This box should lie exactly on top of the Camtasia video area frame.
This is a blank screen.
What is a hyperproperty? ©
An ordinary property is a predicate that’s true or false of a single execution of a system.

For example, the property that every request receives a response.

Verifying that a system satisfies a property means showing that every execution of the system satisfies the property.
Hyperproperties

Ordinary property: True or false of an execution.

© An ordinary property is a predicate that’s true or false of a single execution of a system. ©

For example, the property that every request receives a response. ©

Verifying © that a system satisfies a property © means showing that every execution of the system satisfies the property. ©
Hyperproperties

Ordinary property: True or false of an execution.

Example: Every request receives response.

© An ordinary property is a predicate that’s true or false of a single execution of a system. ©

For example, the property that every request receives a response. ©

Verifying © that a system satisfies a property © means showing that every execution of the system satisfies the property. ©
Hyperproperties

Ordinary property: True or false of an execution.

Verification:

© An ordinary property is a predicate that’s true or false of a single execution of a system. ©

For example, the property that every request receives a response. ©

Verifying © that a system satisfies a property ©
means showing that every execution of the system satisfies the property. ©
Hyperproperties

Ordinary property: True or false of an execution.

Verification:

\[ S \models P \]

system property

© An ordinary property is a predicate that’s true or false of a single execution of a system. ©

For example, the property that every request receives a response. ©

Verifying © that a system satisfies a property © means showing that every execution of the system satisfies the property. ©
Hyperproperties

Ordinary property: True or false of an execution.

Verification:

\[ S \models P \quad \text{means} \quad \sigma \models P \]

- system property
- any execution of system property

© An ordinary property is a predicate that’s true or false of a single execution of a system. ©

For example, the property that every request receives a response. ©

Verifying that a system satisfies a property ©
means showing that every execution of the system satisfies the property. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

A hyperproperty is a predicate that’s true or false on the set of executions of a system, not just on single executions.

Some security conditions are naturally expressed as hyperproperties—for example,

Observational Determinism or OD. OD assumes that an execution is a sequence of states, and a state consists of two parts: a public state and a secret state.
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

A hyperproperty is a predicate that’s true or false on the set of executions of a system, not just on single executions.

Some security conditions are naturally expressed as hyperproperties—for example:

Observational Determinism or OD. OD assumes that an execution is a sequence of states, and a state consists of two parts: a public state and a secret state.
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinism)

A hyperproperty is a predicate that’s true or false on the set of executions of a system, not just on single executions.

Some security conditions are naturally expressed as hyperproperties—for example:

Observational Determinism or OD. OD assumes that an execution is a sequence of states, and a state consists of two parts: a public state and a secret state.
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinism)

\[ \text{state}_1 \rightarrow \text{state}_2 \rightarrow \text{state}_3 \rightarrow \cdots \]

A hyperproperty is a predicate that’s true or false on the set of executions of a system, not just on single executions.

Some security conditions are naturally expressed as hyperproperties—for example,

Observational Determinism or OD. OD assumes that an execution is a sequence of states, and a state consists of two parts: a public state and a secret state.
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinism)

\[ \text{state}_1 \rightarrow \text{state}_2 \rightarrow \text{state}_3 \rightarrow \cdots \]

\((4, “foo”)\)

A hyperproperty is a predicate that’s true or false on the set of executions of a system, not just on single executions.©

Some security conditions are naturally expressed as hyperproperties—for example©

Observational Determinism or OD. OD assumes that an execution is a © sequence of states, and a state © consists of two parts:© a public state and a secret state. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinism)

\[ \text{state}_1 \rightarrow \text{state}_2 \rightarrow \text{state}_3 \rightarrow \cdots \]

\( \text{(4,"foo")} \)

\( \text{public} \quad \text{secret} \)

a public state and a secret state. ©

OD requires that if any two executions © have the same initial public states, then they © always have the same public states. ©

This is an assertion about pairs of executions, not about a single execution. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinism)

\[
\text{state}_1 \rightarrow \text{state}_2 \rightarrow \text{state}_3 \rightarrow \cdots
\]

\[
\text{state}_a \rightarrow \text{state}_b \rightarrow \text{state}_c \rightarrow \cdots
\]

a public state and a secret state.

OD requires that if any two executions have the same initial public states, then they always have the same public states.

This is an assertion about pairs of executions, not about a single execution.
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinism)

\[
\text{state}_1 \rightarrow \text{state}_2 \rightarrow \text{state}_3 \rightarrow \cdots
\]

\[
\uparrow
\]

public =

\[
\downarrow
\]

\[
\text{state}_a \rightarrow \text{state}_b \rightarrow \text{state}_c \rightarrow \cdots
\]

a public state and a secret state. ©

OD requires that if any two executions © have the same initial public states, then they © always have the same public states. ©

This is an assertion about pairs of executions, not about a single execution. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinisim)

\[
\begin{align*}
\text{state}_1 & \rightarrow \text{state}_2 & \rightarrow \text{state}_3 & \rightarrow \cdots \\
\uparrow & \uparrow & \uparrow \\
\text{public} = & \text{public} = & \text{public} = & \cdots \\
\downarrow & \downarrow & \downarrow \\
\text{state}_a & \rightarrow \text{state}_b & \rightarrow \text{state}_c & \rightarrow \cdots 
\end{align*}
\]

a public state and a secret state. ©

OD requires that if any two executions © have the same initial public states, then they © always have the same public states. ©

This is an assertion about pairs of executions, not about a single execution. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

OD (Observational Determinism)

\[ \text{state}_1 \rightarrow \text{state}_2 \rightarrow \text{state}_3 \rightarrow \cdots \]
\[ \uparrow \quad \uparrow \quad \uparrow \quad \cdots \]
\[ \text{public} = \quad \text{public} = \quad \text{public} = \quad \cdots \]
\[ \downarrow \quad \downarrow \quad \downarrow \quad \cdots \]
\[ \text{state}_a \rightarrow \text{state}_b \rightarrow \text{state}_c \rightarrow \cdots \]

an assertion about pairs of executions

da public state and a secret state. ©

OD requires that if any two executions © have the same initial public states, then they © always have the same public states. ©

This is an assertion about pairs of executions, not about a single execution. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

Another example is GNI (short for Generalized NonInterference). It assumes that an execution is a sequence of public and secret events. It’s a way of saying that the public events give you no information about the secret events.

For any two possible system executions
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

\[
\text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots
\]

Another example is GNI (short for Generalized NonInterference). It assumes that an execution is a sequence of public and secret events. It’s a way of saying that the public events give you no information about the secret events. For any two possible system executions
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

\[ \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \]

Public events must reveal nothing about secret events.

Another example is GNI (short for Generalized NonInterference). It assumes that an execution is a sequence of public and secret events. It’s a way of saying that the public events give you no information about the secret events.

For any two possible system executions
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

\[
\text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_a \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots
\]

Another example is GNI (short for Generalized NonInterference). It assumes that an execution is a sequence of public and secret events. It's a way of saying that the public events give you no information about the secret events.

