Temporal Logic:
The Lesser of Three Evils

Leslie Lamport
Microsoft Research

The evil that men do lives after them.

Julius Caesar, by William Shakespeare

Where I Started

Where I Started

Making sure my concurrent algorithms were right.

Where I Started

Making sure my concurrent algorithms were right.

Proving the Correctness of Multiprocess Programs
(*IEEE TSE*, 1977)

Where I Started

Making sure my concurrent algorithms were right.

Proving the Correctness of Multiprocess Programs
(*IEEE TSE*, 1977)

Proved:

Safety Properties: Invariance

Where I Started

Making sure my concurrent algorithms were right.

Proving the Correctness of Multiprocess Programs
(*IEEE TSE*, 1977)

Proved:

Safety Properties: Invariance

Liveness Properties: $P \leadsto Q$

My Introduction to Temporal Logic

My Introduction to Temporal Logic

In 1977–78, Susan Owicki started a little seminar on Amir's 1977 FOCS paper.

My Introduction to Temporal Logic

In 1977–78, Susan Owicki started a little seminar on Amir's 1977 FOCS paper.

It sounded like formal nonsense to me, but I attended anyway.

My Introduction to Temporal Logic

In 1977–78, Susan Owicki started a little seminar on Amir's 1977 FOCS paper.

It sounded like formal nonsense to me, but I attended anyway.

I discovered that:

It was simple: One primitive temporal operator $\Box$

My Introduction to Temporal Logic

In 1977–78, Susan Owicki started a little seminar on Amir's 1977 FOCS paper.

It sounded like formal nonsense to me, but I attended anyway.

I discovered that:

It was simple: One primitive temporal operator $\Box$

$$\Diamond \triangleq \neg\Box\neg$$

My Introduction to Temporal Logic

In 1977–78, Susan Owicki started a little seminar on Amir's 1977 FOCS paper.

It sounded like formal nonsense to me, but I attended anyway.

I discovered that:

It was simple: One primitive temporal operator $\Box$

$$\Diamond \triangleq \neg \Box \neg$$

It worked beautifully for liveness: $P \leadsto Q \triangleq \Box(P \Rightarrow \Diamond Q)$

My Introduction to Temporal Logic

In 1977–78, Susan Owicki started a little seminar on Amir's 1977 FOCS paper.

It sounded like formal nonsense to me, but I attended anyway.

I discovered that:

It was simple: One primitive temporal operator $\Box$

$$\Diamond \triangleq \neg\Box\neg$$

It worked beautifully for liveness: $P \leadsto Q \triangleq \Box(P \Rightarrow \Diamond Q)$

Eventually, Susan and I wrote *Proving Liveness Properties of Concurrent Programs* (*TOPLAS*, 1982).

Specification

Specification

Around 1980, my colleagues and I started trying to write specifications.

Specification

Around 1980, my colleagues and I started trying to write specifications.

Instead of stating some properties about an algorithm, say exactly what it has to do.

Specification

Around 1980, my colleagues and I started trying to write specifications.

Instead of stating some properties about an algorithm, say exactly what it has to do.

Write the properties an algorithm/system/protocol should have.

Temporal logic seemed ideal for this.

Temporal logic seemed ideal for this.

We had been using an exogenous logic:

Temporal logic seemed ideal for this.

We had been using an exogenous logic:

$\models F$ (validity of F) depends on underlying system.

Temporal logic seemed ideal for this.

We had been using an exogenous logic:

$\models F$ (validity of F) depends on underlying system.

Just had to switch to an endogenous logic:

Temporal logic seemed ideal for this.

We had been using an exogenous logic:

$\models F$ (validity of F) depends on underlying system.

Just had to switch to an endogenous logic:

Single notion of $\models$.

Temporal logic seemed ideal for this.

We had been using an exogenous logic:

$\models F$ (validity of F) depends on underlying system.

Just had to switch to an endogenous logic:

Single notion of $\models$.

System specified by temporal logic formula S

Temporal logic seemed ideal for this.

We had been using an exogenous logic:

$\models F$ (validity of F) depends on underlying system.

Just had to switch to an endogenous logic:

Single notion of $\models$.

System specified by temporal logic formula S

$\models F$ becomes $\models S \Rightarrow F$

It Didn’t Work!

It Didn’t Work!

My colleagues spent days unsuccessfully trying to specify a FIFO queue.

It Didn’t Work!

My colleagues spent days unsuccessfully trying to specify a FIFO queue.

The reason was obvious: the simple logic of Amir’s 1977 paper was not expressive enough.

It Didn't Work!

