Coding Isn't Programming

Leslie Lamport

Microsoft Researcher Emeritus

What This Talk is About

What This Talk is About

You probably expect me to talk about concurrency.

What This Talk is About

You probably expect me to talk about concurrency.

I have nothing new to say about that, and I'm tired of saying the same old stuff.

What This Talk is About

You probably expect me to talk about concurrency.

I have nothing new to say about that, and I'm tired of saying the same old stuff.

I've realized that some things that I've learned about writing concurrent programs apply to all programs.

What This Talk is About

You probably expect me to talk about concurrency.

I have nothing new to say about that, and I'm tired of saying the same old stuff.

I've realized that some things that I've learned about writing concurrent programs apply to all programs.

So I'm going to talk about programming.

Algorithms

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

It can be implemented in many programming languages.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

We don't have to write algorithms in a programming language.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

We shouldn't write algorithms in a programming language.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

We shouldn't write algorithms in a programming language.

Programming languages are complicated.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

We shouldn't write algorithms in a programming language.

Programming languages are complicated.

They have to be efficiently executed.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

We shouldn't write algorithms in a programming language.

Programming languages are complicated.

- They have to be efficiently executed.
- They have to handle large programs.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

We shouldn't write algorithms in a programming language.

Programming languages are complicated.

Algorithms

I'm here because I wrote concurrent algorithms, not concurrent programs.

An algorithm is not a program, it's higher-level, more abstract.

We shouldn't write algorithms in a programming language.

Programming languages are complicated.

Algorithms are neither executed nor large.

What language should we use?

~~What language should we use?~~

Don't get hung up on languages.

~~What language should we use?~~

Don't get hung up on languages.

Think about ideas, not the language they're expressed in.

Concurrent Programs

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Distributed programs are ones in which the threads are executed on different computers.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

- Actions of different threads can be interleaved in many ways.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

- Actions of different threads can be interleaved in many ways.
- This implies a huge number of possible executions.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

- Actions of different threads can be interleaved in many ways.
- This implies a huge number of possible executions.
- Hard to think of all the ways they can go wrong.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Debugging doesn't work.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Debugging doesn't work.

Can't be sure you've checked all relevant cases.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Debugging doesn't work.

- Can't be sure you've checked all relevant cases.
- A program can work fine

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Debugging doesn't work.

- Can't be sure you've checked all relevant cases.
- A program can work fine, and a change that alters the relative execution rates of the threads can reveal a bug.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Debugging doesn't work.

- Can't be sure you've checked all relevant cases.
- A program can work fine, and a change that alters the relative execution rates of the threads can reveal a bug.
- Fixing one bug is likely to introduce a new bug.

Concurrent Programs

Contain multiple threads of control that can be executed concurrently.

Concurrent programs are very hard to get right.

Debugging doesn't work.

How to Write a Concurrent Program

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

The part that synchronizes the threads.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

The part that synchronizes the threads.

Usually a small part of what the program does.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Maybe it's in a textbook.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Maybe it's in a textbook.

Maybe it's almost like one in a textbook.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

- Maybe it's in a textbook.
- Maybe it's almost like one in a textbook.
- Maybe it's brand new.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

- Maybe it's in a textbook.
- Maybe it's almost like one in a textbook.
- Maybe it's brand new.
- Hard for the same reason concurrent programs are hard to get right.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

- Maybe it's in a textbook.
- Maybe it's almost like one in a textbook.
- Maybe it's brand new.
- Hard for the same reason concurrent programs are hard to get right.
- But algorithms are simpler than programs.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Implement the algorithm.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Implement the algorithm.

That's coding.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Implement the algorithm.

That's coding.

Lot's of languages and tools have been developed for coding.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Implement the algorithm.

- That's coding.
- Lot's of languages and tools have been developed for coding.
- Programmers are good at coding.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Implement the algorithm.

- That's coding.
- Lot's of languages and tools have been developed for coding.
- Programmers are good at coding.
- They're not so good at finding correct algorithms.

How to Write a Concurrent Program

Figure out what part of what the program does involves concurrency.

Find a correct algorithm to do that part.

Implement the algorithm.

Abstraction

Abstraction

Something is usually called an *algorithm* only if it can be useful in many applications.

Abstraction

Something is usually called an *algorithm* only if
it can be useful in many applications.

The algorithm that describes how concurrency is implemented
in a program is often useful only for that program.

Abstraction

Something is usually called an *algorithm* only if it can be useful in many applications.

The algorithm that describes how concurrency is implemented in a program is often useful only for that program.

I therefore call it an *abstract view*, or an *abstract program*, or simply an *abstraction*.

How to Write a Concurrent Program

How to Write a Concurrent Program

Find an abstract view of the program that describes how it handles concurrency.

How to Write a Concurrent Program

Find an abstract view of the program that describes how it handles concurrency.

A higher-level abstraction than the code.

How to Write a Concurrent Program

Find an abstract view of the program that describes how it handles concurrency.

How to Write a Concurrent Program

Find an abstract view of the program that describes how it handles concurrency.

Programmers are taught how to code, not how to abstract.

How to Write a Concurrent Program

Find an abstract view of the program that describes how it handles concurrency.

Programmers are taught how to code, not how to abstract.

It involves a new kind of thinking.

How to Write a Concurrent Program

Find an abstract view of the program that describes how it handles concurrency.

Programmers are taught how to code, not how to abstract.

It involves a new kind of thinking.

Thinking before you code.

How to Write a Concurrent Program

Find an abstract view of the program that describes how it handles concurrency.

Programmers are taught how to code, not how to abstract.

It involves a new kind of thinking.

Thinking before you code.

Thinking at a higher level than code

How to Write a Concurrent Program

Find an abstract view of the program that describes
how it handles concurrency.

Programmers are taught how to code, not how to abstract.

It involves a new kind of thinking.

How to Write a Concurrent Program

Find an abstract view of the program that describes
how it handles concurrency.

Programmers are taught how to code, not how to abstract.

It involves a new kind of thinking.

Programmers should learn how to do it.

How to Write a ~~Concurrent~~ Program

Find an abstract view of the program that describes how it handles concurrency.

Programmers are taught how to code, not how to abstract.

It involves a new kind of thinking.

Programmers should learn how to do it for all programs.

What's a Program?

What's a Program?

Any piece of code that requires thinking before coding.

What's a Program?

Any piece of code that requires thinking before coding.

Perhaps a complete program

What's a Program?

Any piece of code that requires thinking before coding.

- Perhaps a complete program or a method

What's a Program?

Any piece of code that requires thinking before coding.

Perhaps a complete program or a method or a complicated loop.

What's a Program?

Any piece of code that requires thinking before coding.

What's a Program?

Any piece of code that requires thinking before coding.

Any piece of code to be used by someone who doesn't want to read the code.

How to Write an Abstraction

How to Write an Abstraction

Programs are written for many purposes.

How to Write an Abstraction

Programs are written for many purposes.

No method is best for all programs.

How to Write an Abstraction

Programs are written for many purposes.

No method is best for all programs.

For most programs, you should write two things:

How to Write an Abstraction

Programs are written for many purposes.

No method is best for all programs.

For most programs, you should write two things:

What the program does.

How to Write an Abstraction

Programs are written for many purposes.

No method is best for all programs.

For most programs, you should write two things:

- What the program does.
- How the program does it.

What language should we use?

~~What language should we use?~~

Don't get hung up on languages.

