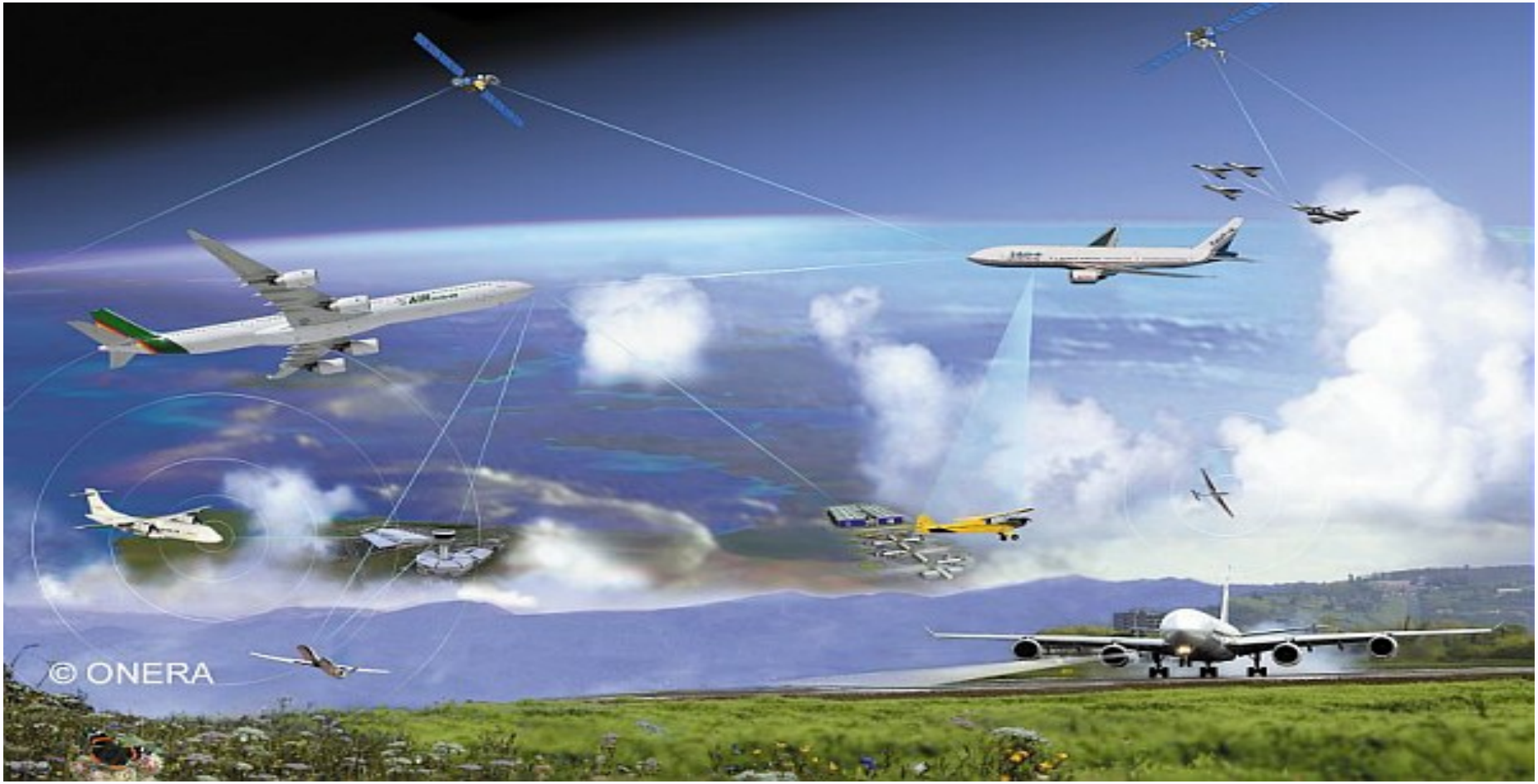


Formal verification of Cyber Physical Systems for everyone

Georgia Tech, GA

Pierre-Loic Garoche – Onera

16th of November 2014

Contents

- **Motivation**
- **Formal methods**
  SMT-based model-checking
  Abstract Interpretation - reasoning on set of states as abstract states
  Deductive method
- **Dealing with Floating points computation**
- **Example of use**
- **Current trends – Challenges**

Contents

Cyber Physical Systems

Critical Embedded systems are everywhere: from the small embedded devices such as insulin pump or small UAV to huge jet airliners or latest electric or hybrid cars such as the GM Volt.

Ambition

sustain the development of CPS by achieving comparable safety/security with simpler/cheaper development processes

Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft

Differential Equations (plant)

Control theorists

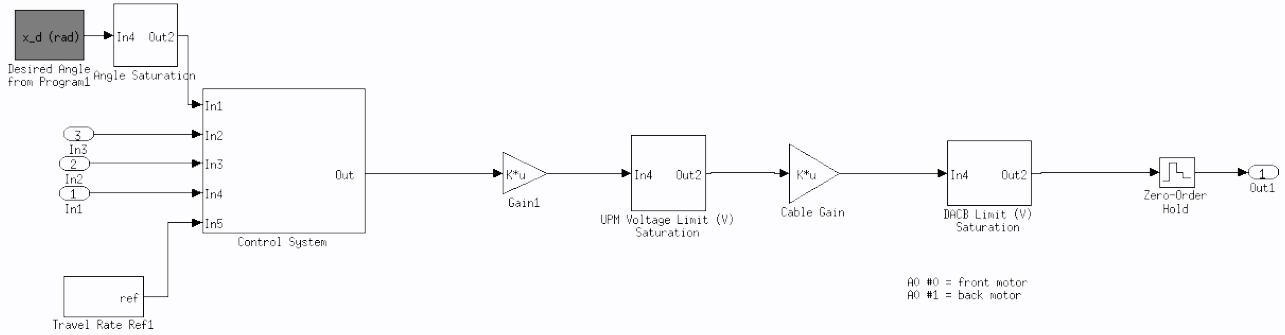
Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft

Differential Equations (plant)

↳ Continuous controller

Control theorists



Today's Target: control systems for a UAV

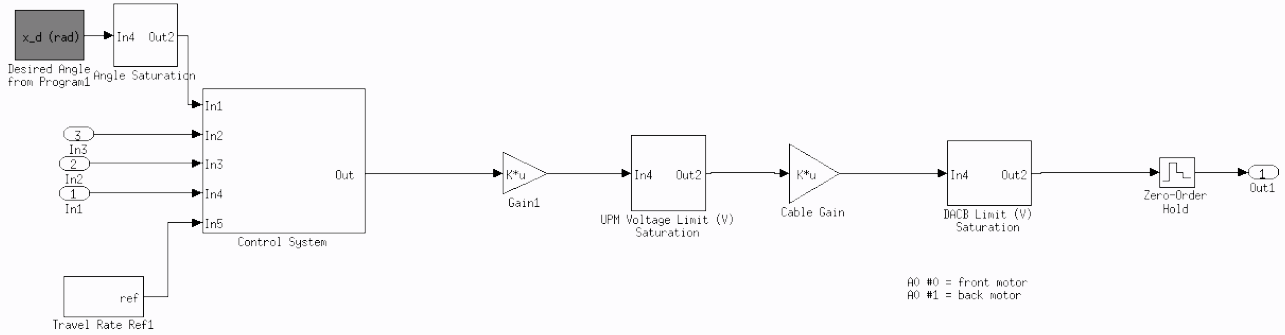
Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



Today's Target: control systems for a UAV

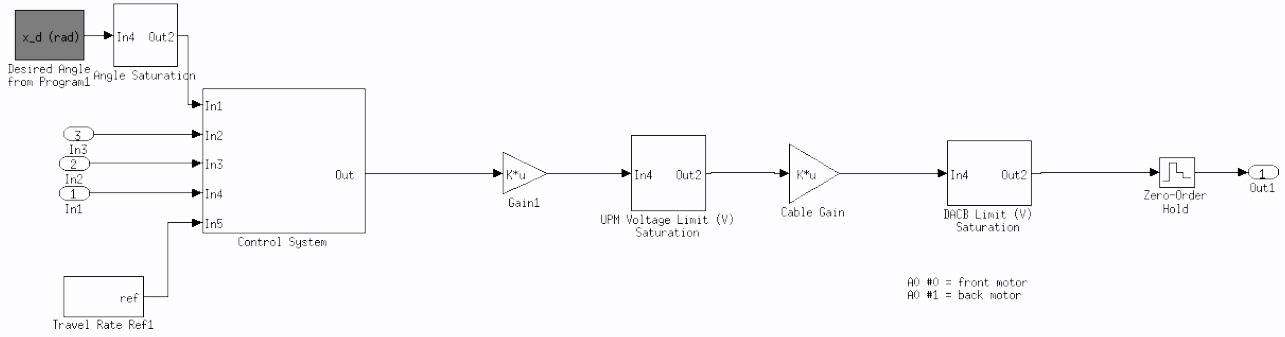
Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design: typically w. synchronous models
  - usually simplification of the plant around specific points and controlers proposed for these

Today's Target: control systems for a UAV

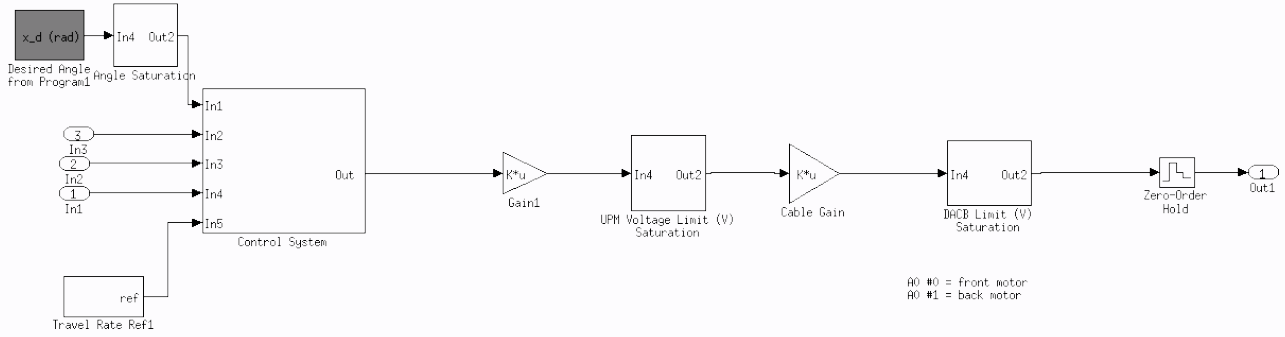
Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design: typically w. synchronous models
  - usually simplification of the plant around specific points and controlers proposed for these
  - lots of arguments/evidences on those simple cases

Today's Target: control systems for a UAV

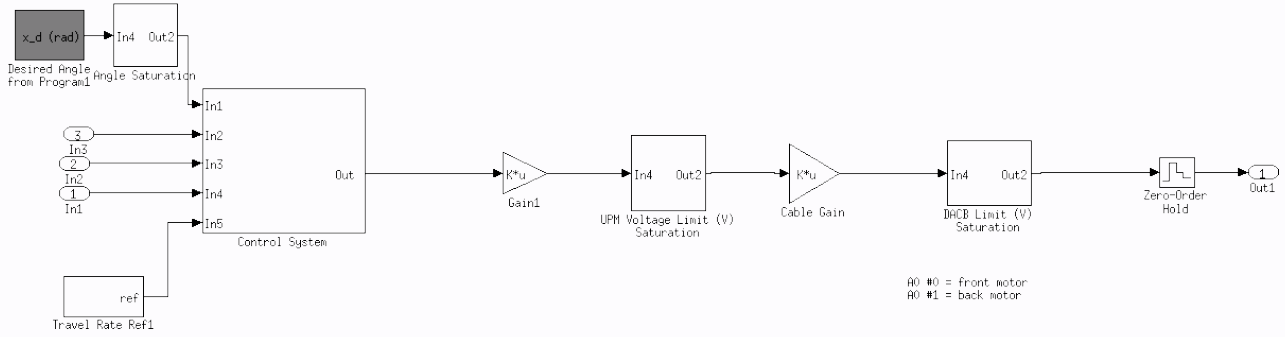
Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design: typically w. synchronous models
  - usually simplification of the plant around specific points and controlers proposed for these
  - lots of arguments/evidences on those simple cases
  - which property? stability, robustness, performances (need the plant!)

Today's Target: control systems for a UAV

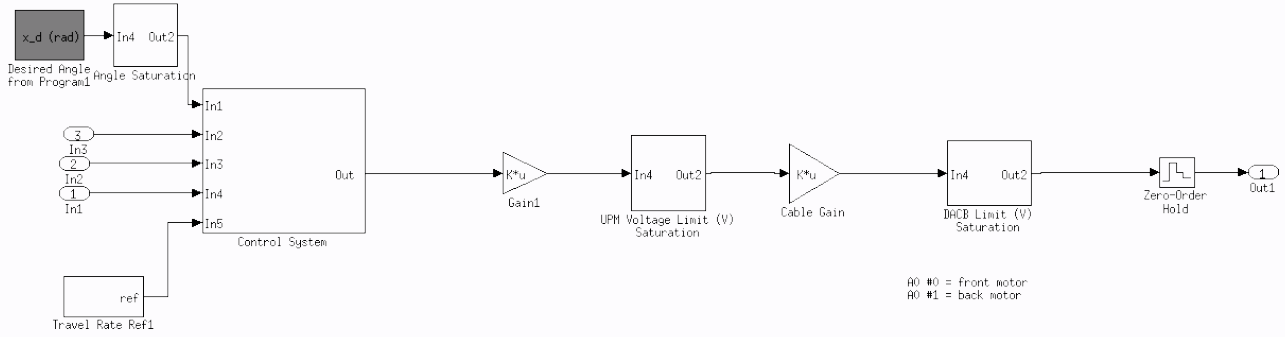
Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists



- Control laws design: typically w. synchronous models
  - usually simplification of the plant around specific points and controlers proposed for these
  - lots of arguments/evidences on those simple cases
  - which property? stability, robustness, performances (need the plant!)
  - frequency domain proof argument vs state space domain (ie. Lyapunov functions)

Today's Target: control systems for a UAV

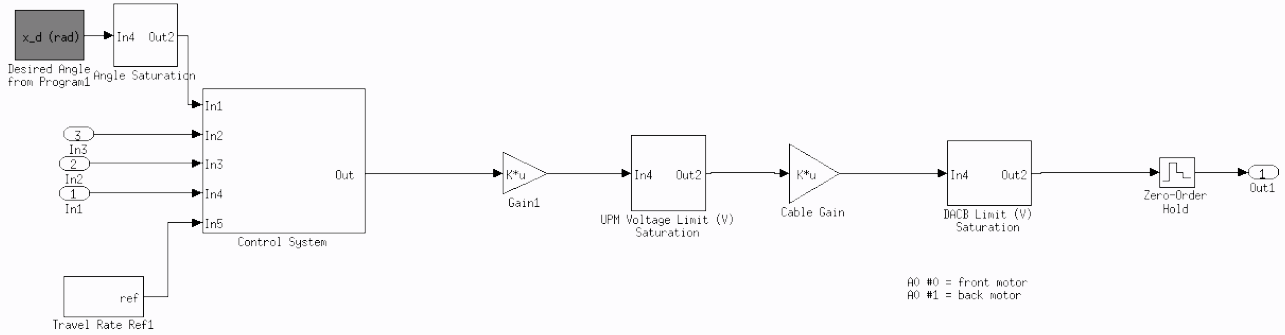
Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

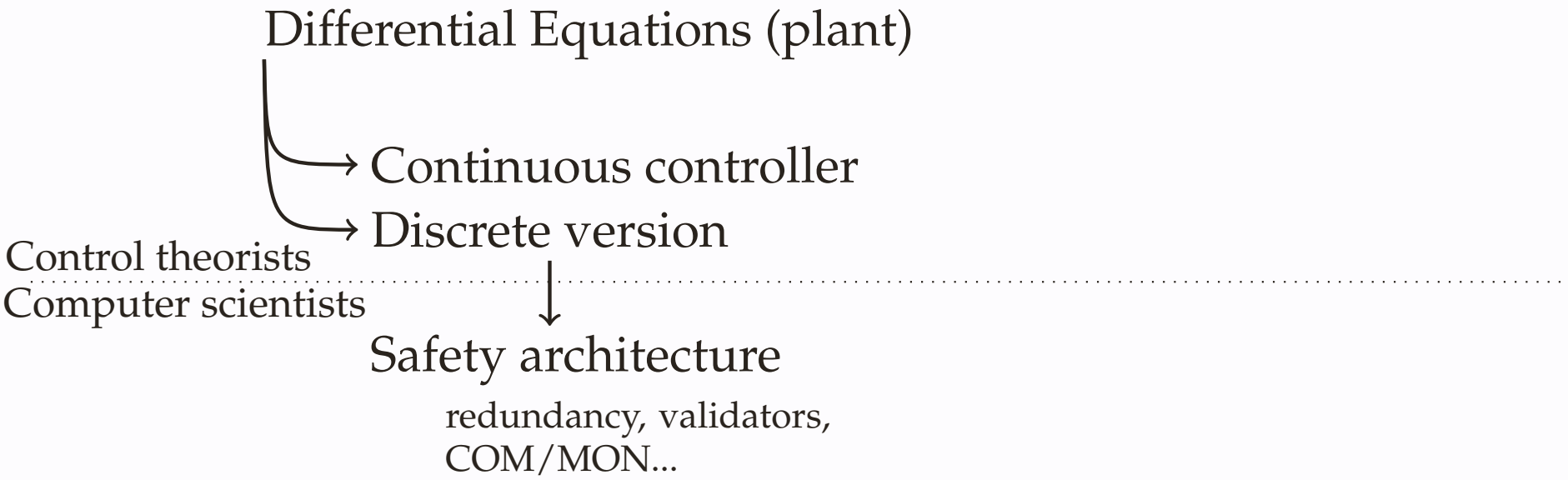
→ Discrete version

Control theorists



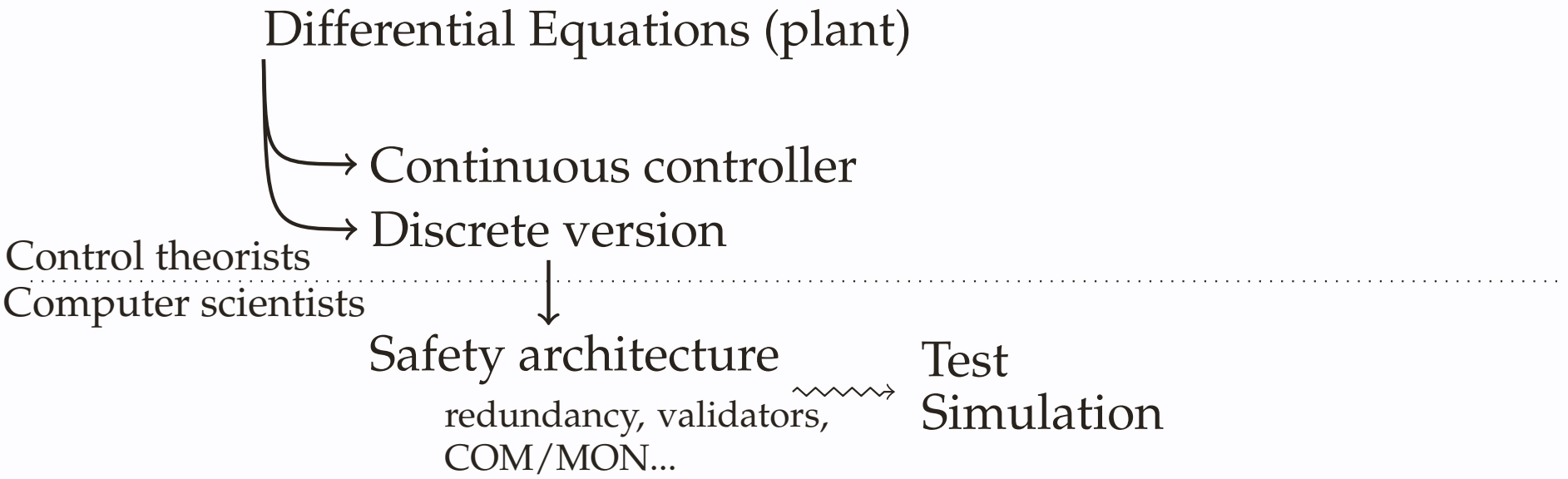
Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft



TODAY'S TARGET: CONTROL SYSTEMS FOR A UAV

TYPICAL DEVELOPMENT CYCLE OF A CONTROLLER FOR AN AIRCRAFT



- Fault tolerance: set of constructs to recover from system/hardware failures
  - is this architecture sound (ie. when there is less than n simultaneaous error, the output is still valid or there will still be a working controler)
  - protection against software error (bug, Run Time Error)
  - protection against hardware rror (SEU, crashed computer, deadlock, deadline misses)

Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists

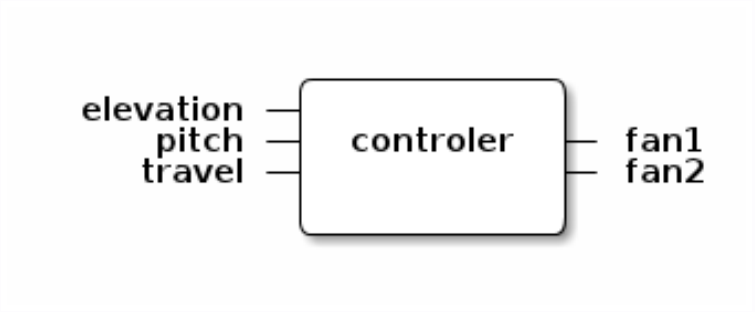
Computer scientists

↓

Safety architecture

redundancy, validators, COM/MON...

⇝ Test Simulation



Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists

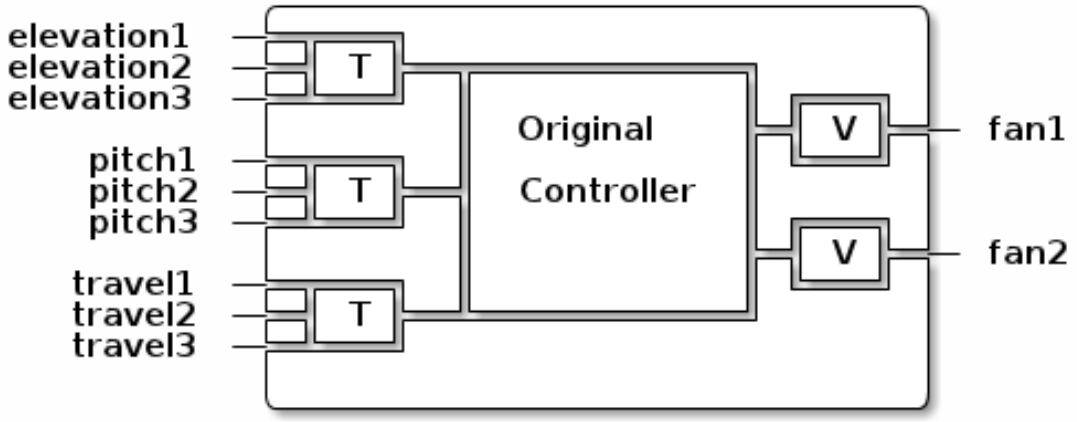
Computer scientists

↓

Safety architecture

redundancy, validators, COM/MON...

⇝ Test Simulation



Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft

Differential Equations (plant)

→ Continuous controller

→ Discrete version

Control theorists

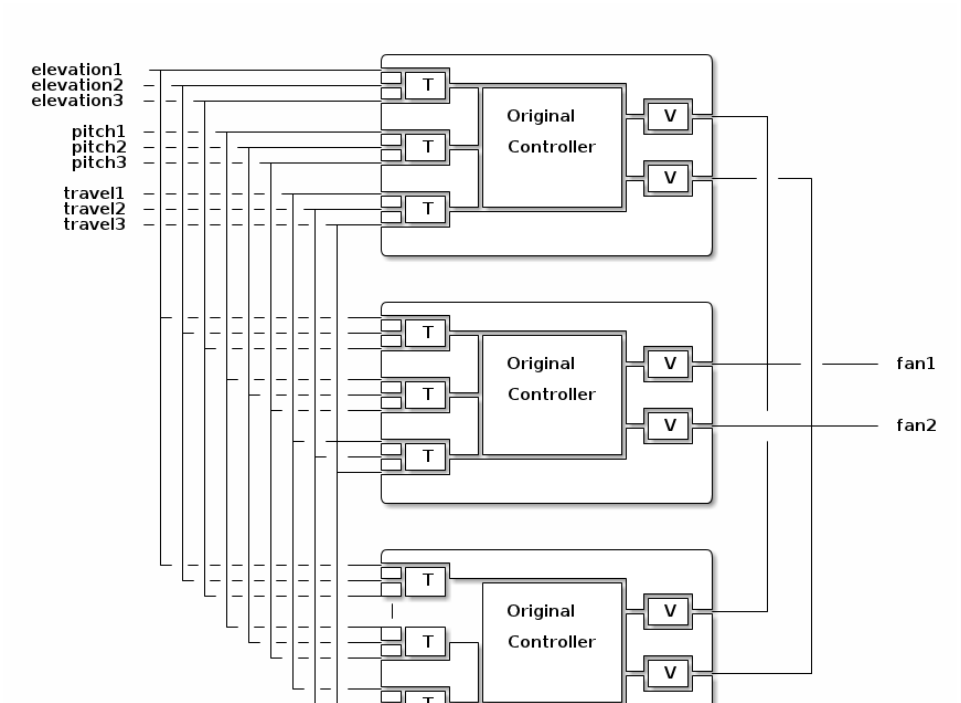
Computer scientists

↓

Safety architecture

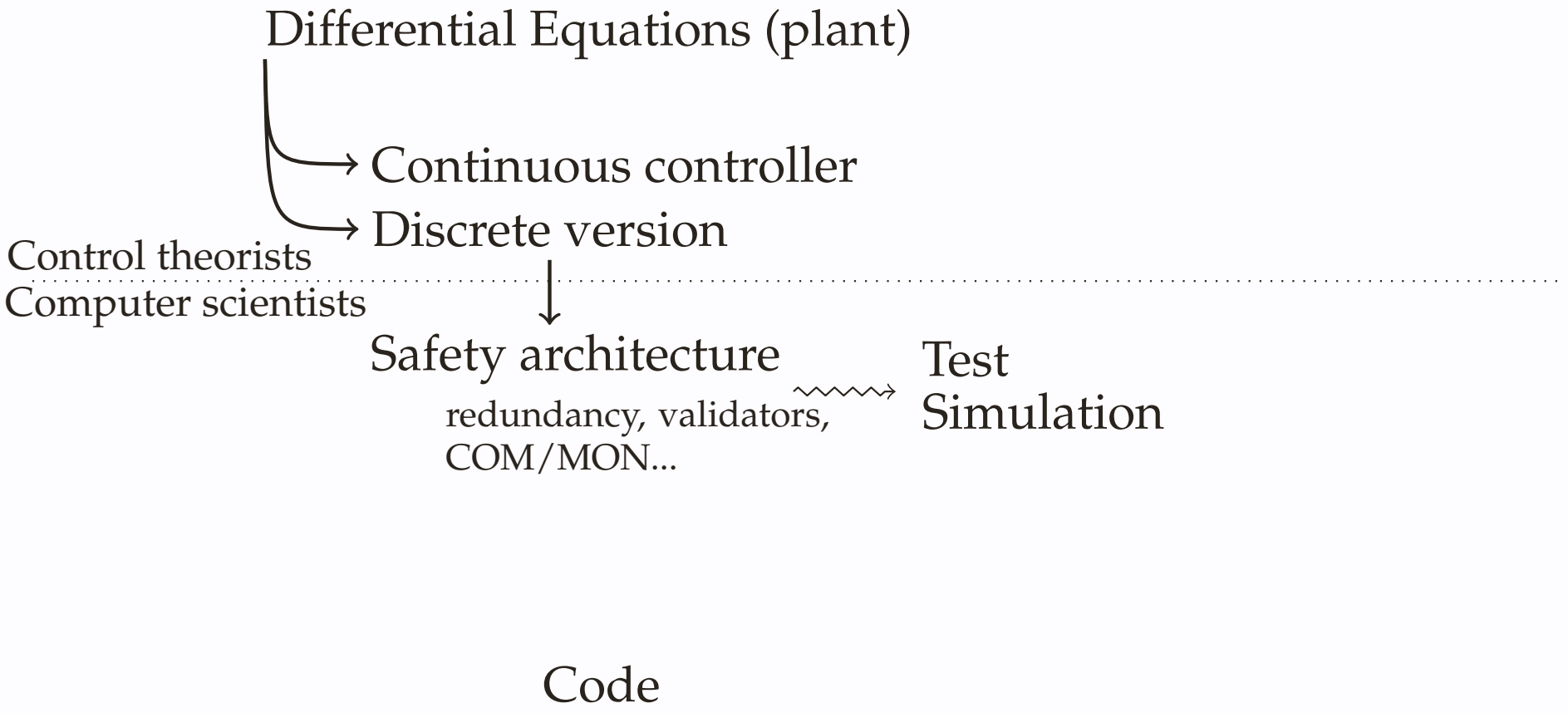
redundancy, validators, COM/MON...