For any two possible system executions
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

\[
\text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots
\]

\[
\text{public}_a \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots
\]

For any two possible system executions ©
GNI requires that you can get a possible system execution by combining ©
the public events of the first © with the secret events of the second. © ©

Again, it’s an assertion about more than one execution. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

For any two possible system executions GNI requires that you can get a possible system execution by combining the public events of the first with the secret events of the second.

Again, it’s an assertion about more than one execution.
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

\[
\begin{align*}
\text{public}_1 & \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_1 & \rightarrow \text{secret}_a \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_a & \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots
\end{align*}
\]

For any two possible system executions ©

GNI requires that you can get a possible system execution by combining © the public events of the first © with the secret events of the second. © ©

Again, it’s an assertion about more than one execution. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

\[
\begin{align*}
\text{public}_1 & \rightarrow \text{secret}_1 & \rightarrow \text{secret}_2 & \rightarrow \text{public}_2 & \rightarrow \cdots \\
\text{public}_1 & \rightarrow \text{secret}_a & \rightarrow \text{public}_2 & \rightarrow \cdots \\
\text{public}_a & \rightarrow \text{secret}_a & \rightarrow \text{public}_b & \rightarrow \text{public}_c & \rightarrow \cdots
\end{align*}
\]

For any two possible system executions ©
GNI requires that you can get a possible system execution by combining ©
the public events of the first © with the secret events of the second. © ©

Again, it’s an assertion about more than one execution. ©
Hyperproperties

Hyperproperty: True or false of a set of executions.

Some security conditions are hyperproperties.

GNI (Generalized NonInterference)

\[
\begin{align*}
\text{public}_1 & \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_1 & \rightarrow \text{secret}_a \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_a & \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots
\end{align*}
\]

For any two possible system executions ☑
GNI requires that you can get a possible system execution by combining ☑ the public events of the first ☑ with the secret events of the second. ☑ ☑

Again, it’s an assertion about more than one execution. ☑
Hyperproperties

Verification:

How do we verify that a system satisfies a hyperproperty?
Hyperproperties

Verification:

\[ S \models H \]

- system
- hyperproperty

How do we verify that a system satisfies a hyperproperty?
Hyperproperties

Verification:

\[ S \models H \]

system \hspace{1cm} hyperproperty

How do we verify that a system satisfies a hyperproperty?
Verifying Properties

\[ S \models P \]

Verifying ordinary properties has been well-studied. We want to make use of methods and tools developed to solve it. So people have reduced verifying hyperproperties to verifying ordinary properties. Here’s how. Define two mappings.
Verifying ordinary properties has been well-studied.

We want to make use of methods and tools developed to solve it.

So people have reduced verifying hyperproperties to verifying ordinary properties. Here’s how.

Define two mappings.

[slide 33]
Verifying Properties

\[ S \models P \]

A well-studied problem.

We want to use its solutions.

Verifying ordinary properties \(\square\) has been well-studied. \(\square\)

We want to make use of methods and tools developed to solve it. \(\square\)

So people have reduced verifying hyperproperties to verifying ordinary properties. Here’s how. \(\square\) Define two mappings. \(\square\)
Verifying Hyperproperties by Verifying Properties

\[ S \models P \]
A well-studied problem.
We want to use its solutions.

Verifying ordinary properties has been well-studied.

We want to make use of methods and tools developed to solve it.

So people have reduced verifying hyperproperties to verifying ordinary properties. Here’s how.

Define two mappings.
Define two mappings.

The first maps a system $S$ to another system $\Omega(S)$. 

The second maps a hyperproperty $H$ to an ordinary property $H$-tilde. 

These mappings are defined so that system $S$ satisfies hyperproperty $H$ if and only if the system $\Omega(S)$ satisfies the ordinary property $H$-tilde.
Define two mappings.

The first maps a system $S$ to another system $\Omega(S)$.  

The second maps a hyperproperty $H$ to an ordinary property $H$-tilde.  

These mapping are defined so that system $S$ satisfies hyperproperty $H$ if and only if the system $\Omega(S)$ satisfies the ordinary property $H$-tilde.
Define two mappings. ©

The first maps a system $S$ to another system $\Omega(S)$. ©

The second maps a hyperproperty $H$ to an ordinary property $\tilde{H}$. ©

These mappings are defined so that © system $S$ satisfies hyperproperty $H$ © if and only if the system $\Omega(S)$ satisfies the ordinary property $\tilde{H}$. ©
Define two mappings.©

The first maps a system $S$ to another system $\Omega(S)$. ©

The second maps a hyperproperty $H$ to an ordinary property $\tilde{H}$-tilde. ©

These mapping are defined so that © system $S$ satisfies hyperproperty $H$ © if and only if the system $\Omega(S)$ satisfies the ordinary property $\tilde{H}$-tilde. ©
Verifying Hyperproperties by Verifying Properties

Two mappings: $\Omega : S \rightarrow \Omega(S)$

$\tilde{H} : H \rightarrow \tilde{H}$

Such that: $S \models H$

Define two mappings. The first maps a system $S$ to another systems $\Omega(S)$. The second maps a hyperproperty $H$ to an ordinary property $H$-tilde. These mapping are defined so that system $S$ satisfies hyperproperty $H$ if and only if the system $\Omega(S)$ satisfies the ordinary property $H$-tilde.
Define two mappings. ©

The first maps a system $S$ to another systems $\Omega(S)$. ©

The second maps a hyperproperty $H$ to an ordinary property $\tilde{H}$. ©

These mapping are defined so that © system $S$ satisfies hyperproperty $H$ if and only if the system $\Omega(S)$ satisfies the ordinary property $\tilde{H}$. ©
Verification has been done this way with a method called self-composition where if hyperproperty $H$ is an assertion about $n$ executions then $\Omega(S)$ is a big system that executes $n$ copies of $S$ in lock-step and $H$ tilde is $H$ restated in terms of executions of the individual processes $S$ in an execution of $\Omega(S)$.
Self-Composition

If $H$ is an assertion about $n$ executions

Verification has been done this way with a method called self-composition

where if hyperproperty $H$ is an assertion about $n$ executions

then $\Omega(S)$ is a big system that executes $n$ copies of $S$ in lock-step

and $H$ tilde is $H$ restated in terms of executions of the individual processes $S$ in an execution of $\Omega(S)$. 