~~What language should we use?~~

Don't get hung up on languages.

If we want to build a tool to check that the *how* implies the *what*,

~~What language should we use?~~

Don't get hung up on languages.

If we want to build a tool to check that the *how* implies the *what*, then we need a precise language.

~~What language should we use?~~

Don't get hung up on languages.

If we want to build a tool to check that the *how* implies the *what*, then we need a precise language.

Needed to ensure that a concurrent program does what it should.

~~What language should we use?~~

Don't get hung up on languages.

If we want to build a tool to check that the *how* implies the *what*, then we need a precise language.

Needed to ensure that a concurrent program does what it should.

I designed TLA$^+$ for that.

~~What language should we use?~~

Don't get hung up on languages.

If we want to build a tool to check that the *how* implies the *what*, then we need a precise language.

Needed to ensure that a concurrent program does what it should.

Not needed for most programs.

A Trivial Example

A Trivial Example

Write a function to compute the largest element in an array of 32-bit integers.

A Trivial Example

Write a function to compute the largest element in an array of 32-bit integers.

Too simple to require much thinking before coding.

A Trivial Example

Write a function to compute the largest element in an array of 32-bit integers.

Too simple to require much thinking before coding.

If I were really compulsive, I might write:

A Trivial Example

Write a function to compute the largest element in an array of 32-bit integers.

Too simple to require much thinking before coding.

If I were really compulsive, I might write:

What: Return the largest element in an integer array A.

A Trivial Example

Write a function to compute the largest element in an array of 32-bit integers.

Too simple to require much thinking before coding.

If I were really compulsive, I might write:

What: Return the largest element in an integer array A.

How: Examine $A[0]$, $A[1]$, ... in turn, letting x be the largest value found, and return x.

Here's the code, written in Rust

```
fn ArrayMax(A: &[i32]) -> i32 {
    let mut x = A[0];
    let mut i = 1;
    while i < A.len() {
        if A[i] > x {
            x = A[i];}
        i += 1;  }
    x }
```

Here's the code, written in Rust (which I don't know)

```
fn ArrayMax(A: &[i32]) -> i32 {
    let mut x = A[0];
    let mut i = 1;
    while i < A.len() {
        if A[i] > x {
            x = A[i];}
        i += 1;  }
    x }
```

Here's the code, written in Rust

```
fn ArrayMax(A: &[i32]) -> i32 {
    let mut x = A[0];
    let mut i = 1;
    while i < A.len() {
        if A[i] > x {
            x = A[i];}
        i += 1;   }
    x }
```

Raise your hand if you see the bug.

Here's the code, written in Rust

```
fn ArrayMax(A: &[i32]) -> i32 {
    let mut x = A[0];
    let mut i = 1;
    while i < A.len() {
        if A[i] > x {
            x = A[i];}
        i += 1;  }
    x }
```

Raise your hand if you see the bug.

It's not a coding bug.

Here's the code, written in Rust

```
fn ArrayMax(A: &[i32]) -> i32 {
    let mut x = A[0];
    let mut i = 1;
    while i < A.len() {
        if A[i] > x {
            x = A[i];}
        i += 1;  }
    x }
```

Raise your hand if you see the bug.

It's not a coding bug. It's a bug in the *What*.

What: Return the largest element in an integer array A.

What: Return the largest element in an integer array A.

If A has no element, then there is no largest one.

What: Return the largest element in an integer array A.

If A has no element, then there is no largest one.

There's an obvious fix to the *What*

What: Return the largest element in an integer array A,
or an error value if A has no element.

If A has no element, then there is no largest one.

There's an obvious fix to the *What*

What: Return the largest element in an integer array A, or an error value if A has no element.

If A has no element, then there is no largest one.

There's an obvious fix to the *What* and to the *How*:

What: Return the largest element in an integer array A, or an error value if A has no element.

If A has no element, then there is no largest one.

There's an obvious fix to the *What* and to the *How*:

```
fn ArrayMax(A: &[i32]) -> Result<i32, &'static str> {
    if A.is_empty() {
        return Err("EmptyArray"); }
    let mut x = A[0];
    let mut i = 1;
    while i < A.len() {
        if A[i] > x {
            x = A[i];}
        i += 1;  }
    Ok(x) }
```

Abstraction

Abstraction

Let's now see how abstraction removes coding details and helps us understand why the program does the right thing.

Abstraction

Let's now see how abstraction removes coding details and helps us understand why the program does the right thing.

Remember, we wouldn't really do this on such a tiny example.

Abstraction

Let's now see how abstraction removes coding details and helps us understand why the program does the right thing.

Remember, we wouldn't really do this on such a tiny example.

But it will illustrate how abstraction works.

Abstracting the What

What: Return the largest element in an integer array A or an error value if A has no element.

Abstracting the What

What: Return the largest element in an integer array A or an error value if A has no element.

The function call/return is a coding detail.

Abstracting the What

What: Return the largest element in an integer array A or an error value if A has no element.

The function call/return is a coding detail.

We're interested in how the largest element of A is found.

Abstracting the What

What: Set x to the largest element in an integer array A or an error value if A has no element.

The function call/return is a coding detail.

We're interested in how the largest element of A is found.

Abstracting the What

What: Set x to the largest element in an integer array A or an error value if A has no element.

Why just integers?

Abstracting the What

What: Set x to the largest element in an array A of numbers or an error value if A has no element.

Why just integers?

Abstracting the What

What: Set x to the largest element in an array A of numbers or an error value if A has no element.

Why an array?

Abstracting the What

What: Set x to the largest element in an array A of numbers or an error value if A has no element.

Why an array?

I assume we're interested in the elements, not how they're arranged.

Abstracting the What

What: Set x to the largest element in a multiset A of numbers or an error value if A has no element.

Why an array?

I assume we're interested in the elements, not how they're arranged.

Abstracting the What

What: Set x to the largest element in a multiset A of numbers or an error value if A has no element.

Why an array?

I assume we're interested in the elements, not how they're arranged.

It's a multiset, not a set, because it can contain multiple copies of the same element.

What: Set x to the largest element in a multiset A of numbers or an error value if A has no element.

What: Set x to the largest element in a multiset A of numbers or an error value if A has no element.

I don't like this part because it complicates the code.

What: Set x to the largest element in a multiset A of numbers
or an error value if A has no element.

I don't like this part because it complicates the code.

Can we find a different *What* that's just as good
but has a simpler implementation?

Of course this is silly for such a simple example.

Of course this is silly for such a simple example.
The complication is about two lines of code.

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

– Decide what I wanted the program to do.

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

– Decide what I wanted the program to do.

– Realize coding it would be a lot of work.

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

- Decide what I wanted the program to do.
- Realize coding it would be a lot of work.
- Figure out what I really needed it to do.

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

- Decide what I wanted the program to do.
- Realize coding it would be a lot of work.
- Figure out what I really needed it to do.

The result was a *What* that might not have everything I wanted

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

- Decide what I wanted the program to do.
- Realize coding it would be a lot of work.
- Figure out what I really needed it to do.

The result was a *What* that might not have everything I wanted, but had everything I needed.

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

- Decide what I wanted the program to do.
- Realize coding it would be a lot of work.
- Figure out what I really needed it to do.

The result was a *What* that might not have everything I wanted, but had everything I needed.

This took time

Of course this is silly for such a simple example.

But when I wrote a program, often I would:

– Decide what I wanted the program to do.

– Realize coding it would be a lot of work.