⇝ Test Simulation



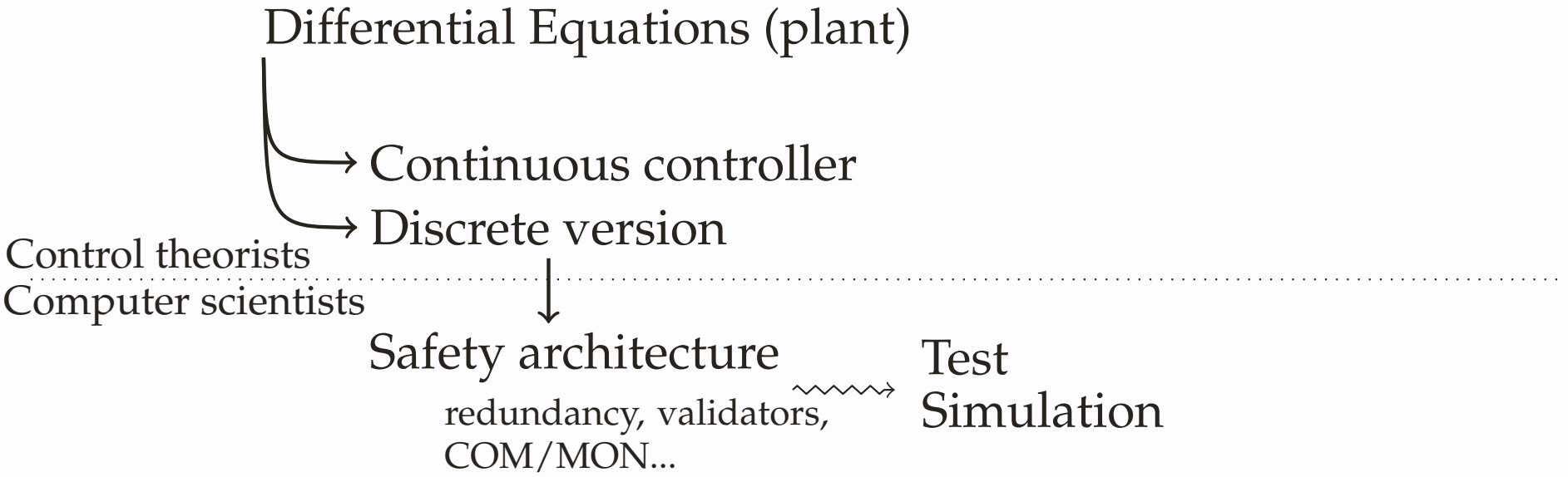
Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft



Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft

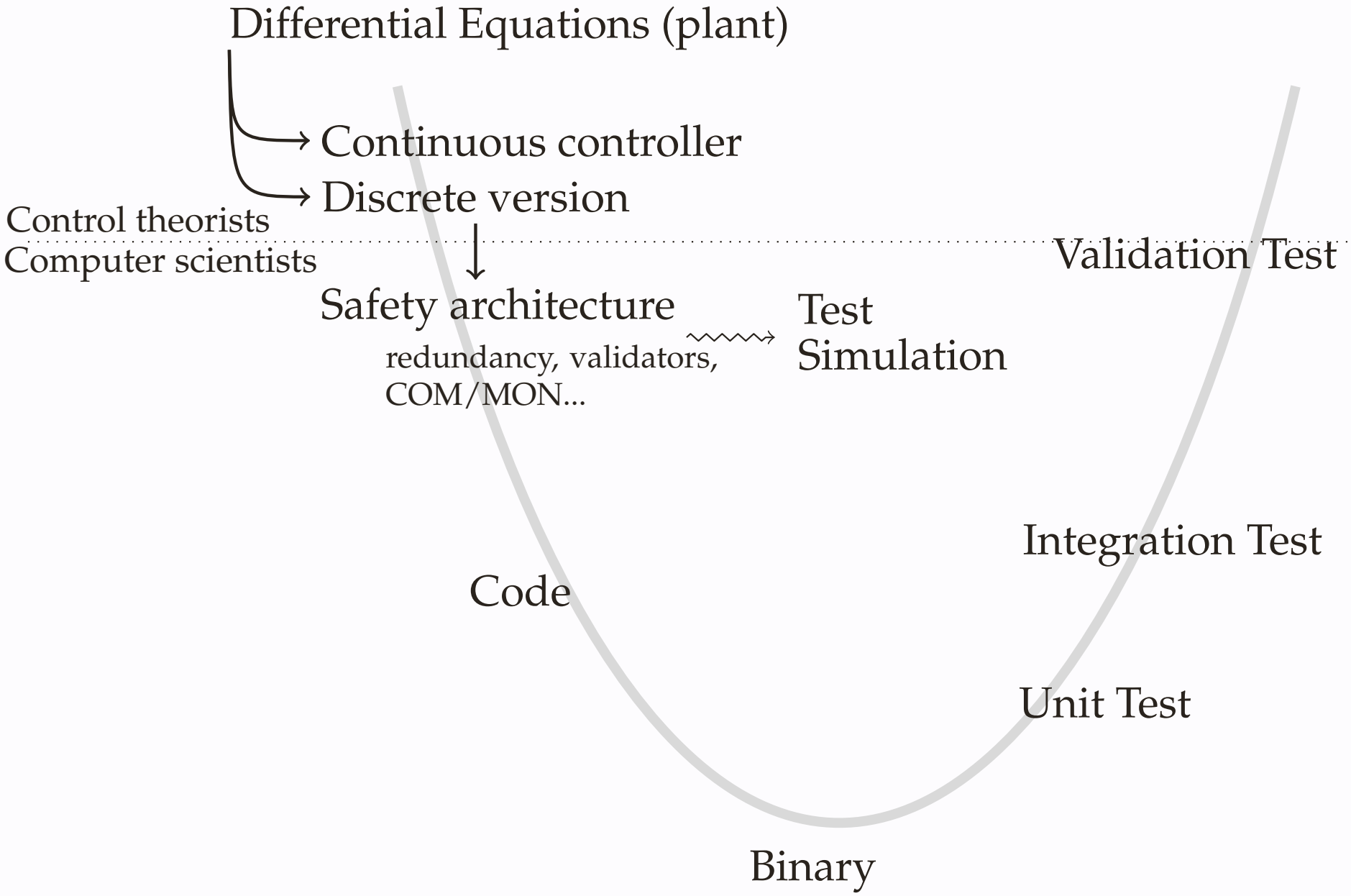


Code

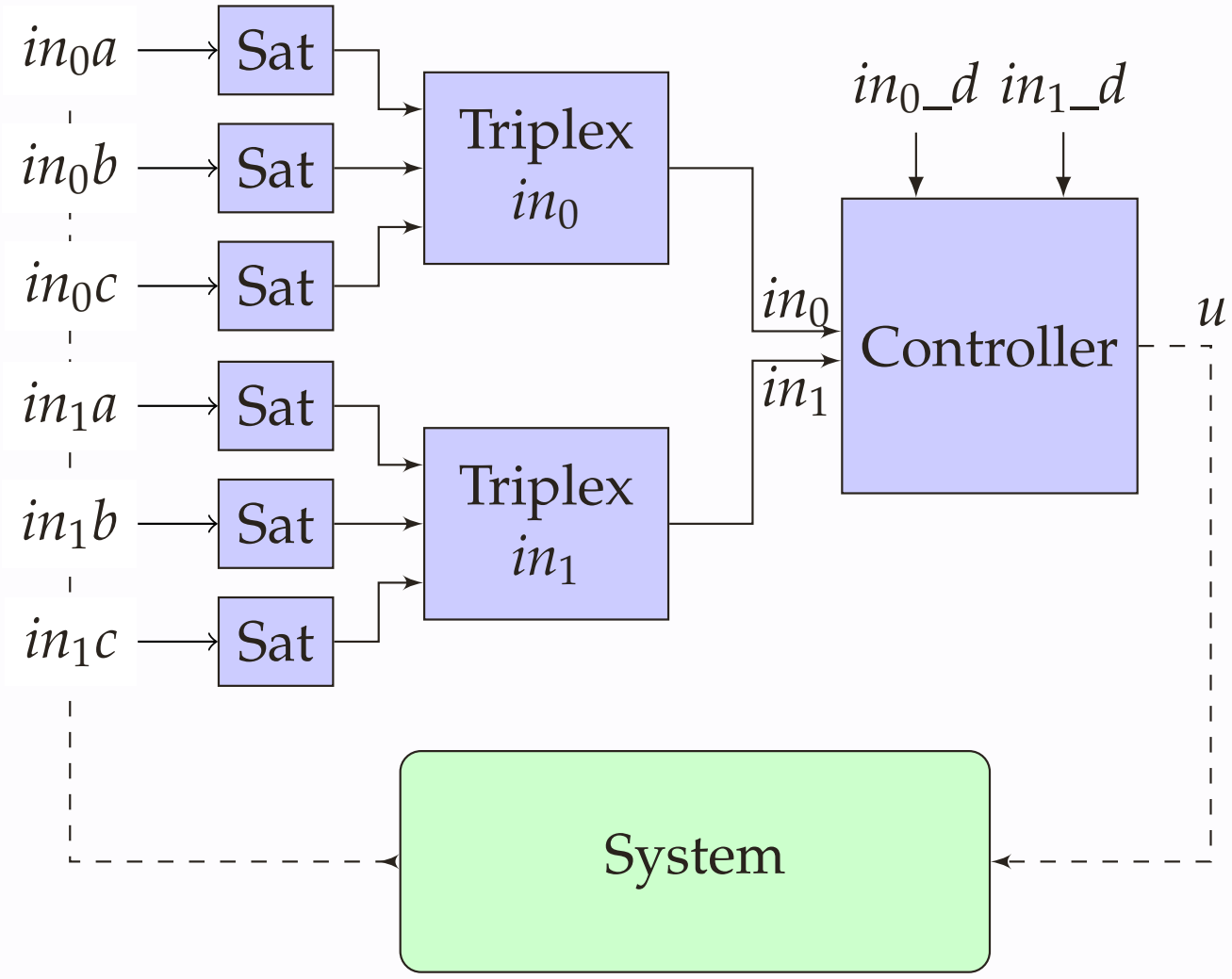
- Actual implementation:
  - floats not reals
  - pointers, arrays, memory access $\rightarrow$ potential failure
  - real world: overflows

Today's Target: control systems for a UAV

Typical development cycle of a controller for an aircraft



System Example: basic triplication pattern



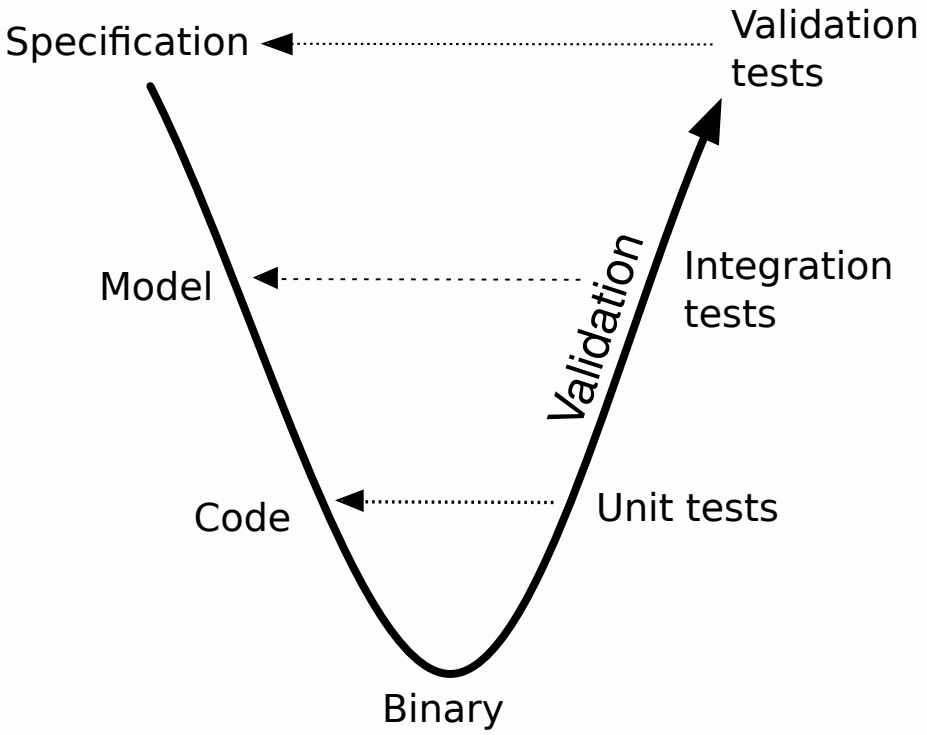
Regulation : certification authorities

Each industry has its own certification process and associated authorities

- Different rules for different domains :
  - civil aircrafts
  - railway
  - space
  - automotive
- Regulation are usually international, or at least European for France.
- National authorities have the duty to receive your files and validate your plane/car/train with respect to regulation
  - in France, for civil aircrafts, this is the role of the DGATA, en entity of the french defense department

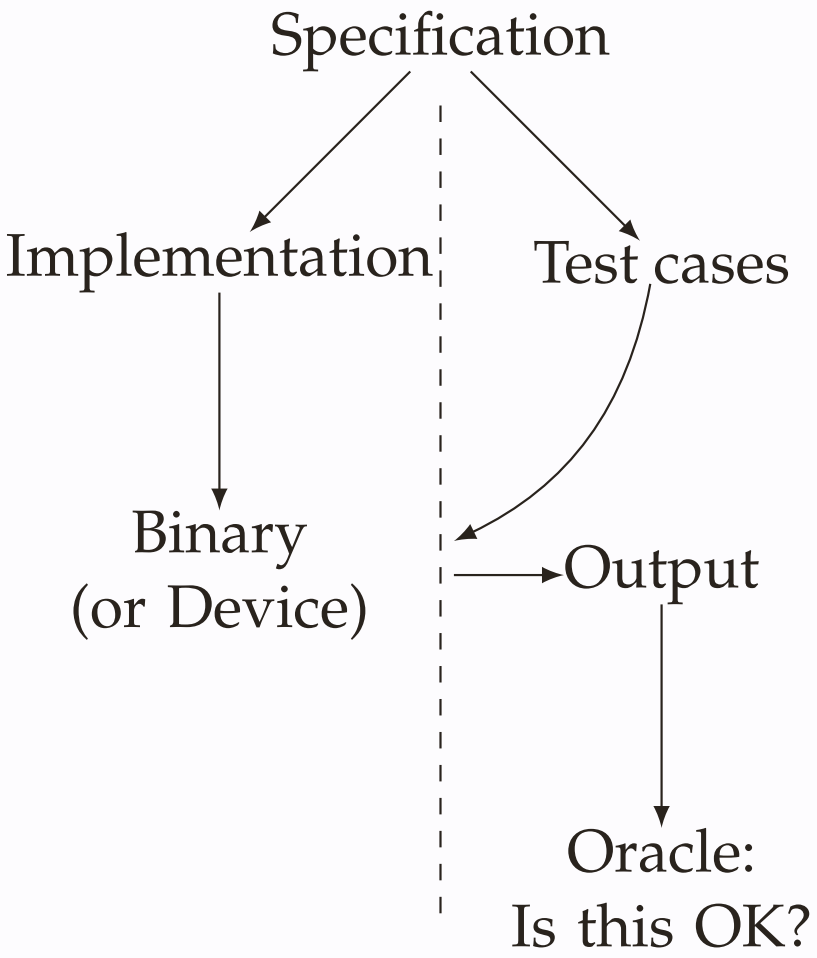
Not quite clear yet for UAVs

GUARANTYING SOFTWARE: A PROCESS BASED APPROACH

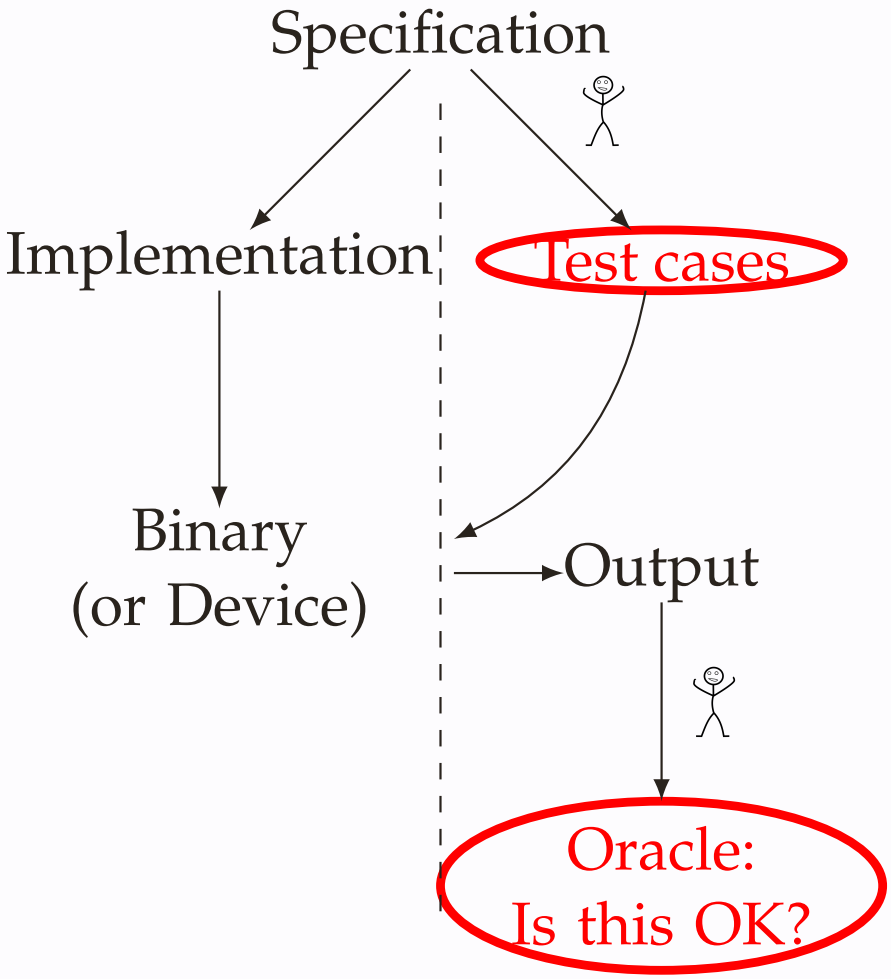


- Relative cost of systems in an aircraft: 30% for civil planes, 50% for military
- Relative cost of validation for systems: « [..] costly testing and validation phases that can take up to 80% of the cost of designs. » [IST Report]

What is testing ?



WHAT IS TESTING ?



Manual parts are the costly ones.

CIVIL AIRCRAFTS: THE DO 178 B & C

- Old certification document: first version in the 80s, B version in 92, now switching to C.
- Not so big: 140 pages for B version

- 3 main words: traceability, conformance, verification
  - traceability with respect to upper level requirements
    - each item at any place should be linked to its original requirement, no dead code, you need to master everything
  - conformance to standards at a given level
  - verification w.r.t higher level specification

The norm:

- the norm define goals and not means.
- you could do what you want.
- formal methods can be applied.

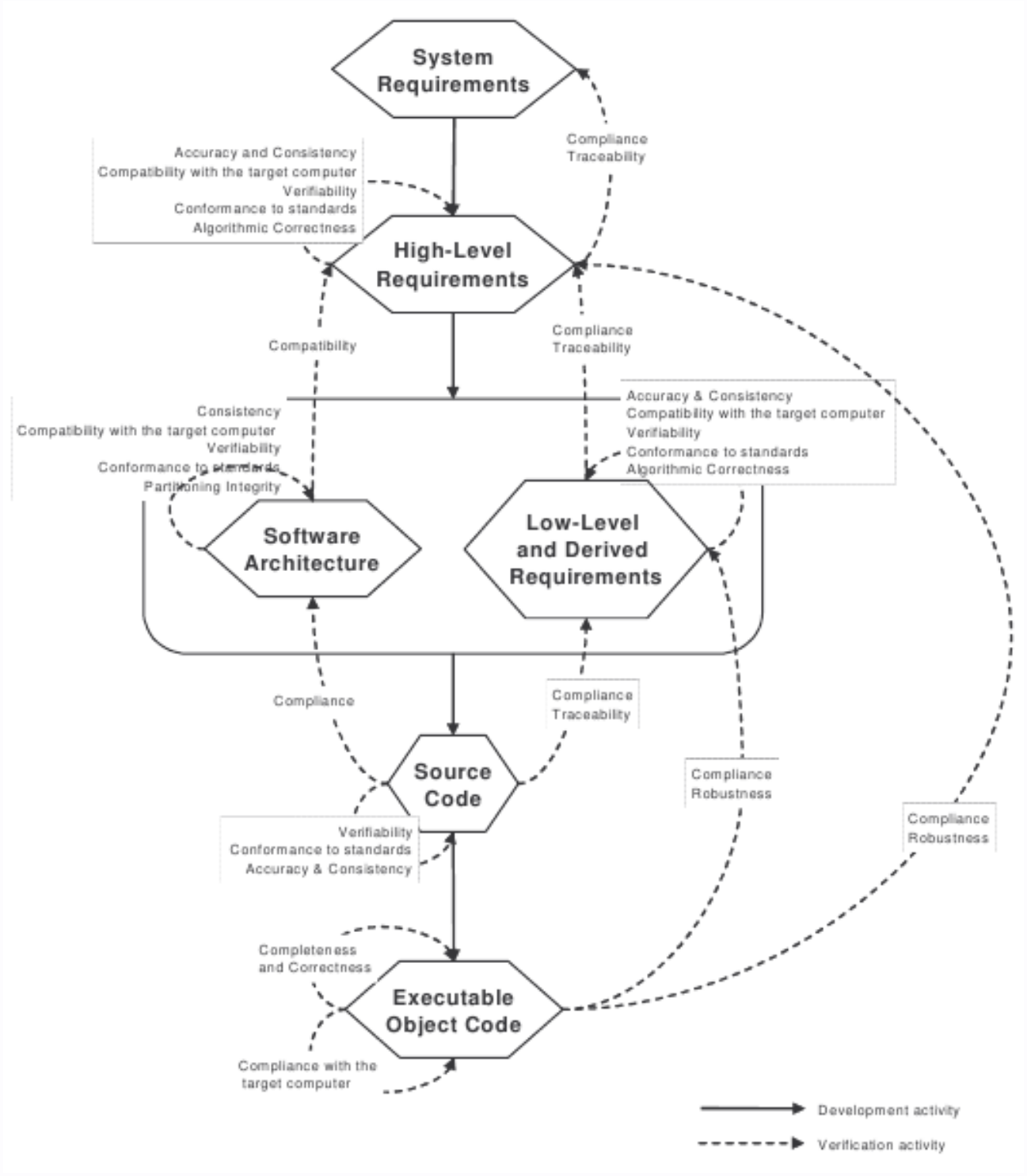
CIVIL AIRCRAFTS: THE DO 178 B & C

- Old certification document: first version in the 80s, B version in 92, now switching to C.
- Not so big: 140 pages for B version

- 3 main words: traceability, conformance, verification
  - traceability with respect to upper level requirements
    - each item at any place should be linked to its original requirement, no dead code, you need to master everything
  - conformance to standards at a given level
  - verification w.r.t higher level specification

The norm:

- the norm define goals and not means, but ... test oriented.
- you could do what you want but ... the test results should be.
- formal methods can be applied but ... tests should be.


System Requirements
Compliance
Traceability
Accuracy and Consistency
Compatibility with the target computer
Verifiability
Conformance to standards
Algorithmic Correctness
High-Level Requirements
Compatibility
Compliance
Traceability
Consistency
Compatibility with the target computer
Verifiability
Conformance to standards
Partitioning Integrity
Accuracy & Consistency
Compatibility with the target computer
Verifiability
Conformance to standards
Algorithmic Correctness
Software Architecture
Low-Level and Derived Requirements
Compliance
Compliance
Traceability
Source Code
Compliance
Robustness
Compliance
Robustness
Verifiability
Conformance to standards
Accuracy & Consistency
Completeness and Correctness
Executable Object Code
Compliance with the target computer
Development activity
Verification activity

DO TESTING PROCESS AND CRITICALITY LEVELS

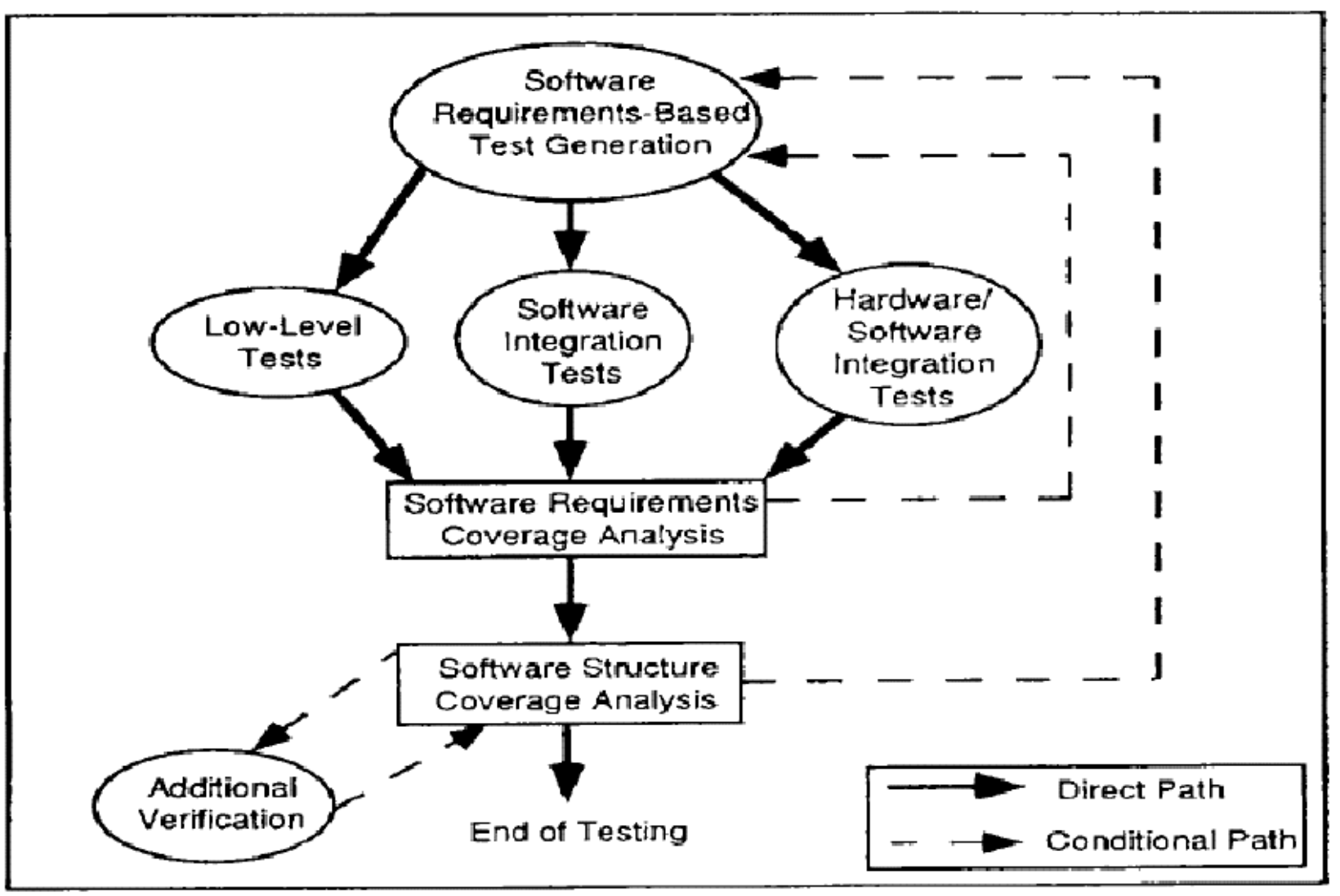
DO testing process

- Tests should be only requirement-based
  - hence: no test generation based on code is allowed
  - normal range tests, robustness tests, based on specification
  - 3 kinds: integration HW/SW, integration SW, low level tests
- When to stop testing: structural coverage criteria

DO criticality levels

- four levels from most critical (A) to less (D)
- in practice each level is associated with a coverage criteria: for A level, MC/DC modified condition/decision coverage exhaustive coverage of the atomic conditions of the boolean formula

DO TESTING PROCESS CONT'D



Certifying a system under DO178B

The practice:

- the aircraft manufacturer argues with a certification authority
- production of heavy certification document
- specific authorization for specific industrial projects

$\Longrightarrow$ it is possible to do something else than just tests such as formal methods

Classical approach to Verification and Validation (V&V): Test!