[slide 43]
Self-Composition

If $H$ is an assertion about $n$ executions

then $\Omega(S) = S \otimes S \ldots \otimes S$

executed in lock-step

Verification has been done this way with a method called self-composition where if hyperproperty $H$ is an assertion about $n$ executions then $\Omega(S)$ is a big system that executes $n$ copies of $S$ in lock-step and $H$ tilde is $H$ restated in terms of executions of the individual processes $S$ in an execution of $\Omega(S)$.
Self-Composition

If $H$ is an assertion about $n$ executions

then $\Omega(S) = S \otimes S \ldots \otimes S$

$\tilde{H} = H$ stated in terms of executions of individual processes $S$

Verification has been done this way with a method called self-composition where if hyperproperty $H$ is an assertion about $n$ executions then $\Omega(S)$ is a big system that executes $n$ copies of $S$ in lock-step and $H$ tilde is $H$ restated in terms of executions of the individual processes $S$ in an execution of $\Omega(S)$. [slide 45]
Self-Composition

Example: OD

For example, suppose the hyperproperty $H$ is Observational Determinism. Then $\Omega(S)$ consists of two copies of $S$ run in lock step, and $H$ tilde asserts that, if those two copies of $S$ start with equal public states, then they will always have equal public states.
Self-Composition

Example: OD

\[ \Omega(S) = S \otimes S \]

For example, suppose the hyperproperty \( H \) is Observational Determinism. Then \( \Omega(S) \) consists of two copies of \( S \) run in lock step, and \( H \) tilde asserts that, if those two copies of \( S \) start with equal public states, then they will always have equal public states.
Self-Composition

Example: OD

\[ \Omega(S) = S \otimes S \]

\[ \widetilde{\text{OD}} = \text{If the copies of } S \text{ start with equal public states} \]

For example, suppose the hyperproperty $H$ is Observational Determinism. Then $\Omega(S)$ consists of two copies of $S$ run in lock step, and $H$ tilde asserts that, if those two copies of $S$ start with equal public states, then they will always have equal public states.
Self-Composition

Example: OD

\[ \Omega(S) = S \otimes S \]

\[ \tilde{\text{OD}} = \text{If the copies of } S \text{ start with equal public states then they always have equal public states.} \]

For example, suppose the hyperproperty \( H \) is Observational Determinism. \( \copyright \) Then \( \Omega(S) \) consists of two copies of \( S \) run in lock step, and \( \copyright \) \( H \) tilde asserts that, if those two copies of \( S \) start with equal public states, \( \copyright \) then they will always have equal public states. \( \copyright \)
There’s a problem with this kind of Self-Composition. ©

It doesn’t work for some security hyperproperties, including GNI. ©

GNI says that, for any two behaviors of $S$ © there exists a 3rd behavior of $S$ satisfying a certain condition. ©

With self-composition ©
There’s a problem with this kind of Self-Composition. ©

It doesn’t work for some security hyperproperties, including GNI. ©

GNI says that, for any two behaviors of $S$ © there exists a 3rd behavior of $S$ satisfying a certain condition. ©

With self-composition ©
GNI: For any two behaviors,

\[ \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \]

\[ \text{public}_a \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots \]

There’s a problem with this kind of Self-Composition. ©
It doesn’t work for some security hyperproperties, including GNI. ©
GNI says that, for any two behaviors of \( S \) © there exists a 3rd behavior of \( S \) satisfying a certain condition. ©
With self-composition ©
Self-Composition – The Problem

GNI: For any two behaviors, there exists a 3rd behavior.

\[
\begin{align*}
&\text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
&\text{public}_1 \rightarrow \text{secret}_a \rightarrow \text{public}_2 \rightarrow \cdots \\
&\text{public}_a \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots
\end{align*}
\]

There’s a problem with this kind of Self-Composition. It doesn’t work for some security hyperproperties, including GNI. GNI says that, for any two behaviors of \( S \) there exists a 3rd behavior of \( S \) satisfying a certain condition. With self-composition
Self-Composition – The Problem

GNI: For any two behaviors, there exists a 3\(^{rd}\) behavior.

Have to verify:

\[
\begin{align*}
\text{public}_1 & \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_1 & \rightarrow \text{secret}_a \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_a & \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots 
\end{align*}
\]

With self-composition © these two behaviors © are described by this big system \(\Omega(S)\). ©

The 3rd behavior of \(S\) © is described by GNI, ©

So GNI-tilde must contain system \(S\), which it can’t because GNI-tilde is a property, and how can you put a system in a property?
Self-Composition – The Problem

GNI: For any two behaviors, there exists a 3rd behavior.

Have to verify:

\[
\begin{align*}
\text{public}_1 & \to \text{secret}_1 & \to \text{secret}_2 & \to \text{public}_2 & \to \cdots \\
\text{public}_1 & \to \text{secret}_a & \to \text{public}_2 & \to \cdots \\
\text{public}_a & \to \text{secret}_a & \to \text{public}_b & \to \text{public}_c & \to \cdots 
\end{align*}
\]

With self-composition these two behaviors are described by this big system \( \Omega(S) \). The 3rd behavior of \( S \) is described by GNI, So GNI-tilde must contain system \( S \), which it can’t because GNI-tilde is a property, and how can you put a system in a property?
Self-Composition – The Problem

GNI: For any two behaviors, there exists a 3\textsuperscript{rd} behavior.

Have to verify: \( S \otimes S \in \Omega(S) \)

\[
\begin{align*}
\text{public}_1 & \to \text{secret}_1 & \to \text{secret}_2 & \to \text{public}_2 & \to \cdots \\
\text{public}_1 & \to \text{secret}_a & \to \text{public}_2 & \to \cdots \\
\text{public}_a & \to \text{secret}_a & \to \text{public}_b & \to \text{public}_c & \to \cdots
\end{align*}
\]

With self-composition \( \circ \) these two behaviors \( \circ \) are described by this big system \( \Omega(S) \).

The 3\textsuperscript{rd} behavior of \( S \) \( \circ \) is described by GNI, \( \circ \)

So GNI-tilde must contain system \( S \), which it can’t because GNI-tilde is a property, and how can you put a system in a property?
Self-Composition – The Problem

GNI: For any two behaviors, there exists a 3rd behavior.

Have to verify: \( S \otimes S \)

\[
\begin{align*}
\text{public}_1 & \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_1 & \rightarrow \text{secret}_a \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_a & \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots
\end{align*}
\]

With self-composition \( \odot \) these two behaviors \( \odot \) are described by this big system \( \Omega(S) \).

The 3rd behavior of \( S \) \( \odot \) is described by GNI, \( \odot \)

So GNI-tilde must contain system \( S \), which it can’t because GNI-tilde is a property, and how can you put a system in a property?

[slide 57]
Self-Composition – The Problem

GNI: For any two behaviors, there exists a 3\textsuperscript{rd} behavior.

Have to verify: $S \otimes S \models \tilde{GNI}$

described by $GNI$

\[
\begin{align*}
\text{public}_1 & \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_1 & \rightarrow \text{secret}_a \rightarrow \text{public}_2 \rightarrow \cdots \\
\text{public}_a & \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots
\end{align*}
\]

With self-composition $\circ$ these two behaviors $\circ$ are described by this big system $\Omega(S)$. $\circ$

The 3\textsuperscript{rd} behavior of $S$ $\circ$ is described by GNI, $\circ$

So GNI-tilde must contain system $S$, which it can’t because GNI-tilde is a property, and how can you put a system in a property?

[slide 58] 3 min 51 sec
Self-Composition – The Problem

GNI: For any two behaviors, there exists a 3rd behavior.

Have to verify: \( S \otimes S \models \widehat{GNI} \)

\begin{align*}
\text{described by } GNI \\
\text{must contain another copy of system } S
\end{align*}

public_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots \\
public_1 \rightarrow \text{secret}_a \rightarrow \text{public}_2 \rightarrow \cdots \\
public_a \rightarrow \text{secret}_a \rightarrow \text{public}_b \rightarrow \text{public}_c \rightarrow \cdots

With self-composition \( \circ \) these two behaviors \( \circ \) are described by this big system \( \Omega(S) \). \( \circ \)

The 3rd behavior of \( S \) \( \circ \) is described by GNI, \( \circ \)

So GNI-tilde must contain system \( S \), which it can’t because GNI-tilde is a property, and how can you put a system in a property?