– Figure out what I really needed it to do.

The result was a *What* that might not have everything I wanted, but had everything I needed.

This took time, but it saved much more time writing the code.

What: Set x to the largest element in a multiset A of numbers or an error value if A has no element.

What: Set x to the largest element in a multiset A of numbers ~~or an error value if A has no element.~~

We want to eliminate this

What: Set x to the largest element in a multiset A of numbers

We want to eliminate this, which requires modifying this.

What: Set x to the largest element in a multiset A of numbers

We want to eliminate this, which requires modifying this.

How to do that is not obvious to most programmers.

What: Set x to the largest element in a multiset A of numbers

We want to eliminate this, which requires modifying this.

How to do that is not obvious to most programmers.

It's obvious to me because I was educated as a mathematician

What: Set x to the largest element in a multiset A of numbers.

We want to eliminate this, which requires modifying this.

How to do that is not obvious to most programmers.

It's obvious to me because I was educated as a mathematician, so I know what the largest element of an empty set should equal.

Set x to the smallest number $\geq$ all elements of A.

What: ~~Set x to the largest element in a multiset A of numbers.~~

We want to eliminate this, which requires modifying this.

Here's how.

Set x to the smallest number $\geq$ all elements of A.

What: ~~Set x to the largest element in a multiset A of numbers.~~

We want to eliminate this, which requires modifying this.

Here's how.

The two are equivalent if A is not empty.

Set x to the smallest number $\geq$ all elements of A.

What: ~~Set x to the largest element in a multiset A of numbers.~~

We want to eliminate this, which requires modifying this.

Here's how.

The two are equivalent if A is not empty.

But what if A is empty?

What: Set x to the smallest number $\geq$ all elements of A.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

Why is this true?

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

I'm richer than everyone living in Bodie, CA.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

I'm richer than everyone living in Bodie, CA.

I'm poorer than everyone living in Bodie, CA.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

I'm richer than everyone living in Bodie, CA.

I'm poorer than everyone living in Bodie, CA.

No one lives in Bodie, CA.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

I'm richer than everyone living in Bodie, CA.

I'm poorer than everyone living in Bodie, CA.

No one lives in Bodie, CA.

This is simple (mathematical) logic.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

I'm richer than everyone living in Bodie, CA.

I'm poorer than everyone living in Bodie, CA.

No one lives in Bodie, CA.

This is simple (mathematical) logic.

To think rationally, you have to understand simple logic.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

I'm richer than everyone living in Bodie, CA.

I'm poorer than everyone living in Bodie, CA.

No one lives in Bodie, CA.

This is simple (mathematical) logic.

To think rationally, you have to understand simple logic.

Programmers should think rationally.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

If A is empty, then x is set to the smallest number.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

If A is empty, then x is set to the smallest number.

Mathematicians (sometimes) define $-\infty$ to be the smallest number.

What: Set x to the smallest number $\geq$ all elements of A.

If A is empty, then every number is $\geq$ all elements of A.

If A is empty, then x is set to the smallest number.

Mathematicians (sometimes) define $-\infty$ to be the smallest number.

So this *What* implies that if A is empty, then x is set to the number $-\infty$.

The *How*

The *How*

let $B = A$ **and** $x = -\infty$ **;**

The *How*

let $B = A$ **and** $x = -\infty$;
while B not empty {

}

The *How*

let $B = A$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;

}

The *How*

let $B = A$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;

}

This is nondeterministic; there are many possible executions.

The *How*

let $B = A$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
}

The *How*

```
let  B = A  and  x = −∞ ;
while  B not empty  {
      let  i = any element of B ;
      let  B = B with i removed ;
      if  i > x  { let x = i }        }
```

The *How*

let $B = A$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

I used an *ad hoc* combination of programming-language notation

The *How*

```
let B = A and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

I used an *ad hoc* combination of programming-language notation

The *How*

```
let  B = A  and  x = −∞ ;
while  B not empty  {
    let  i = any element of B ;
    let  B = B with i removed ;
    if  i > x  { let x = i }        }
```

I used an *ad hoc* combination of programming-language notation and English

The *How*

let $B = A$ **and** $x = -\infty$;
while B not empty {
  let i = any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

I used an *ad hoc* combination of programming-language notation and English

let $B = A$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

```
let B = A and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }      }
```

How can we convince someone that this implements the *What*?

```
let B = A and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

How can we convince someone that this implements the *What*?

I believe most programmers could only show that some executions compute the right value of x.

let $B = A$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

How can we convince someone that this implements the *What*?

I believe most programmers could only show that some executions compute the right value of x.

But we must show **every** execution computes the right value of x.

let $B = A$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

How can we convince someone that this implements the *What*?

I believe most programmers could only show that some executions compute the right value of x.

But we must show **every** execution computes the right value of x.

Seeing how to do this requires thinking about executions.

What is an Execution?

What is an Execution?

An obvious answer:

What is an Execution?

An obvious answer:

A sequence of steps.

What is an Execution?

An obvious answer:

A sequence of steps.

Each step is the execution of a part of the code.

What is an Execution?

A usually better answer:

What is an Execution?

A usually better answer:

No rule applies to all programs.

What is an Execution?

A usually better answer:

A sequence of states.

What is an Execution?

A usually better answer:

A sequence of states.

A step is a pair of consecutive states

What is an Execution?

A usually better answer:

A sequence of states.

A step is a pair of consecutive states that describes execution of a part of the code.

What is an Execution?

A usually better answer:

A sequence of states.

A step is a pair of consecutive states that describes execution of a part of the code.

To see how we describe an execution this way,
we look at one possible execution.

What is an Execution?

A usually better answer:

A sequence of states.

A step is a pair of consecutive states that describes execution of a part of the code.

To see how we describe an execution this way, we look at one possible execution.

We suppose A has two copies of 2 and one copy of 3.

What is an Execution?

A usually better answer:

A sequence of states.

A step is a pair of consecutive states that describes execution of a part of the code.

To see how we describe an execution this way,
we look at one possible execution.

We suppose A has two copies of 2 and one copy of 3.
Let's write this multiset as $\{\{2, 2, 3\}\}$ or $\{\{2, 3, 2\}\}$ or $\{\{3, 2, 2\}\}$.

What is an Execution?

A usually better answer:

A sequence of states.

A step is a pair of consecutive states that describes execution of a part of the code.

To see how we describe an execution this way, we look at one possible execution.

We suppose A has two copies of 2 and one copy of 3.

Let's write this multiset as $\{\{2, 2, 3\}\}$ or $\{\{2, 3, 2\}\}$ or $\{\{3, 2, 2\}\}$.

I'll write it this way

```
let  B = {{2, 3, 2}}  and  x = −∞ ;
while  B not empty  {
      let  i = any element of B ;
      let  B = B with i removed ;
      if  i > x  { let x = i }          }
```

```
let  B = {{2, 3, 2}}  and  x = −∞ ;
while  B not empty  {
      let  i = any element of  B ;
      let  B = B with i removed ;
      if  i > x  { let x = i }        }
```

A state will describe the values of B and x and perhaps other stuff.

```
let B = {{2,3,2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = & \\ x = & \end{bmatrix}$$

A state will describe the values of B and x and perhaps other stuff.

I'll just show the values of B and x.

```
let B = {{2,3,2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \\ x = \end{bmatrix}$$

What are their initial values?

let $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = ? \\ x\ = ? \end{bmatrix}$$

What are their initial values?

