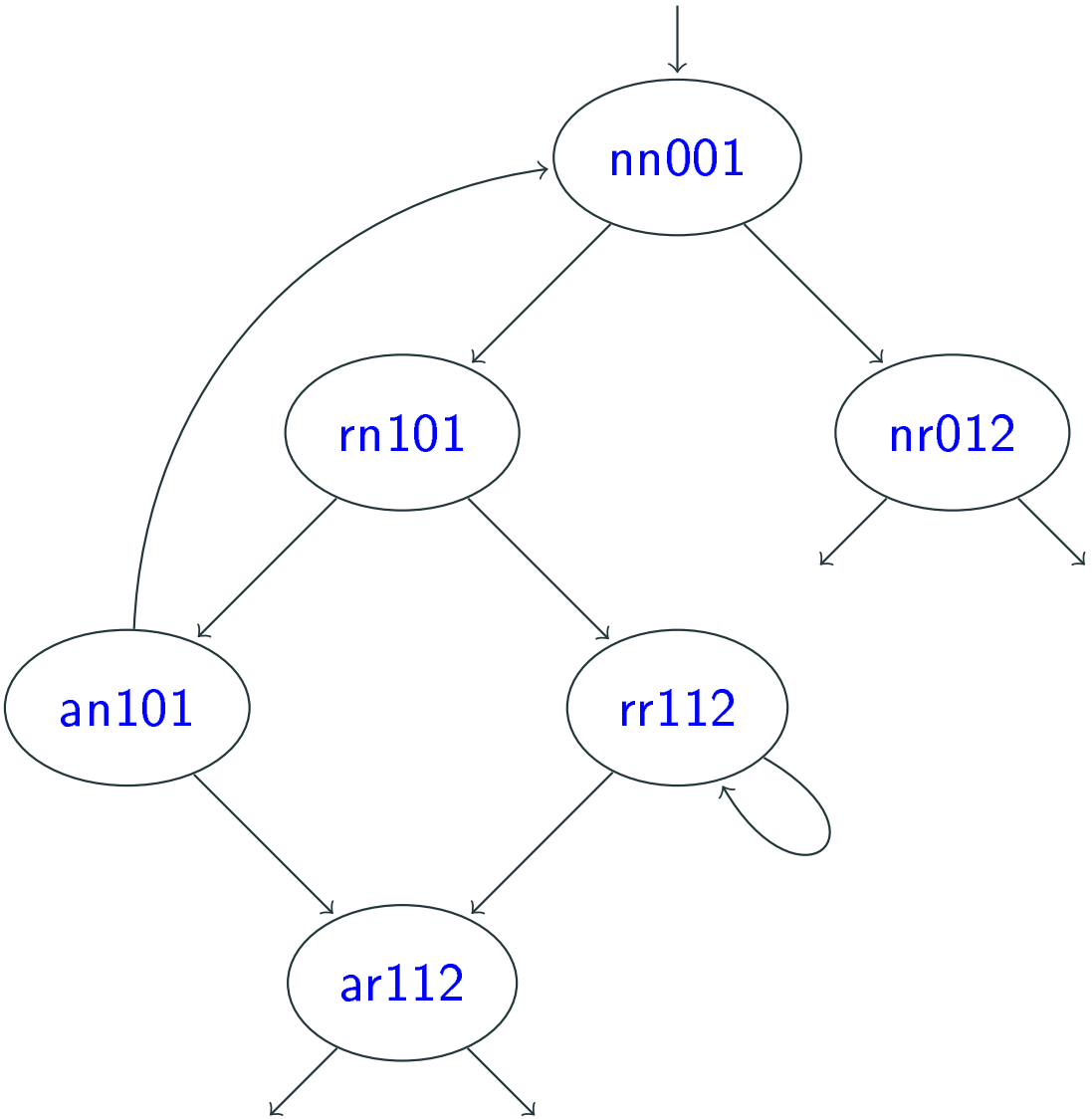


Third Attempt



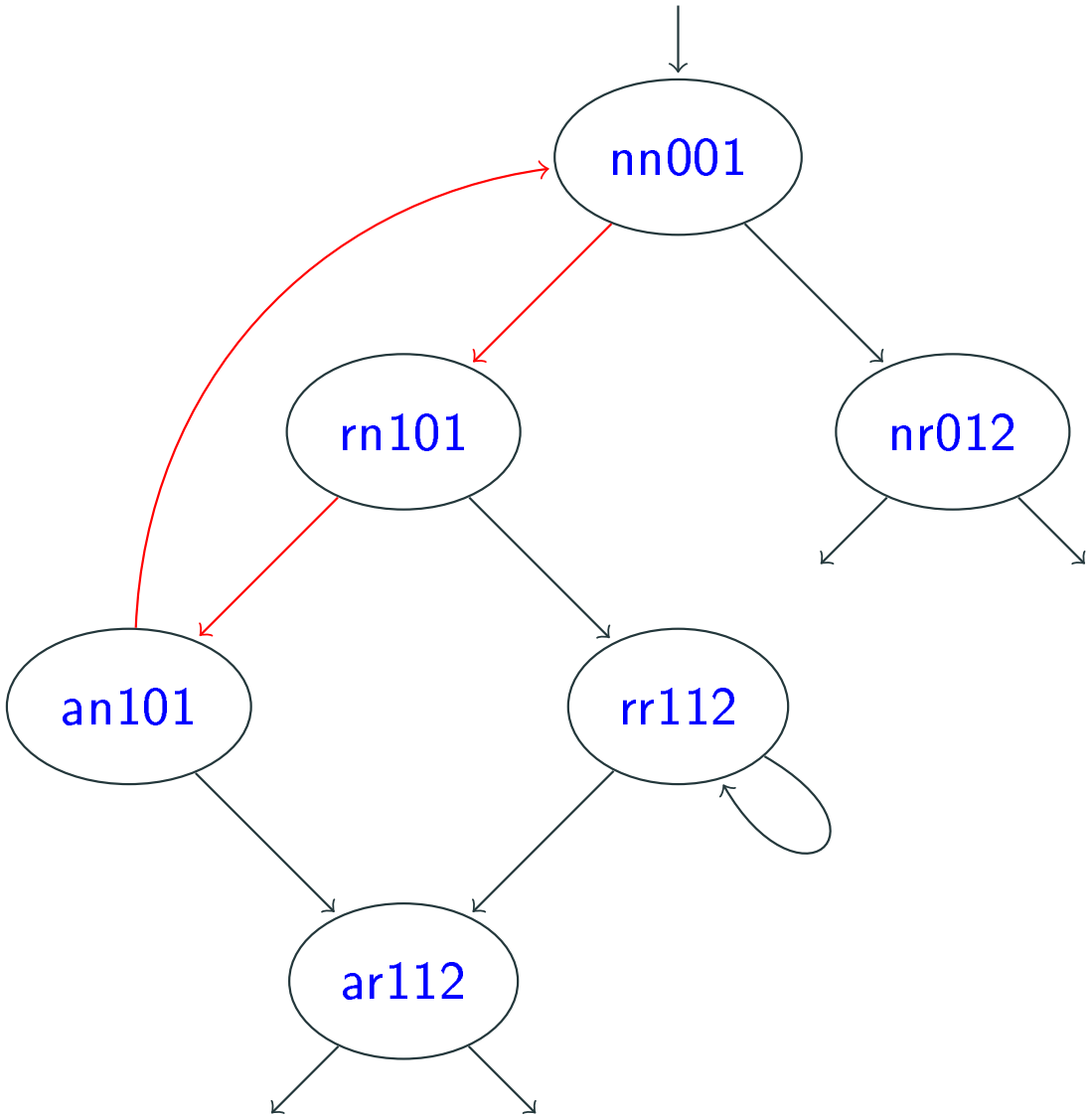


The State Graph



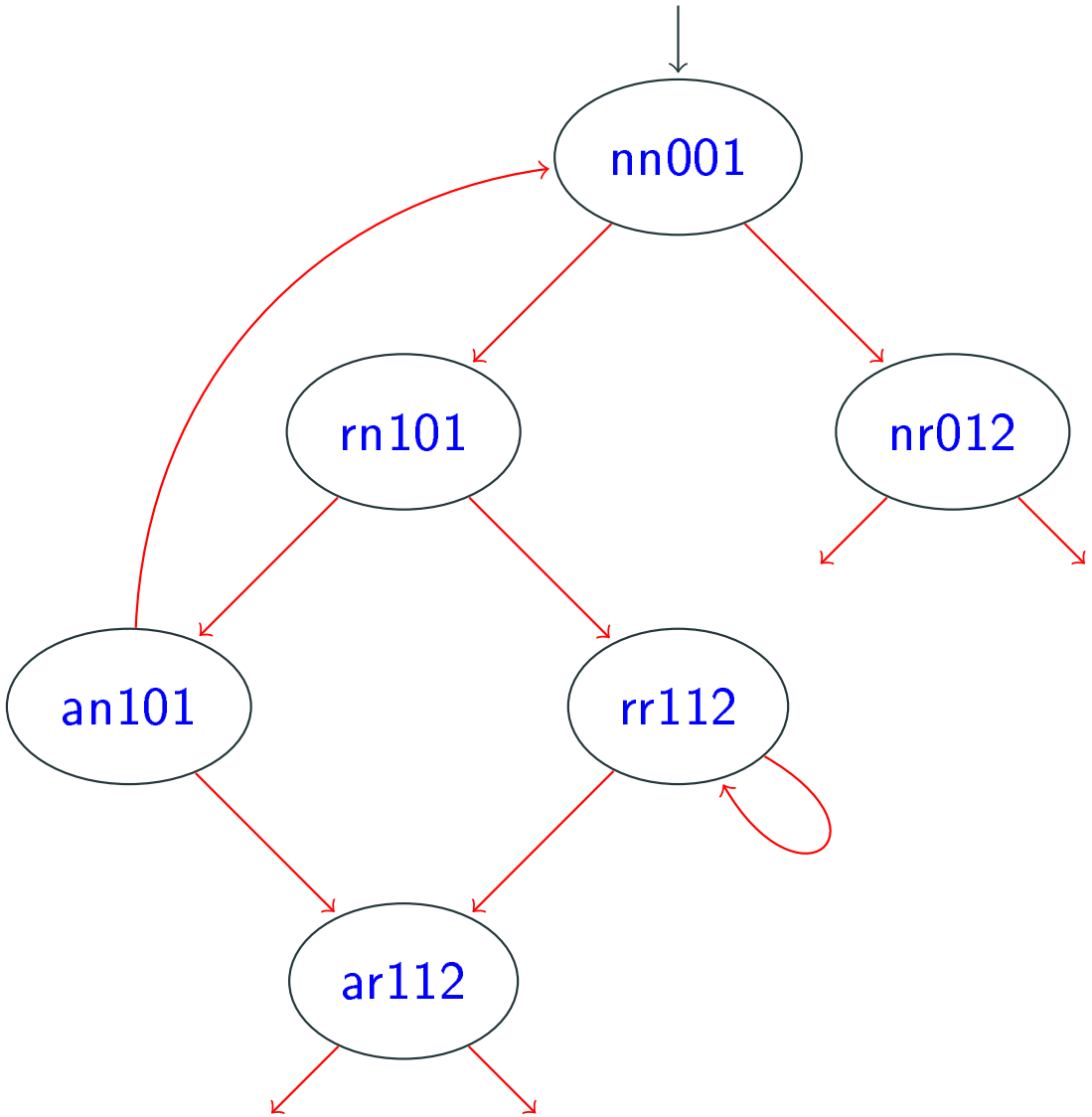
The State Graph

Testing / Simulation: Explore one path at a time