[slide 59] 3 min 51 sec
Here is our solution ©
Self-Composition – Our Solution

Works for GNI and . . .

Here is our solution © It works for some additional security properties, including GNI. ©
We use the temporal logic TLA. ©
**TLA** (temporal logic of actions)

Used by Microsoft, Amazon Web Services, Oracle, . . .

TLA has industrial-strength tools and is used by engineers who build large, distributed systems. ©

TLA describes systems, as well as properties, as formulas. ©

System $S$ satisfies property $P$ © means that the formula, $S$ implies $P$ is true. ©

The system obtained by running $n$ copies of $S$ in lock-step is defined as follows. ©

The definition begins with $S$ (of $x$-sub-1), which is the formula obtained by substituting a new set of variables, $x$-sub-1, for the variables of $S$. ©

[slide 63] 5 min 56 sec
Properties & systems are formulas.

TLA has industrial-strength tools and is used by engineers who build large, distributed systems. ©

TLA describes systems, as well as properties, as formulas. ©

System S satisfies property P © means that the formula, S implies P is true. ©

The system obtained by running n copies of S in lock-step is defined as follows. ©

The definition begins with S (of x-sub-1), which is the formula obtained by substituting a new set of variables, x-sub-1, for the variables of S. ©
Properties & systems are formulas.

\[
S \models P
\]

TLA has industrial-strength tools and is used by engineers who build large, distributed systems.

TLA describes systems, as well as properties, as formulas.

System \( S \) satisfies property \( P \) means that the formula, \( S \) implies \( P \) is true.

The system obtained by running \( n \) copies of \( S \) in lock-step is defined as follows.

The definition begins with \( S \) (of \( x\)-sub-1), which is the formula obtained by substituting a new set of variables, \( x\)-sub-1, for the variables of \( S \).
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

TLA has industrial-strength tools and is used by engineers who build large, distributed systems. ©

TLA describes systems, as well as properties, as formulas. ©

System \( S \) satisfies property \( P \) © means that the formula, \( S \implies P \) is true. ©

The system obtained by running \( n \) copies of \( S \) in lock-step is defined as follows. ©

The definition begins with \( S \) (of \( x \)-sub-1), which is the formula obtained by substituting a new set of variables, \( x \)-sub-1, for the variables of \( S \). ©
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \] equals

TLA has industrial-strength tools and is used by engineers who build large, distributed systems. ©

TLA describes systems, as well as properties, as formulas. ©

System \( S \) satisfies property \( P \) © means that the formula, \( S \) implies \( P \) is true. ©

The system obtained by running \( n \) copies of \( S \) in lock-step is defined as follows. ©

The definition begins with \( S \) (of \( x \)-sub-1), which is the formula obtained by substituting a new set of variables, \( x \)-sub-1, for the variables of \( S \). ©
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \quad \text{equals} \]

\[ S(x_1) \]

\( S \) with new set \( x_1 \) of variables

The definition begins with \( S \) (of \( x \)-sub-1) which is the formula obtained by substituting a new set of variables, \( x \)-sub-1, for the variables of \( S \). ©

Formula \( S \) (of \( x \)-sub-1) asserts that the values assumed by the variables of \( x \)-sub-1 during an execution satisfy the specification of system \( S \). ©

And similarly for \( S \) of \( x \)-sub-2 through \( n \), all different sets of variables. ©

In TLA conjunction is parallel composition, so this is a system composed of \( n \) copies of system \( S \) executing in parallel. ©
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \text{ equals } S(x_1) \]

\( S \) with new set \( x_1 \) of variables that satisfy spec of \( S \)

The definition begins with \( S \) (of \( x \)-sub-1) which is the formula obtained by substituting a new set of variables, \( x \)-sub-1, for the variables of \( S \).

Formula \( S \) (of \( x \)-sub-1) asserts that the values assumed by the variables of \( x \)-sub-1 during an execution satisfy the specification of system \( S \).

And similarly for \( S \) of \( x \)-sub-2 through \( n \), all different sets of variables.

In TLA conjunction is parallel composition, so this is a system composed of \( n \) copies of system \( S \) executing in parallel.
Properties & systems are formulas.

\[ S \models P \quad \equiv \quad \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \text{ equals} \]

\[ S(x_1) \quad S(x_2) \ldots \quad S(x_n) \]

*\( S \) with new set \( x_n \) of variables

The definition begins with \( S \) (of \( x\)-sub-1) which is the formula obtained by substituting a new set of variables, \( x\)-sub-1, for the variables of \( S \). ©

Formula \( S \) (of \( x\)-sub-1) asserts that the values assumed by the variables of \( x\)-sub-1 during an execution satisfy the specification of system \( S \). ©

And similarly for \( S \) of \( x\)-sub-2 through \( n \), all different sets of variables. ©

In TLA conjunction is parallel composition, so this is a system composed of \( n \) copies of system \( S \) executing in parallel. ©
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \] equals

\[ S(x_1) \land S(x_2) \ldots \land S(x_n) \]
parallel composition of \( n \) copies of \( S \)

In TLA conjunction is parallel composition, so this is a system composed of \( n \) copies of system \( S \) executing in parallel. ©

and \( K \) asserts that the copies run in lock-step. I don’t have time to explain how \( K \) is defined. ©

It’s now easy to define the property asserting that \( S \) satisfies GNI. ©

Here are the 2 copies of \( S \) that execute in lock-step.
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \text{ equals } \]
\[ S(x_1) \land S(x_2) \ldots \land S(x_n) \land K(x_1, \ldots, x_n) \]

copies run in lock-step

In TLA conjunction is parallel composition, so this is a system composed of \( n \) copies of system \( S \) executing in parallel.

and \( K \) asserts that the copies run in lock-step. I don’t have time to explain how \( K \) is defined.

It’s now easy to define the property asserting that \( S \) satisfies GNI.

Here are the 2 copies of \( S \) that execute in lock-step.
Properties & systems are formulas.

\[
S \models P \equiv \models (S \Rightarrow P)
\]

\[
S \otimes S \ldots \otimes S \text{ equals }
S(x_1) \land S(x_2) \ldots \land S(x_n) \land K(x_1, \ldots, x_n)
\]

\[
S \models GNI
\]

In TLA conjunction is parallel composition, so this is a system composed of \( n \) copies of system \( S \) executing in parallel. ©

and \( K \) asserts that the copies run in lock-step. I don’t have time to explain how \( K \) is defined. ©

It’s now easy to define the property asserting that \( S \) satisfies GNI. ©

Here are the 2 copies of \( S \) that execute in lock-step.
Properties & systems are formulas.

\[ S \vdash P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \] equals

\[ S(x_1) \land S(x_2) \ldots \land S(x_n) \land K(x_1, \ldots, x_n) \]

\[ S \vdash GNI \equiv \models (S(x_1) \land S(x_2) \land K(x_1, \ldots, x_n)) \]