An obvious answer: some standard initial value we'll call "?" .

```
let  B = {{2, 3, 2}}  and  x = −∞ ;
while  B not empty  {
    let  i = any element of B ;
    let  B = B with i removed ;
    if  i > x  { let x = i }      }
```

$$\begin{bmatrix} B = ? \\ x = ? \end{bmatrix}$$

We first execute this statement.

let $B = \{\{2,3,2\}\}$ **and** $x = -\infty$ **;**
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = ? \\ x = ? \end{bmatrix}$$

We first execute this statement.

The obvious representation is as these two steps.

```
let  B = {{2,3,2}}  and  x = −∞ ;
while  B not empty  {
    let  i = any element of B ;
    let  B = B with i removed ;
    if  i > x  { let x = i }        }
```

$$\begin{bmatrix} B = ? \\ x = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = ? \end{bmatrix}$$

We first execute this statement.

The obvious representation is as these two steps.

let $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = ? \\ x = ? \end{bmatrix} \to \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = ? \end{bmatrix} \to \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

We first execute this statement.

The obvious representation is as these two steps.

```
let  B = {{2, 3, 2}}  and  x = −∞ ;
while  B not empty  {
      let  i = any element of  B ;
      let  B = B  with  i  removed ;
      if  i > x  { let x = i }        }
```

$$\begin{bmatrix} B = ? \\ x = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

To understand this abstract program

let $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = ? \\ x\ = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x\ = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x\ = -\infty \end{bmatrix}$$

To understand this abstract program, we want an execution to be as simple as possible.

let $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = ? \\ x = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

To understand this abstract program, we want an execution to be as simple as possible.

This means making an execution have as few steps as possible.

let $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = ? \\ x = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = ? \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

These two steps are of no interest.

```
let  B = {{2,3,2}}  and  x = −∞ ;
while  B not empty  {
      let  i = any element of B ;
      let  B = B with i removed ;
      if  i > x  { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

These two steps are of no interest.

So we let this statement define the initial state.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

These two steps are of no interest.

So we let this statement define the initial state.

And we can rewrite it like this.

```
initially B = {{2,3,2}} and x = −∞ ;
while B not empty {
      let i = any element of B ;
      let B = B with i removed ;
      if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x\ = -\infty \end{bmatrix}$$

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

What's the next state?

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

What's the next state?

Code doesn't say what constitutes a step.

```
initially B = {{2, 3, 2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

What's the next state?

Code doesn't say what constitutes a step.

Is executing this statement one step?

```
initially B = {{2, 3, 2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

What's the next state?

Code doesn't say what constitutes a step.

Is executing this statement one step?

Or are choosing the element of B

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

What's the next state?

Code doesn't say what constitutes a step.

Is executing this statement one step?

Or are choosing the element of B and setting i

```
initially B = {{2,3,2}} and x = −∞ ;
while B not empty {
     let i = any element of B ;
     let B = B with i removed ;
     if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

What's the next state?

Code doesn't say what constitutes a step.

Is executing this statement one step?

Or are choosing the element of B and setting i separate steps?

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2, 3, 2\}\} \\ x = -\infty \end{bmatrix}$$

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2, 3, 2\}\} \\ x = -\infty \end{bmatrix}$$

To keep the abstraction simple

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2, 3, 2\}\} \\ x = -\infty \end{bmatrix}$$

To keep the abstraction simple, I will make evaluating the **while** loop's test

```
initially B = {{2,3,2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

To keep the abstraction simple, I will make evaluating the **while** loop's test and evaluating its body

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

To keep the abstraction simple, I will make evaluating the **while** loop's test and evaluating its body one step.

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
    let i = any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = \{\{2, 3, 2\}\} \\ x = -\infty \end{bmatrix}$$

To keep the abstraction simple, I will make evaluating the **while** loop's test and evaluating its body one step.

I won't bother defining a way to make the pseudocode say that's all one step.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

```
initially B = {{2,3,2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

Executing the next step can let i equal 2 or 3.

```
initially B = {{2, 3, 2}} and x = −∞ ;
while B not empty {
      let i = any element of B ;
      let B = B with i removed ;
      if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix}$$

Executing the next step can let i equal 2 or 3.

I will let it choose 3.

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2, 3, 2\}\} \\ x = -\infty \end{bmatrix} \to \begin{bmatrix} B = \{\{2, 2\}\} \\ x = 3 \end{bmatrix}$$

Executing the next step can let i equal 2 or 3.

I will let it choose 3.

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
  let i = any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x\ = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x\ = 3 \end{bmatrix}$$

The next two steps have to choose 2.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix}$$

The next two steps have to choose 2.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The next two steps have to choose 2.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \to \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \to \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \to \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix} \to \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The next step finds B empty, so it exits the loop without changing B or x.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The next step finds B empty, so it exits the loop without changing B or x.

That's an uninteresting step.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x \ = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x \ = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x \ = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x \ = 2 \end{bmatrix}$$

The next step finds B empty, so it exits the loop without changing B or x.

That's an uninteresting step.

So we declare that the execution terminates when B is empty.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The execution terminates when B is empty.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \to \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \to \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \to \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The execution terminates when B is empty.

I don't know how to say that with code.

```
initially B = {{2, 3, 2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \to \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \to \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \to \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The execution terminates when B is empty.

I don't know how to say that with code.
(It's easy to say in TLA$^+$.)

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The execution terminates when B is empty.

I don't know how to say that with code.

But don't get hung up on languages.

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The execution terminates when B is empty.

I don't know how to say that with code.

But don't get hung up on languages.

Understand what the executions are.