My colleagues spent days unsuccessfully trying to specify a FIFO queue.

The reason was obvious: the simple logic of Amir's 1977 paper was not expressive enough.

An arms race ensued. Who could invent the biggest, most powerful temporal logic?

It Didn't Work!

My colleagues spent days unsuccessfully trying to specify a FIFO queue.

The reason was obvious: the simple logic of Amir's 1977 paper was not expressive enough.

An arms race ensued. Who could invent the biggest, most powerful temporal logic?

I was not immune:

TIMESETS — A New Method for Temporal Reasoning About Programs

(in *LNCS 131*, 1981)

The Real Problem

The Real Problem

Writing a specification as a list of properties doesn't work.

The Real Problem

Writing a specification as a list of properties doesn't work.

No one can understand the consequences of a list of properties.

An Example: Weak Memory Models

An Example: Weak Memory Models

Typically specified by axioms.

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Alpha memory specification model allowed this:

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Alpha memory specification model allowed this:

Initially: $x = y = 0$

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Alpha memory specification model allowed this:

Initially: $x = y = 0$

Process 1: **if** $x = 23$ **then** $y := 42$

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Alpha memory specification model allowed this:

Initially: $x = y = 0$

Process 1: **if** $x = 23$ **then** $y := 42$

Process 2: **if** $y = 42$ **then** $x := 23$

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Alpha memory specification model allowed this:

Initially: $x = y = 0$

Process 1: **if** $x = 23$ **then** $y := 42$

Process 2: **if** $y = 42$ **then** $x := 23$

After execution: $x = 23$, $y = 42$

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Itanium memory specification document.

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Itanium memory specification document.

We wrote a TLA$^+$ specification and used our tools to check the document's tiny examples.

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

The original Itanium memory specification document.

We wrote a TLA$^+$ specification and used our tools to check the document's tiny examples.

We found several errors.

An Example: Weak Memory Models

Typically specified by axioms.

Even their designers don't understand them.

No one can figure out from a list of axioms
what a tiny bit of concurrent code can do.

What works

What works

Specify liveness with Amir's original temporal logic.

What works

Specify liveness with Amir's original temporal logic.

Specify safety by a state machine (abstract program).

What works

Specify liveness with Amir's original temporal logic.

Specify safety by a state machine (abstract program).

How to do this in temporal logic:

What works

Specify liveness with Amir's original temporal logic.

Specify safety by a state machine (abstract program).

How to do this in temporal logic:

Generalize Amir's temporal logic.

What works

Specify liveness with Amir's original temporal logic.

Specify safety by a state machine (abstract program).

How to do this in temporal logic:

Generalize Amir's temporal logic.

Don't add new temporal operators.

What works

Specify liveness with Amir's original temporal logic.

Specify safety by a state machine (abstract program).

How to do this in temporal logic:

- Generalize Amir's temporal logic.
- Don't add new temporal operators.
- Do generalize elementary formulas

What works

Specify liveness with Amir's original temporal logic.

Specify safety by a state machine (abstract program).

How to do this in temporal logic:

Generalize Amir's temporal logic.

Don't add new temporal operators.

Do generalize elementary formulas from state predicates to transition predicates.

What works

Specify liveness with Amir's original temporal logic.

Specify safety by a state machine (abstract program).

How to do this in temporal logic:

Generalize Amir's temporal logic.

Don't add new temporal operators.

Do generalize elementary formulas from state predicates to transition predicates.

But that's another story.

What is Evil About Temporal Logic

What is Evil About Temporal Logic

A fundamental rule of ordinary math: to prove $A \Rightarrow B$, we assume A and prove B.

What is Evil About Temporal Logic

A fundamental rule of ordinary math: to prove $A \Rightarrow B$, we assume A and prove B.

The Deduction Principle:

$$\frac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

What is Evil About Temporal Logic

A fundamental rule of ordinary math: to prove $A \Rightarrow B$, we assume A and prove B.

The Deduction Principle:

$$\frac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

The deduction principle is not valid for temporal logic (and other modal logics).

What is Evil About Temporal Logic

A fundamental rule of ordinary math: to prove $A \Rightarrow B$, we assume A and prove B.

The Deduction Principle:

$$\frac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

The deduction principle is not valid for temporal logic (and other modal logics).

For example, a basic rule of temporal logic asserts that if P is true then it is always true.

$$\frac{P}{\Box P}$$

From

$$\frac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

From

$$\dfrac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

and

$$\dfrac{P}{\Box P}$$

From

$$\frac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

and

$$\frac{P}{\Box P}$$

by substituting $\Box P$ for Q we deduce

$$P \Rightarrow \Box P$$

From

$$\frac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

and

$$\frac{P}{\Box P}$$

by substituting $\Box P$ for Q we deduce

$$P \Rightarrow \Box P$$

which asserts that if P is true now then it is always true.