2 copies of \( S \) run in lock-step

Here are the 2 copies of \( S \) that execute in lock-step.
This composite system must satisfy—that is, this formula must imply—

that there exists another execution of \( S \)
represented by the values of variables \( x \)-sub 3

with the right relation among the 3 executions—
that is, the values of the public variables of \( x \)-sub 3 equal those of \( x \)-sub 1
and the values of its secret variables equal those of \( x \)-sub-2.
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \text{ equals } S(x_1) \land S(x_2) \ldots \land S(x_n) \land K(x_1, \ldots, x_n) \]

\[ S \models GNI \equiv \models (S(x_1) \land S(x_2) \land K(x_1, \ldots, x_n)) \]

\[ \Rightarrow \exists x_3 : S(x_3) \]

exists a 3rd execution of \( S \)

Here are the 2 copies of \( S \) that execute in lock-step.
This composite system must satisfy—that is, this formula must imply—
that there exists another execution of \( S \)
represented by the values of variables \( x \)-sub 3

with the right relation among the 3 executions—
that is, the values of the public variables of \( x \)-sub 3 equal those of \( x \)-sub 1
and the values of its secret variables equal those of \( x \)-sub-2.
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \] equals

\[ S(x_1) \land S(x_2) \ldots \land S(x_n) \land K(x_1, \ldots, x_n) \]

\[ S \models GNI \equiv \models (S(x_1) \land S(x_2) \land K(x_1, \ldots, x_n) \]

\[ \Rightarrow \exists x_3 : S(x_3) \land L(x_1, x_2, x_3) \]

relation among the 3 executions

Here are the 2 copies of \( S \) that execute in lock-step. This composite system must satisfy—that is, this formula must imply— that there exists another execution of \( S \) represented by the values of variables \( x \)-sub 3 with the right relation among the 3 executions—that is, the values of the public variables of \( x \)-sub 3 equal those of \( x \)-sub 1 and the values of its secret variables equal those of \( x \)-sub-2. © ©

[slide 76] 5 min 56 sec
Properties & systems are formulas.

\[ S \models P \equiv \models (S \Rightarrow P) \]

\[ S \otimes S \ldots \otimes S \] equals

\[ S(x_1) \land S(x_2) \ldots \land S(x_n) \land K(x_1, \ldots, x_n) \]

\[ S \models GNI \equiv \models (S(x_1) \land S(x_2) \land K(x_1, \ldots, x_n) \Rightarrow \exists x_3 : S(x_3) \land L(x_1, x_2, x_3)) \]

Here are the 2 copies of \( S \) that execute in lock-step. This composite system must satisfy—that is, this formula must imply—\( \odot \)
that there exists another execution of \( S \)
represented by the values of variables \( x \)-sub 3 \( \odot \)
with the right relation among the 3 executions—
that is, the values of the public variables of \( x \)-sub 3 equal those of \( x \)-sub 1
and the values of its secret variables equal those of \( x \)-sub-2. \( \odot \) \( \odot \)
Unfortunately, expressing GNI is not this easy.

TLA, like most temporal logics, models a system execution as a sequence of states.

GNI and some other security hyperproperties were originally described in terms of executions as sequences of events.

To translate from events to states, we model an event as a change of state.
It’s Not So Easy

TLA models executions as sequences of states.

Unfortunately, expressing GNI is not this easy. ©

TLA, like most temporal logics, models a system execution as a sequence of states. ©

GNI and some other security hyperproperties were originally described in terms of executions as sequences of events. © ©

To translate from events to states, we model an event as a change of state.
It’s Not So Easy

TLA models executions as sequences of **states**.

GNI based on executions as sequences of **events**.

Unfortunately, expressing GNI is not this easy. ©

TLA, like most temporal logics, models a system execution as a sequence of **states**. ©

GNI and some other security hyperproperties were originally described in terms of executions as sequences of **events**. ©

To translate from events to states, we model an event as a change of state.

[slide 80] 6 min 22 sec
It's Not So Easy

TLA models executions as sequences of states.

GNI based on executions as sequences of events.

\[
\text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots
\]

Public events must reveal nothing about secret events.

Unfortunately, expressing GNI is not this easy. ©

TLA, like most temporal logics, models a system execution as a sequence of states. ©

GNI and some other security hyperproperties were originally described in terms of executions as sequences of events. ©

To translate from events to states, we model an event as a change of state.
It’s Not So Easy

TLA models executions as sequences of states.

GNI based on executions as sequences of events.

Translate events to states by: \( \text{event} = \text{state}_1 \rightarrow \text{state}_2 \)

Unfortunately, expressing GNI is not this easy. ©

TLA, like most temporal logics, models a system execution as a sequence of states. ©

GNI and some other security hyperproperties were originally described in terms of executions as sequences of events. © ©

To translate from events to states, we model an event as a change of state.
It’s Not So Easy

For GNI, assume a state is \((\text{public state}, \text{secret state})\).

To translate GNI, we assume a state is a \((\text{public-state}, \text{secret-state})\) pair. ©

Like this. ©

A public event © is one that changes the public state. ©

A secret event © is one that changes the secret state. ©
It’s Not So Easy

For GNI, assume a state is `(public state, secret state)`.

```
(4, "foo")
```

A public event is one that changes the public state.

A secret event is one that changes the secret state.
It’s Not So Easy

For GNI, assume a state is (public state, secret state).

A public event (4, “foo”)

A public event is one that changes the public state.

A secret event is one that changes the secret state.
For GNI, assume a state is \((\text{public state, secret state})\).

A public event \( (9, \text{“foo”}) \)

A public event \( \) is one that changes the public state.

A secret event \( \) is one that changes the secret state.
It's Not So Easy

For GNI, assume a state is (public state, secret state).

A secret event (9,"foo")

To translate GNI, we assume a state is a (public-state, secret-state) pair. Like this.

A public event is one that changes the public state.

A secret event is one that changes the secret state.
It’s Not So Easy

For GNI, assume a state is \((\text{public state, secret state})\).  

A secret event \((9, "far")\)

To translate GNI, we assume a state is a \((\text{public-state, secret-state})\) pair.  

Like this.  

A public event is one that changes the public state.  

A secret event is one that changes the secret state.
For GNI, assume a state is \((\text{public state}, \text{secret state})\).

\[
\cdots \rightarrow \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{public}_2 \rightarrow \cdots
\]

So this sequence of events becomes this sequence of states.

This public event changes the public state.

This secret event changes the secret state.

So every event is either a public event or a secret event.

And the events are replaced by the state changes.

And similarly, this sequence of events becomes this sequence of states.
It’s Not So Easy

For GNI, assume a state is \((\text{public state, secret state})\).

\[
\cdots \rightarrow \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{public}_2 \rightarrow \cdots
\]

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

So this sequence of events becomes this sequence of states.

This public event changes the public state. This secret event changes the secret state.

So every event is either a public event or a secret event. And the events are replaced by the state changes.

And similarly, this sequence of events becomes this sequence of states.
It’s Not So Easy

For GNI, assume a state is \((\text{public state}, \text{secret state})\).

\[
\cdots \rightarrow \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{public}_2 \rightarrow \cdots \\
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

So this sequence of events becomes this sequence of states. This public event changes the public state. This secret event changes the secret state. So every event is either a public event or a secret event. And the events are replaced by the state changes. And similarly, this sequence of events becomes this sequence of states.
It’s Not So Easy

For GNI, assume a state is $(\text{public state}, \text{secret state})$.