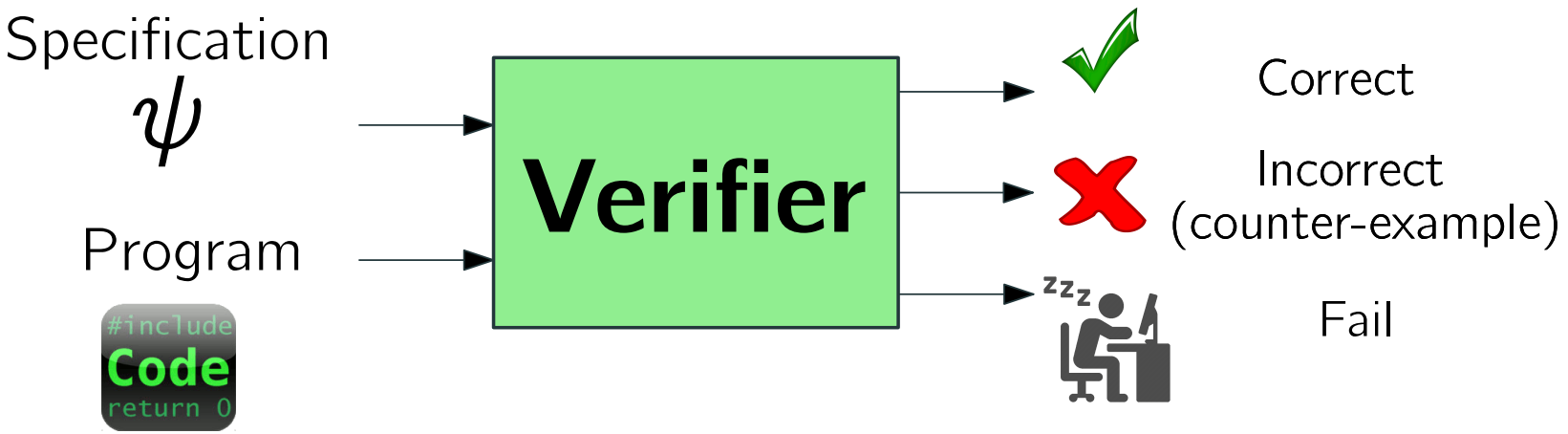
The State Graph

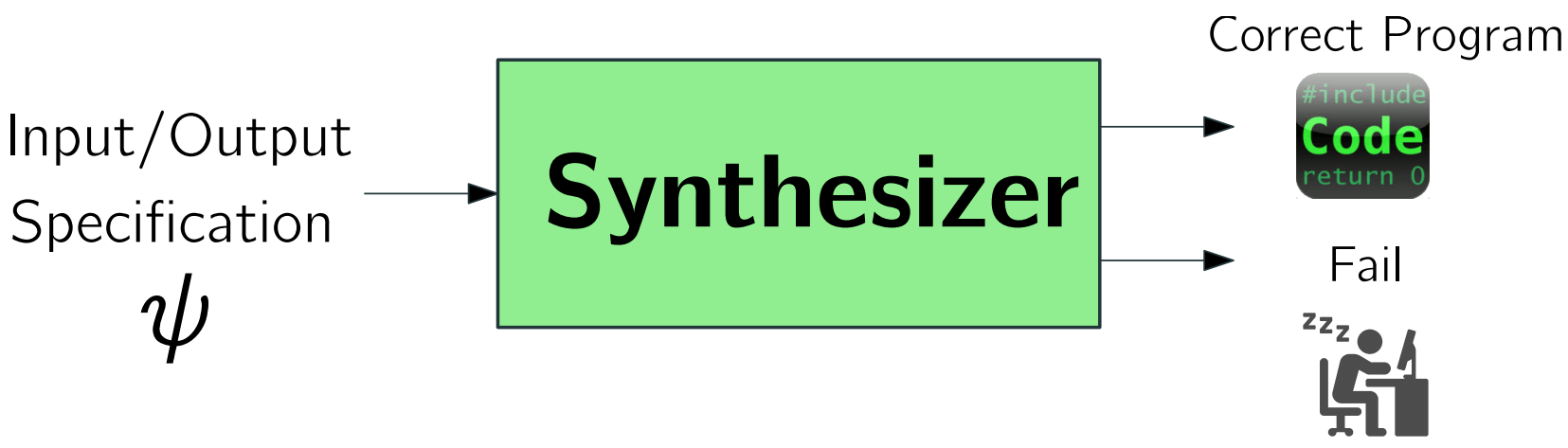
Model Checking: Explore the whole graph ($3 \times 3 \times 2 \times 2 \times 2 = 72$ states)



Formal Methods

"Formal Methods" are mathematically rigorous techniques and tools for specification, synthesis and verification of systems

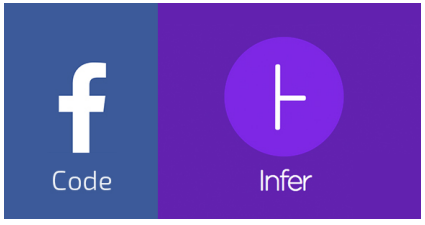




Recent Success Stories in Industry

Formal specification language TLA+ and model checking
Solve difficult design problems in critical systems

(Chris Newcombe et al. 2015)



Infer static analyzer to verify every code modification
in Facebook's mobile apps

(`http://fbinfer.com/`)

