

# System Modelling and Design

## Traffic Lights

Revision: 1.2, May 19, 2004

Ken Robinson

School of Computer Science & Engineering  
The University of New South Wales, Sydney Australia

16th March 2005

©Ken Robinson 2005

`mailto::k.robinson@unsw.edu.au`

# Outline I

- 1 Objectives of this lecture
- 2 A simple 2-way intersection
- 3 The Context Machine
- 4 The SimpleTwoWay Machine
- 5 The Invariant
  - Animation
  - A Safe state
  - Further Animation
  - Strengthening the invariant
  - Other Invariants
  - Strengthening the Precondition
  - The SimpleTwoWay machine
- 6 Sequencing
  - The precondition of ToRed

# Outline II

- The precondition of ToGreen
- Equivalent predicates
- The precondition of ToAmber

## 7 The TwoWay Machine

# Objectives of this lecture

- To explore the specification of some simple traffic light controllers.
- To explore the use of a state invariant to ensure safety.
- To explore the use of preconditions that ensure an operation will not violate the state invariant, when the precondition is satisfied.
- To use the animator to illustrate these explorations.

# Objectives of this lecture

- To explore the specification of some simple traffic light controllers.
- To explore the use of a state invariant to ensure safety.
- To explore the use of preconditions that ensure an operation will not violate the state invariant, when the precondition is satisfied.
- To use the animator to illustrate these explorations.

# Objectives of this lecture

- To explore the specification of some simple traffic light controllers.
- To explore the use of a state invariant to ensure safety.
- To explore the use of preconditions that ensure an operation will not violate the state invariant, when the precondition is satisfied.
- To use the animator to illustrate these explorations.

# Objectives of this lecture

- To explore the specification of some simple traffic light controllers.
- To explore the use of a state invariant to ensure safety.
- To explore the use of preconditions that ensure an operation will not violate the state invariant, when the precondition is satisfied.
- To use the animator to illustrate these explorations.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads,  
one running *North-South*  
and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green**  
and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red** → **Green** → **Amber** → **Red** → ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads,  
one running *North-South*  
and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green**  
and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red**  $\longrightarrow$  **Green**  $\longrightarrow$  **Amber**  $\longrightarrow$  **Red**  $\longrightarrow$  ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads,  
one running *North-South*  
and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green**  
and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red**  $\longrightarrow$  **Green**  $\longrightarrow$  **Amber**  $\longrightarrow$  **Red**  $\longrightarrow$  ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads, one running *North-South* and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green** and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red** → **Green** → **Amber** → **Red** → ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads,  
one running *North-South*  
and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green**  
and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red** → **Green** → **Amber** → **Red** → ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads, one running *North-South* and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green** and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

Red  $\longrightarrow$  Green  $\longrightarrow$  Amber  $\longrightarrow$  Red  $\longrightarrow$  ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads, one running *North-South* and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green** and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

Red  $\longrightarrow$  Green  $\longrightarrow$  Amber  $\longrightarrow$  Red  $\longrightarrow$  ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads, one running *North-South* and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green** and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red** → **Green** → **Amber** → **Red** → ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads, one running *North-South* and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green** and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

Red  $\longrightarrow$  Green  $\longrightarrow$  Amber  $\longrightarrow$  Red  $\longrightarrow$  ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads, one running *North-South* and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green** and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red** → **Green** → **Amber** → **Red** → ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# A simple 2-way intersection

Consider traffic lights at the intersection of two roads, one running *North-South* and the other *East-West*.

There are four sets of lights, each capable of showing **Red**, **Green** and **Amber**, placed at *North*, *East*, *South* and *West* positions.

The *North* and *South* lights are always identical, as are the *East* and *West* lights.

There are no *right-turn* lights.

Lights should change in the sequence:

**Red** → **Green** → **Amber** → **Red** → ...

We wish to specify a traffic light controller that ensures safety and the correct sequencing.

# The Context Machine

We will introduce a context machine containing the enumerated sets *DIRECTION* and *LIGHT*.