- simulation at model level (among other issues: work on a simplification of the plant description, an ideal representation)
- informal specification (natural language) of each component (HLR / LLR)
- manual coding
- Unit Tests / Integration tests for software
- Hardware in the loop tests, validation tests, flight tests ... once integrated with the OS, the platform, the aircraft

Critical software

Target systems:

Control command, flight control, safety tests, engines ...

Most critical : level A

Reactive systems : software that computes a loop body forever

- reads the control input, the environment input, produces a feedback
- complex logic of alarm handling and reporting
- necessity to guaranty time constraint: execution time bounds

Important issue: we never look for bug, we rather ensure their absence !

Critical software: what to verify ?

Due to certification issues, programs should be mastered: reasonable subset of C, no malloc, no recursion, but floats. $\Longrightarrow$ C0 like code

Conformance (general properties):

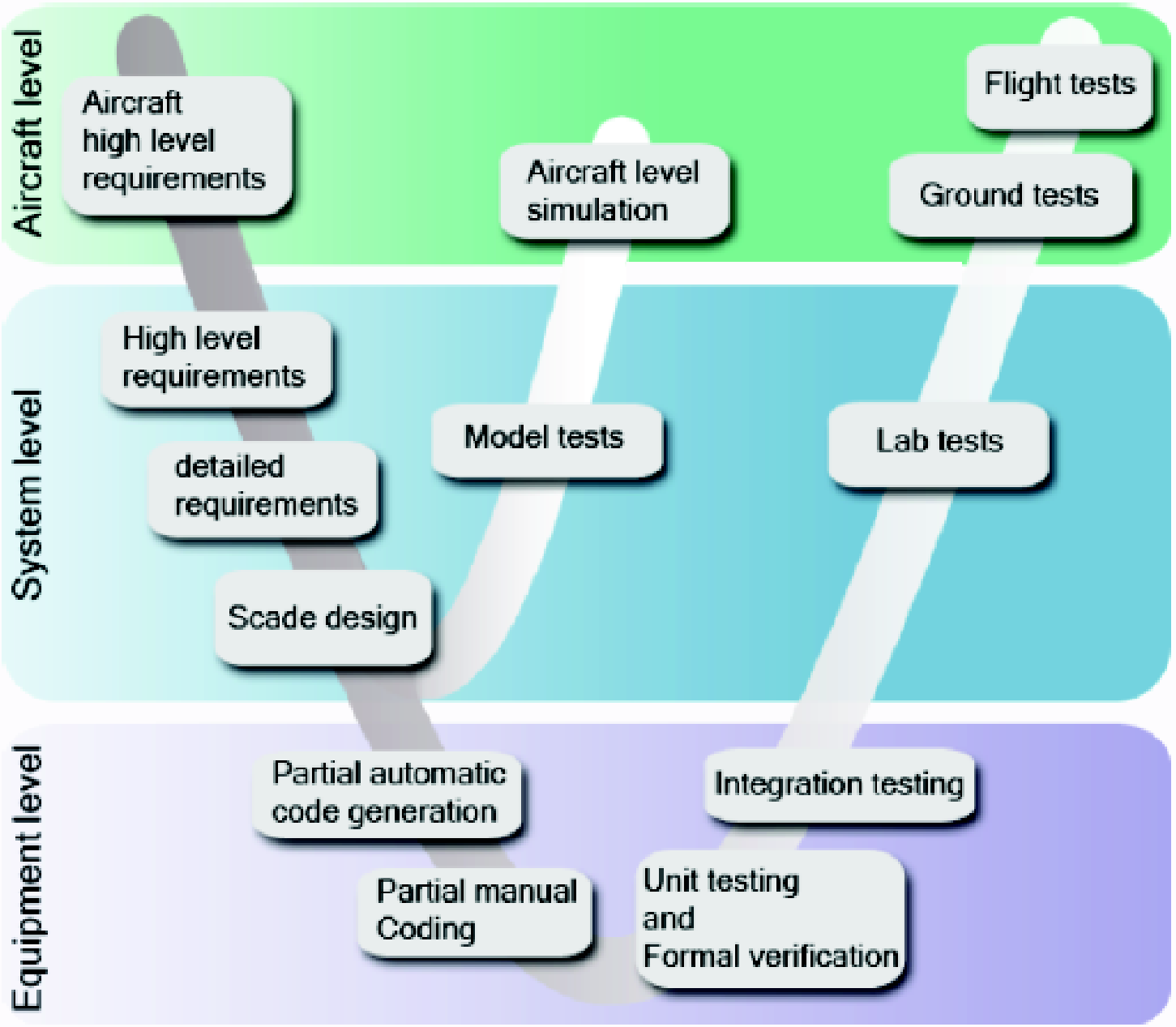
- Reactive systems: real time issues (Worst Case Execution Time)
- Run Time Errors: overflow, pointer dereferencing, out-of-bound access, illegal arithmetic ops, ...
- Floating points vs Reals

Verification (specific properties):

- Low level specification

Rely as much as possible on autocoders to generate code from models.

Example: Airbus Development cycle



Contents

Program semantics

Semantics ≡ behavior of the program

Differents means to describe it

- Operational semantics
  - system described as transitions from one step to the other

$$c_1 \rightarrow_\alpha c_2 \rightarrow_\beta c_3 \rightarrow_\beta c_4 \rightarrow_\gamma \ldots c_{n-1} \rightarrow_\alpha c_n$$

- Denotational semantics
  - interested in the result, not the intermediate states

$$[\![e]\!] = \text{ meaning of e ie. its value}$$

- Axiomatic semantics less precise
  - does not define the output as a function over the input
  - does not describe the computation steps

  but - specify the expected behavior - is independent on the implementation

Trace Semantics vs Collecting Semantics

Properties of the systems are expressed/observed/verified over executions.

Consider a transition system S defined as (Σ, I, R)

- Σ: set of states
- $I \subseteq \Sigma$: initial states
- $R \in \Sigma \times \Sigma$: computation steps

Trace semantics

$$TS(S) \triangleq \{s_0 s_1 \ldots | s_0 \in I, \forall i \geq 0, (s_i, s_{i+1}) \in R\}$$

Properties: temporal logics

Collecting semantics

$$CS(S) \triangleq \{s_n | \exists n \geq 0 \exists (s) \in TS(S)\}$$

Properties: safety

Formal methods

Why using formal methods ?

- Strong mathematical evidence: exhaustive analysis
- Could be automatized for some kind of programs
- Reusability (software evolution)
- Cost killer (less time to verify, less people needed)
- More interesting for humans :)

Safety properties

- FM provide means to verify that a program verifies a property
- Here, safety properties: observable on the collecting semantics. Ie. true for all reachable state independently of their past.

BAD NEWS: UNDECIDABILITY

Assuming a method allows to verify a property on a program. It should be

sound complete terminating

Bad news: undecidability

Assuming a method allows to verify a property on a program. It should be

sound complete terminating

Infortunately, as reported by Rice's undecidability theorem, no such method exists in general for non trivial properties.
Choice: loose the completeness property
When it fails: find alternatives, tests, manual reviews, etc.

BAD NEWS: UNDECIDABILITY

Assuming a method allows to verify a property on a program. It should be

sound complete terminating

Infortunately, as reported by Rice's undecidability theorem, no such method exists in general for non trivial properties.
Choice: loose the completeness property
When it fails: find alternatives, tests, manual reviews, etc.

Different approaches to reason on programs

- SMT based model-checking
- Abstract Interpretation
- Deductive methods

Contents

SMT-BASED MODEL-CHECKING

- Encode the model semantics as a predicate in SMT logics: M(x,y)
  - ie. in a tooled decidable subset of first order logic.
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination
  - prove the non reachability of a set describing bad state

SMT-BASED MODEL-CHECKING

- Encode the model semantics as a predicate in SMT logics: M(x,y)
  - ie. in a tooled decidable subset of first order logic.
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination
  - prove the non reachability of a set describing bad state

Example of algorithms: k-induction, PDR/IC3

SMT-BASED MODEL-CHECKING

- Encode the model semantics as a predicate in SMT logics: M(x,y)
  - ie. in a tooled decidable subset of first order logic.
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination
  - prove the non reachability of a set describing bad state

Example of algorithms: k-induction, PDR/IC3
pros:

- capable of producing a concrete counter example (bounded model checking)
- Useful to debug or understand the origin of the property violation

SMT-BASED MODEL-CHECKING

- Encode the model semantics as a predicate in SMT logics: M(x,y)
  - ie. in a tooled decidable subset of first order logic.
- Perform inductive reasoning for a given property:
  - eg: $true \models P(init)$ and $P(x) \wedge M(x,y) \models P(y)$
- Compute backward analysis using quantifier elimination
  - prove the non reachability of a set describing bad state

Example of algorithms: k-induction, PDR/IC3
pros:

- capable of producing a concrete counter example (bounded model checking)
- Useful to debug or understand the origin of the property violation

cons:

- are restricted to linear inductive or k-inductive properties;
- bad results in presence of complex numerical computations
- depend on the power of SMT solvers
- some properties may be hard to analyze

Model-checking: a simple Lustre example

```
node accu(i: int) returns (o: int);
let
      o = 0 -> i + pre o;
tel;
–@ ensures out ≥ 0;
node f(reset: bool) returns (out: int);
var cpt: int; ok : bool;
let
      cpt = if reset then 0 else (0 -> 1 + pre cpt);
      out = if reset then 0 else accu(cpt);
tel;
```

- Not k-inductive (for any k): if cpt < 0 then out could be negative
- Reinforcing the property: out $\geq 0 \wedge$ cpt ≥ 0 is 1-inductive

Contents

Abstract Interpretation - reasoning on set of states as abstract states

$$\begin{array}{rcl} Prog \models Prop & \equiv & CollectingS(Prog) \subseteq Prop \\ x \text{ is even where } x \in \{2,4,6\} & \equiv & \{2,4,6\} \subseteq \{\ldots,-4,-2,0,2,4,\ldots\} \end{array}$$

Idea: compute an overapproximation $x^{\#} \supseteq x$:

$x^{\#} = \{2,4,6,8\} \subseteq Even$ OK, 8 denoting a spurious value

$x^{\#} = \{2,4,5,6\} \not\subseteq Even$ we are not able to conclude

Algorithm: Kleene fixpoint computation on the monotonic abstract semantics of the program

Example of abstractions

- intervals
- Convex close polyhedra
- Zonotopes (vector of affine forms)
- Ellipsoids (level set of quadratic polynomials)

Abstract Interpretation - Application

- Recast the semantics as fixpoint

$$CS = lfp\left(\lambda X.I \cup \{s' | s \in X, (s, s') \in R\}\right)$$

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain
- Compute an over approximation of reachable states in the abstract domain
  - operators are used to

widening: force termination through additional abstraction, ie. imprecision

narrowing: regain part of the lost precision

Abstract Interpretation - Application

- Recast the semantics as fixpoint

$$CS = lfp\left(\lambda X.I \cup \{s' | s \in X, (s, s') \in R\}\right)$$

- Choose an appropriate abstraction depending on the property to be proved (boundedness, relationship between variables, memory issues, etc)
- Express the model semantics in the abstract domain
- Compute an over approximation of reachable states in the abstract domain
  - operators are used to

widening: force termination through additional abstraction, ie. imprecision

narrowing: regain part of the lost precision

Stable linear controllers with or without saturations are analyzed using a specific abstract domain:

- The control flow graph of the controller is identified
- The stability of each linear subsystem is analyzed and provides a quadratic Lyapunov function (ellipsoid)
- The set of reachable states is bounded using the generated ellipsoids.

EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

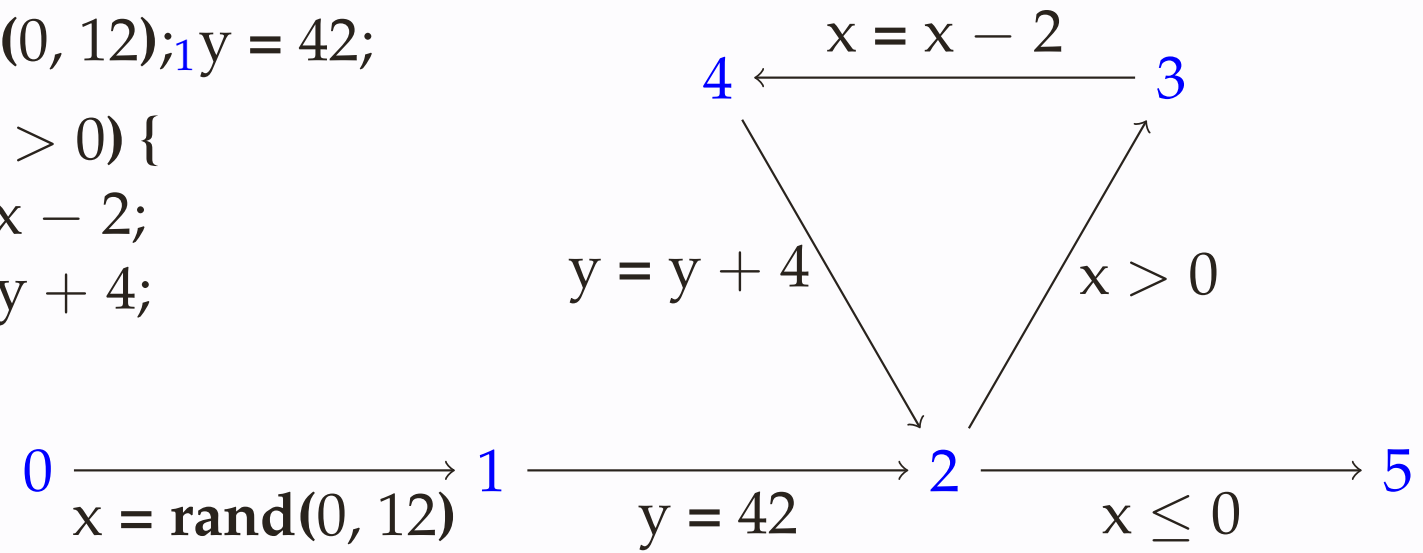
$_0$x = **rand**(0, 12);$_1$y = 42;

while $_2$(x > 0) {

    $_3$x = x − 2;

    $_4$y = y + 4;

}$_5$



EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

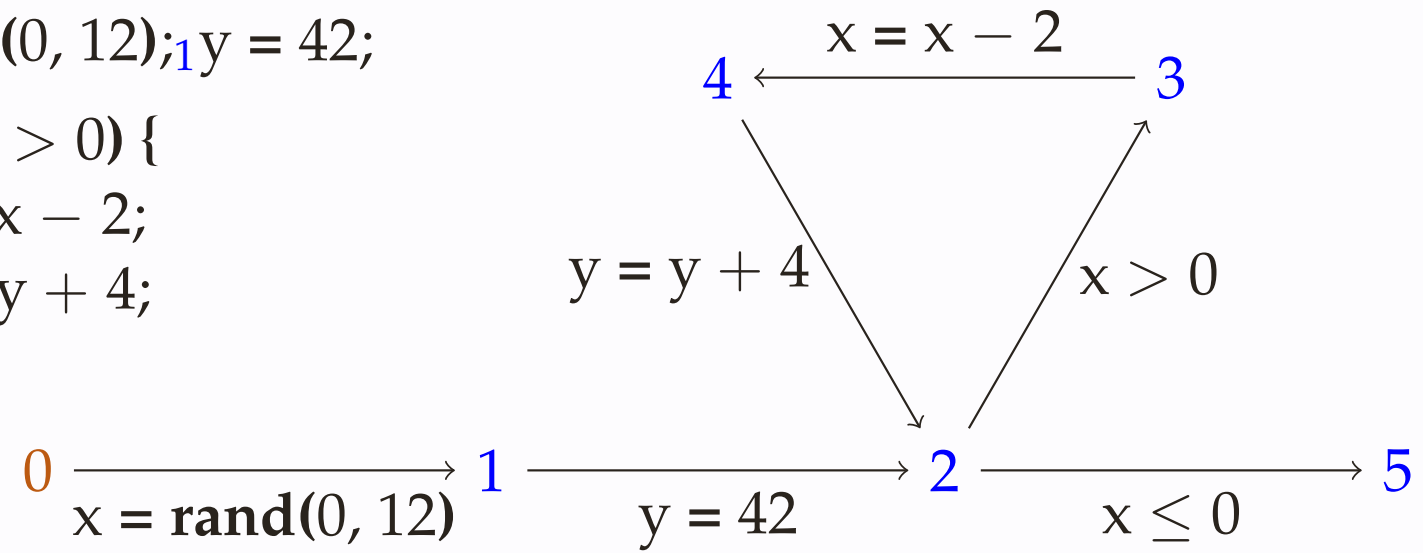
$_0$x = **rand**(0, 12);$_1$y = 42;

while $_2$(x > 0) {

  $_3$x = x − 2;

  $_4$y = y + 4;

}$_5$

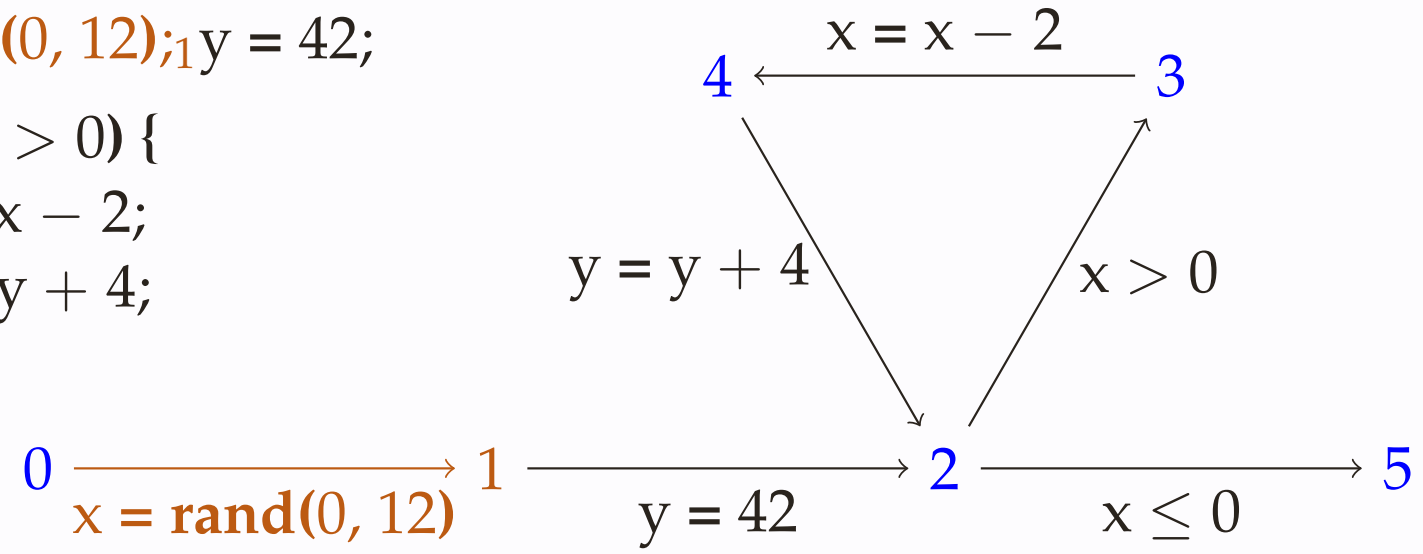


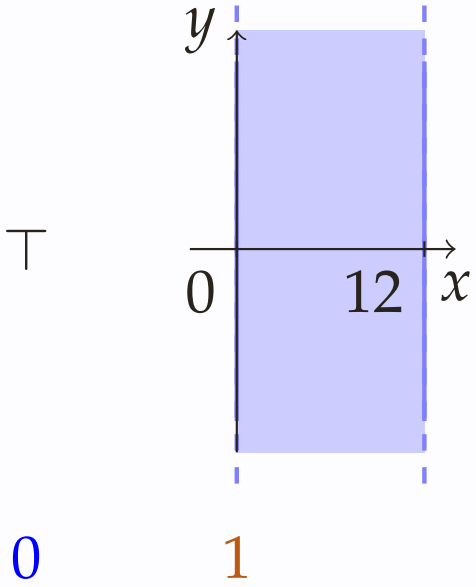
$\top$

0

EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

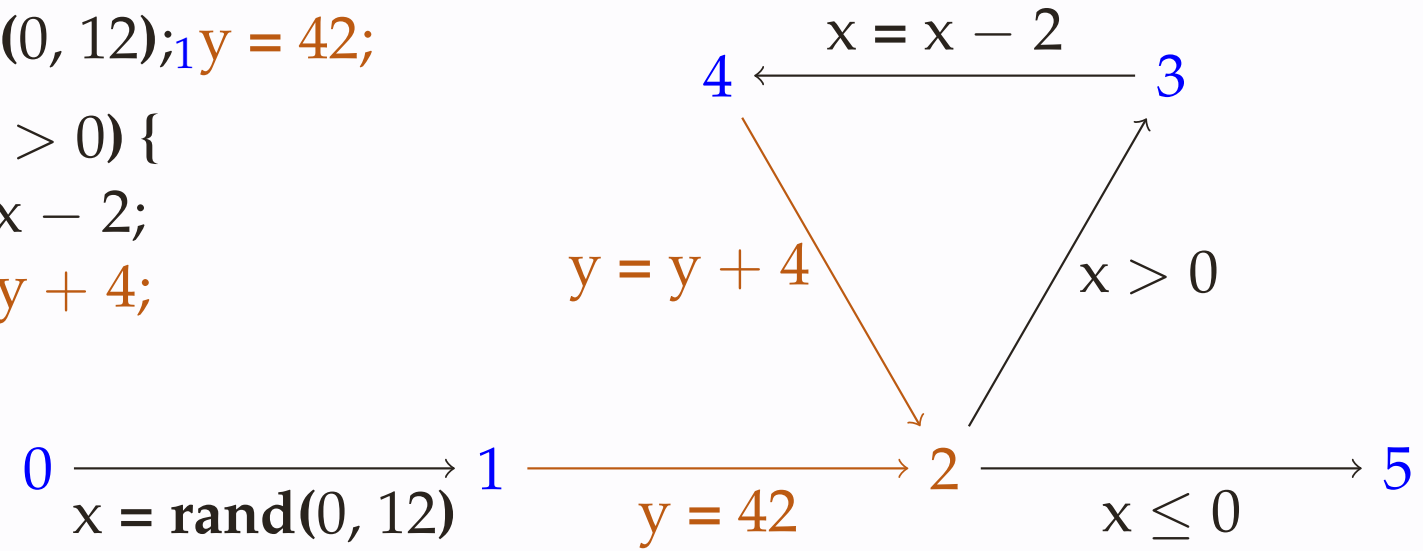
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

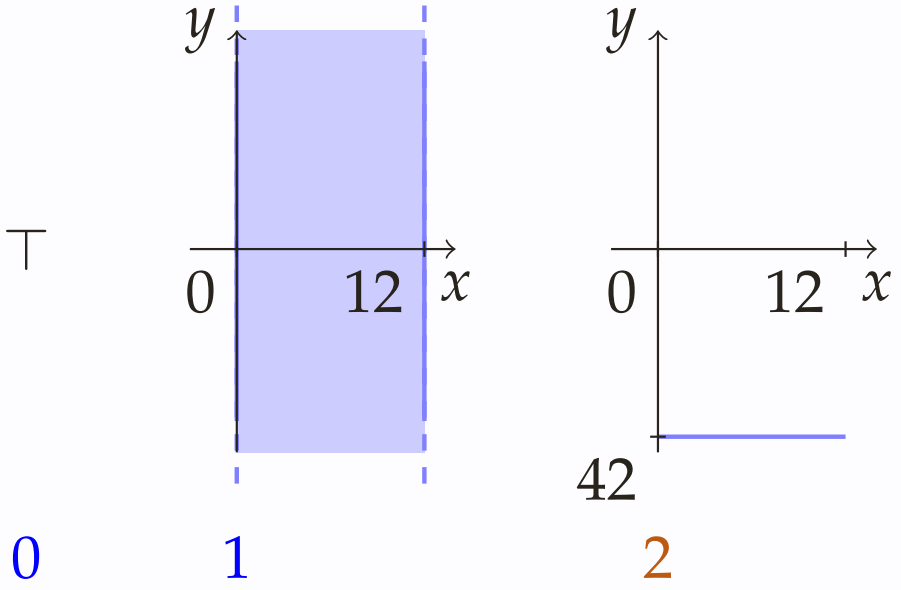




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

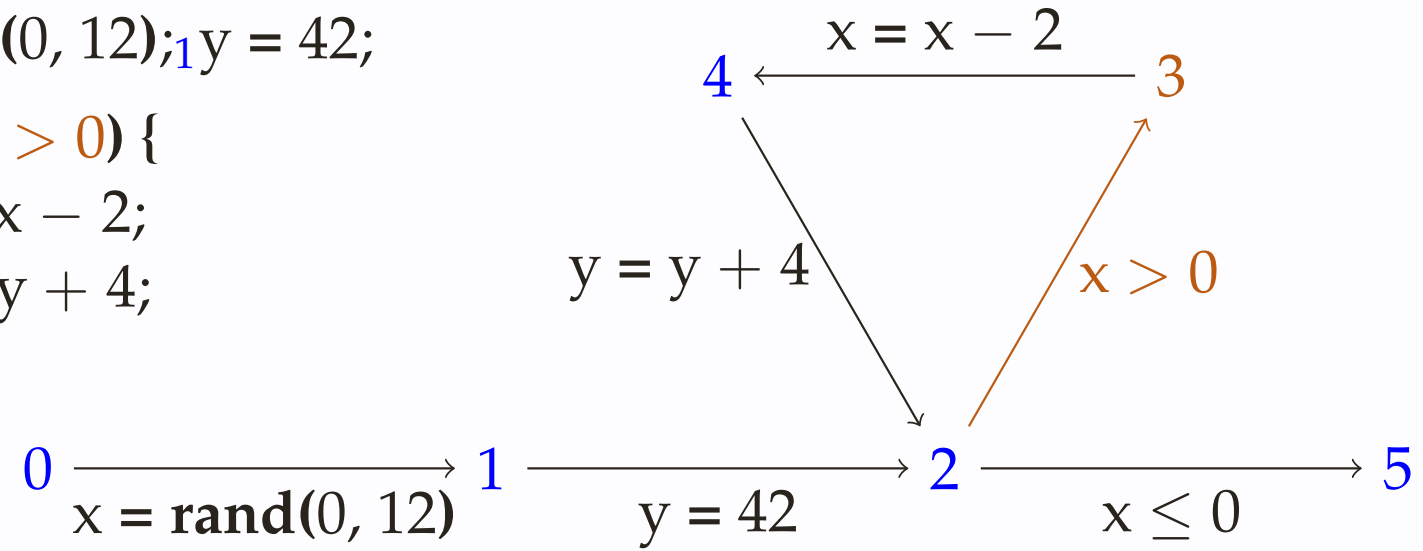
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