Astrée static analyzer to check
flight control program for the A380 series

(`http://www.astree.ens.fr/`)

SLAM static verifier for debugging device drivers
Based on predication abstraction and CEGAR

(`http://research.microsoft.com/en-us/projects/slam/`)

Tools

Z3 SMT solver

`https://github.com/Z3Prover/z3`

The Coq Proof Assistant

`https://coq.inria.fr/`

SPIN Model Checker

`http://spinroot.com/`

Course Work

- **50%** 6 homework assignments (each $\sim$ 8%)
- **50%** final examination

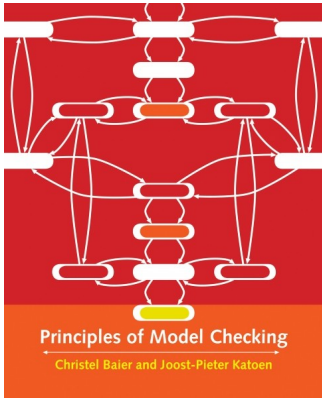
- Assignments must be completed individually
  - Unless the assignment explicitly says that collaboration is possible
- Workload depends on planning well: Start early!

Textbook

- Materials for reading will be posted with lecture notes

Suggested Book

- "Principles of Model Checking"
  Christel Baier and Joost-Pieter Katoen
- Covers some of the course material



Course Staff

- **Instructor:** Hossein Hojjat (`https://www.cs.rit.edu/~hh/`)
  - University of Tehran
    (Bs. Software Engineering 2001 - 2005)
  - University of Tehran & TU Eindhoven
    (Msc. Software Engineering 2005 - 2007)
  - EPFL Lausanne, Switzerland
    (PhD Computer Science 2008 - 2013)
  - Cornell University
    (Postdoctoral Researcher 2014 - 2016)
  - Rochester Institute of Technology
    (Tenure Track Assistant Professor 2016 - 2018)
- **Email:** hojjat@cornell.edu
- **Office:** 615

Icebreaker

Tell us about your background,
how do you (usually) ensure that your programs are correct,
story of a nasty bug that took you a while to debug! (if any)

Why functional programming?

Parallelism

- Moore's law:
  Transistors of CPU doubles approximately every two years
- No longer true: Number of cores has been increasing recently

GPU programs can spawn millions of threads during execution

- Software has to take advantage of all the additional processors
- Programmers use sequential algorithms

Concurrent Programming

Models

- Shared Memory with locking

(mutex, semaphore,...)

- Message Passing

(Actor model)

- Software transactional memory

Concurrent Programming

Models

- Shared Memory with locking (mutex, semaphore,...)
- Message Passing (Actor model)
- Software transactional memory

```
class Person(val name:String,
             val age:  Int)
class actor extends Actor {
def receive = {
 case people:  Set[Person] =>
  val (minors, adults) =
   people partition (_.age < 18)
  Facebook !  minors
  LinkedIn !  adults
 }
}
```

Concurrent Programming

Models

- Shared Memory with locking (mutex, semaphore,...)
- Message Passing (Actor model)
- Software transactional memory

```
class Person(val name:String,
             val age:  Int)
class actor extends Actor {
def receive = {
 case people:  Set[Person] =>
  val (minors, adults) =
   people partition (_.age < 18)
  Facebook !  minors
  LinkedIn !  adults
 }
}
```

- None of the concurrent models is the ultimate solution
- Fundamental problem: Non-determinism
- Heisenbug: Bug that seems to disappear when attempting to study it

Non-determinism

- Non-determinism: concurrent threads are accessing shared mutable state
- We can encapsulate state in actors or transactions, but the fundamental problem is the same

```
var x = 0;
thread {
x = 1;
x = x + 1;
}
thread {
x = x * 2;
}
```

value of x finally: $2, 3, 4$

(assignments are atomic)

non-determinism = parallel processing + mutable state

Functional Programming

- To get deterministic processing, avoid the mutable state
- Avoid mutable state means programming functionally
- Rebirth of interest in functional programming triggered by multi-core hardware

Functional Programming

- To get deterministic processing, avoid the mutable state
- Avoid mutable state means programming functionally
- Rebirth of interest in functional programming triggered by multi-core hardware

- No mutable state: variables are immutable
- No assignment statement
- Functions are first-class values
- Functional program: collection of mathematical functions