```
initially B = {{2,3,2}} and x = −∞ ;
while B not empty {
    let i = any element of B ;
    let B = B with i removed ;
    if i > x { let x = i }        }
```

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

The execution terminates when B is empty.

I don't know how to say that with code.

But don't get hung up on languages.

Understand what the executions are.

Don't worry about how they're described.

What's a State?

What's a State?

For executions to be a useful way to think about a program:

What's a State?

For executions to be a useful way to think about a program:

The possible next states must depend only on the current state

What's a State?

For executions to be a useful way to think about a program:

The possible next states must depend only on the current state, not on any previous state.

What's a State?

For executions to be a useful way to think about a program:

The possible next states must depend only on the current state, not on any previous state.

We can choose states this way for abstractions that describe actual programs because:

What's a State?

For executions to be a useful way to think about a program:

The possible next states must depend only on the current state, not on any previous state.

We can choose states this way for abstractions that describe actual programs because:

Programs are executed on computers.

What's a State?

For executions to be a useful way to think about a program:

The possible next states must depend only on the current state, not on any previous state.

We can choose states this way for abstractions that describe actual programs because:

Programs are executed on computers.

What a computer does next depends only on its current state

What's a State?

For executions to be a useful way to think about a program:

The possible next states must depend only on the current state, not on any previous state.

We can choose states this way for abstractions that describe actual programs because:

Programs are executed on computers.

What a computer does next depends only on its current state (and perhaps external inputs)

What's a State?

For executions to be a useful way to think about a program:

The possible next states must depend only on the current state, not on any previous state.

We can choose states this way for abstractions that describe actual programs because:

Programs are executed on computers.

What a computer does next depends only on its current state, not on any previous state.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x = -\infty \end{bmatrix} \to \begin{bmatrix} B = \{\{2,2\}\} \\ x = 3 \end{bmatrix} \to \begin{bmatrix} B = \{\{2\}\} \\ x = 2 \end{bmatrix} \to \begin{bmatrix} B = \{\{\ \}\} \\ x = 2 \end{bmatrix}$$

For our simple example:

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
  let $i =$ any element of B ;
  let $B = B$ with i removed ;
  if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x \;= -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x \;= 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x \;= 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x \;= 2 \end{bmatrix}$$

For our simple example:

The state has to describe only the values of B and x.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let i = any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x \ = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x \ = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x \ = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x \ = 2 \end{bmatrix}$$

For our simple example:

The state has to describe only the values of B and x.

The variable i is used only to indicate how the step that ends in the current state changes B.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let i = any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x \ = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x \ = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x \ = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x \ = 2 \end{bmatrix}$$

For our simple example:
  The state has to describe only the values of B and x.

The variable i is used only to indicate how the step that ends in the current state changes B. Its value doesn't affect future states.

initially $B = \{\{2,3,2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ }   }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x\ = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x\ = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x\ = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x\ = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x\ = 2 \end{bmatrix}$$

For our simple example:

The state has to describe only the values of B and x.

The variable i is used only to indicate how the step that ends in the current state changes B. Its value doesn't affect future states.

Had I not eliminated the step representing an execution of the **while** statement with B empty

initially $B = \{\{2, 3, 2\}\}$ **and** $x = -\infty$;
while B not empty {
    let $i =$ any element of B ;
    let $B = B$ with i removed ;
    if $i > x$ { **let** $x = i$ } }

$$\begin{bmatrix} B = \{\{2,3,2\}\} \\ x\ = -\infty \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2,2\}\} \\ x\ = 3 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{2\}\} \\ x\ = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x\ = 2 \end{bmatrix} \rightarrow \begin{bmatrix} B = \{\{\ \}\} \\ x\ = 2 \end{bmatrix}$$

For our simple example:

The state has to describe only the values of B and x.

The variable i is used only to indicate how the step that ends in the current state changes B. Its value doesn't affect future states.

Had I not eliminated the step representing an execution of the **while** statement with B empty, the state would also have needed to indicate whether the execution had terminated.

Why Is the Abstract Program Correct?

Why Is the Abstract Program Correct?

Why does any possible execution terminate with x equal to the correct value?

Why Is the Abstract Program Correct?

Why does any possible execution terminate with x equal to the correct value?

At any point in the execution, what can happen in the future can depend only on the current state.

Why Is the Abstract Program Correct?

Why does any possible execution terminate with
x equal to the correct value?

At any point in the execution, what can happen
in the future can depend only on the current state.

Therefore, at any point in the execution, the value x can
have when it terminates depends only on the current state.

Why Is the Abstract Program Correct?

Why does any possible execution terminate with
x equal to the correct value?

At any point in the execution, what can happen
in the future can depend only on the current state.

Therefore, at any point in the execution, the value x can
have when it terminates depends only on the current state.

Why x can only have the correct value when it terminates
must depend on something that's true of every state.

Why Is the Abstract Program Correct?

Why does any possible execution terminate with x equal to the correct value?

At any point in the execution, what can happen in the future can depend only on the current state.

Therefore, at any point in the execution, the value x can have when it terminates depends only on the current state.

Why x can only have the correct value when it terminates must depend on something that's true of every state.

Something true of every state of every execution is called an *invariant* of the program.

You don't understand why a program does the right thing

You don't understand why a program does the right thing,

for example terminating with the right answer

You don't understand why a program does the right thing,
unless you know the invariant that ensures it does the right thing.

Here is the invariant for our example

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

so the program should terminate with x equal to $max(A)$

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M:

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

To show that this is an invariant, we show that it satisfies these two conditions:

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\{\{x, max(B)\}\}) = max(A)$$

To show that this is an invariant, we show that it satisfies these two conditions:

1. It is true of the initial state.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

To show that this is an invariant, we show that it satisfies these two conditions:

1. It is true of the initial state.
2. If it's true in any state, then it's true in the next state.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\!\{x, max(B)\}\!\}\,) \;=\; max(A)$$

To show that this is an invariant, we show that it satisfies these two conditions:

1. It is true of the initial state.
2. If it's true in any state, then it's true in the next state.

And to show that the invariant implies correctness, we show:

3. It implies $x = max(A)$ in a terminated state.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

1. It is true of the initial state.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

1. It is true of the initial state.

   Because $B = A$ and $x = -\infty$,

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\ \{\{x, max(B)\}\}\ )\ =\ max(A)$$

1. It is true of the initial state.

   Because $B = A$ and $x = -\infty$, using $max(\{\{-\infty, max(A)\}\}) = max(A)$.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

3. It implies $x = max(A)$ in a terminated state.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\, \{\{x, max(B)\}\} \,) \;=\; max(A)$$

3. It implies $x = max(A)$ in a terminated state.

   Because $B = \{\{ \}\}$,

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

3. It implies $x = max(A)$ in a terminated state.

   Because $B = \{\{\,\}\}$, using $max(\{\{\,\}\}) = -\infty$ and $max(\{\{x, -\infty\}\}) = x$.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\{\{x, max(B)\}\}) = max(A)$$

2. If it's true in any state, then it's true in the next state.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

2. If it's true in any state, then it's true in the next state.