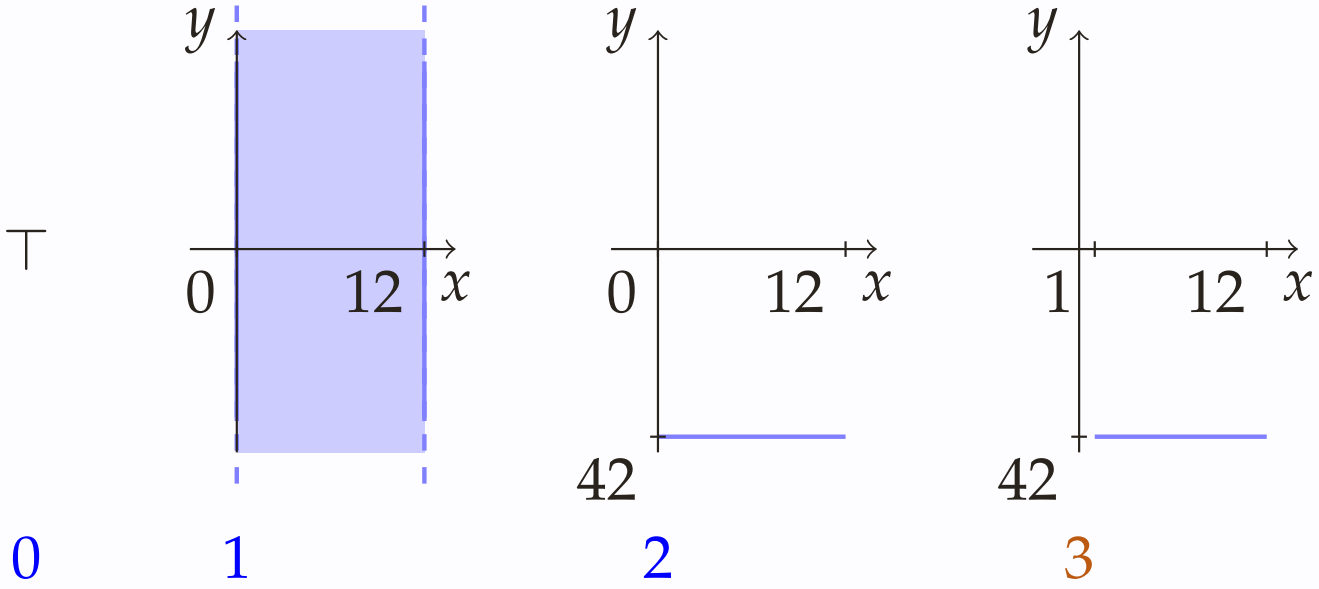




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

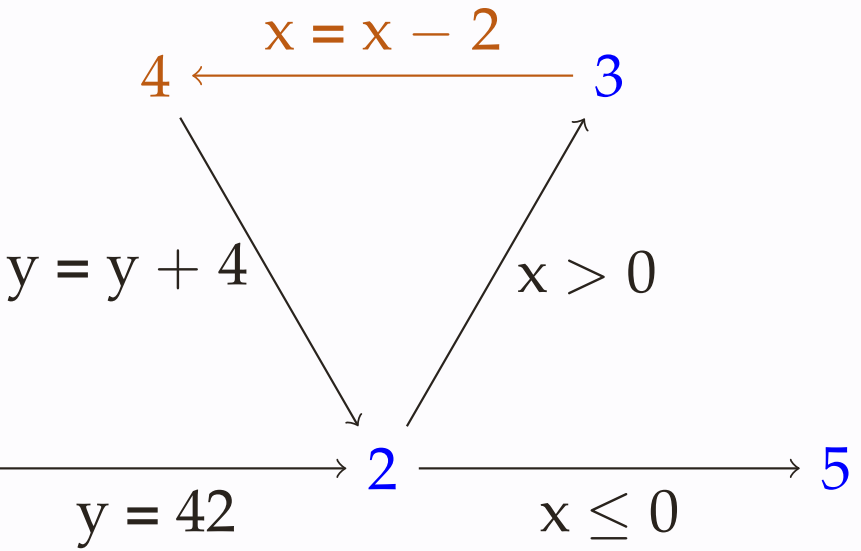
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

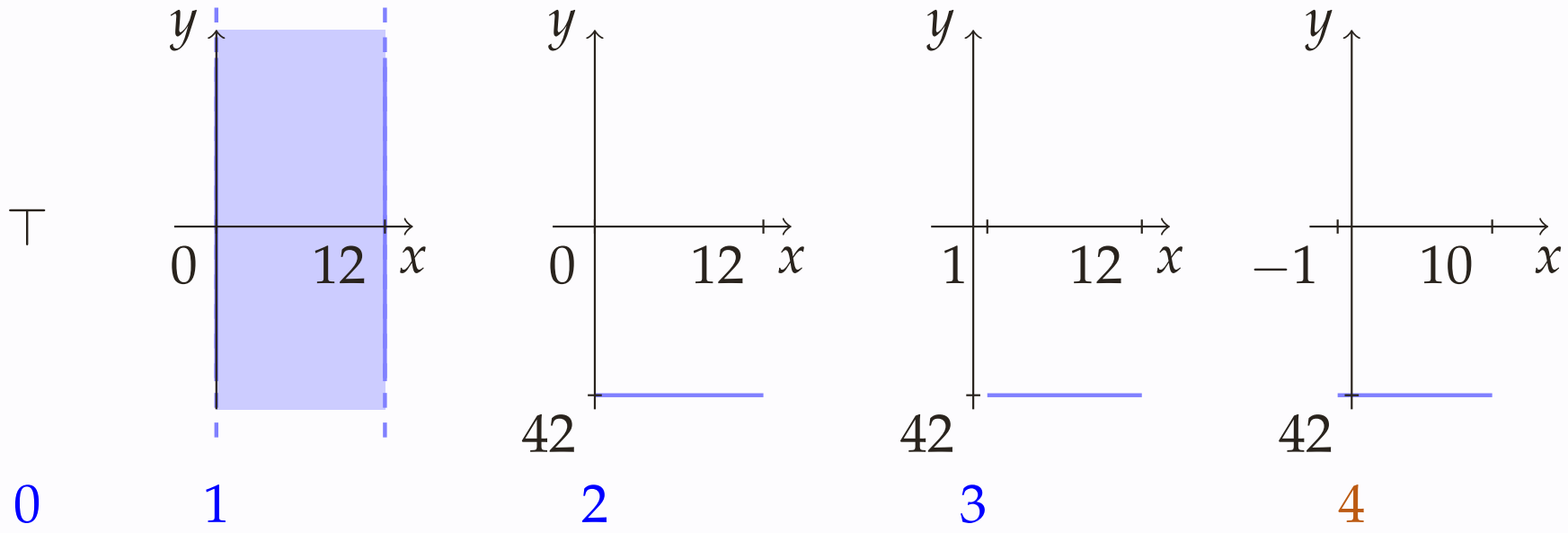




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

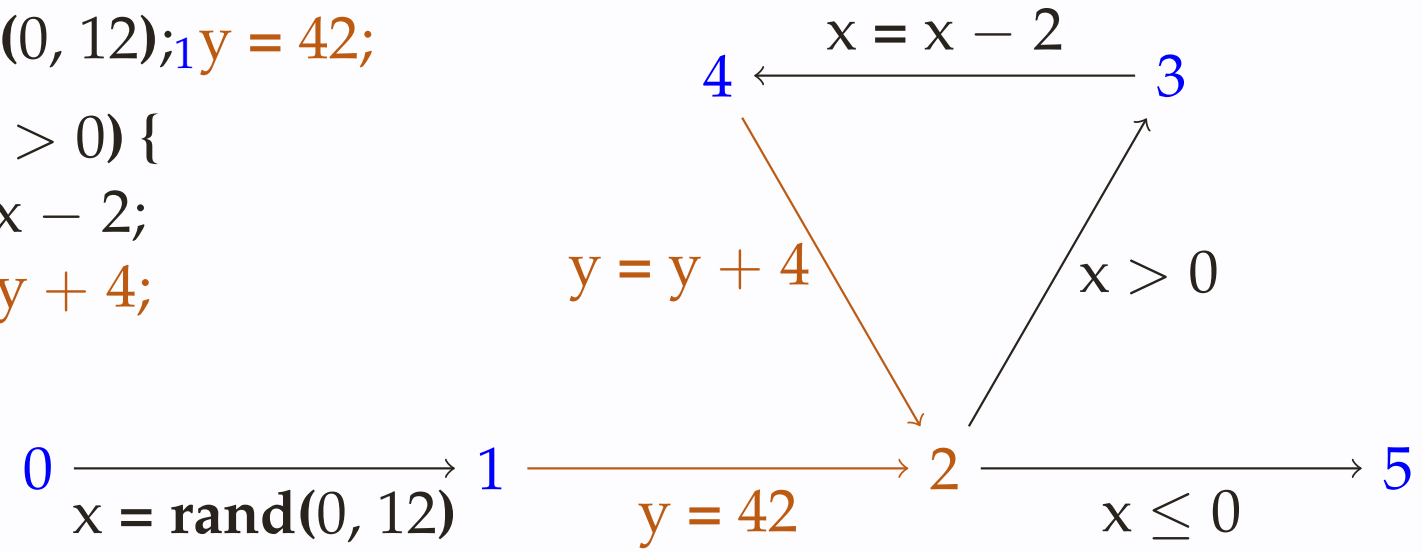
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

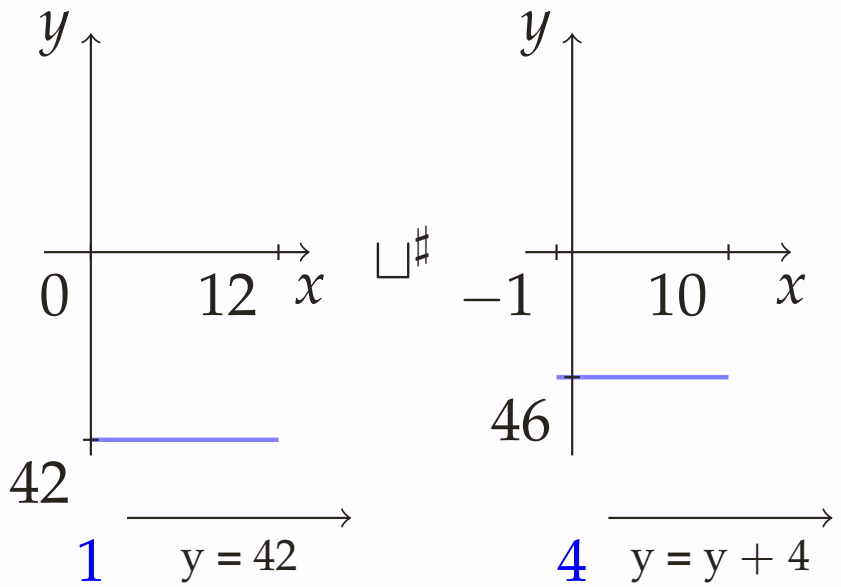




Example: a simple polyhedral analysis on integers

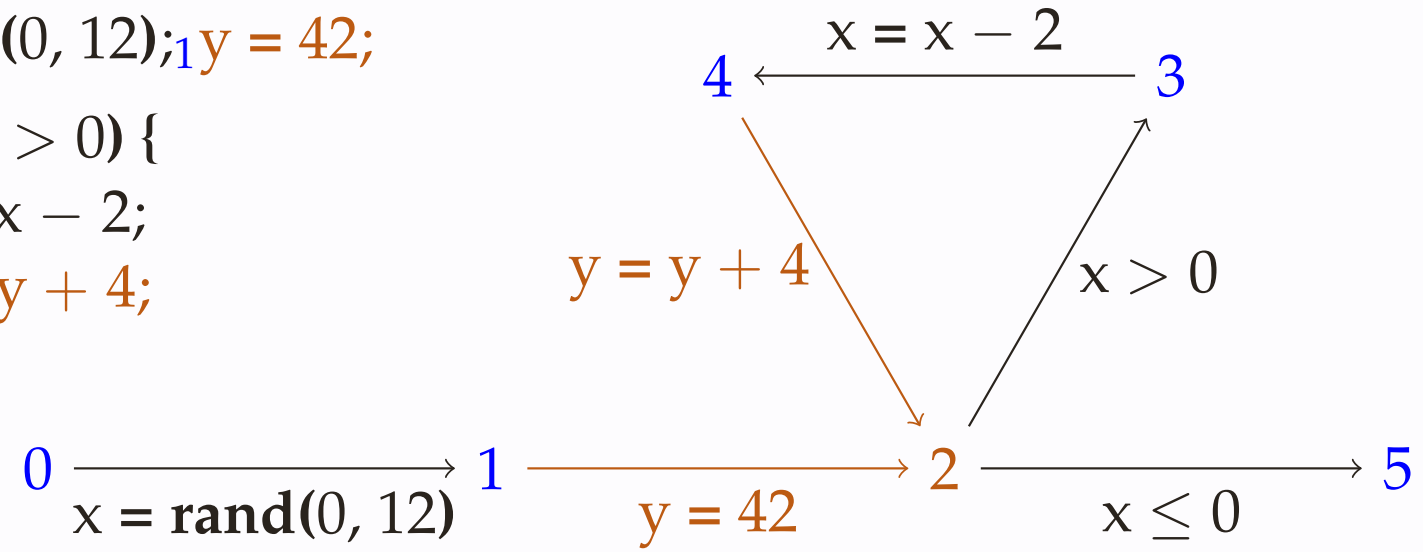
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

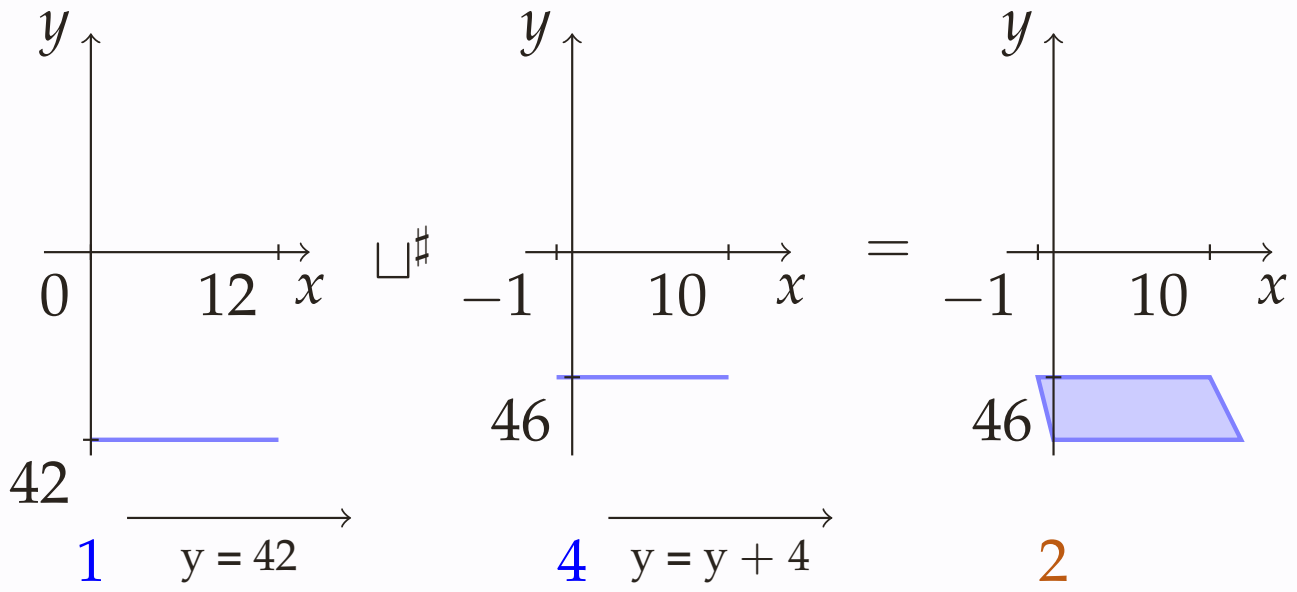




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

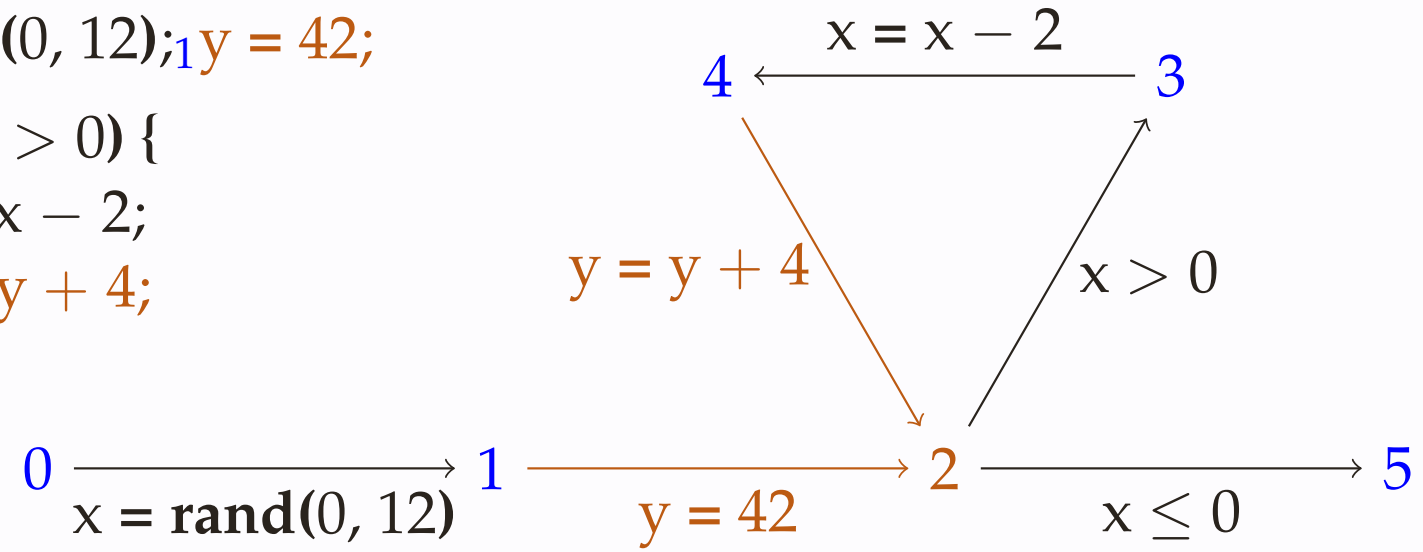
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

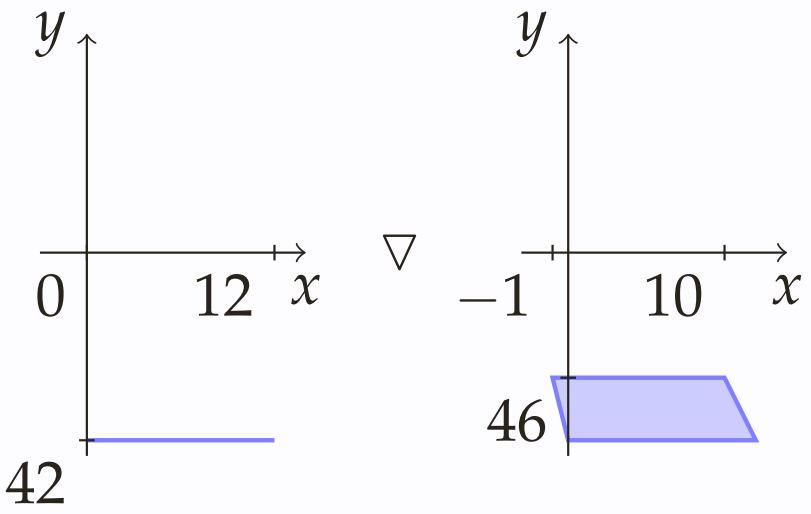




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

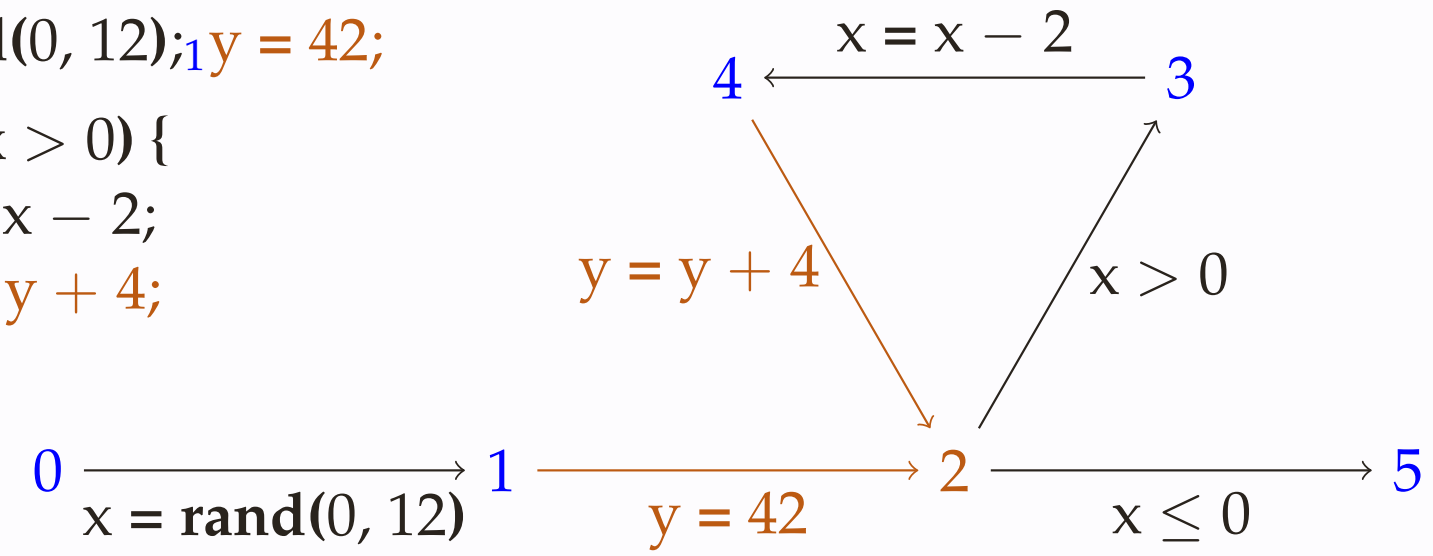
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

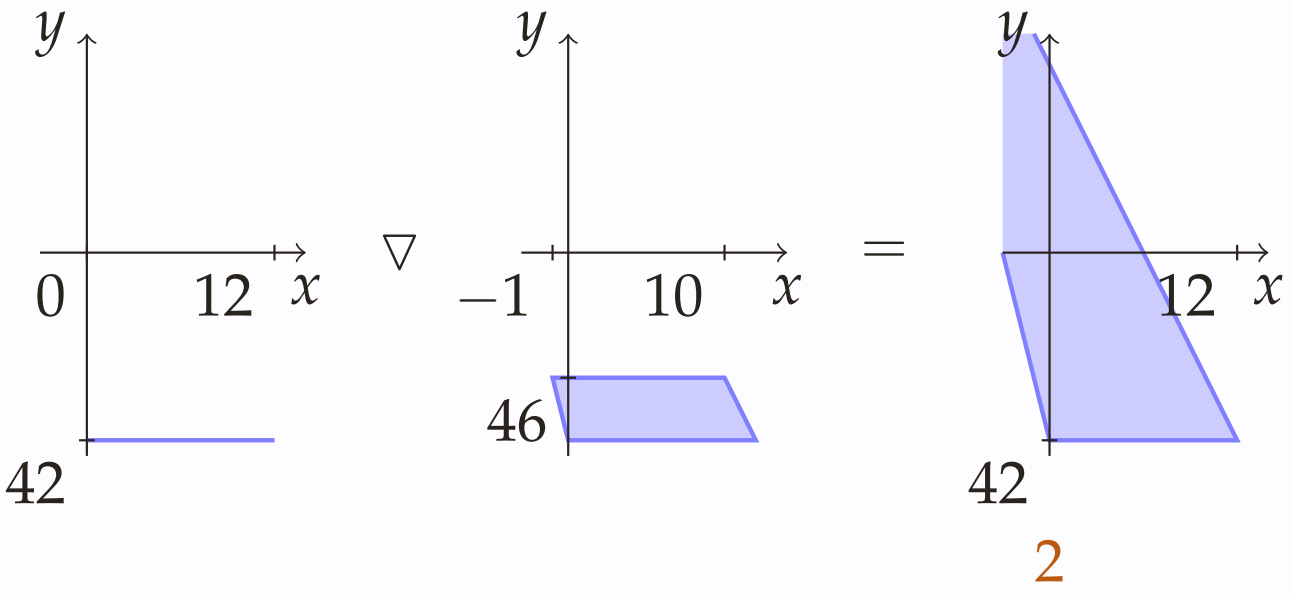




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```





EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

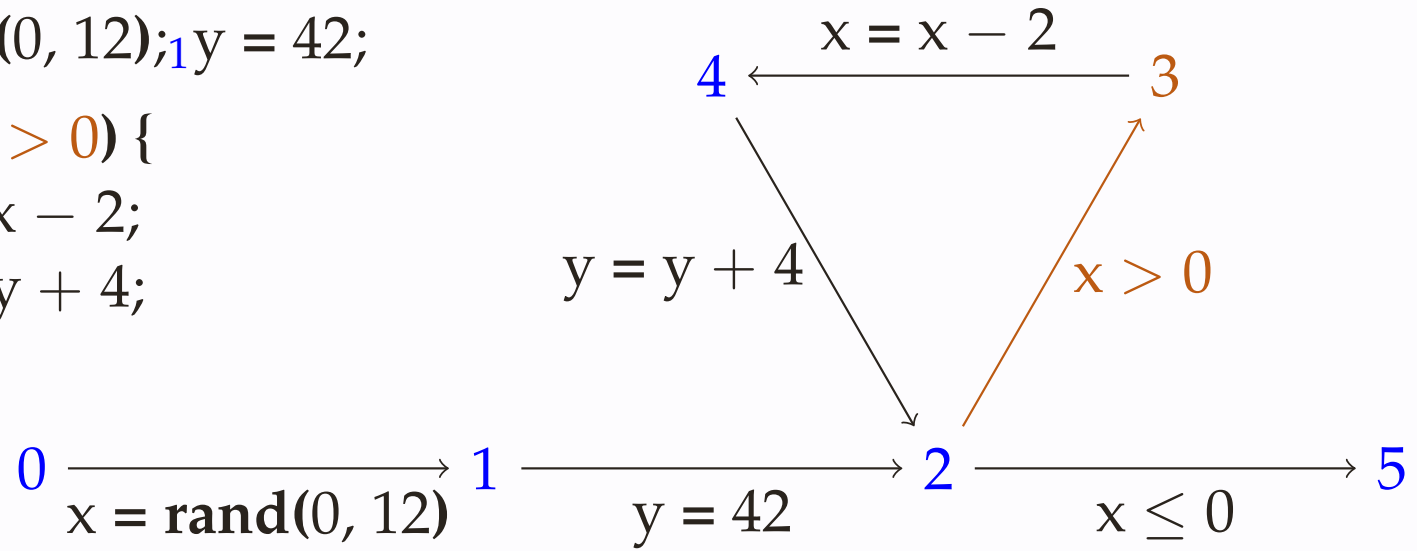
$_0$x = **rand**(0, 12);$_1$y = 42;