$\cdots \rightarrow \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{public}_2 \rightarrow \cdots$

$\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots$

So this sequence of events becomes this sequence of states.

This public event changes the public state. This secret event changes the secret state.

So every event is either a public event or a secret event. And the events are replaced by the state changes.

And similarly, this sequence of events becomes this sequence of states.
It’s Not So Easy

For GNI, assume a state is \((\text{public state, secret state})\).

\[
\cdots \rightarrow \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{public}_2 \rightarrow \cdots \\
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

So this sequence of events becomes this sequence of states.

This public event changes the public state. This secret event changes the secret state.

So every event is either a public event or a secret event.

And the events are replaced by the state changes.

And similarly, this sequence of events becomes this sequence of states.
For GNI, assume a state is \((\text{public state}, \text{secret state})\).

\[
\begin{align*}
\text{public}_1 & \quad \text{secret}_1 & \quad \text{public}_2 \\
\cdots & \to (p_1, s_1) & \to (p_2, s_1) & \to (p_2, s_2) & \to (p_3, s_2) & \to \cdots
\end{align*}
\]

So this sequence of events becomes this sequence of states.

This public event changes the public state. This secret event changes the secret state.

So every event is either a public event or a secret event. And the events are replaced by the state changes.

And similarly, this sequence of events becomes this sequence of states.
It’s Not So Easy

For GNI, assume a state is \((\text{public state}, \text{secret state})\).

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

\[
\cdots \rightarrow \text{public}_1 \rightarrow \text{secret}_1 \rightarrow \text{secret}_2 \rightarrow \text{public}_2 \rightarrow \cdots
\]

So this sequence of events \(\circ\) becomes this sequence of states. \(\circ\)

This public event changes the public state. \(\circ\) This secret event changes the secret state. \(\circ\)

So every event is either a public event or a secret event. \(\circ\)

And the events are replaced by the state changes. \(\circ\)

And similarly, this sequence of events \(\circ\)

becomes this sequence of states. \(\circ\) \(\circ\)

[slide 95] 7 min 4 sec
It’s Not So Easy

For GNI, assume a state is \((\text{public state, secret state})\).

\[ \cdots \to (p_1, s_1) \to (p_2, s_1) \to (p_2, s_2) \to (p_3, s_2) \to \cdots \]

\[
\begin{array}{cccc}
\text{public}_1 & \text{secret}_1 & \text{secret}_2 & \text{public}_2 \\
\cdots \to (p_1, s_1) & \to (p_2, s_1) & \to (p_2, s_2) & \to (p_2, s_3) \to (p_3, s_3) \to \cdots \\
\end{array}
\]

So this sequence of events becomes this sequence of states.
This public event changes the public state.
This secret event changes the secret state.
So every event is either a public event or a secret event.
And the events are replaced by the state changes.
And similarly, this sequence of events becomes this sequence of states.
It’s Not So Easy

For GNI, assume a state is \((\text{public state, secret state})\).

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \cdots
\]

So this sequence of events becomes this sequence of states.

This public event changes the public state. This secret event changes the secret state.

So every event is either a public event or a secret event. And the events are replaced by the state changes.

And similarly, this sequence of events becomes this sequence of states.
It’s Not So Easy

GNI: If these system executions are run in lock-step

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots \]

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \cdots \]

The TLA version of GNI which I showed you before asserts that, if these two system executions are run in lockstep ©

Then there exists a 3rd system execution ©
whose public states come from the 1st execution ©
and whose secret states come from the 2nd execution. ©

But there’s a problem here. ©
This state change changes the public state so it’s is a public event. ©
It’s Not So Easy

GNI: If these system executions are run in lock-step
then there exists a 3rd system execution...

\[ \ldots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \ldots \]

\[ \ldots \rightarrow (\ , \ ) \rightarrow (\ , \ ) \rightarrow (\ , \ ) \rightarrow (\ , \ ) \rightarrow \ldots \]

\[ \ldots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \ldots \]

The TLA version of GNI which I showed you before asserts that,
if these two system executions are run in lockstep ©
Then there exists a 3rd system execution ©
whose public states come from the 1st execution ©
and whose secret states come from the 2nd execution. ©
But there’s a problem here. ©
This state change changes the public state so it’s a public event. ©
It’s Not So Easy

GNI: If these system executions are run in lock-step then there exists a 3\textsuperscript{rd} system execution... 

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots \]

\[ \cdots \rightarrow (p_1, ) \rightarrow (p_2, ) \rightarrow (p_2, ) \rightarrow (p_3, ) \rightarrow \cdots \]

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \cdots \]

The TLA version of GNI which I showed you before asserts that, if these two system executions are run in lockstep, then there exists a 3\textsuperscript{rd} system execution whose public states come from the 1\textsuperscript{st} execution and whose secret states come from the 2\textsuperscript{nd} execution. But there’s a problem here. This state change changes the public state so it’s a public event.
It’s Not So Easy

GNI: If these system executions are run in lock-step then there exists a 3\textsuperscript{rd} system execution...

\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots

\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_3) \rightarrow \cdots

The TLA version of GNI which I showed you before asserts that, if these two system executions are run in lockstep
Then there exists a 3\textsuperscript{rd} system execution
whose public states come from the 1\textsuperscript{st} execution
and whose secret states come from the 2\textsuperscript{nd} execution.
But there’s a problem here. This state change changes the public state so it’s is a public event.
It’s Not So Easy

GNI: If these system executions are run in lock-step then there exists a 3\textsuperscript{rd} system execution...

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots \]

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_3) \rightarrow \cdots \]

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \cdots \]

The TLA version of GNI which I showed you before asserts that, if these two system executions are run in lockstep ©

Then there exists a 3\textsuperscript{rd} system execution ©

whose public states come from the 1\textsuperscript{st} execution ©

and whose secret states come from the 2\textsuperscript{nd} execution. ©

But there’s a problem here. ©

This state change changes the public state so it’s is a public event. ©
It’s Not So Easy

GNI: If these system executions are run in lock-step then there exists a 3rd system execution...

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots \]

\[ \text{public} \]
\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_3) \rightarrow \cdots \]

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \cdots \]

This state change changes the public state so it’s a public event. ©

This state change changes the secret state so it’s is a secret event. ©

But this state change changes both the secret and public states, which makes it both a public & secret event. ©

So this isn’t a system execution, because GNI assumes that the system allows state changes that are either public or secret events, but not both. ©
It’s Not So Easy

GNI: If these system executions are run in lock-step then there exists a 3rd system execution...

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

\[
\begin{align*}
\text{public} \\
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_3) \rightarrow \cdots
\end{align*}
\]

\[
\begin{align*}
\text{secret} \\
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \cdots
\end{align*}
\]

This state change changes the public state so it’s a public event. ©
This state change changes the secret state so it’s a secret event. ©
But this state change changes both the secret and public states, which makes it both a public & secret event. ©
So this isn’t a system execution, because GNI assumes that the system allows state changes that are either public or secret events, but not both. ©
It’s Not So Easy

GNI: If these system executions are run in lock-step
then there exists a 3rd system execution...