We will also specify a constant (function) *OTHER\_DIR* that maps each direction into the other direction.

Context machines are used commonly and frequently in *B Method (B)* to define sets and constants.

# The Context Machine

We will introduce a context machine containing the enumerated sets *DIRECTION* and *LIGHT*.

We will also specify a constant (function) *OTHER\_DIR* that maps each direction into the other direction.

Context machines are used commonly and frequently in B to define sets and constants.

# The Context Machine

We will introduce a context machine containing the enumerated sets *DIRECTION* and *LIGHT*.

We will also specify a constant (function) *OTHER\_DIR* that maps each direction into the other direction.

Context machines are used commonly and frequently in B to define sets and constants.

# TwoWay\_Ctx.mch

**MACHINE** *TwoWay\_Ctx*

**SETS**

*DIRECTION* = { *NorthSouth* , *EastWest* } ;

*LIGHT* = { *Red* , *Green* , *Amber* }

**CONSTANTS** *OTHER\_DIR*

**PROPERTIES**

$OTHER\_DIR \in DIRECTION \rightarrow DIRECTION \wedge$

$OTHER\_DIR = \{ NorthSouth \mapsto EastWest , EastWest \mapsto NorthSouth \}$

**END**

# The SimpleTwoWay Machine

- 1 We will write a basic machine with no non-trivial state invariant and no non-trivial preconditions.
- 2 The machine will see the context machine and have a state of one variable, *lights*, which is a total function from *DIRECTION* to *LIGHT*.
- 3 There is one operation: *ChangeLight(dir, colour)* that changes the light to *colour* in the direction *dir*

# The SimpleTwoWay Machine

- 1 We will write a basic machine with no non-trivial state invariant and no non-trivial preconditions.
- 2 The machine will see the context machine and have a state of one variable, *lights*, which is a total function from *DIRECTION* to *LIGHT*.
- 3 There is one operation: *ChangeLight(dir, colour)* that changes the light to *colour* in the direction *dir*

# The SimpleTwoWay Machine

- 1 We will write a basic machine with no non-trivial state invariant and no non-trivial preconditions.
- 2 The machine will see the context machine and have a state of one variable, *lights*, which is a total function from *DIRECTION* to *LIGHT*.
- 3 There is one operation: *ChangeLight(dir, colour)* that changes the light to *colour* in the direction *dir*

# The SimpleTwoWay Machine

- 1 We will write a basic machine with no non-trivial state invariant and no non-trivial preconditions.
- 2 The machine will see the context machine and have a state of one variable, *lights*, which is a total function from *DIRECTION* to *LIGHT*.
- 3 There is one operation: *ChangeLight(dir, colour)* that changes the light to *colour* in the direction *dir*

# SimpleTwoWay0.mch

**MACHINE** *SimpleTwoWay0*

**SEES** *TwoWay\_Ctx*

**VARIABLES** *lights*

**INVARIANT**  $lights \in DIRECTION \rightarrow LIGHT$

**INITIALISATION**  $lights := \{ NorthSouth \mapsto Red, EastWest \mapsto Red \}$

## OPERATIONS

**ChangeLight** ( *dir* , *colour* )  $\hat{=}$

**PRE**  $dir \in DIRECTION \wedge colour \in LIGHT$

**THEN**  $lights ( dir ) := colour$

**END**

**END**

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 **Try to formulate an invariant that will ensure safety.**
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 **Whenever the state is unsafe, the invariant must be *false*.**

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be *safe*.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# The Invariant

- 1 Use the animator to explore the behaviour. When animating, choose to display the invariant, normally turned off.
- 2 It is, of course, trivial to establish that the controller is unsafe.
- 3 Try to formulate an invariant that will ensure safety.
- 4 Whenever the state is unsafe, the invariant must be *false*.

$$\neg(\text{safe}) \Rightarrow \neg(\text{invariant}) \quad (1)$$

- 5 Conversely, whenever the invariant is *true*, the state should be safe.

$$\text{invariant} \Rightarrow \text{safe} \quad (2)$$

- 6 Of course, 1 and 2 are equivalent; one is the *contrapositive* of the other.
- 7 If we have the *weakest* invariant, then whenever the state is safe, the invariant will be *true*.
- 8 Find adequately strong preconditions.