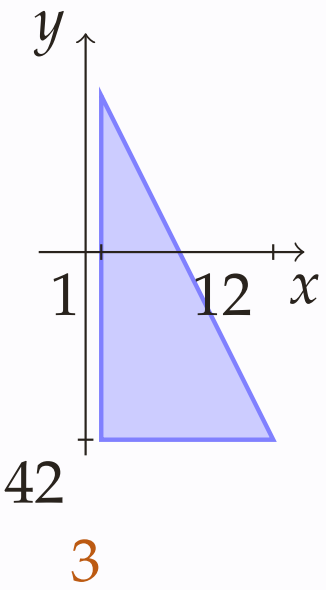
while $_2$(x > 0) {

  $_3$x = x − 2;

  $_4$y = y + 4;

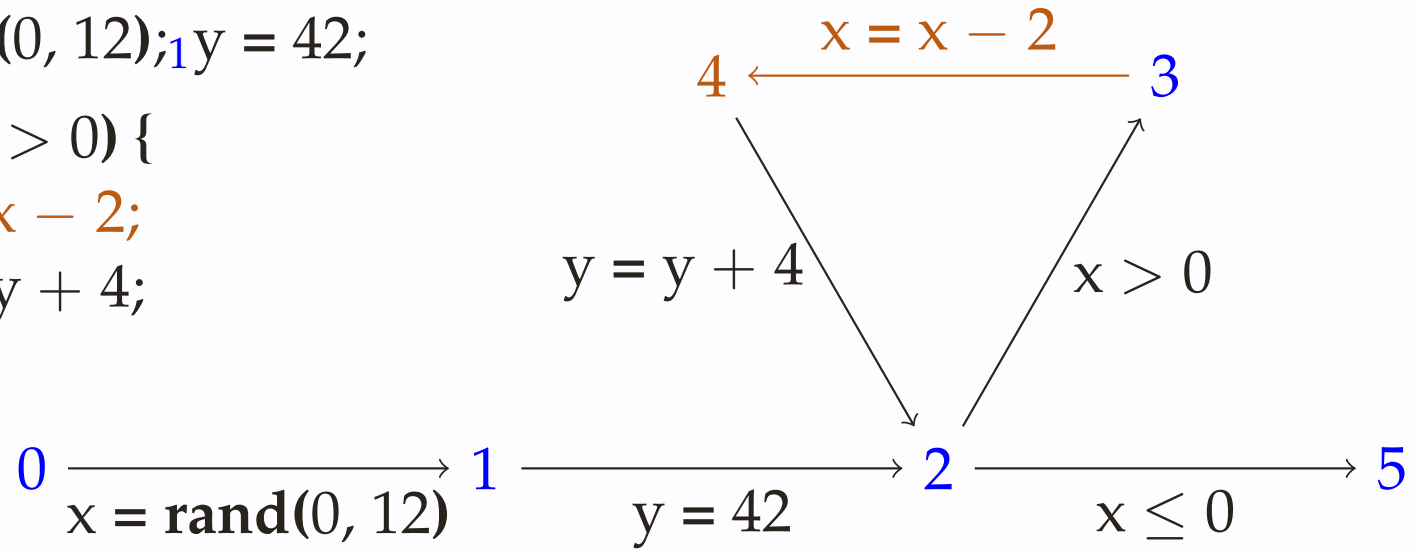
}$_5$

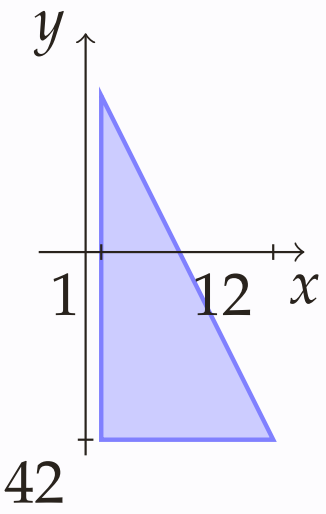




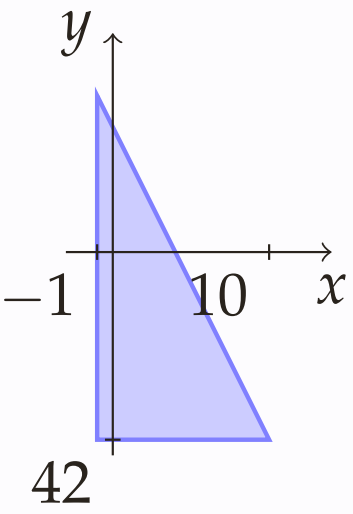
EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```





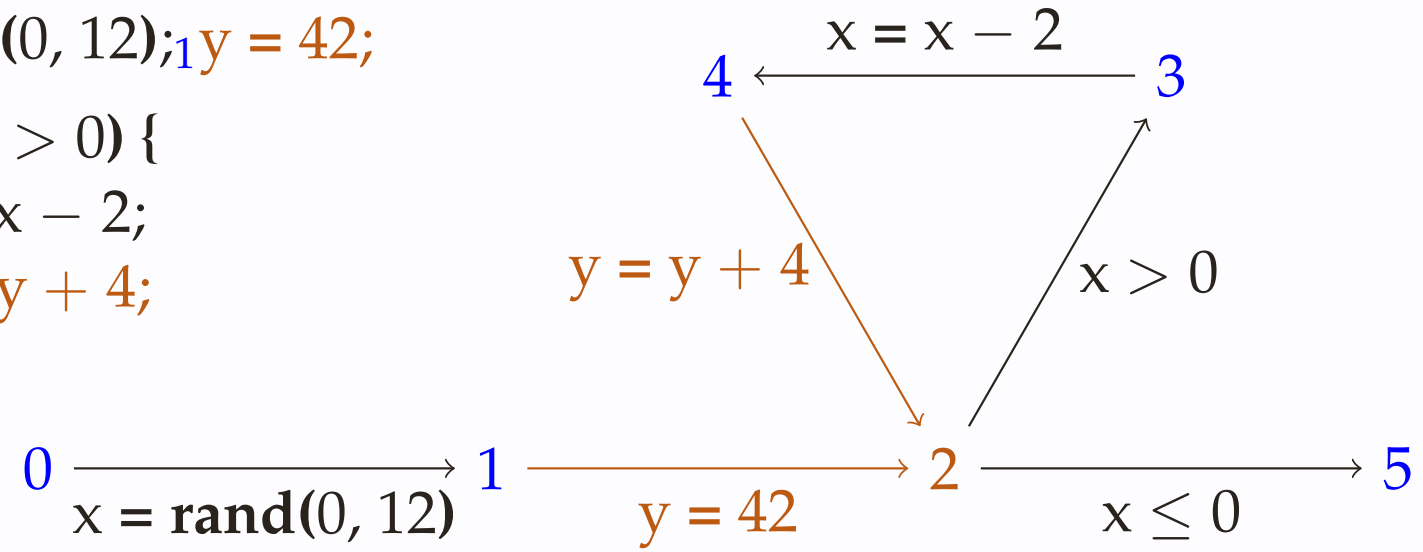
3

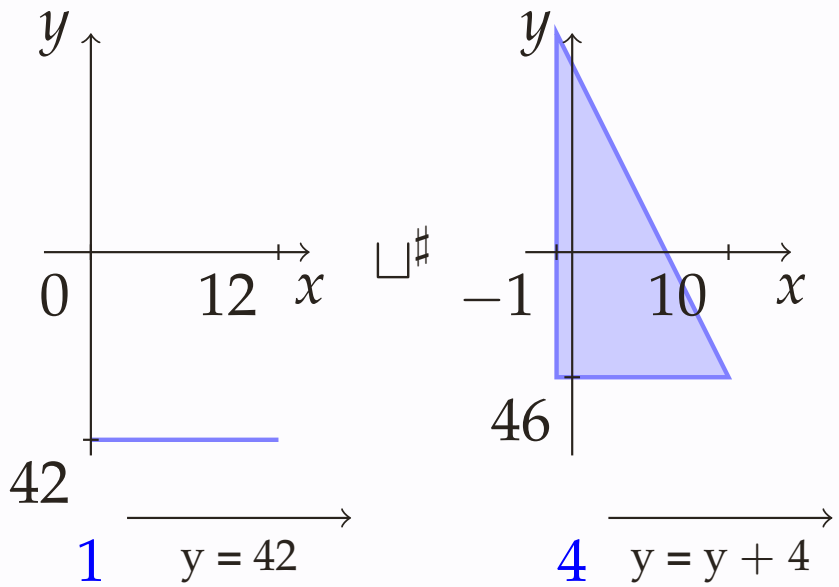


4

EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

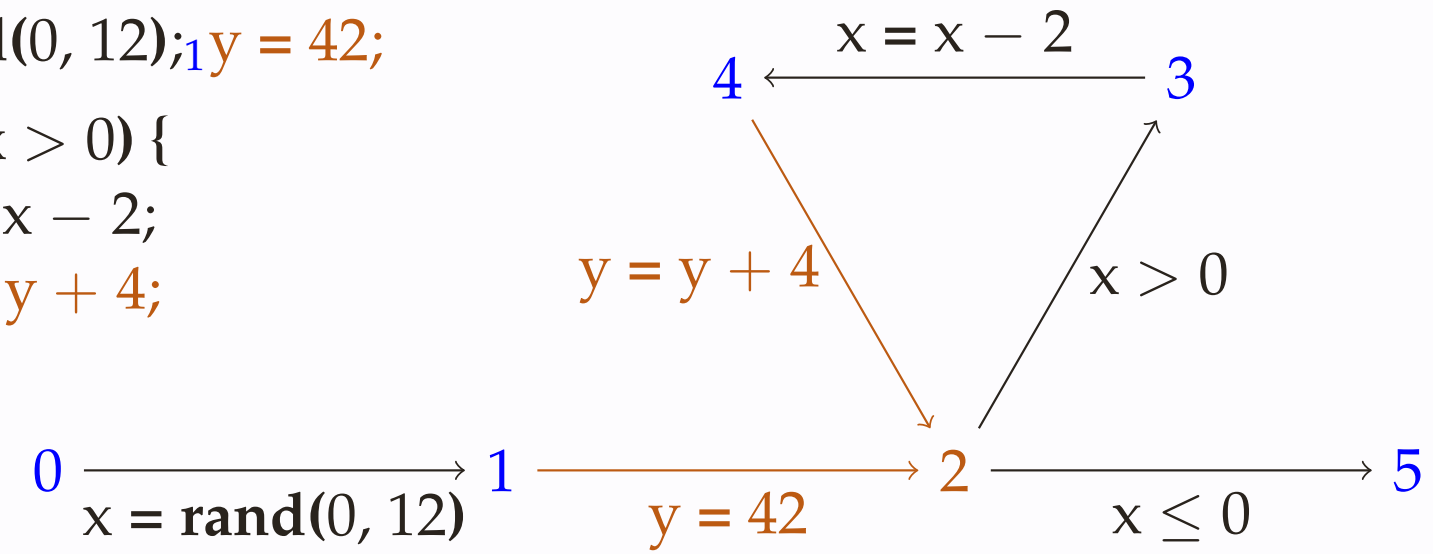
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

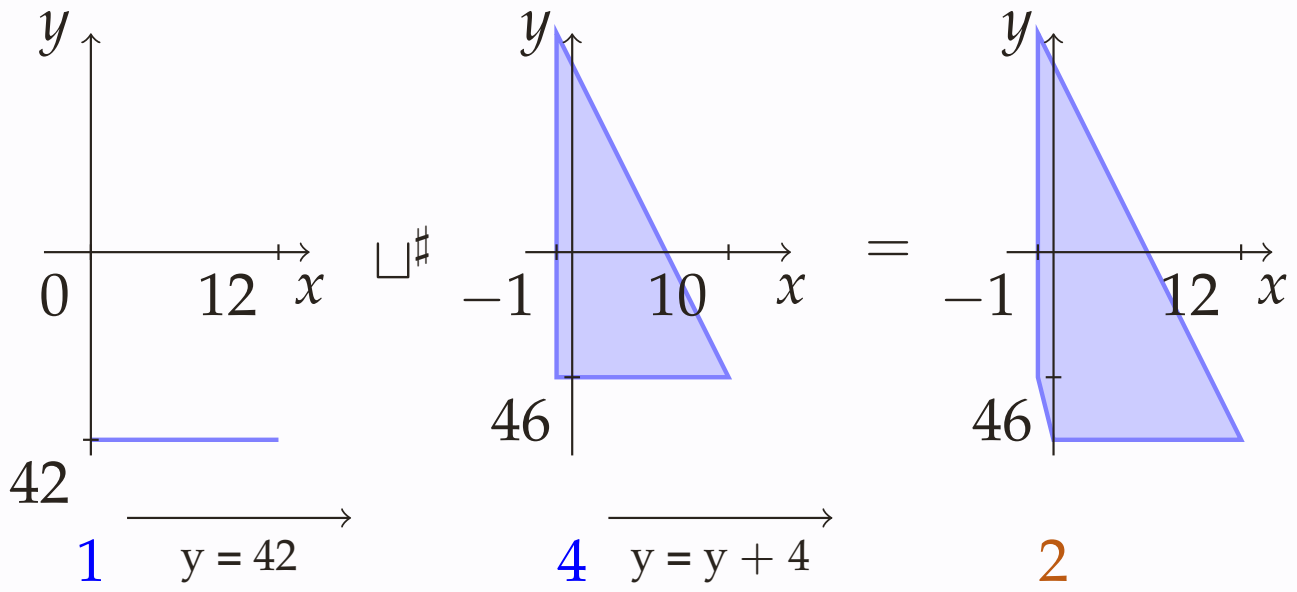




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

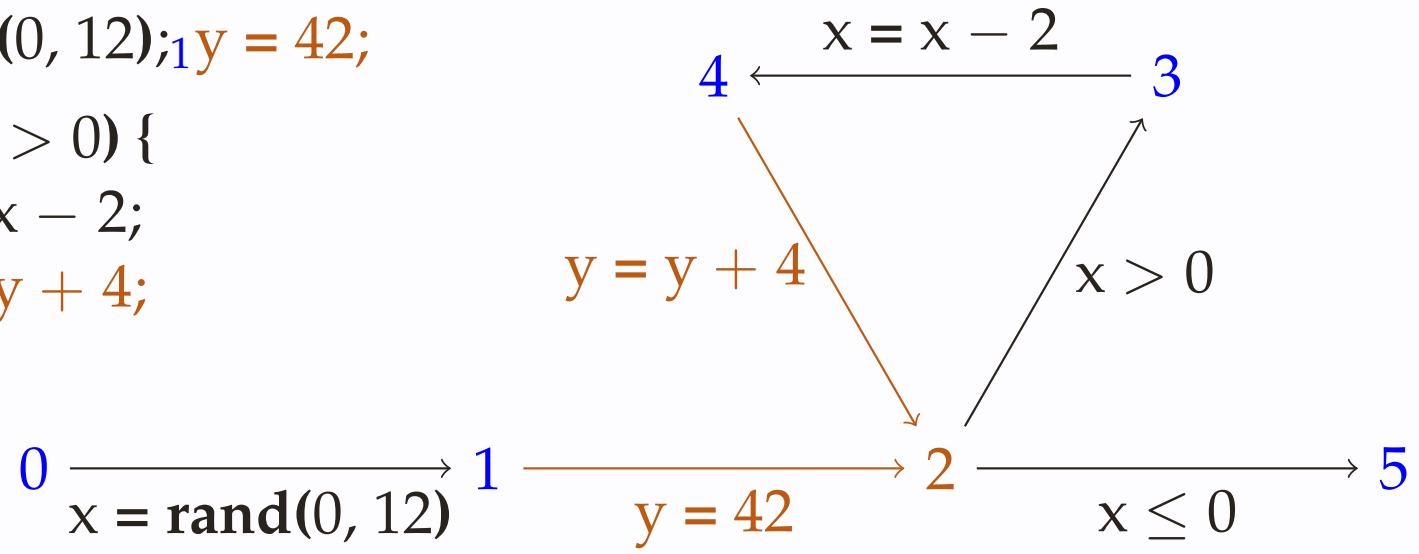
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

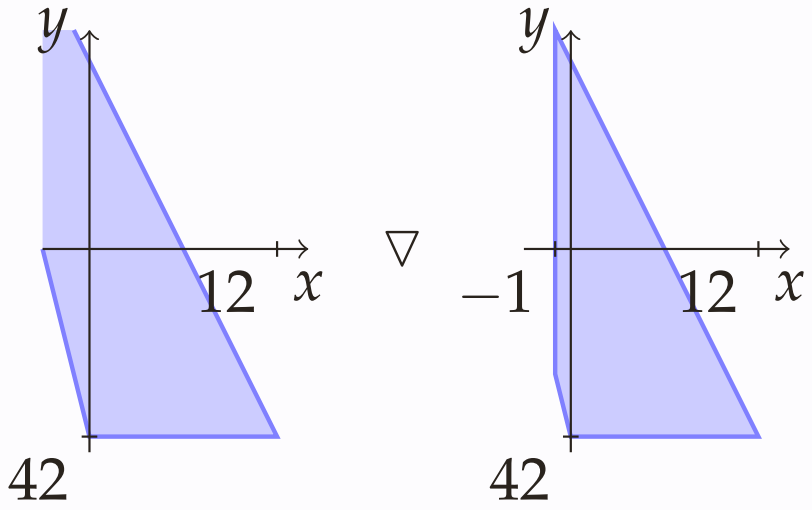




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

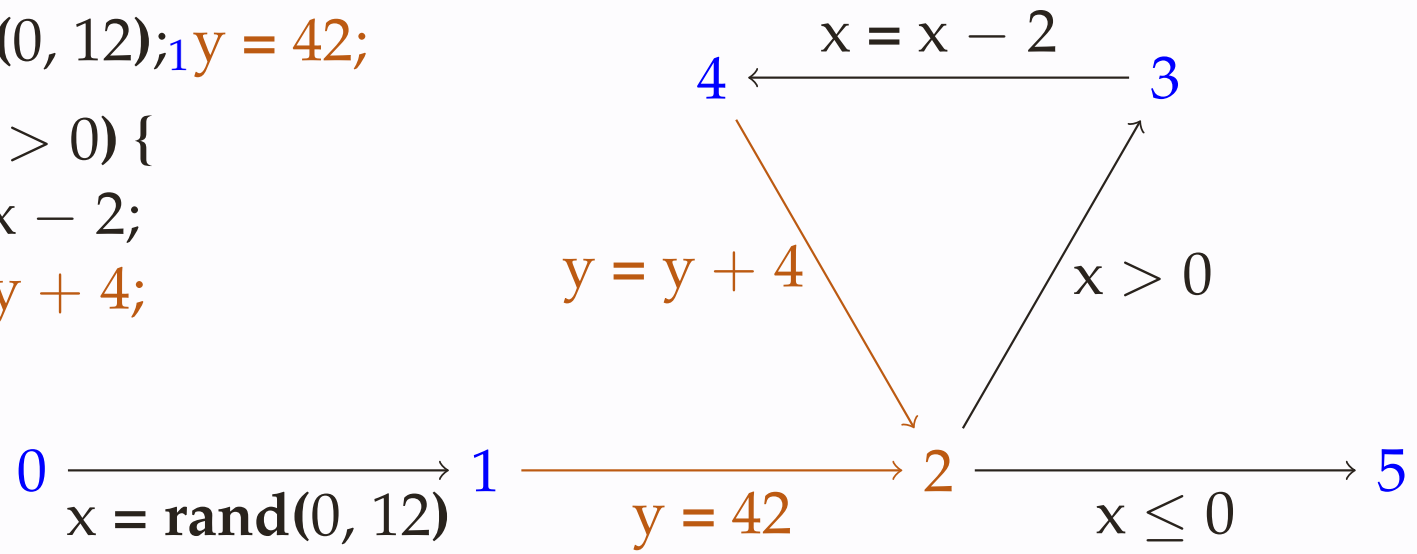
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

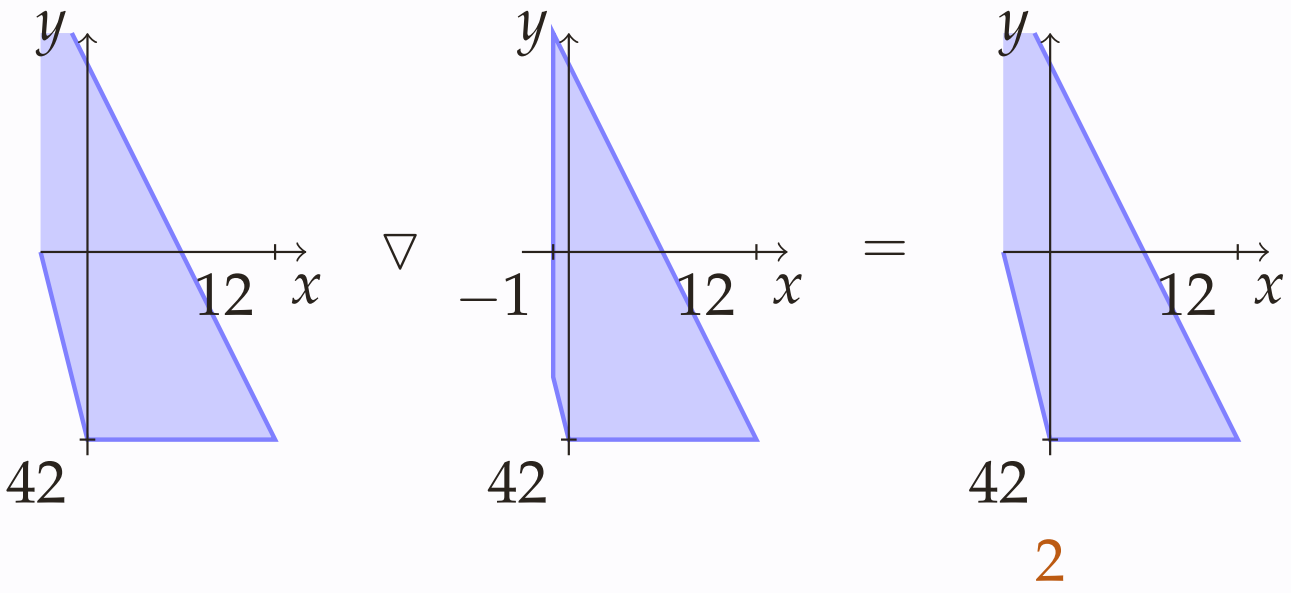




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

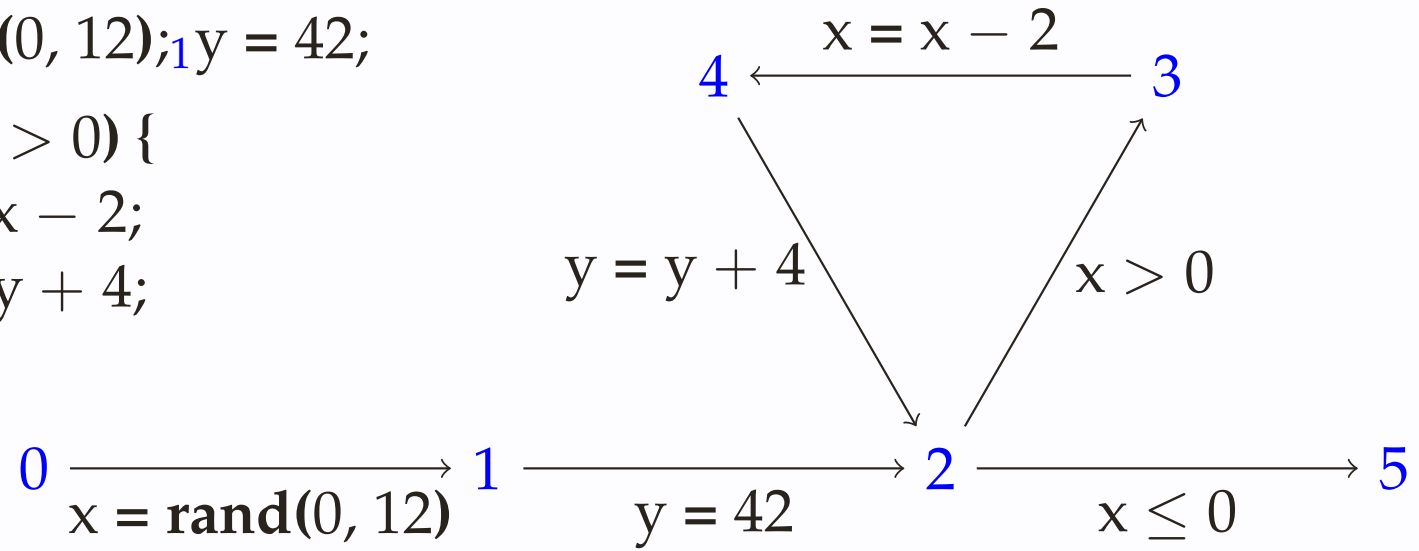
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