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_3) \rightarrow \cdots
\]

This state change changes the public state so it’s a public event.

This state change changes the secret state so it’s is a secret event.

But this state change changes both the secret and public states,
which makes it both a public & secret event.

So this isn’t a system execution, because GNI assumes that the system allows
state changes that are either public or secret events, but not both.
It’s Not So Easy

GNI: If these system executions are run in lock-step then there exists a 3rd `system` execution...

\[ \cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots \]

This state change changes the public state so it’s a public event. ©
This state change changes the secret state so it’s is a secret event. ©
But this state change changes both the secret and public states, which makes it both a public & secret event. ©
So this isn’t a system execution, because GNI assumes that the system allows state changes that are either public or secret events, but not both. ©
GNI: If any two system executions are run in lock-step then ...
This definition is wrong.

This problem is inherent in our TLA definition of GNI. ©

The definition is wrong. ©
Getting It Right

Instead of being in lock-step,

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \cdots
\]

\[
\cdots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \cdots
\]

Instead of having to execute the two copies of the system in lock-step ©
A correct definition of GNI should allow them to be executed like this. ©
GNI assumes these two executions appear the same to public users ©
who just see this. ©
Most formalisms consider these to be different executions because of this extra state. ©
but that means users can tell when secret events occur between public events. ©
TLA considers these two executions to be the same because that extra step leaves the
state seen by the user unchanged. ©
Instead of being in lock-step, the executions should be:

\[
\ldots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \ldots \\
\ldots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \ldots
\]

Instead of having to execute the two copies of the system in lock-step.

A correct definition of GNI should allow them to be executed like this.

GNI assumes these two executions appear the same to public users who just see this.

Most formalisms consider these to be different executions because of this extra state, but that means users can tell when secret events occur between public events.

TLA considers these two executions to be the same because that extra step leaves the state seen by the user unchanged.

[slide 110]
Getting It Right

Instead of being in lock-step, the executions should be:

GNI assumes they are the same to public users.

\[ \ldots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_3, s_2) \rightarrow \ldots \]

\[ \ldots \rightarrow (p_1, s_1) \rightarrow (p_2, s_1) \rightarrow (p_2, s_2) \rightarrow (p_2, s_3) \rightarrow (p_3, s_3) \rightarrow \ldots \]

Instead of having to execute the two copies of the system in lock-step ©
A correct definition of GNI should allow them to be executed like this. ©
GNI assumes these two executions appear the same to public users ©
who just see this. ©
Most formalisms consider these to be different executions because of this extra state. ©
but that means users can tell when secret events occur between public events. ©
TLA considers these two executions to be the same because that extra step leaves the
state seen by the user unchanged. ©
Getting It Right

Instead of being in lock-step, the executions should be:

GNI assumes they are the same to public users.

\[
\cdots \rightarrow (p_1, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow \ (p_3, \ ) \rightarrow \cdots
\]

\[
\cdots \rightarrow (p_1, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow (p_3, \ ) \rightarrow \cdots
\]

Instead of having to execute the two copies of the system in lock-step.

A correct definition of GNI should allow them to be executed like this.

GNI assumes these two executions appear the same to public users who just see this.

Most formalisms consider these to be different executions because of this extra state. But that means users can tell when secret events occur between public events.

TLA considers these two executions to be the same because that extra step leaves the state seen by the user unchanged.
Getting It Right

Instead of being in lock-step, the executions should be: GNI assumes they are the same to public users.

\[
\cdots \to (p_1, s_1) \to (p_2, s_1) \to (p_2, s_2) \to (p_3, s_2) \to \cdots
\]

\[
\cdots \to (p_1, s_1) \to (p_2, s_1) \to (p_2, s_2) \to \boxed{(p_2, s_2)} \to (p_3, s_3) \to \cdots
\]

Most formalisms consider these to be different executions.

Instead of having to execute the two copies of the system in lock-step ©
A correct definition of GNI should allow them to be executed like this. ©
GNI assumes these two executions appear the same to public users © who just see this. ©

Most formalisms consider these to be different executions because of this extra state. ©
but that means users can tell when secret events occur between public events. ©
TLA considers these two executions to be the same because that extra step leaves the state seen by the user unchanged. ©
Getting It Right

Instead of being in lock-step, the executions should be:
GNI assumes they are the same to public users.

\[
\cdots \rightarrow (p_1, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow (p_3, \ ) \rightarrow \cdots \\
\cdots \rightarrow (p_1, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow (p_2, \ ) \rightarrow (p_3, \ ) \rightarrow \cdots 
\]

Most formalisms consider these to be different executions, implying users can tell when secret events occur between public ones.

Instead of having to execute the two copies of the system in lock-step ©
A correct definition of GNI should allow them to be executed like this. ©
GNI assumes these two executions appear the same to public users ©
who just see this. ©
Most formalisms consider these to be different executions because of this extra state. ©
but that means users can tell when secret events occur between public events. ©
TLA considers these two executions to be the same because that extra step leaves the
state seen by the user unchanged. ©
Getting It Right

Instead of being in lock-step, the executions should be:
GNI assumes they are the same to public users.

\[
\cdots \rightarrow (p_1, \quad) \rightarrow (p_2, \quad) \rightarrow (p_2, \quad) \rightarrow (p_3, \quad) \rightarrow \cdots
\]

\[
\cdots \rightarrow (p_1, \quad) \rightarrow (p_2, \quad) \rightarrow (p_2, \quad) \rightarrow (p_2, \quad) \rightarrow (p_3, \quad) \rightarrow \cdots
\]

TLA considers them the same because the extra step leaves the visible state unchanged.

Instead of having to execute the two copies of the system in lock-step ©
A correct definition of GNI should allow them to be executed like this. ©
GNI assumes these two executions appear the same to public users ©
who just see this. ©
Most formalisms consider these to be different executions because of this extra state. ©
but that means users can tell when secret events occur between public events. ©
TLA considers these two executions to be the same because that extra step leaves the state seen by the user unchanged. ©
TLA

TLA seems strange because steps that leave the state unchanged can’t be required or forbidden by a formula. This helps ensure that a spec can assert only what it should. A TLA spec can’t say that. That’s why implementation is simply implication.
TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

TLA at first seems strange to most people because steps that leave the state unchanged can’t be required or forbidden by a TLA formula. But that’s one reason TLA is simple. This restriction helps ensure that a spec can assert only what it should. For example, a specification of an hour minute clock should not assert that the clock does not display the temperature or doesn’t display seconds. A TLA spec can’t say that. That’s why implementation is simply implication.
TLA

TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

This helps ensure that a spec can assert only what it should.

TLA at first seems strange to most people because steps that leave the state unchanged can’t be required or forbidden by a TLA formula. But that’s one reason TLA is simple.

This restriction helps ensure that a spec can assert only what it should.

For example, a specification of an hour minute clock should not assert that the clock does not display the temperature or doesn’t display seconds. A TLA spec can’t say that.