# A Safe state

The state invariant should be:

❶ *false* for all unsafe states    *true* for all safe states

❷ An initial attempt at an invariant might be

$$\neg (\text{lights}(\text{NorthSouth}) = \text{Green} \wedge \text{lights}(\text{EastWest}) = \text{Green})$$

which, using the predicate identities

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q \equiv P \Rightarrow \neg Q,$$

may be written

$$\text{lights}(\text{NorthSouth}) = \text{Green} \Rightarrow \neg (\text{lights}(\text{EastWest}) = \text{Green})$$

To avoid an error during analysis caused by the left associativity of operators in B, the above implication must be parenthesised.

# A Safe state

The state invariant should be:

❶ *false* for all unsafe states    *true* for all safe states

❷ An initial attempt at an invariant might be

$$\neg (\text{lights}(\text{NorthSouth}) = \text{Green} \wedge \text{lights}(\text{EastWest}) = \text{Green})$$

which, using the predicate identities

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q \equiv P \Rightarrow \neg Q,$$

may be written

$$\text{lights}(\text{NorthSouth}) = \text{Green} \Rightarrow \neg (\text{lights}(\text{EastWest}) = \text{Green})$$

To avoid an error during analysis caused by the left associativity of operators in B, the above implication must be parenthesised.

# A Safe state

The state invariant should be:

- 1 *false* for all unsafe states    *true* for all safe states
- 2 An initial attempt at an invariant might be

$$\neg (\text{lights}(\text{NorthSouth}) = \text{Green} \wedge \text{lights}(\text{EastWest}) = \text{Green})$$

which, using the predicate identities

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q \equiv P \Rightarrow \neg Q,$$

may be written

$$\text{lights}(\text{NorthSouth}) = \text{Green} \Rightarrow \neg (\text{lights}(\text{EastWest}) = \text{Green})$$

To avoid an error during analysis caused by the left associativity of operators in B, the above implication must be parenthesised.

# A Safe state

The state invariant should be:

- 1 *false* for all unsafe states    *true* for all safe states
- 2 An initial attempt at an invariant might be

$$\neg (\text{lights}(\text{NorthSouth}) = \text{Green} \wedge \text{lights}(\text{EastWest}) = \text{Green})$$

which, using the predicate identities

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q \equiv P \Rightarrow \neg Q,$$

may be written

$$\text{lights}(\text{NorthSouth}) = \text{Green} \Rightarrow \neg (\text{lights}(\text{EastWest}) = \text{Green})$$

To avoid an error during analysis caused by the left associativity of operators in B, the above implication must be parenthesised.

# A Safe state

The state invariant should be:

- 1 *false* for all unsafe states    *true* for all safe states
- 2 An initial attempt at an invariant might be

$$\neg (\text{lights}(\text{NorthSouth}) = \text{Green} \wedge \text{lights}(\text{EastWest}) = \text{Green})$$

which, using the predicate identities

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q \equiv P \Rightarrow \neg Q,$$

may be written

$$\text{lights}(\text{NorthSouth}) = \text{Green} \Rightarrow \neg (\text{lights}(\text{EastWest}) = \text{Green})$$

To avoid an error during analysis caused by the left associativity of operators in B, the above implication must be parenthesised.

# A Safe state

The state invariant should be:

- 1 *false* for all unsafe states    *true* for all safe states
- 2 An initial attempt at an invariant might be

$$\neg (\text{lights}(\text{NorthSouth}) = \text{Green} \wedge \text{lights}(\text{EastWest}) = \text{Green})$$

which, using the predicate identities

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q \equiv P \Rightarrow \neg Q ,$$

may be written

$$\text{lights}(\text{NorthSouth}) = \text{Green} \Rightarrow \neg (\text{lights}(\text{EastWest}) = \text{Green})$$

To avoid an error during analysis caused by the left associativity of operators in B, the above implication must be parenthesised.