I don't know how many programmers can figure out why this condition holds.

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\{\{x, max(B)\}\}) = max(A)$$

2. If it's true in any state, then it's true in the next state.

I don't know how many programmers can figure out why this condition holds.

I think you should learn how

Here is the invariant for our example, where $max(M)$ is the smallest number $\geq$ every element of a multiset M :

$$max(\,\{\{x, max(B)\}\}\,) \;=\; max(A)$$

2. If it's true in any state, then it's true in the next state.

I don't know how many programmers can figure out why this condition holds.

I think you should learn how, because I expect those who can't to be among the first programmers replaced by AI.

Termination

Termination

I explained how to show that x has the correct value when the abstract program terminates.

Termination

I explained how to show that x has the correct value when the abstract program terminates.

I haven't explained how to show that it always terminates.

Termination

I explained how to show that x has the correct value when the abstract program terminates.

I haven't explained how to show that it always terminates.

I don't have time to discuss termination.

Termination

I explained how to show that x has the correct value when the abstract program terminates.

I haven't explained how to show that it always terminates.

I don't have time to discuss termination.

If I did, we would see that the program doesn't terminate for some values of A.

Termination

I explained how to show that x has the correct value when the abstract program terminates.

I haven't explained how to show that it always terminates.

I don't have time to discuss termination.

If I did, we would see that the program doesn't terminate for some values of A.

Can you figure out what values those are?

Implementing $-\infty$

Implementing $-\infty$

Suppose A is an array of 32-bit integers.

Implementing $-\infty$

Suppose A is an array of 32-bit integers.

Method 1: Implement $-\infty$ as the smallest 32-bit integer.

Implementing $-\infty$

Suppose A is an array of 32-bit integers.

Method 1: Implement $-\infty$ as the smallest 32-bit integer.
If that's acceptable, then an implementation that does not test if A is empty is a satisfactory implementation.

Implementing $-\infty$

Suppose A is an array of 32-bit integers.

Method 1: Implement $-\infty$ as the smallest 32-bit integer.
If that's acceptable, then an implementation that does not test if A is empty is a satisfactory implementation.

Method 2: Implement $-\infty$ as an error value.

Implementing $-\infty$

Suppose A is an array of 32-bit integers.

Method 1: Implement $-\infty$ as the smallest 32-bit integer.
If that's acceptable, then an implementation that does not test if A is empty is a satisfactory implementation.

Method 2: Implement $-\infty$ as an error value.
The Rust code that tests if A is empty implements our abstract program.

Implementing $-\infty$

Suppose A is an array of 32-bit integers.

Method 1: Implement $-\infty$ as the smallest 32-bit integer.
If that's acceptable, then an implementation that does not test if A is empty is a satisfactory implementation.

Method 2: Implement $-\infty$ as an error value.
The Rust code that tests if A is empty implements our abstract program.

Few programmers or computer scientist know what it means for a program to implement an abstract program (or algorithm).

Implementing $-\infty$

Suppose A is an array of 32-bit integers.

Method 1: Implement $-\infty$ as the smallest 32-bit integer.
If that's acceptable, then an implementation that does not test if A is empty is a satisfactory implementation.

Method 2: Implement $-\infty$ as an error value.
The Rust code that tests if A is empty implements our abstract program.

Few programmers or computer scientist know what it means for a program to implement an abstract program (or algorithm).

I don't have time to explain what it means.

Real Concurrent Programs

Real Concurrent Programs

The *Why* and *How* should be precise.

Real Concurrent Programs

The *Why* and *How* should be precise.

Tools should check that the *How* implements the *Why*.

Real Concurrent Programs

The *Why* and *How* should be precise.

Tools should check that the *How* implements the *Why*.

Here's an example of how TLA^+ works in practice.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- **Formal methods find bugs in system designs that cannot be found through any other technique we know of.**
- **Formal methods are surprisingly feasible for mainstream software development and give good return on investment.**
- **At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.**

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other

They build Amazon's cloud infrastructure.

to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- **Formal methods find bugs in system designs that cannot be found through any other technique we know of.**
- **Formal methods are surprisingly feasible for mainstream software development and give good return on investment.**
- **At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.**

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary com-

The formal method they use is TLA^{+}.

pend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

» key insights

- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» **key insights**

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment

66 COMMUNICATIONS OF THE ACM | APRIL 2015 | VOL. 58 | NO. 4

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.

- **Formal methods find bugs in system designs that cannot be found through any other technique we know of.**

TLA^+ finds

- ~~Formal methods find~~ bugs in system designs that cannot be found through any other technique we know of.

TLA$^+$ finds

- ~~Formal methods find~~ bugs in system designs that cannot be found through any other technique we know of.

These are fundamental design flaws

TLA^+ finds

- ~~Formal methods find~~ bugs in system designs that cannot be found through any other technique we know of.

These are fundamental design flaws, not just simple coding errors.

TLA$^+$ finds

- ~~Formal methods find~~ **bugs** in system designs that cannot be found through any other technique we know of.

These are fundamental design flaws.

Very expensive to fix after the code is written

TLA^+ finds

- ~~Formal methods find~~ **bugs** in system designs that cannot be found through any other technique we know of.

These are fundamental design flaws.

Very expensive to fix after the code is written because it requires extensive recoding.

TLA$^+$ finds

■ ~~Formal methods find~~ bugs in system designs that **cannot be found** through any other technique we know of.

These are fundamental design flaws.

Very expensive to fix after the code is written because it requires extensive recoding.

And often not found until the code has been released to users.

TLA$^+$ finds

- ~~Formal methods find~~ bugs in system designs that cannot be found through any other technique we know of.

These are fundamental design flaws.

Very expensive to fix after the code is written because it requires extensive recoding.

But Amazon engineers find these flaws before any code is written.

Is Abstraction Useful Just for Concurrency?

Is Abstraction Useful Just for Concurrency?

The code for handling concurrency is important, but it's small.

Is Abstraction Useful Just for Concurrency?

The code for handling concurrency is important, but it's small.

What about the rest of the program?

Is Abstraction Useful Just for Concurrency?

The code for handling concurrency is important, but it's small.

What about the rest of the program?

I know of just one case in which an entire system system was built starting with a TLA$^+$ abstraction.

Rosetta

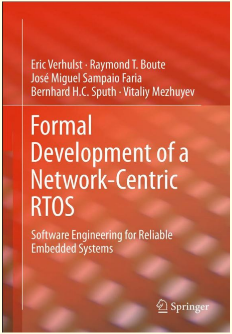
Rosetta

European Space Agency spacecraft that explored a comet.

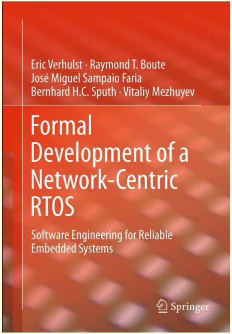
Rosetta

European Space Agency spacecraft that explored a comet.

Several of its instruments were controlled by
the Virtuoso real-time operating system.

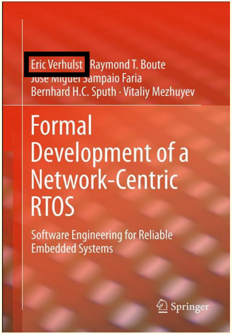


The next version of Virtuoso.



The next version of Virtuoso.

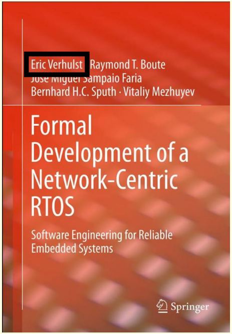
Its high-level design is described in TLA^+.



The next version of Virtuoso.

Its high-level design is described in TLA$^+$.

Here's an email from Eric Verhulst,
the head of the development team.

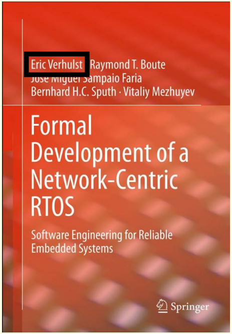


The next version of Virtuoso.

Its high-level design is described in TLA$^+$.

Here's an email from Eric Verhulst,
the head of the development team.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture

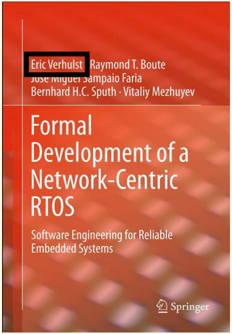


The next version of Virtuoso.

Its high-level design is described in TLA^{+}.

Here's an email from Eric Verhulst,
the head of the development team.

The [TLA^{+}] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming).



The next version of Virtuoso.

Its high-level design is described in TLA$^+$.

Here's an email from Eric Verhulst, the head of the development team.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10× less than in [Virtuoso].

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10× less than in [Virtuoso].

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10× less than in [Virtuoso].

You don't produce 10× less code by better coding.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10× less than in [Virtuoso].

You don't produce 10× less code by better coding.

You do it with a cleaner architecture

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10× less than in [Virtuoso].

You don't produce 10× less code by better coding.

You do it with a ~~cleaner architecture~~ better high-level design

The [TLA+] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10× less than in [Virtuoso].

You don't produce 10× less code by better coding.

better high-level design

You do it with a ~~cleaner architecture~~, which comes from thinking about an abstraction, not about the code.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10× less than in [Virtuoso].