From

$$\frac{\dfrac{P}{Q}}{P \Rightarrow Q}$$

and

$$\frac{P}{\Box P}$$

NOT

by substituting $\Box P$ for Q we deduce

$$P \Rightarrow \Box P$$

which asserts that if P is true now then it is always true.

In modal logics, implication ($P \Rightarrow Q$) and inference ($\frac{P}{Q}$) are different.

In modal logics, implication ($P \Rightarrow Q$) and inference ($\frac{P}{Q}$) are different.

This is confusing.

In modal logics, implication ($P \Rightarrow Q$) and inference ($\frac{P}{Q}$) are different.

This is confusing.

Martín Abadi and I once believed a false result for several days because this confused us.

In modal logics, implication ($P \Rightarrow Q$) and inference ($\frac{P}{Q}$) are different.

This is confusing.

Martín Abadi and I once believed a false result for several days because this confused us.

A logic that can confuse Martín is evil.

Greater Evil #1

Temporal logic is modal because it has an implicit *time* variable.

Greater Evil #1

Temporal logic is modal because it has an implicit *time* variable.

A solution: make time explicit.

Greater Evil #1

Temporal logic is modal because it has an implicit *time* variable.

A solution: make time explicit.

For example: $P \leadsto Q$ becomes $\forall\, t : (P(t) \Rightarrow \exists\, s \geq t : Q(s))$.

Greater Evil #1

Temporal logic is modal because it has an implicit *time* variable.

A solution: make time explicit.

For example: $P \leadsto Q$ becomes $\forall\, t : (P(t) \Rightarrow \exists\, s \geq t : Q(s))$.

This makes formulas ugly and hard to understand.

Greater Evil #1

Temporal logic is modal because it has an implicit *time* variable.

A solution: make time explicit.

For example: $P \leadsto Q$ becomes $\forall\, t : (P(t) \Rightarrow \exists\, s \geq t : Q(s))$.

This makes formulas ugly and hard to understand.

Trying to eliminate this is what led Amir to temporal logic.

Greater Evil #1

Temporal logic is modal because it has an implicit *time* variable.

A solution: make time explicit.

For example: $P \leadsto Q$ becomes $\forall\, t : (P(t) \Rightarrow \exists\, s \geq t : Q(s))$.

This makes formulas ugly and hard to understand.

Trying to eliminate this is what led Amir to temporal logic.

(He was inspired by Nissim Francez's thesis.)

Greater Evil #2

Use a programming logic.

Greater Evil #2

Use a programming logic.

Some programming logics:

Greater Evil #2

Use a programming logic.

Some programming logics:

Hoare Logic (Tony Hoare 1968)

Greater Evil #2

Use a programming logic.

Some programming logics:

Hoare Logic (Tony Hoare 1968)

Dynamic Logic (Vaughan Pratt 1974)

Greater Evil #2

Use a programming logic.

Some programming logics:

Hoare Logic (Tony Hoare 1968)

Dynamic Logic (Vaughan Pratt 1974)

Weakest Preconditions (Edsger Dijkstra 1975)

Greater Evil #2

Use a programming logic.

Some programming logics:

Hoare Logic (Tony Hoare 1968)

Dynamic Logic (Vaughan Pratt 1974)

Weakest Preconditions (Edsger Dijkstra 1975)

Action Systems (Ralph Back ∼1983)

Greater Evil #2

Use a programming logic.

Some programming logics:

Hoare Logic (Tony Hoare 1968)

Dynamic Logic (Vaughan Pratt 1974)

Weakest Preconditions (Edsger Dijkstra 1975)

Action Systems (Ralph Back $\sim$1983)

What they have in common:

programs appear in formulas of the “logic”.

Greater Evil #2

Use a programming logic.

Some programming logics:

Hoare Logic (Tony Hoare 1968)

Dynamic Logic (Vaughan Pratt 1974)

Weakest Preconditions (Edsger Dijkstra 1975)

Action Systems (Ralph Back $\sim$1983)

What they have in common:

programs appear in formulas of the “logic”.

Why are they evil?

Greater Evil #2

Use a programming logic.

Some programming logics:

Hoare Logic (Tony Hoare 1968)

Dynamic Logic (Vaughan Pratt 1974)

Weakest Preconditions (Edsger Dijkstra 1975)

Action Systems (Ralph Back ∼1983)

What they have in common:

programs appear in formulas of the "logic".