# A Safe state

The state invariant should be:

- 1 *false* for all unsafe states    *true* for all safe states
- 2 An initial attempt at an invariant might be

$$\neg (\text{lights}(\text{NorthSouth}) = \text{Green} \wedge \text{lights}(\text{EastWest}) = \text{Green})$$

which, using the predicate identities

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q \equiv P \Rightarrow \neg Q ,$$

may be written

$$\text{lights}(\text{NorthSouth}) = \text{Green} \Rightarrow \neg (\text{lights}(\text{EastWest}) = \text{Green})$$

To avoid an error during analysis caused by the left associativity of operators in B, the above implication must be parenthesised.

## Further Animation

Try animating with the invariant on the preceding slide, using the animation script on the following slide.

ANIMATE

SimpleTwoWay0.test1.anm

PARAMETER\_VALUES

?

SETS\_VALUES

?

CONSTANTS\_VALUES

OTHER\_DIR = {NorthSouth |-> EastWest , EastWest |-> NorthSouth}

ENUM\_SETS\_VALUES

DIRECTION = {NorthSouth , EastWest};

LIGHT = {Red , Green , Amber}

OPERATIONS

INI\_SimpleTwoWay0;

ChangeLight(NorthSouth,Green);

ChangeLight(EastWest,Green);

undo;

ChangeLight(EastWest,Amber);

undo;

ChangeLight(NorthSouth,Amber);

ChangeLight(EastWest,Amber)

END

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

$$\equiv$$

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

$$\equiv$$

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

$$\equiv$$

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

$$\equiv$$

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

$$\equiv$$

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

$$\equiv$$

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Strengthening the invariant

Clearly, the light in both directions cannot be either **Green** or **Amber**.

	Red	Green	Amber
Red	safe	safe	safe
Green	safe	unsafe	unsafe
Amber	safe	unsafe	unsafe

This leads to the invariant:

$$\neg (\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \wedge \text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\})$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \neg (\text{lights}(\text{EastWest}) \in \{\text{Green}, \text{Amber}\}))$$

≡

$$(\text{lights}(\text{NorthSouth}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{EastWest}) = \text{Red})$$

# Other Invariants

There are other invariants that adequately express safety for a two-way intersection:

$$\text{lights}(\text{NorthSouth}) = \text{Red} \vee \text{lights}(\text{EastWest}) = \text{Red}$$

$$\text{Red} \in \text{ran}(\text{lights})$$

But these conditions do not generalise to intersections with more than two ways. Indeed the expression of the invariant that best generalises is

$$\forall \text{dir}. (\text{dir} \in \text{DIRECTION} \wedge \text{lights}(\text{dir}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red})$$

# Other Invariants

There are other invariants that adequately express safety for a two-way intersection:

$$\text{lights}(\text{NorthSouth}) = \text{Red} \vee \text{lights}(\text{EastWest}) = \text{Red}$$

$$\text{Red} \in \text{ran}(\text{lights})$$

But these conditions do not generalise to intersections with more than two ways. Indeed the expression of the invariant that best generalises is

$$\forall \text{dir}. (\text{dir} \in \text{DIRECTION} \wedge \text{lights}(\text{dir}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red})$$

# Other Invariants

There are other invariants that adequately express safety for a two-way intersection:

$$\text{lights}(\text{NorthSouth}) = \text{Red} \vee \text{lights}(\text{EastWest}) = \text{Red}$$

$$\text{Red} \in \text{ran}(\text{lights})$$

But these conditions do not generalise to intersections with more than two ways. Indeed the expression of the invariant that best generalises is

$$\forall \text{dir}. (\text{dir} \in \text{DIRECTION} \wedge \text{lights}(\text{dir}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red})$$