You don't produce 10× less code by better coding.

You do it with a ~~cleaner architecture~~ better high-level design, which comes from thinking about an abstraction, not about the code.

It doesn't come from thinking in a programming language.

Sometimes what a program should do can't be stated precisely.

Sometimes what a program should do can't be stated precisely.

Here's an example I encountered.

TLATEX — the TLA$^+$ pretty-printer

TLATEX — the TLA$^+$ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

TLATEX — the TLA$^+$ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

The naive output:

$$\begin{array}{l} Foo \;\Rightarrow\; \land\, a = b \\ \qquad\quad\;\; \land\, ccc = d \end{array}$$

TLATEX — the TLA+ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

The user probably wanted these aligned.

The naive output:

$$\begin{array}{l} Foo \;\Rightarrow\; \land\, a = b \\ \qquad\quad\;\; \land\, ccc = d \end{array}$$

TLATEX — the TLA$^+$ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

The right output:

$$\begin{aligned} Foo \;\Rightarrow\; &\land\, a \;\;\;\;= b \\ &\land\, ccc = d \end{aligned}$$

The user probably wanted these aligned.

TLATEX — the TLA+ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

TLATEX — the TLA+ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

The naive output:

$$\land\ aaa + bb = c$$
$$\land\ iii = jj * k$$

TLATEX — the TLA+ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

The naive output:

$$\land\ aaa + bb = c$$
$$\land\ iii = jj * k$$

The user probably didn't wanted these aligned.

TLATEX — the TLA$^+$ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

The right output:

$$\wedge\ aaa + bb = c$$
$$\wedge\ iii = jj * k$$

The user probably didn't wanted these aligned.

There is no precise definition of correct alignment.

There is no precise definition of correct alignment.

We can't describe precisely what the user wants.

There is no precise definition of correct alignment.

We can't describe precisely what the user wants.

If we can't describe *What* precisely, abstraction is useless.

There is no precise definition of correct alignment.

We can't describe precisely what the user wants.

~~If we can't describe *What* precisely, abstraction is useless.~~

Wrong.

There is no precise definition of correct alignment.

We can't describe precisely what the user wants.

~~If we can't describe *What* precisely, abstraction is useless.~~

Wrong.

The program has to do something.

There is no precise definition of correct alignment.

We can't describe precisely what the user wants.

~~If we can't describe *What* precisely, abstraction is useless.~~

Wrong.

The program has to do something.

Not knowing precisely what it *should* do means
we have to think abstractly about what it *will* do.

It's impossible to specify the best pretty-printer.

It's impossible to specify the best pretty-printer.

But the program has to do something.

It's impossible to specify the best pretty-printer.

But the program has to do something.

Writing stream-of-consciousness code doesn't produce a good program.

My Abstraction

My Abstraction

6 rules plus definitions (in comments).

My Abstraction

6 rules plus definitions (in comments).

Example:

A left-comment token is LeftComment aligned with its covering token.

My Abstraction

6 rules plus definitions (in comments).

Example:

A left-comment token is LeftComment aligned with its covering token.

This is defined precisely (mostly in English).

Why Did I Write This Abstraction?

Why Did I Write This Abstraction?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

Why Did I Write This Abstraction?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules

Why Did I Write This Abstraction?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules, aided by debugging code to report what rules were being used.

Why Did I Write This Abstraction?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules, aided by debugging code to report what rules were being used.

The few bugs in implementing the rules were easy to catch.

Why Did I Write This Abstraction?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules, aided by debugging code to report what rules were being used.

The few bugs in implementing the rules were easy to catch.

Had I just written code, it would have taken me much longer and not produced formatting as good.

What is Typical About This Abstraction

What is Typical About This Abstraction

It's at a higher-level than the code.

What is Typical About This Abstraction

It's at a higher-level than the code.

It could have been implemented in any language.

What is Typical About This Abstraction

It's at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the abstraction.

What is Typical About This Abstraction

It's at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the abstraction.

No method or tool for writing better code would have made the abstraction unnecessary.

What is Typical About This Abstraction

It's at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the abstraction.

No method or tool for writing better code would have made the abstraction unnecessary.

It says nothing about how to write the code.

What is Typical About This Abstraction

It's at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the abstraction.

No method or tool for writing better code would have made the abstraction unnecessary.

It says nothing about how to write the code.

You write an abstraction to help you think about the problem before you think about the code.

What is Not Typical About This Abstraction

What is Not Typical About This Abstraction

It's quite subtle.

What is Not Typical About This Abstraction

It's quite subtle.

Perhaps 95% of programs require less thought,
so abstractions that are shorter and simpler suffice.

What is Not Typical About This Abstraction

It's quite subtle.

Perhaps 95% of programs require less thought,
so abstractions that are shorter and simpler suffice.

It's a set of rules.

What is Not Typical About This Abstraction

It's quite subtle.

Perhaps 95% of programs require less thought,
so abstractions that are shorter and simpler suffice.

It's a set of rules.

A set of rules/requirements/axioms is usually a bad abstraction.

What is Not Typical About This Abstraction

It's quite subtle.

Perhaps 95% of programs require less thought,
so abstractions that are shorter and simpler suffice.

It's a set of rules.

A set of rules/requirements/axioms is usually a bad abstraction.

It's hard to understand the consequences of a set of rules.

What is Not Typical About This Abstraction

It's quite subtle.

Perhaps 95% of programs require less thought,
so abstractions that are shorter and simpler suffice.

It's a set of rules.

A set of rules/requirements/axioms is usually a bad abstraction.

It's hard to understand the consequences of a set of rules.

No method is best for all programs.

Thinking is always better than not thinking before coding.

Thinking is always better than not thinking before coding.

Some people say that you shouldn't think too much before coding.

Thinking is always better than not thinking before coding.

Some people say that you shouldn't think too much before coding.

I say that too little thinking is a much more serious and much more common problem than too much thinking.

How to Think

How to Think

Write !

How to Think

Write !

> "Writing is nature's way of letting you know how sloppy your thinking is."
>
> *Guindon*

How to Think

Write!

"Writing is nature's way of letting you know how sloppy your thinking is."

Guindon

"If you think without writing, you only think you're thinking."

Lamport

How to Think

Write !

> "Writing is nature's way of letting you know how sloppy your thinking is."
>
> *Guindon*

> "If you think without writing, you only think you're thinking."
>
> *Lamport*

Writing helps you think better.

How to Think

Write!

> "Writing is nature's way of letting you know how sloppy your thinking is."
>
> *Guindon*

> "If you think without writing, you only think you're thinking."
>
> *Lamport*

Writing helps you think better.
Thinking better helps you write better.

How to Think

Write !

> “Writing is nature’s way of letting you know how sloppy your thinking is.”
>
> *Guindon*

> “If you think without writing, you only think you’re thinking.”
>
> *Lamport*

Writing helps you think better.
Thinking better helps you write better.

It’s a virtuous cycle.

To begin, most people must learn to write better.

To begin, most people must learn to write better.

This means writing to convince others.

To begin, most people must learn to write better.

This means writing to convince others.

It's too easy to convince yourself of something that's not true.

To begin, most people must learn to write better.

This means writing to convince others.

You have to learn to read what you wrote the way others might read it.

To begin, most people must learn to write better.

This means writing to convince others.

You have to learn to read what you wrote the way others might read it.

Perhaps other readers can teach you that.

How to Think Abstractly

How to Think Abstractly

Abstraction is what I'm good at.

How to Think Abstractly

Abstraction is what I'm good at.

And being good at it is why I was invited to speak to you.

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

In large part by being educated as a mathematician.

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

Abstraction is at the heart of mathematics.

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

Abstraction is at the heart of mathematics.

Math abstracts from two sheep and two goats to the number 2.

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

Abstraction is at the heart of mathematics.

I don't know how you should learn to be better at abstraction.

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

Abstraction is at the heart of mathematics.

I don't know how you should learn to be better at abstraction.

Because it's mathematics, TLA^+ teaches some users

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

Abstraction is at the heart of mathematics.