Why are they evil? First a digression.

Program 1:

Program 1:

initially $x = 0$

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1 \text{ mod } 2$

end while

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1 \bmod 2$

end while

Program 2:

initially $p = q = 0$

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1 \bmod 2$

end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$;

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1$ mod 2

end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ;

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1$ mod 2
end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1$ mod 2
end while

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1$ mod 2

end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1$ mod 2
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons* ; $q := q + 1$ mod 2
end while

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1 \bmod 2$

end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1 \bmod 2$
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons* ; $q := q + 1 \bmod 2$
end while

Two-Phase Handshake, an important hardware protocol

We can derive Program 2 from Program 1 by substituting $p + q$ mod 2 for x .

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1$ mod 2
end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1$ mod 2
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons* ; $q := q + 1$ mod 2
end while

We can derive Program 2 from Program 1 by substituting $p + q$ mod 2 for x. See festschrift for Willem-Paul de Roever.

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1$ mod 2
end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod*; $p := p + 1$ mod 2
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons*; $q := q + 1$ mod 2
end while

A derivation is a refinement proof run backwards.

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1 \bmod 2$
end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1 \bmod 2$
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons* ; $q := q + 1 \bmod 2$
end while

A derivation is a refinement proof run backwards.

Refinement is substitution.

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1 \bmod 2$
end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1 \bmod 2$
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons* ; $q := q + 1 \bmod 2$
end while

How do you substitute $p + q$ mod 2 for x in a program?

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1$ mod 2
end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1$ mod 2
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons* ; $q := q + 1$ mod 2
end while

How do you substitute $p + q$ mod 2 for x in a program?

It can't be done.

Program 1:

initially $x = 0$

while TRUE **do** **if** $x = 0$ **then** $Prod$ **else** $Cons$ **end if**;
$x := x + 1$ mod 2

end while

Program 2:

initially $p = q = 0$

Process 1: **while** TRUE **do** **await** $p = q$; *Prod* ; $p := p + 1$ mod 2
end while

Process 2: **while** TRUE **do** **await** $p \neq q$; *Cons* ; $q := q + 1$ mod 2
end while

Why Programming Logics are Evil

Substitution of an expression for a variable is a fundamental operation of mathematics.

Why Programming Logics are Evil

Substitution of an expression for a variable is a fundamental operation of mathematics.

A logic that doesn't permit substitution is evil.

Why Programming Logics are Evil

Substitution of an expression for a variable is a fundamental operation of mathematics.

A logic that doesn't permit substitution is evil.

Program refinement is based on substitution.

Why Programming Logics are Evil

Substitution of an expression for a variable is a fundamental operation of mathematics.

A logic that doesn't permit substitution is evil.

Program refinement is based on substitution.

A programming logic that doesn't permit substitution is especially evil.

Why Programming Logics are Evil

Substitution of an expression for a variable is a fundamental operation of mathematics.

A logic that doesn't permit substitution is evil.

Program refinement is based on substitution.

A programming logic that doesn't permit substitution is especially evil.

Refinement by substitution is not a problem with temporal logic.

Why Programming Logics are Evil

Substitution of an expression for a variable is a fundamental operation of mathematics.

A logic that doesn't permit substitution is evil.

Program refinement is based on substitution.

A programming logic that doesn't permit substitution is especially evil.

Refinement by substitution is not a problem with temporal logic.

Temporal logic is a lesser evil.

A Necessary Evil

Temporal logic is the best way I know of to reason about systems .

A Necessary Evil

Temporal logic is the best way I know of to reason about systems—especially for liveness properties.

A Necessary Evil

Temporal logic is the best way I know of to reason about systems—especially for liveness properties.

Someone as good as Amir would not have done anything evil unless it was necessary.

A Necessary Evil

Temporal logic is the best way I know of to reason about systems—especially for liveness properties.

Someone as good as Amir would not have done anything evil unless it was necessary.

We are all grateful that he did it.

A Necessary Evil

Temporal logic is the best way I know of to reason about systems—especially for liveness properties.

Someone as good as Amir would not have done anything evil unless it was necessary.

We are all grateful that he did it.

I am grateful that I had the privilege of being his colleague.

A Necessary Evil

Temporal logic is the best way I know of to reason about systems—especially for liveness properties.

Someone as good as Amir would not have done anything evil unless it was necessary.

We are all grateful that he did it.

I am grateful that I had the privilege of being his colleague.

He was a great scientist and a wonderful human being.

Thank you.