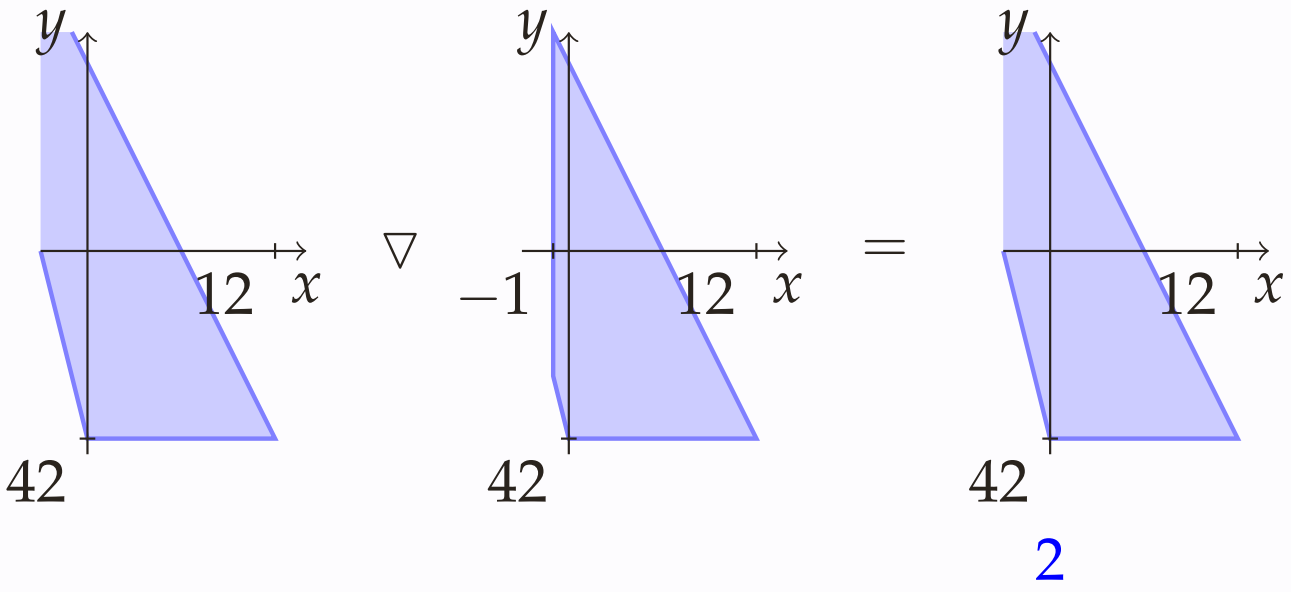




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

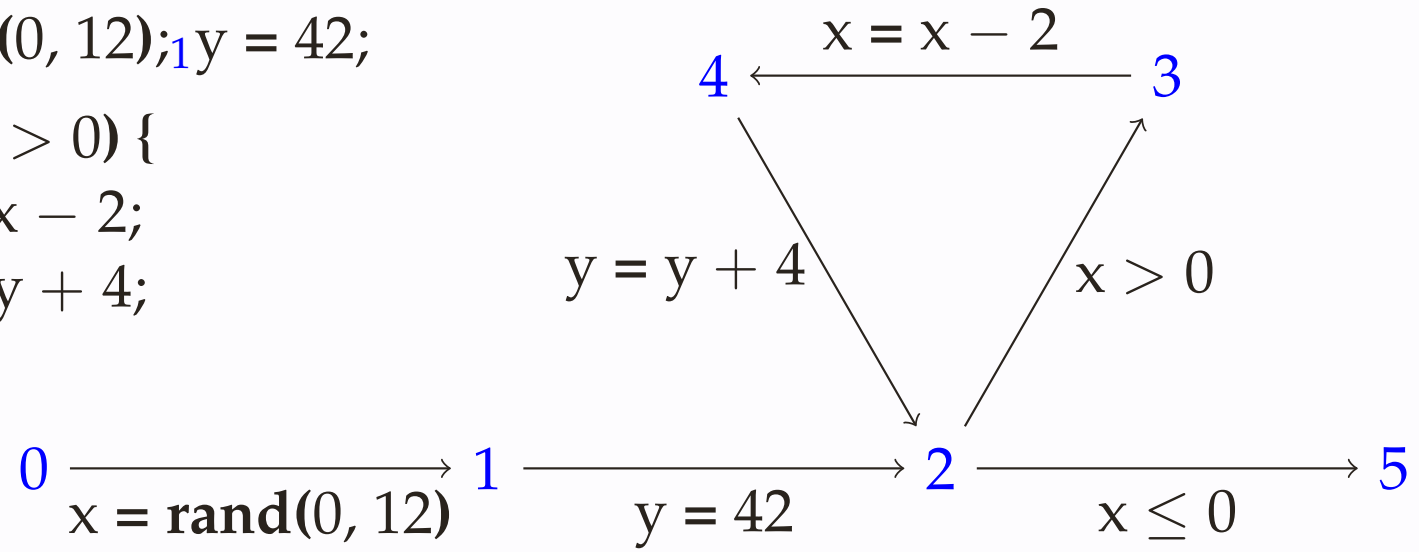


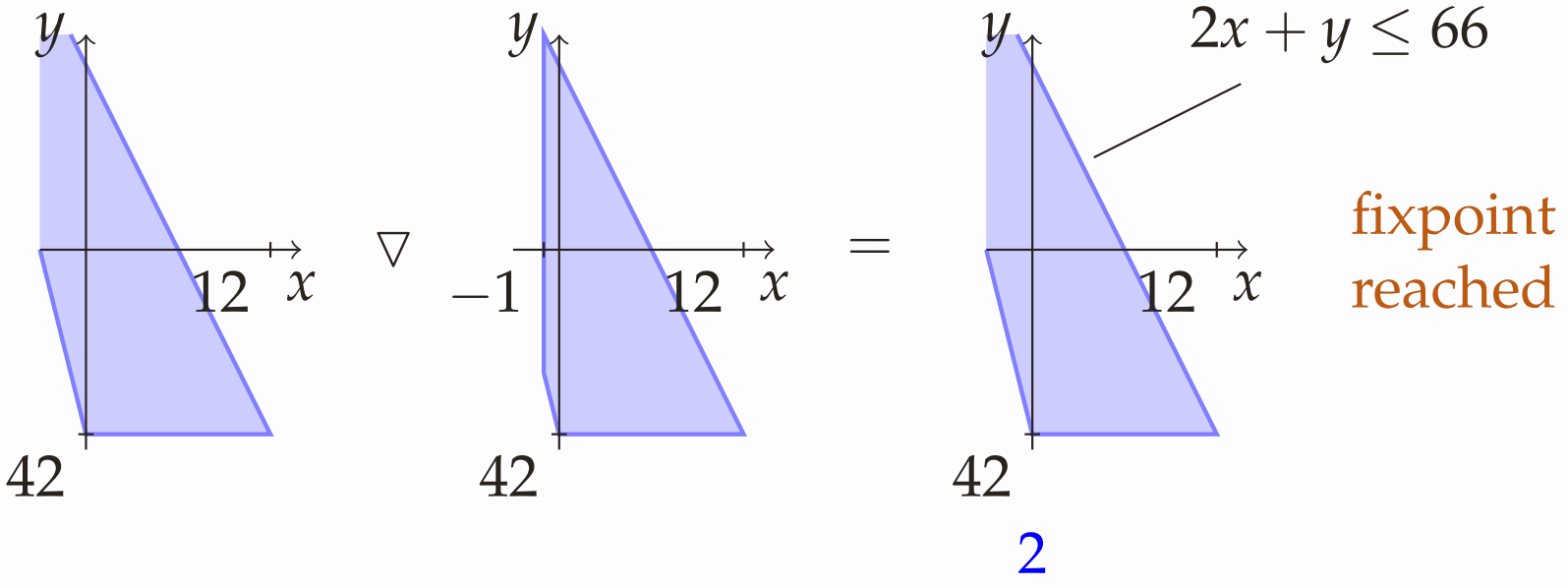


fixpoint reached

EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```





EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

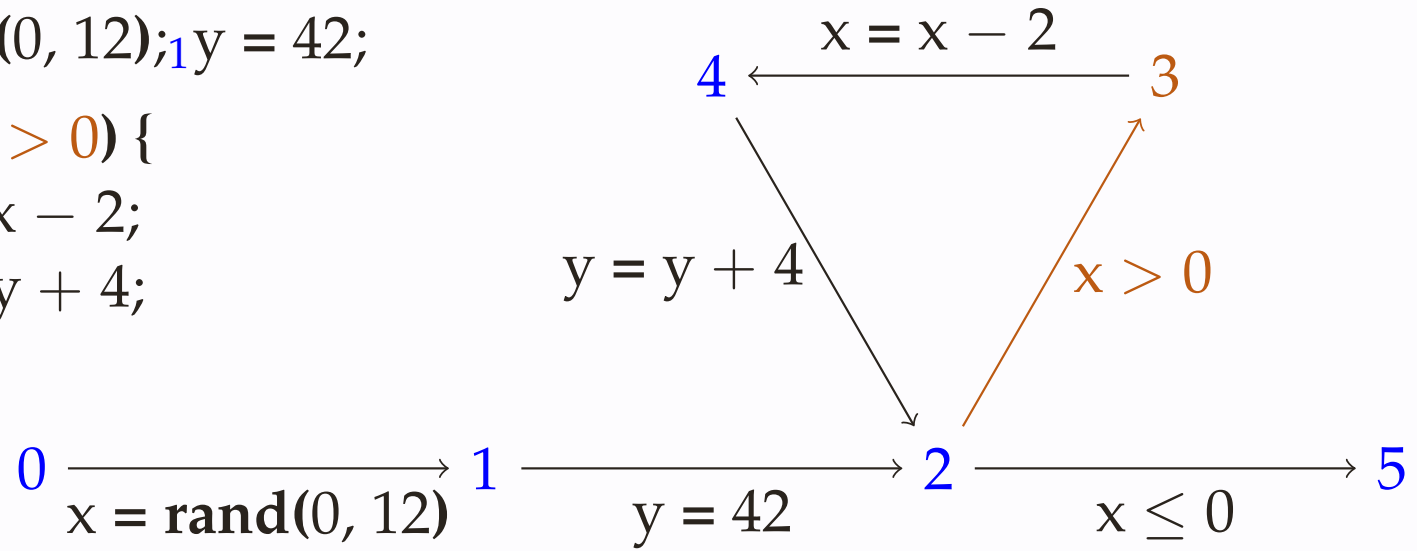
$_0$x = **rand**(0, 12);$_1$y = 42;

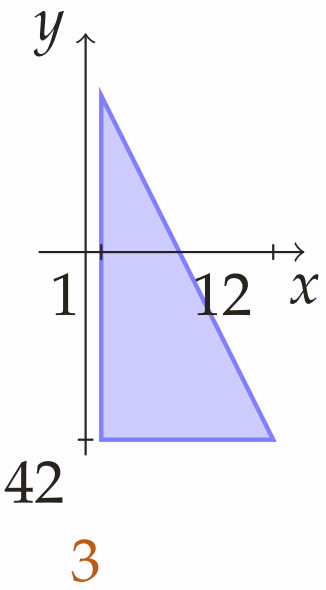
while $_2$(x > 0) {

  $_3$x = x − 2;

  $_4$y = y + 4;

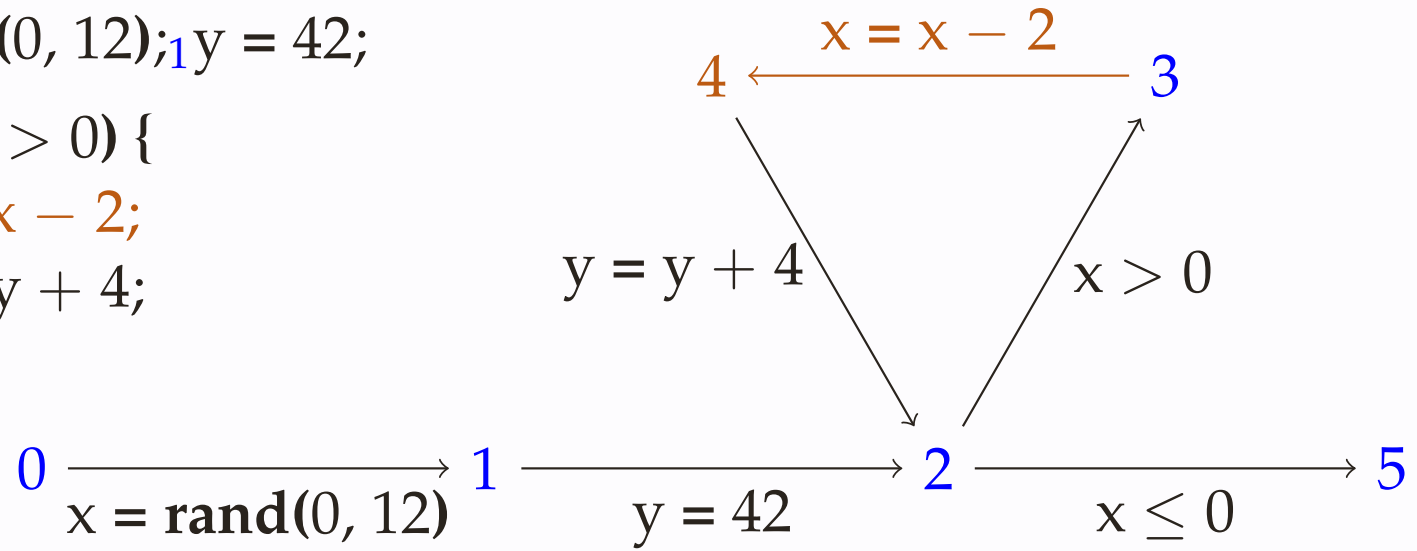
}$_5$

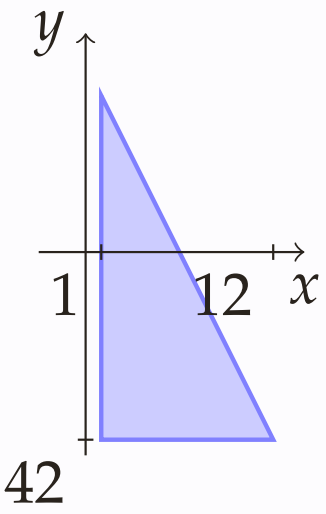




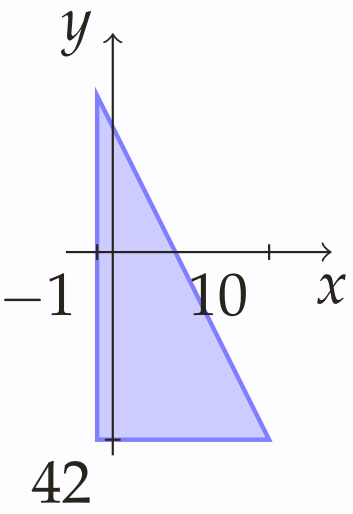
EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```





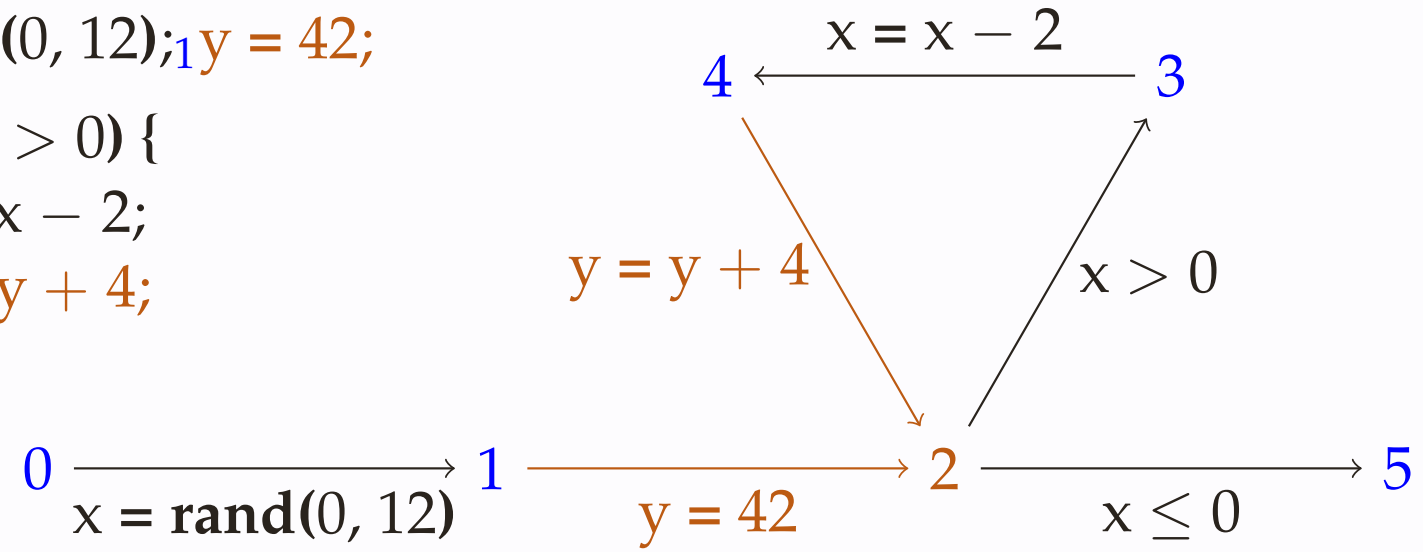
3

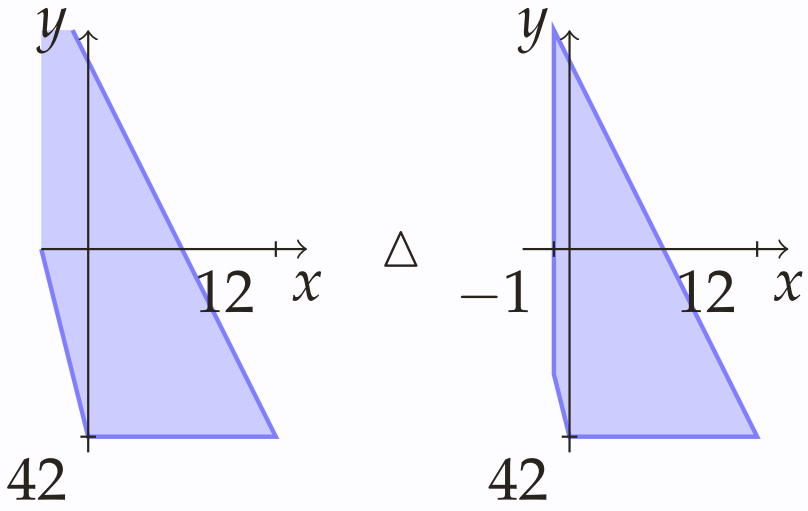


4

Example: a simple polyhedral analysis on integers

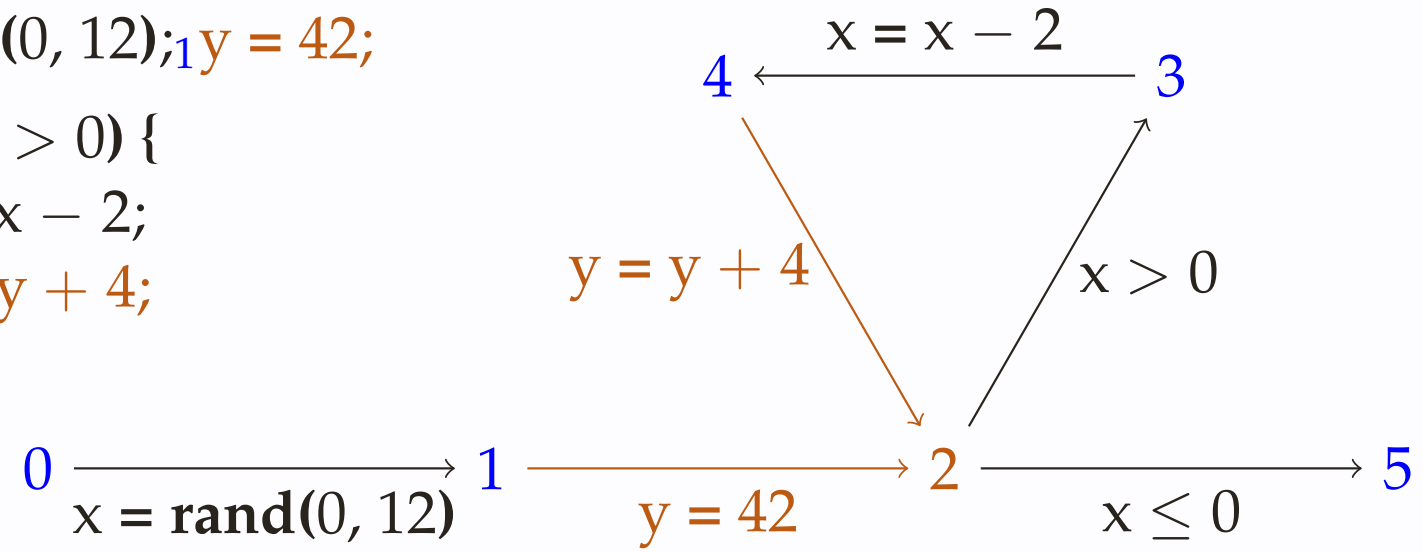
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

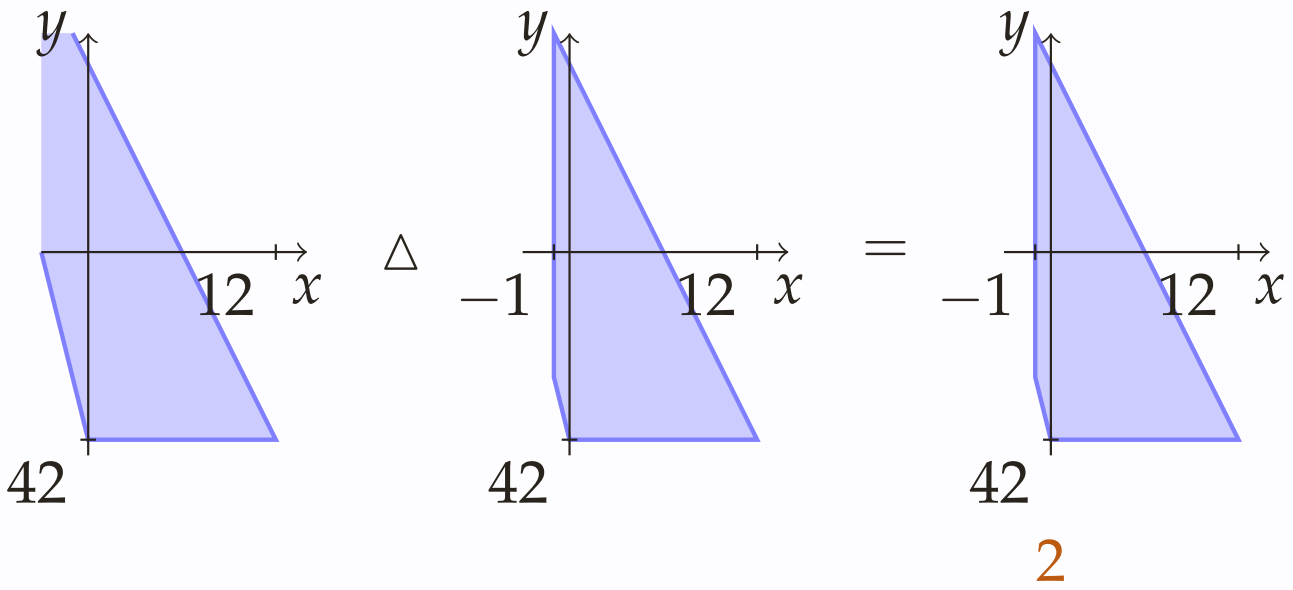




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

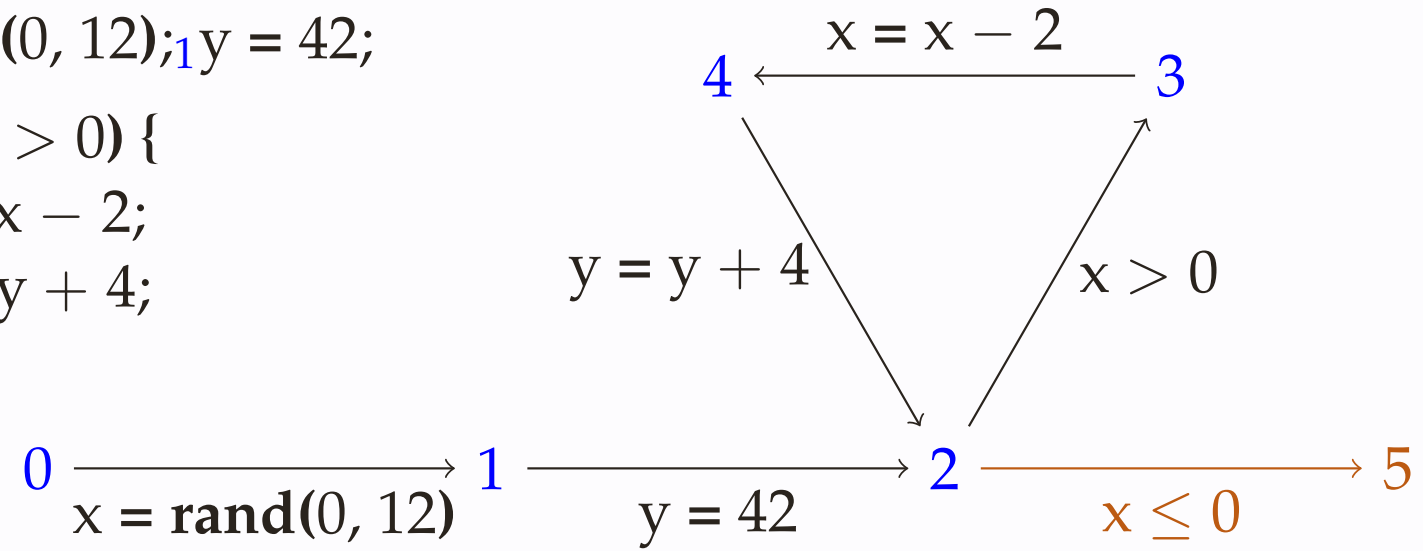
```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```

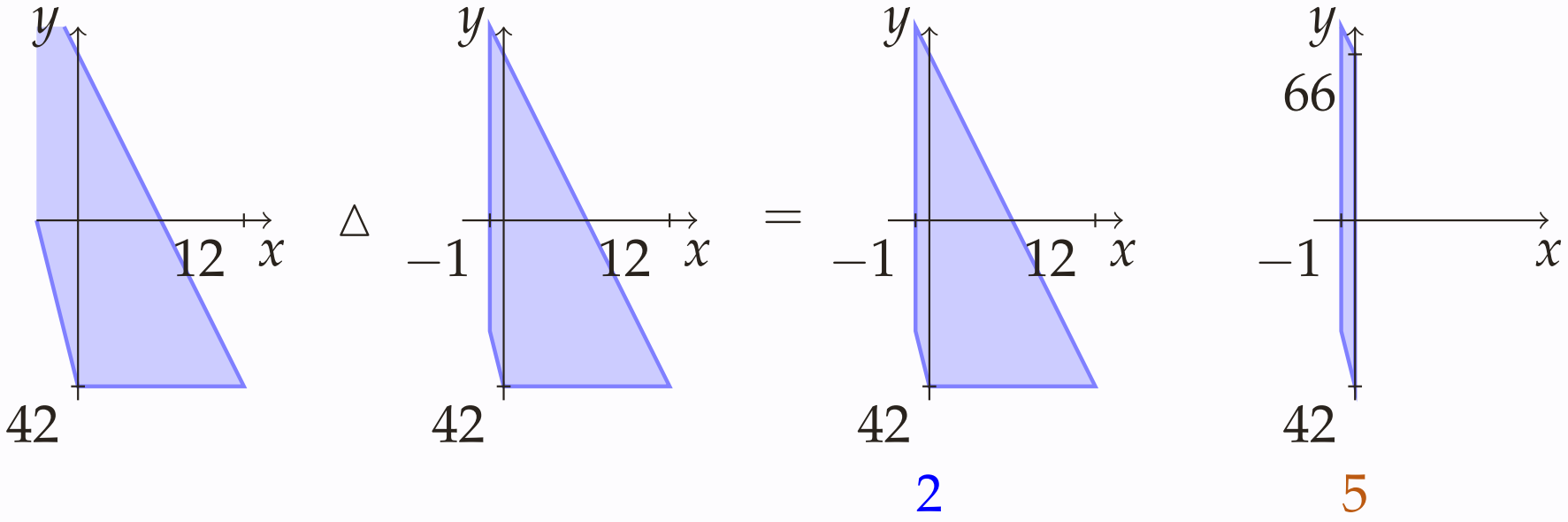




EXAMPLE: A SIMPLE POLYHEDRAL ANALYSIS ON INTEGERS