That’s why implementation is simply implication.
TLA

TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

This helps ensure that a spec can assert only what it should.

A spec of an hour-minute clock

TLA at first seems strange to most people because steps that leave the state unchanged can’t be required or forbidden by a TLA formula. But that’s one reason TLA is simple. This restriction helps ensure that a spec can assert only what it should. For example, a specification of an hour minute clock should not assert that the clock does not display the temperature or doesn’t display seconds. A TLA spec can’t say that. That’s why implementation is simply implication.
TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

This helps ensure that a spec can assert only what it should.

A spec of an hour-minute clock shouldn’t say that it doesn’t display the temperature.

TLA at first seems strange to most people because steps that leave the state unchanged can’t be required or forbidden by a TLA formula. But that’s one reason TLA is simple. This restriction helps ensure that a spec can assert only what it should. For example, a specification of an hour minute clock should not assert that the clock does not display the temperature or doesn’t display seconds. A TLA spec can’t say that. That’s why implementation is simply implication.
TLA

TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

This helps ensure that a spec can assert only what it should.

A spec of an hour-minute clock shouldn’t say that it doesn’t display seconds.

TLA at first seems strange to most people because steps that leave the state unchanged can’t be required or forbidden by a TLA formula. ©

But that’s one reason TLA is simple. ©

This restriction helps ensure that a spec can assert only what it should. ©

For example, a specification of an hour minute clock should not assert that the clock © does not display the temperature © or doesn’t display seconds. ©

A TLA spec can’t say that. ©

That’s why implementation is simply implication. ©
TLA

TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

This helps ensure that a spec can assert only what it should.

A TLA spec of an hour-minute clock can’t say that it doesn’t display seconds.

TLA at first seems strange to most people because steps that leave the state unchanged can’t be required or forbidden by a TLA formula. But that’s one reason TLA is simple. This restriction helps ensure that a spec can assert only what it should. For example, a specification of an hour minute clock should not assert that the clock does not display the temperature or doesn’t display seconds. TLA spec can’t say that. That’s why implementation is simply implication.
TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

This helps ensure that a spec can assert only what it should.

A TLA spec of an hour-minute clock can’t say that it doesn’t display seconds.

TLA at first seems strange to most people because steps that leave the state unchanged can’t be required or forbidden by a TLA formula.

But that’s one reason TLA is simple.

This restriction helps ensure that a spec can assert only what it should.

For example, a specification of an hour minute clock should not assert that the clock does not display the temperature or doesn’t display seconds.

A TLA spec can’t say that.

That’s why implementation is simply implication.
TLA is simple because steps that leave the state unchanged can’t be required or forbidden by a formula.

This ensures that a spec can assert only what it should.

A main contribution of the paper:

A main contribution of the paper is that ©

This feature of TLA is also important for expressing hyperproperties. ©
TLA is simple because steps that leave the state unchanged can't be required or forbidden by a formula.

This ensures that a spec can assert only what it should.

A main contribution of the paper:

It's important for expressing hyperproperties.

A main contribution of the paper is that

This feature of TLA is also important for expressing hyperproperties.
Steps that leave the state unchanged can’t be required or forbidden.

This feature provides flexibility in aligning executions.

It enables simple specifications of a class of hyperproperties that includes GNI.

You’ll have to read the paper to find out how it’s done. It’s not obvious.
Steps that leave the state unchanged can’t be required or forbidden.

Provides flexibility in aligning executions.

This feature provides flexibility in aligning executions.

It enables simple specifications of a class of hyperproperties that includes GNI.

You’ll have to read the paper to find out how it’s done. It’s not obvious.
Steps that leave the state unchanged can’t be required or forbidden.

Provides flexibility in aligning executions.

Enables simple specifications of GNI and other hyperproperties.

This feature provides flexibility in aligning executions.

It enables simple specifications of a class of hyperproperties that includes GNI.

You’ll have to read the paper to find out how it’s done. It’s not obvious.
Steps that leave the state unchanged can’t be required or forbidden.

Provides flexibility in aligning executions.

Enables simple specifications of GNI and other hyperproperties.

You’ll have to read the paper to find out how it’s done.
In the Paper

What’s in the paper? ©

I’ve been doing a lot of hand-waving. The paper contains the details. ©
They’re explained with two toy systems that satisfy GNI. ©
There are TLA specifications of these other security hyperproperties. ©
There’s a characterization of when a hyperproperty is preserved under refinement. ©
The paper contains toy examples, but TLA is not a toy.
In the Paper

The details

What’s in the paper?

I’ve been doing a lot of hand-waving. The paper contains the details.

They’re explained with two toy systems that satisfy GNI.

There are TLA specifications of these other security hyperproperties.

There’s a characterization of when a hyperproperty is preserved under refinement.

The paper contains toy examples, but TLA is not a toy.
 Illustrated with two toy examples that satisfy GNI

They’re explained with two toy systems that satisfy GNI. There are TLA specifications of these other security hyperproperties. There’s a characterization of when a hyperproperty is preserved under refinement. The paper contains toy examples, but TLA is not a toy.
In the Paper

The details

Illustrated with two toy examples that satisfy GNI

TLA specs of: Noninference
             Noninterference
             Possibilistic Noninterference
             Input/output hyperproperties

What’s in the paper? ©
I’ve been doing a lot of hand-waving. The paper contains the details. ©
They’re explained with two toy systems that satisfy GNI. ©
There are TLA specifications of these other security hyperproperties. ©
There’s a characterization of when a hyperproperty is preserved under refinement. ©
The paper contains toy examples, but TLA is not a toy.
In the Paper

The details

Illustrated with two toy examples that satisfy GNI

TLA specs of: Noninference
Noninterference
Possibilistic Noninterference
Input/output hyperproperties

When a hyperproperty is preserved by refinement

What’s in the paper? ©
I’ve been doing a lot of hand-waving. The paper contains the details. ©
They’re explained with two toy systems that satisfy GNI. ©
There are TLA specifications of these other security hyperproperties. ©
There’s a characterization of when a hyperproperty is preserved under refinement. ©
The paper contains toy examples, but TLA is not a toy.
In the Paper

The details

Illustrated with two toy examples that satisfy GNI

TLA specs of:  Noninference
               Noninterference
               Possibilistic Noninterference
               Input/output hyperproperties

When a hyperproperty is preserved by refinement

Relation to machine-checked TLA proof of OD for a real system

The paper contains toy examples, but TLA is not a toy.

Others have written a machine-checked TLA proof of observational determinism for a real-time message passing system that was later commercialized.

The paper explains the relation of that work to ours. ©
On the Web

On the Web, you can find ©

Model-checked TLA specifications of the examples in the paper. ©

And all about TLA so you can try it yourself.

Thank you.

[slide 136]
On the Web

TLA specs of the examples in the paper

On the Web, you can find ©

Model-checked TLA specifications of the examples in the paper. ©

And all about TLA so you can try it yourself.

Thank you.
On the Web

TLA specs of the examples in the paper

TLA documentation & tools:

https://lamport.azurewebsites.net/tla/tla.html

On the Web, you can find ©

Model-checked TLA specifications of the examples in the paper. ©

And all about TLA so you can try it yourself.

Thank you.