I don't know how you should learn to be better at abstraction.

Because it's mathematics, TLA$^+$ teaches some users,
but it may be too hard for most programmers.

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

Abstraction is at the heart of mathematics.

I don't know how you should learn to be better at abstraction.

Perhaps mathematicians can teach abstraction by making it, rather than the math itself, the subject.

How to Think Abstractly

Abstraction is what I'm good at.

How did I become good at it?

Abstraction is at the heart of mathematics.

I don't know how you should learn to be better at abstraction.

Perhaps mathematicians can teach abstraction by making it, rather than the math itself, the subject.

Try asking them.

Things to Remember

Things to Remember

Programming should be thinking followed by coding.

Things to Remember

Programming should be thinking followed by coding.

Thinking requires writing.

Things to Remember

Programming should be thinking followed by coding.

Thinking requires writing.

If the program is simple, very little writing is necessary.

Things to Remember

Programming should be thinking followed by coding.

Thinking requires writing.

If the program is simple, very little writing is necessary.
But it takes thinking to know if it's simple.

Things to Remember

Programming should be thinking followed by coding.

Thinking requires writing.

If the program is simple, very little writing is necessary.

For non-simple programs, abstract thinking (above the code level) can avoid errors and lead to better, easier to write code.

Things to Remember

Programming should be thinking followed by coding.

Thinking requires writing.

If the program is simple, very little writing is necessary.

For non-simple programs, abstract thinking (above the code level) can avoid errors and lead to better, easier to write code.

A non-simple program can be anything from a complete system to a complicated loop.

Things to Remember

No way of abstracting is best for all programs.

Things to Remember

No way of abstracting is best for all programs.

The abstraction of an execution as a sequence of states is often a very good one.

Things to Remember

No way of abstracting is best for all programs.

The abstraction of an execution as a sequence of states is often a very good one.

A state should contain all the information that can affect what future states are possible.

Things to Remember

No way of abstracting is best for all programs.

The abstraction of an execution as a sequence of states is often a very good one.

A state should contain all the information that can affect what future states are possible.

In this abstraction, a program does the right thing because it satisfies an invariant.

Things to Remember

No way of abstracting is best for all programs.

The abstraction of an execution as a sequence of states is often a very good one.

A state should contain all the information that can affect what future states are possible.

In this abstraction, a program does the right thing because it satisfies an invariant.

Understanding the program requires understanding that invariant.

Things to Remember

Don't get hung up on languages.

Things to Remember

Don't get hung up on languages.

Especially not on programming languages.

A Postscript

Why Programs Should Have Bugs

Why Programs Should Have Bugs

I started programming in 1957.

Why Programs Should Have Bugs

I started programming in 1957.

We can write much more complex programs now.

Why Programs Should Have Bugs

I started programming in 1957.

We can write much more complex programs now.

Part of the reason is better programming languages.

Why Programs Should Have Bugs

I started programming in 1957.

We can write much more complex programs now.

Part of the reason is better programming languages.

But the major reason is that we have libraries of programs our programs can use.

The hardest part of programming is now figuring out how to use those library programs,

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

Part of that description is language dependent.

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

Part of that description is language dependent.

  That part is usually simple.

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

Part of that description is language dependent.

That part is usually simple.

It's often implied by how the program is called

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

Part of that description is language dependent.

- That part is usually simple.
- It's often implied by how the program is called, especially for strongly-typed languages.

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

Part of that description is language dependent.

But the most useful programs do more complex things.

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

Part of that description is language dependent.

But the most useful programs do more complex things.

You shouldn't have to read the code to understand those things.

The hardest part of programming is now figuring out how to use those library programs, because many of them can never have bugs.

A program can have bugs only if there is a precise description of what it should do.

Part of that description is language dependent.

But the most useful programs do more complex things.

You shouldn't have to read the code to understand those things.

They should have an abstract, language-independent description.

Few programs have such a description.

Few programs have such a description.

Many have no description at all.

Few programs have such a description.

I won't bother giving you horror stories of how this has made it difficult or impossible for me to use some library programs.

Few programs have such a description.

I won't bother giving you horror stories of how this has made it difficult or impossible for me to use some library programs.

Instead, I'll tell you about an organization that did a pretty good job of providing precise descriptions.

The W3C JavaScript Standard

W3C = World Wide Web Consortium

The W3C JavaScript Standard

There has been a lot of work on verifying that what a program should do is implied by how it does it.

The W3C JavaScript Standard

There has been a lot of work on verifying that what a program should do is implied by how it does it.

Initiated by a 1967 paper of Robert Floyd.

The W3C JavaScript Standard

There has been a lot of work on verifying that what a program should do is implied by how it does it.

Most of it views what a program should do as a relation between its inputs and its outputs.

The W3C JavaScript Standard

There has been a lot of work on verifying that what a program should do is implied by how it does it.

Most of it views what a program should do as a relation between its inputs and its outputs.

That is the view used by the W3C standard.

The W3C JavaScript Standard

There has been a lot of work on verifying that what
a program should do is implied by how it does it.

Most of it views what a program should do as
a relation between its inputs and its outputs.

That is the view used by the W3C standard.

This view works fine for sequential programs.

The W3C JavaScript Standard

There has been a lot of work on verifying that what a program should do is implied by how it does it.

Most of it views what a program should do as a relation between its inputs and its outputs.

That is the view used by the W3C standard.

This view works fine for sequential programs, and I expect most JavaScript programs are sequential.

In 2016 I produced a video course about TLA$^+$,
with web pages for viewing it.

In 2016 I produced a video course about TLA$^+$,
with web pages for viewing it.

Users interact with a JavaScript program.

In 2016 I produced a video course about TLA^+,
with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

In 2016 I produced a video course about TLA$^+$,
with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

The code controlling the video is executed by one thread.

In 2016 I produced a video course about TLA$^+$,
with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

The code controlling the video is executed by one thread.

The code handling mouse clicks is executed by a diferent thread.

In 2016 I produced a video course about TLA$^+$, with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

Executing a library program can change different parts of the state at different times.

In 2016 I produced a video course about TLA$^+$, with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

Executing a library program can change different parts of the state at different times.

The order in which those changes occur matters

In 2016 I produced a video course about TLA$^+$, with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

Executing a library program can change different parts of the state at different times.

The order in which those changes occur matters, but it can't be described by viewing what the program does as a relation between inputs and outputs.

In 2016 I produced a video course about TLA$^+$,
with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

Executing a library program can change different
parts of the state at different times.

The order in which those changes occur matters.

The hard part of writing the program was figuring out how to get
the library programs to interract correctly on all popular browsers.

In 2016 I produced a video course about TLA$^+$, with web pages for viewing it.

Users interact with a JavaScript program.

It's a concurrent program.

Executing a library program can change different parts of the state at different times.

The order in which those changes occur matters.

The hard part of writing the program was figuring out how to get the library programs to interract correctly on all popular browsers.

It required a lot of debugging.

It seems to work correctly, but there's no way to be sure that it will keep working correctly.

It seems to work correctly, but there's no way to be sure that it will keep working correctly.

We should be able to do better than that.

“That’s all folks!”