```
0x = rand(0, 12);1y = 42;
while 2(x > 0) {
    3x = x − 2;
    4y = y + 4;
}5
```





Contents

Deductive method

- Mainly developped for reasoning about imperative code (rather than Synchronous Languages)
- Notion of contract: Precondition, Postcondition
  Hoare triple $\{Pre\}code\{Post\}$
- Symbolic algorithm, by induction of the program instruction
  $\Longrightarrow$ compute the weakest precondition that, when satisfied, guaranty to obtain *Post* after executing Code: $WP(Code, Post)$.
- Proving the contract $\equiv$ prove Pre $\Longrightarrow$ WP(Code, Post)$

Deductive method

- Mainly developped for reasoning about imperative code (rather than Synchronous Languages)
- Notion of contract: Precondition, Postcondition
  Hoare triple $\{Pre\}code\{Post\}$
- Symbolic algorithm, by induction of the program instruction
  $\Longrightarrow$ compute the weakest precondition that, when satisfied, guaranty to obtain *Post* after executing Code: $WP(Code, Post)$.
- Proving the contract $\equiv$ prove Pre $\Longrightarrow$ WP(Code, Post)$

Tools axiomatize (ie. encode in SMT predicates) all the semantics of the considered language, eg. C:

- memory model (overflow in array index may reach other values)
- may use various backend solver to discharge proof objectives

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
x := 1;
$\{ x=1 \wedge n \geq 0 \}$
y := n;
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    x := x × y;
    $\{ x \times (y- 1)! = n! \wedge (y- 1) \geq 0 \}$
    y := y - 1
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ \mathbf{n \geq 0} \}$
$x := 1;$
$\{ x=1 \wedge n \geq 0 \}$
$y := n;$
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    $x := x \times y;$
    $\{ x \times (y- 1)! = n! \wedge (y- 1) \geq 0 \}$
    $y := y - 1$
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ \mathbf{x = n!} \}$

Deductive method - Example of Hoare annotations

$\{ \mathbf{n \geq 0} \}$
$x := 1;$
$\{ x=1 \wedge n \geq 0 \}$
$y := n;$
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    $x := x \times y;$
    $\{ x \times (y- 1)! = n! \wedge (y- 1) \geq 0 \}$
    $y := y - 1$
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ \mathbf{x \times y! = n! \wedge y \geq 0 \wedge y = 0} \}$
$\{ \mathbf{x = n!} \}$

Deductive method - Example of Hoare annotations

$\{ \mathbf{n \geq 0} \}$
$x := 1;$
$\{ \mathbf{x=1 \wedge n \geq 0} \}$
$y := n;$
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    $x := x \times y;$
    $\{ x \times (y - 1)! = n! \wedge (y - 1) \geq 0 \}$
    $y := y - 1$
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
x := 1;
$\{ \mathbf{x=1} \wedge \mathbf{n} \geq \mathbf{0} \}$
y := n;
$\{ \mathbf{x=1} \wedge \mathbf{n} \geq \mathbf{0} \wedge \mathbf{y} = \mathbf{n} \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    x := x × y;
    $\{ x \times (y - 1)! = n! \wedge (y - 1) \geq 0 \}$
    y := y - 1
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
x := 1;
$\{ x=1 \wedge n \geq 0 \}$
y := n;
$\{ \mathbf{x=1 \wedge n \geq 0 \wedge y = n} \}$
$\{ \mathbf{x \times y! = n! \wedge y \geq 0} \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    x := x × y;
    $\{ x \times (y- 1)! = n! \wedge (y- 1) \geq 0 \}$
    y := y - 1
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ \mathbf{x \times y! = n! \wedge y \geq 0 \wedge y = 0} \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
x := 1;
$\{ x=1 \wedge n \geq 0 \}$
y := n;
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ \mathbf{x \times y! = n! \wedge y \geq 0} \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    x := x × y;
    $\{ x \times (y- 1)! = n! \wedge (y- 1) \geq 0 \}$
    y := y - 1
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ \mathbf{x \times y! = n! \wedge y \geq 0 \wedge y = 0} \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
$x := 1;$
$\{ x=1 \wedge n \geq 0 \}$
$y := n;$
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ \mathbf{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0} \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    $x := x \times y;$
    $\{ x \times (y- 1)! = n! \wedge (y- 1) \geq 0 \}$
    $y := y - 1$
$\{ \mathbf{x \times y! = n! \wedge y \geq 0} \}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
$x := 1;$
$\{ x=1 \wedge n \geq 0 \}$
$y := n;$
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ \mathbf{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0} \}$
    $\{ \mathbf{x \times y! = n! \wedge (y - 1) \geq 0} \}$
    $x := x \times y;$
    $\{ x \times (y-1)! = n! \wedge (y-1) \geq 0 \}$
    $y := y - 1$
$\{ \mathbf{x \times y! = n! \wedge y \geq 0} \}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
$x := 1;$
$\{ x=1 \wedge n \geq 0 \}$
$y := n;$
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\mathbf{\{ x \times y! = n! \wedge (y - 1) \geq 0 \}}$
    $x := x \times y;$
    $\mathbf{\{ x \times (y - 1)! = n! \wedge (y - 1) \geq 0 \}}$
    $y := y - 1$
$\{ x \times y! = n! \wedge y \geq 0 \}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ x = n! \}$

Deductive method - Example of Hoare annotations

$\{ n \geq 0 \}$
$x := 1;$
$\{ x=1 \wedge n \geq 0 \}$
$y := n;$
$\{ x=1 \wedge n \geq 0 \wedge y = n \}$
$\{ x \times y! = n! \wedge y \geq 0 \}$
while $y \neq 0$ do
    $\{ x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \}$
    $\{ x \times y! = n! \wedge (y - 1) \geq 0 \}$
    $x := x \times y;$
    $\mathbf{\{ x \times (y - 1)! = n! \wedge (y - 1) \geq 0 \}}$
    $y := y - 1$
$\mathbf{\{ x \times y! = n! \wedge y \geq 0 \}}$
done;
$\{ x \times y! = n! \wedge y \geq 0 \wedge y = 0 \}$
$\{ x = n! \}$

Deductive method - WP example

$\{ n \geq 0 \}$
x := 1;
y := n;
while $y \neq 0$ do
    // Invariant $I \triangleq x \times y! = n! \wedge y \geq 0$
    x := $x \times y$;
    y := y - 1
done;
$\{ x = n! \}$

WP(loop_body, $\{ x = n! \}$) = Inv
WP(fact, $\{ x = n! \}$) = WP(assign, Inv) = $1 \times n! = n! \wedge n \geq 0 = n \geq 0$

Proof Objective: $n \geq 0 \implies$ WP(fact, $\{ x = n! \}$)

Deductive method - Stability analyis

System open-loop stable: existence of a Lyapunov function (LF)

- Quadratic LF supplied by the user
- Automatic generation of quadratic invariants annotations in the code
- Use of specific solver to discharge the generated proof obligations

```
//@ in_ellipsoidQ(QMat_11,vect_of_1_scalar(Sum4));
{
Sum4 = discrete_timeg_no_plant_08b_y -
discrete_timeg_no_plant_08b_yd;
}
//@ ensures in_ellipsoidQ(QMat_12,vect_of_2_scalar(Sum4,D11));
```

Contents

Dealing with Floating points computation

Floats are not reals!!

```
int i = 0; float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%f\n", x);
```

returns 100958.343750 on my computer.

Dealing with Floating points computation

Floats are not reals!!

```
int i = 0; float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%f\n", x);
```

returns 100958.343750 on my computer.

Float operations are non associative, non distributive, etc.

Dealing with Floating points computation

Floats are not reals!!

```
int i = 0; float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%f\n", x);
```

returns 100958.343750 on my computer.

Float operations are non associative, non distributive, etc.

All presented methods have to be adapted to consider floating point computation:

- dedicated SMT solvers
- adapt the proofs to handle floating point errors
  - eg. to prove that $\forall x; x \leq v$ prove instead that
    - $x \leq v'$
    - $v' + flerr(x) < v$ with $flerr(x)$ the floating point error generated when computing x

Dealing with Floating points computation

Floats are not reals!!

```
int i = 0; float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%f\n", x);
```

returns 100958.343750 on my computer.

Float operations are non associative, non distributive, etc.

All presented methods have to be adapted to consider floating point computation:

- dedicated SMT solvers
- adapt the proofs to handle floating point errors
  - eg. to prove that $\forall x; x \leq v$ prove instead that
    - $x \leq v'$
    - $v' + flerr(x) < v$ with $flerr(x)$ the floating point error generated when computing x

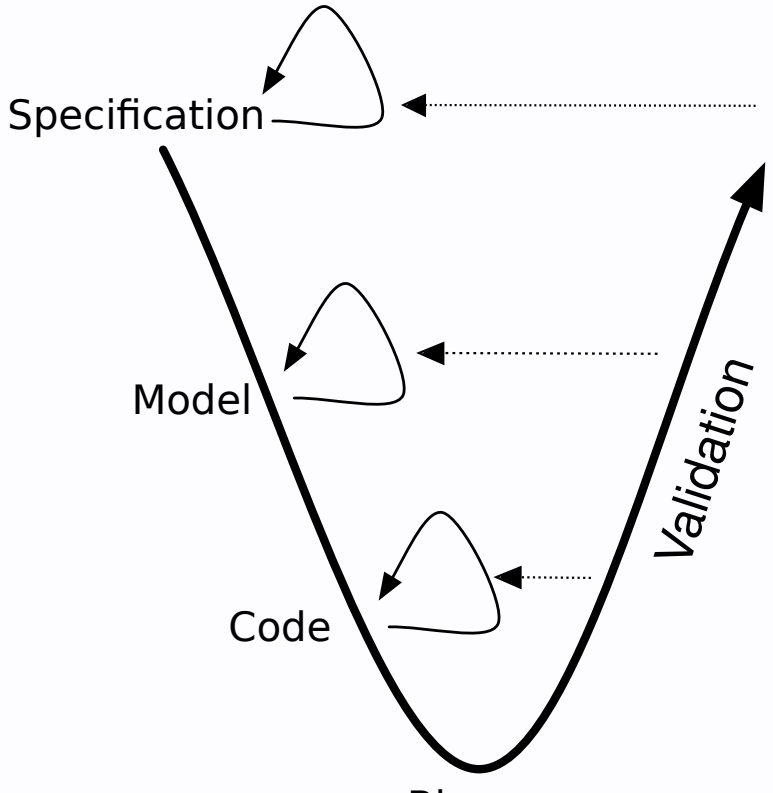
Similar issues at the tool level: they use floats as well!

Contents

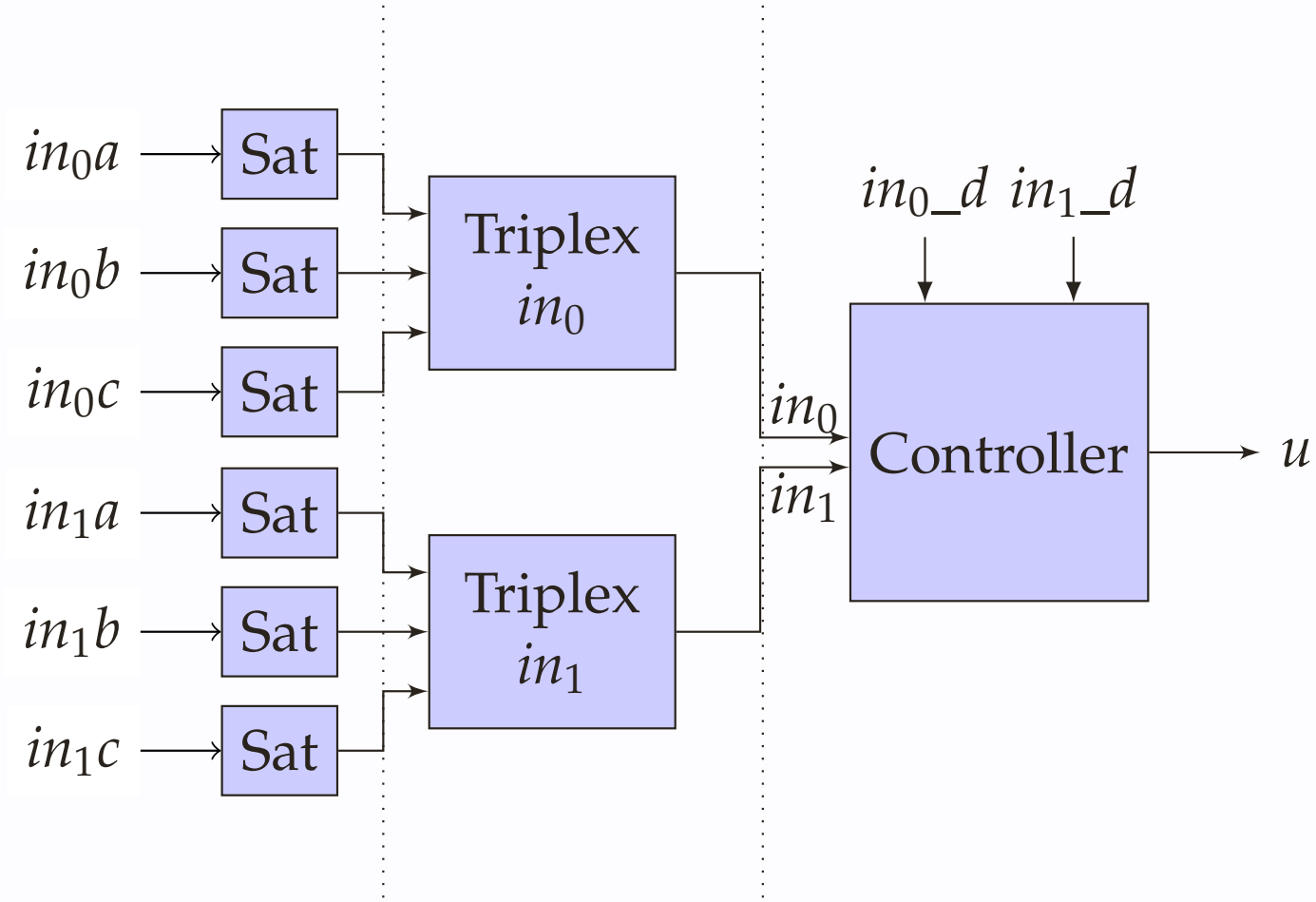
Application of formal method on a simple control systems

In practice formal V& V can be performed at various stages of the development process

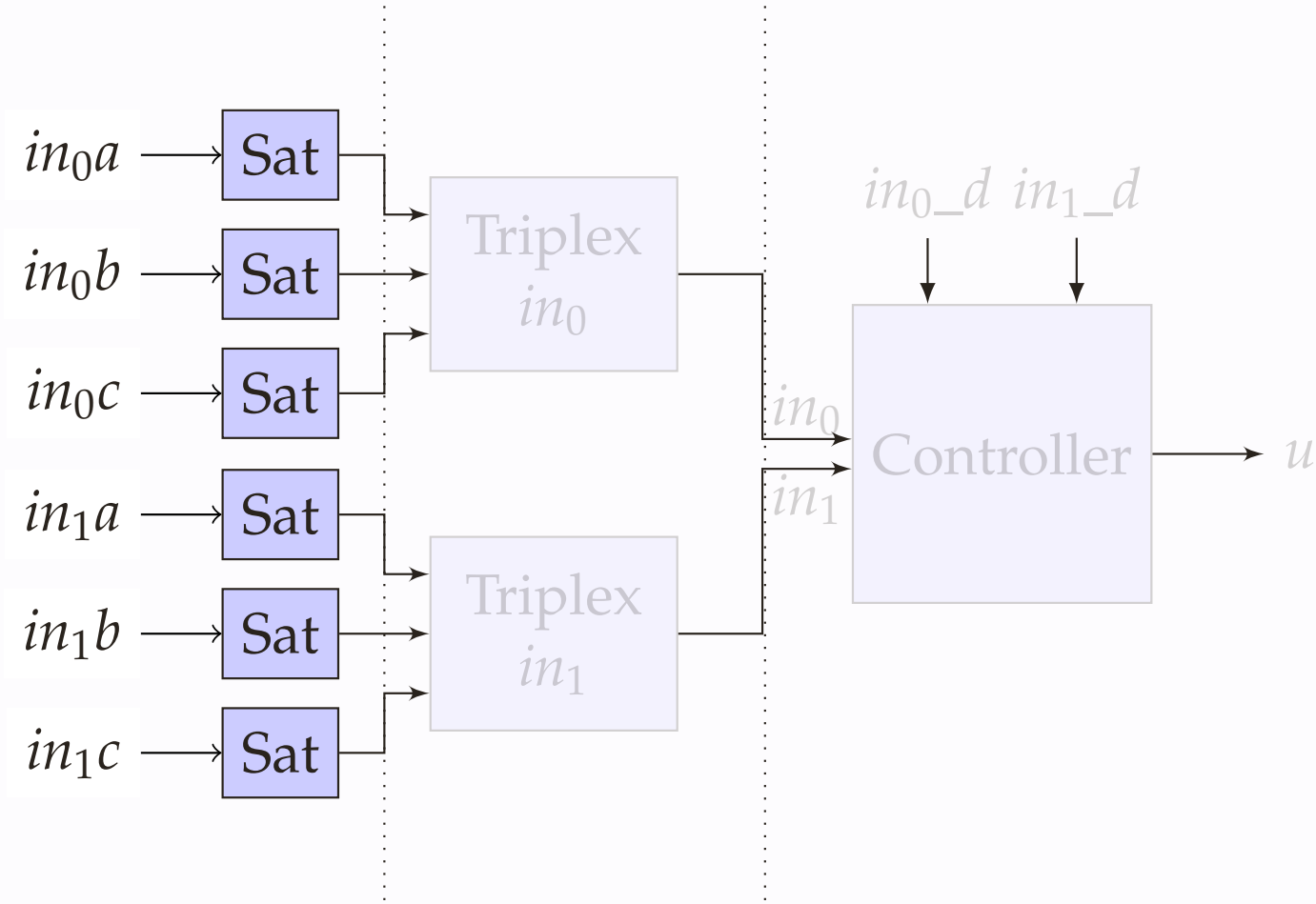
- early at model level on simpler version of the program
- at code level with the added difficulties caused by pointers, heap, stack, RTE.
- it requires the formalization of the specification.



At model level - Combining Analyses

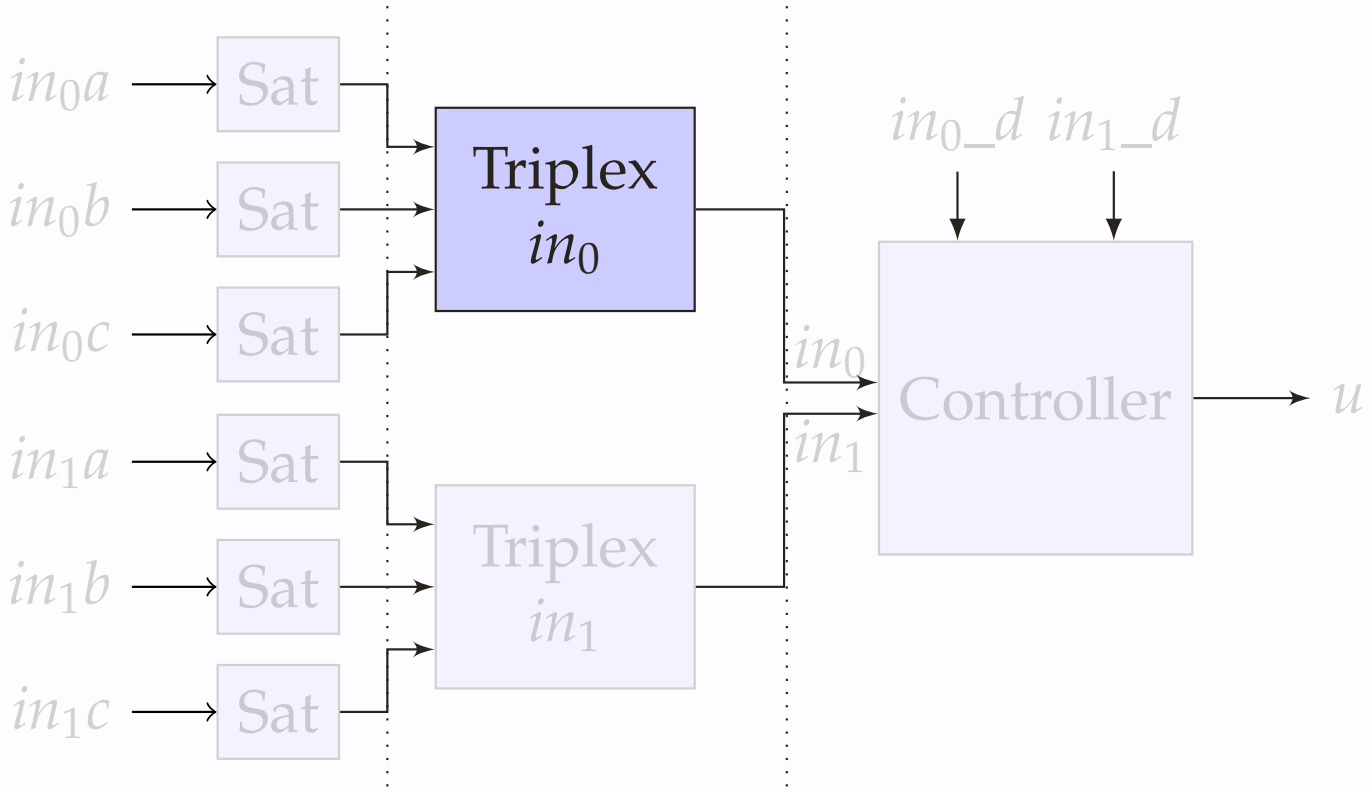


At model level - Basic saturations



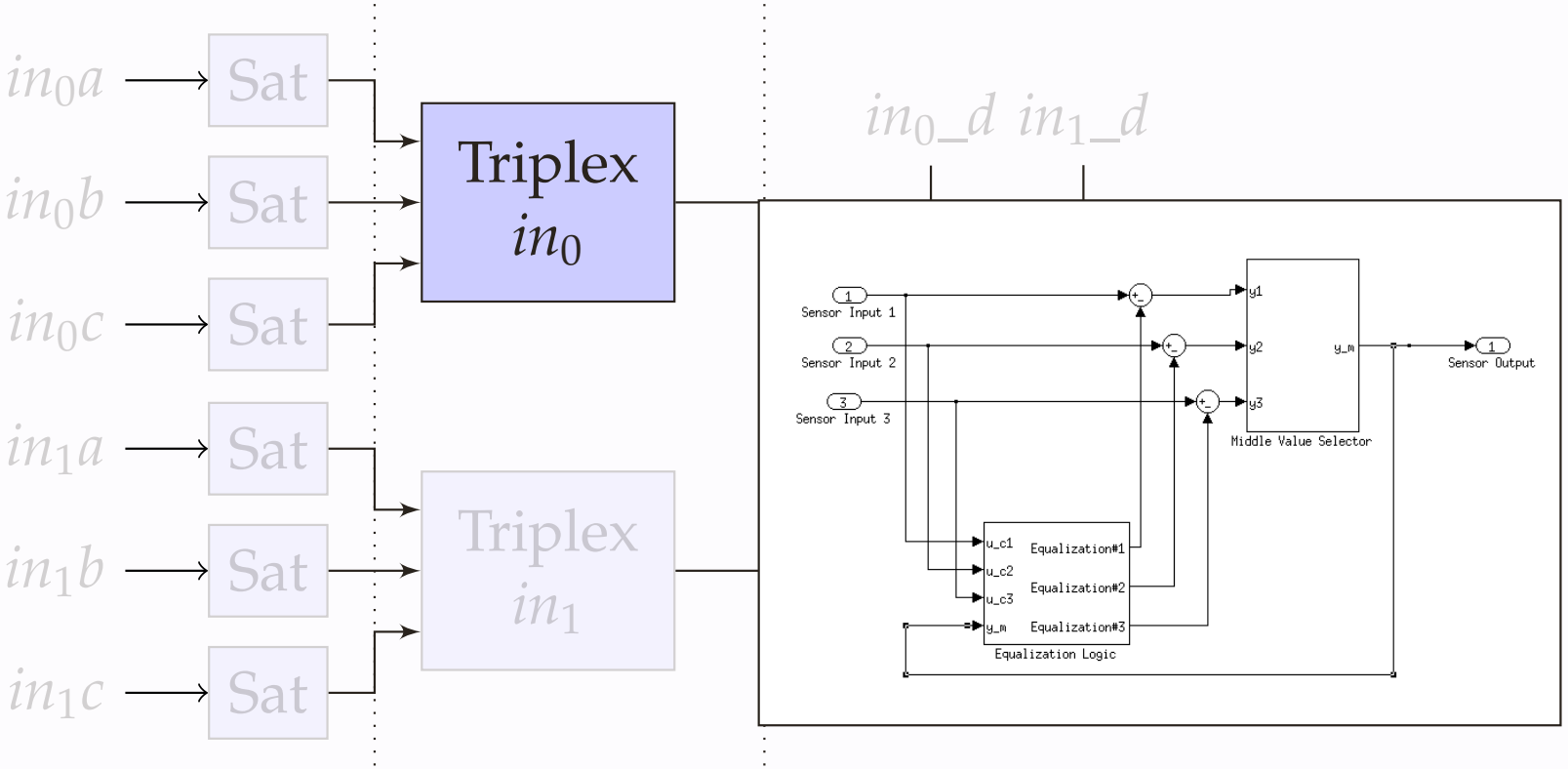
Abstract Interpretation computes a sound bound (1.2) on each ouptut whatever the value of in_xy is.

At model level - Analysis of the triplex voter



Backward analysis applied on each triplex proves the specification BIBO.

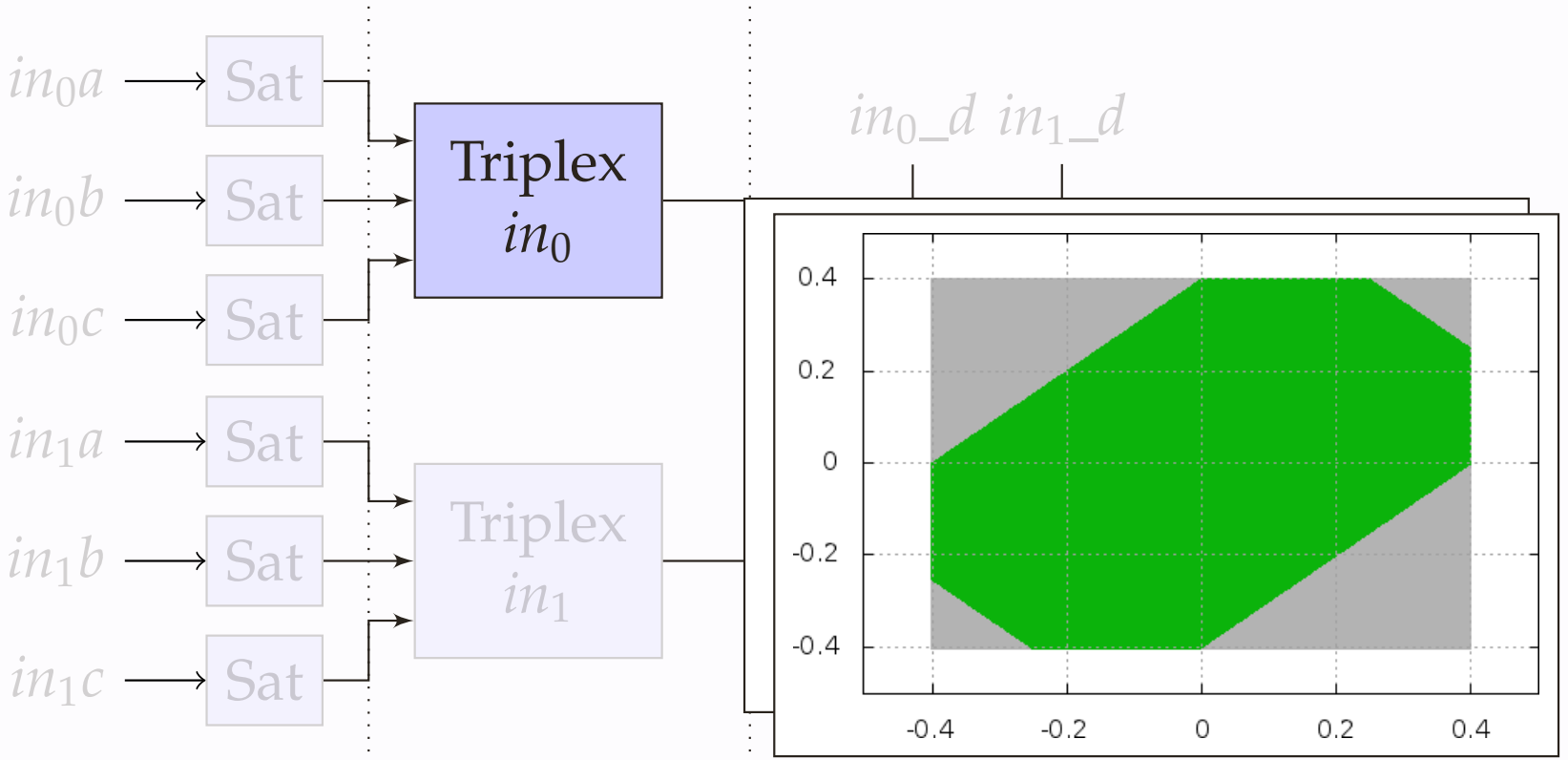
AT MODEL LEVEL - ANALYSIS OF THE TRIPLEX VOTER



Backward analysis applied on each triplex proves the specification BIBO.

$$\forall k \in \mathbb{N},\ |InA_k| \leq a \wedge |InB_k| \leq a \wedge |InC_k| \leq a \implies |Output_k| \leq 3a \wedge$$
$$|EqualizationA_k| \leq 2a \wedge |EqualizationB_k| \leq 2a \wedge |EqualizationC_k| \leq 2a$$

At model level - Analysis of the triplex voter

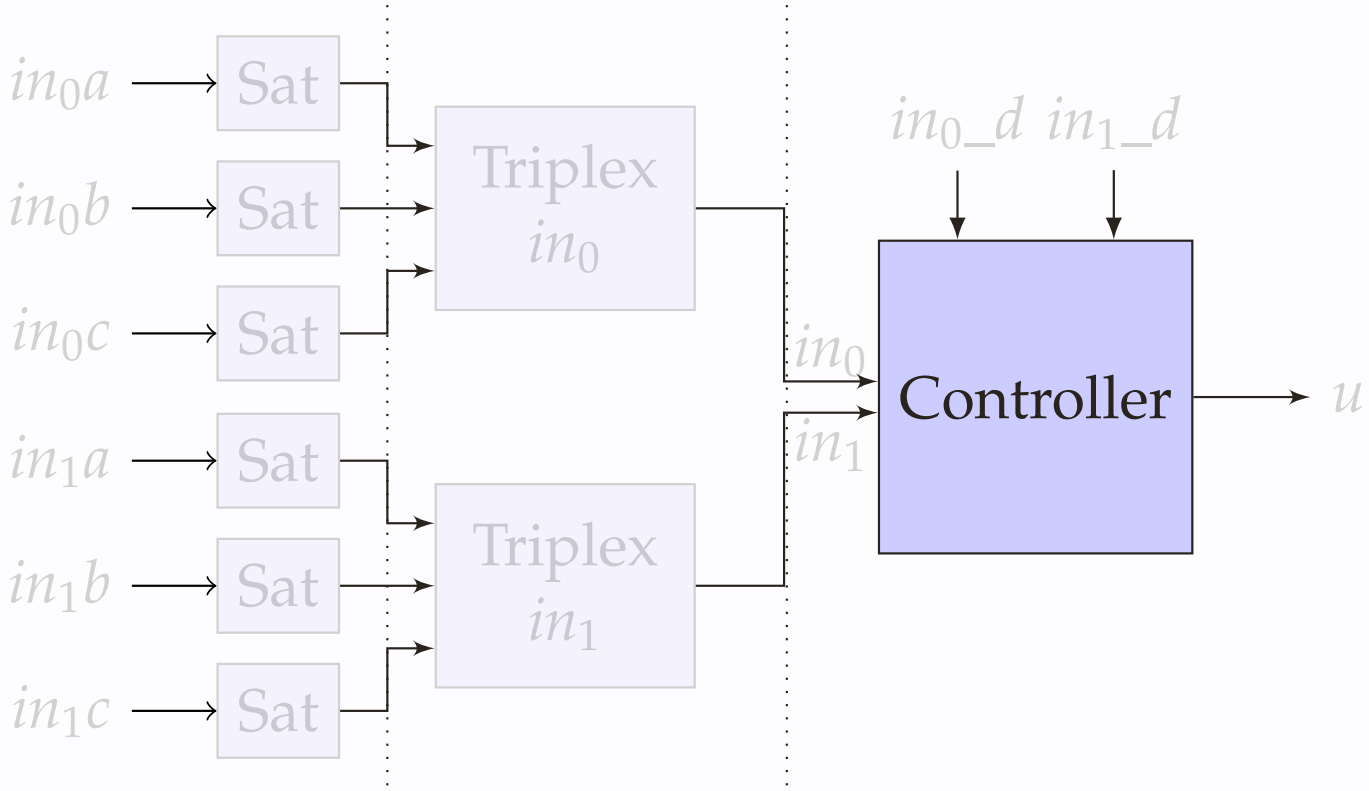


Backward analysis applied on each triplex proves the specification BIBO.

$$\forall k \in \mathbb{N},\ |InA_k| \leq a \wedge |InB_k| \leq a \wedge |InC_k| \leq a \implies |Output_k| \leq 3a \wedge$$
$$|EqualizationA_k| \leq 2a \wedge |EqualizationB_k| \leq 2a \wedge |EqualizationC_k| \leq 2a$$

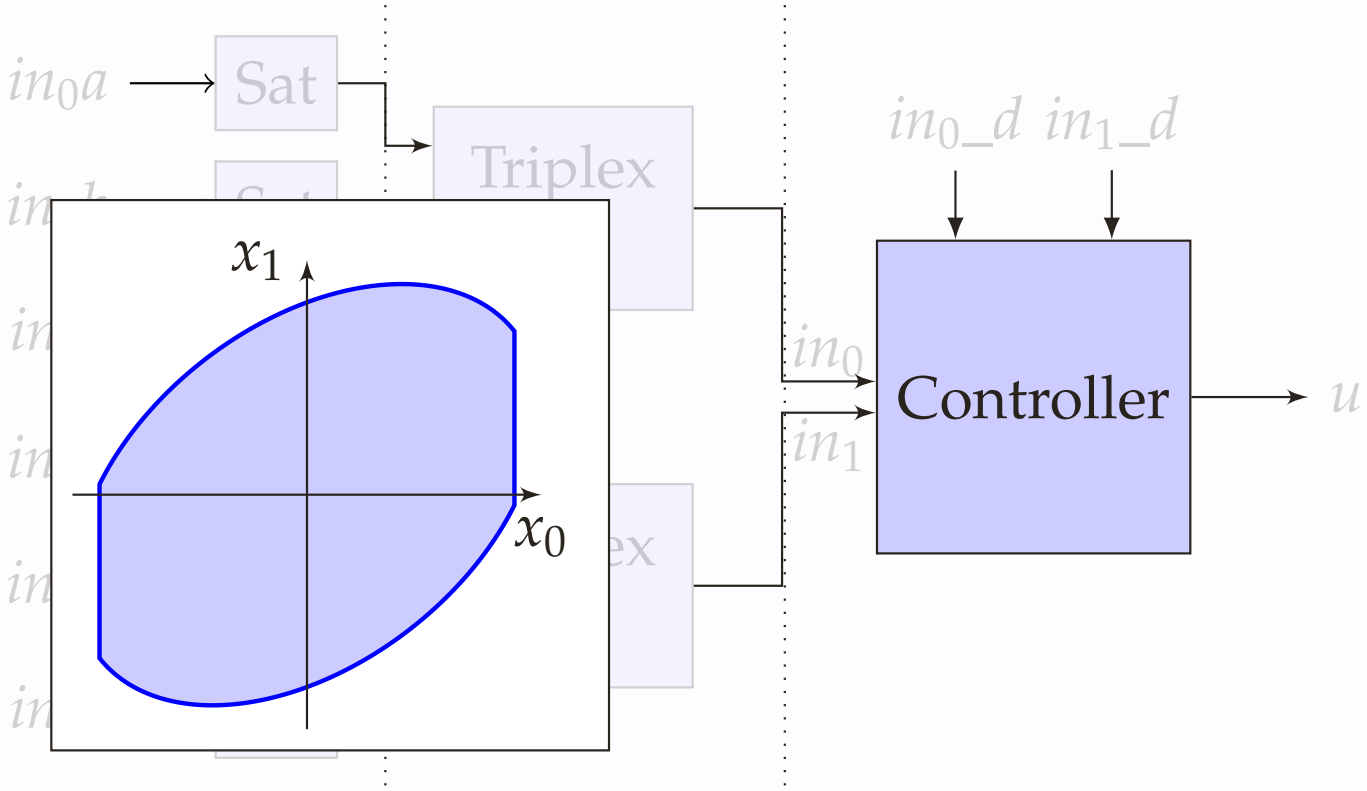
Assuming input is bounded by 1.2, we have output bounded by 3.6.

At model level - Analysis of the controller



Providing a bound on the inputs (3.6) an over-approximation of the output is computed:

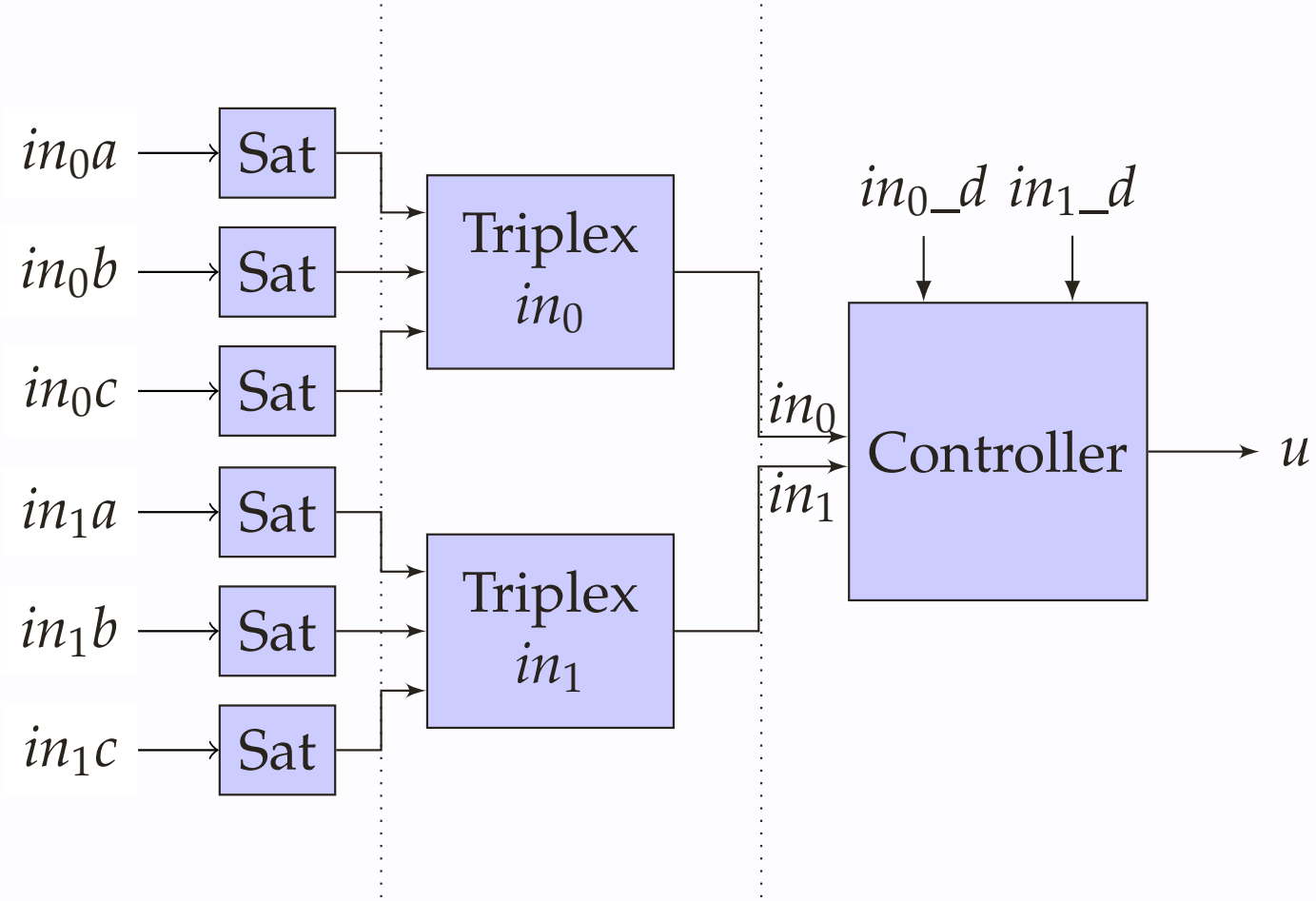
At model level - Analysis of the controller



Providing a bound on the inputs (3.6) an over-approximation of the output is computed:~ $|u| \leq 194.499$.

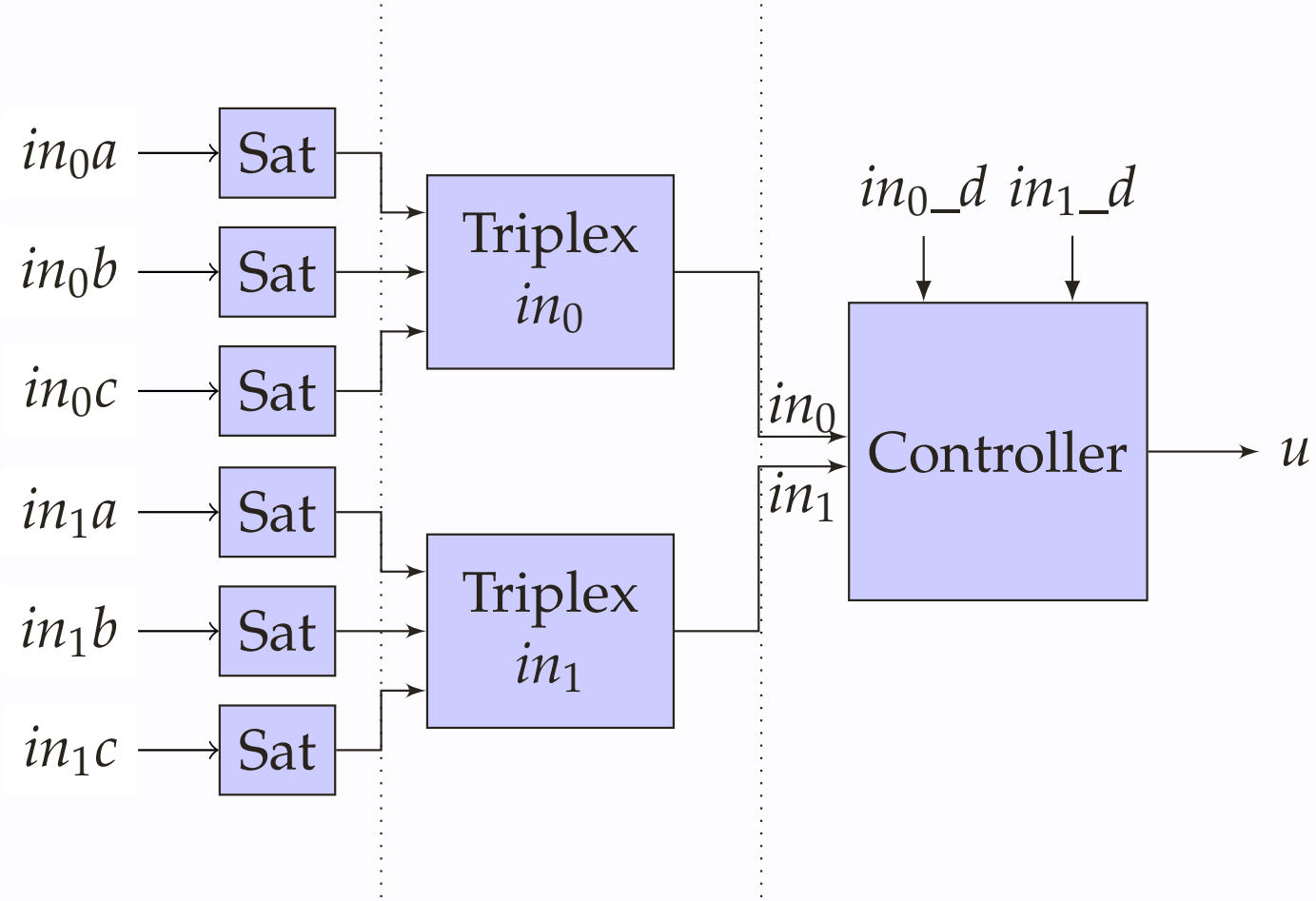
$$0.098\,x_3^2 - 0.224\,x_3\,x_2 + 0.040\,x_3\,x_1 - 0.026\,x_3\,x_0 + 0.141\,x_2^2 - 0.053\,x_2\,x_1 \\ +0.030\,x_2\,x_0 + 0.024\,x_1^2 - 0.017\,x_1\,x_0 + 0.019\,x_0^2 \leq 14.259$$

At model level - Rebuilding the analysis



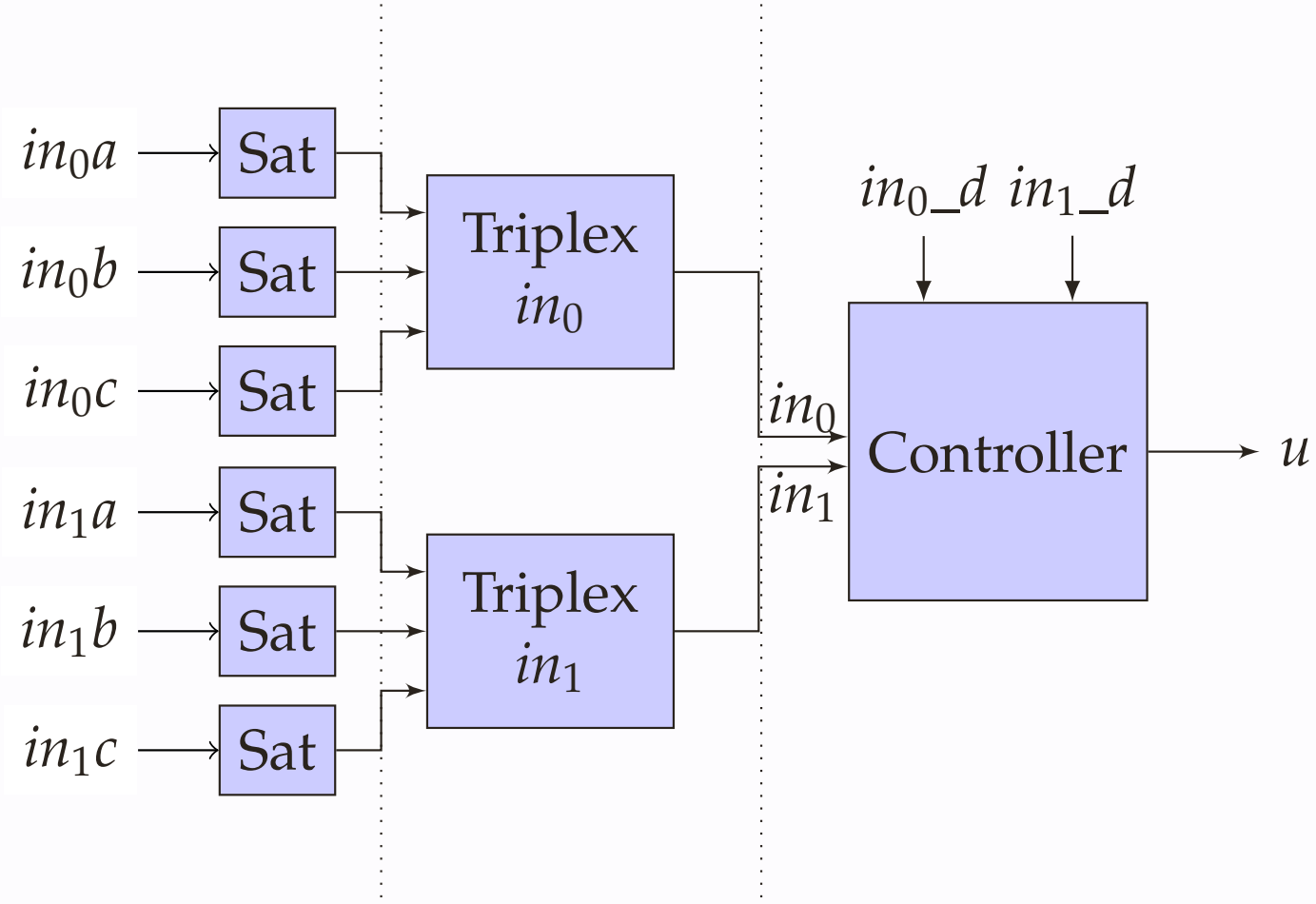
$$]-\infty,+\infty[$$

AT MODEL LEVEL - REBUILDING THE ANALYSIS

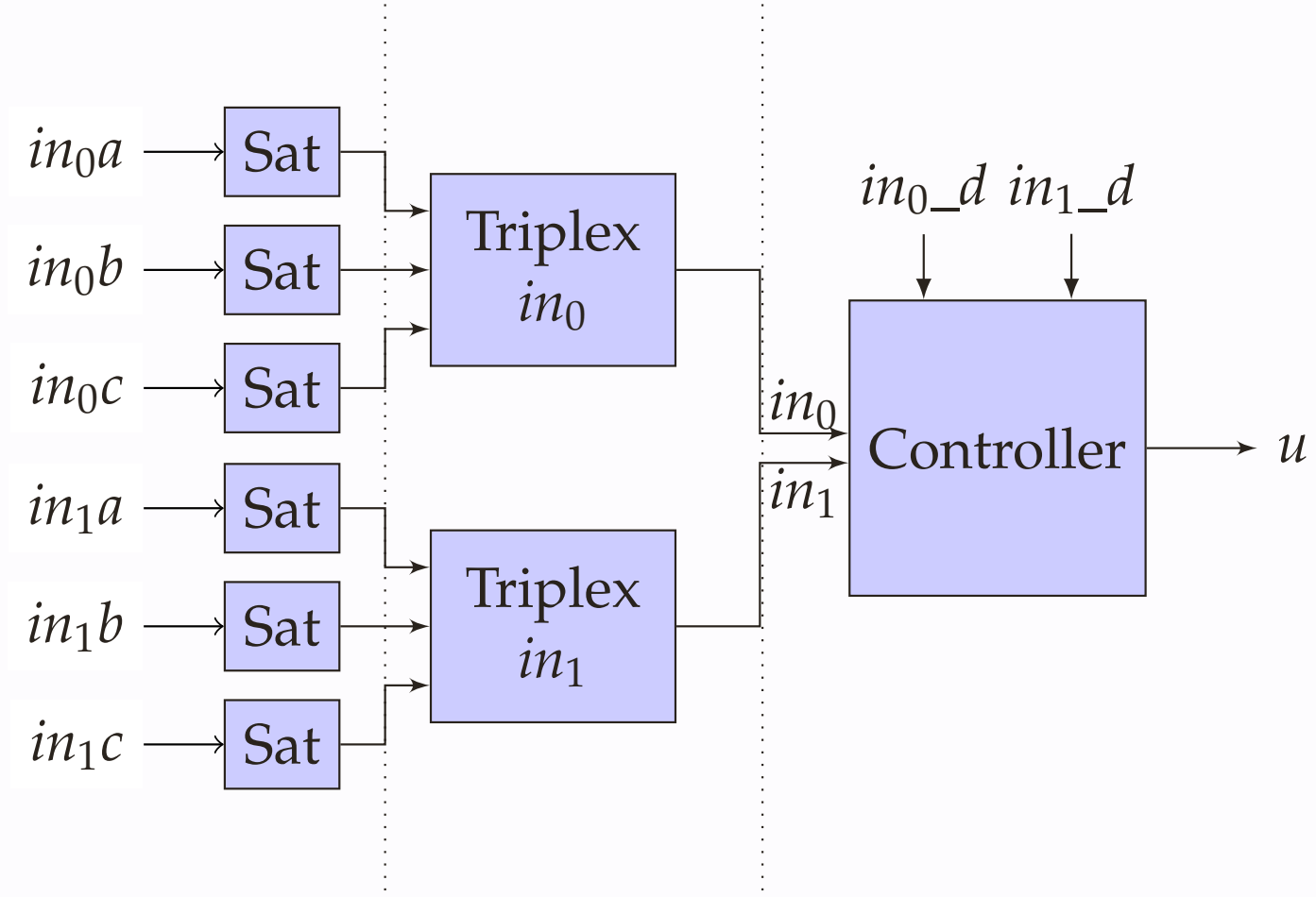


$$]-\infty,+\infty[\qquad 1.2$$

AT MODEL LEVEL - REBUILDING THE ANALYSIS

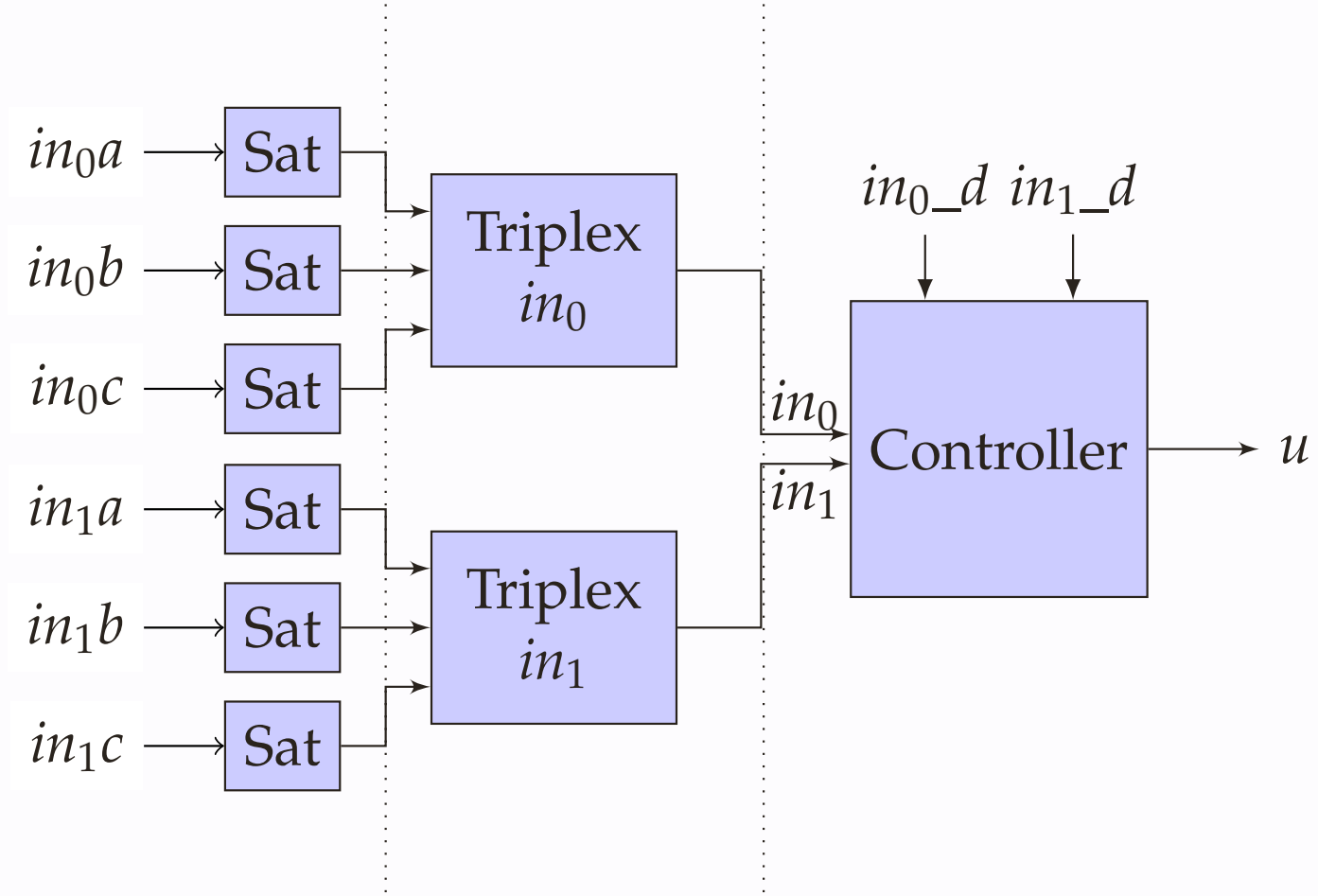


AT MODEL LEVEL - REBUILDING THE ANALYSIS



$]-\infty,+\infty[$ 1.2 3.6 194.499

AT MODEL LEVEL - REBUILDING THE ANALYSIS



$]-\infty,+\infty[$ 1.2 3.6 194.499

System is bounded!

Contents

Current trends – Challenges

- Combining methods:
  - Deductive methods and model checking often require additional invariants
  - use astract interpretation or other techniques to compute invariants
- Extend the analyses to hybrid systems
  - extend the set of properties considered
  - guaranting exhaustively that properties hold on a mix of discrete and continuous systems
- Integrate formal methods in the development process:
  - propagate down specification and proofs
  - propagate back counter examples and invariants for annotation/traceability purposes.
  - in an automatic fashion

Conclusion

- Critical CPS and softwares are everywhere
- They play a major role in transportation system (and elsewhere)
- Certification norms are restrictive but they simplify the use of formal methods
- Formal methods exist since the 70s but they reached a level of maturity that permits their use in the real world now
- They could play a major role in the development of UAVs

- Challenging issues on transfers from research to industry
  - interact with industry, understand their process, make them evolve.
- Challenging issues on the research side to handle new properties
  - have new techniques that scale up, automatic proofs, ...