# Other Invariants

There are other invariants that adequately express safety for a two-way intersection:

$$\mathit{lights}(\mathit{NorthSouth}) = \mathit{Red} \vee \mathit{lights}(\mathit{EastWest}) = \mathit{Red}$$

$$\mathit{Red} \in \mathit{ran}(\mathit{lights})$$

But these conditions do not generalise to intersections with more than two ways. Indeed the expression of the invariant that best generalises is

$$\forall \mathit{dir}. (\mathit{dir} \in \mathit{DIRECTION} \wedge \mathit{lights}(\mathit{dir}) \in \{\mathit{Green}, \mathit{Amber}\} \Rightarrow \mathit{lights}(\mathit{OTHER\_DIR}(\mathit{dir})) = \mathit{Red})$$

# Other Invariants

There are other invariants that adequately express safety for a two-way intersection:

$$\text{lights}(\text{NorthSouth}) = \text{Red} \vee \text{lights}(\text{EastWest}) = \text{Red}$$

$$\text{Red} \in \text{ran}(\text{lights})$$

But these conditions do not generalise to intersections with more than two ways. Indeed the expression of the invariant that best generalises is

$$\forall \text{dir}. (\text{dir} \in \text{DIRECTION} \wedge \text{lights}(\text{dir}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red})$$

# Other Invariants

There are other invariants that adequately express safety for a two-way intersection:

$$\text{lights}(\text{NorthSouth}) = \text{Red} \vee \text{lights}(\text{EastWest}) = \text{Red}$$

$$\text{Red} \in \text{ran}(\text{lights})$$

But these conditions do not generalise to intersections with more than two ways. Indeed the expression of the invariant that best generalises is

$$\forall \text{dir}. (\text{dir} \in \text{DIRECTION} \wedge \text{lights}(\text{dir}) \in \{\text{Green}, \text{Amber}\} \Rightarrow \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red})$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$(colour \in \{Green, Amber\} \Rightarrow \\ lights(OTHER\_DIR(dir)) = Red)$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$(colour \in \{Green, Amber\} \Rightarrow \\ lights(OTHER\_DIR(dir)) = Red)$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$(colour \in \{Green, Amber\} \Rightarrow \\ lights(OTHER\_DIR(dir)) = Red)$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$(colour \in \{Green, Amber\} \Rightarrow \\ lights(OTHER\_DIR(dir)) = Red)$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$(colour \in \{Green, Amber\} \Rightarrow \\ lights(OTHER\_DIR(dir)) = Red)$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$(colour \in \{Green, Amber\} \Rightarrow \\ lights(OTHER\_DIR(dir)) = Red)$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$\begin{aligned} & (\text{colour} \in \{\text{Green}, \text{Amber}\} \Rightarrow \\ & \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}) \end{aligned}$$

# Strengthening the Precondition

We now have states and arguments of the operation *ChangeLight* that lead to a state that violates the state invariant.

This is not satisfactory!

The preconditions need to be strengthened so that the post-state violates the invariant *only if* the precondition is *false*.

This illustrates how preconditions and invariants collaborate in achieving safety.

Notice that preconditions ensure safety by imposing a proof obligation to be discharged in any context in which the operation is used.

A possible precondition is

$$(colour \in \{Green, Amber\} \Rightarrow \\ lights(OTHER\_DIR(dir)) = Red)$$

# The SimpleTwoWay machine I

**MACHINE** *SimpleTwoWay*

**SEES** *TwoWay\_Ctx*

**VARIABLES** *lights*

**INVARIANT**

$lights \in DIRECTION \rightarrow LIGHT \wedge$

$( lights ( NorthSouth ) \in \{ Green , Amber \} \Rightarrow lights ( EastWest ) = Red )$

**INITIALISATION**

$lights := \{ NorthSouth \mapsto Red , EastWest \mapsto Red \}$

**OPERATIONS**

**ChangeLight** ( *dir* , *colour* )  $\hat{=}$

**PRE**  $dir \in DIRECTION \wedge colour \in LIGHT \wedge$

$( colour \in \{ Green , Amber \} \Rightarrow lights ( OTHER\_DIR ( dir ) ) = Red )$

**THEN**  $lights ( dir ) := colour$

**END**

**END**

# Sequencing

The `SimpleTwoWay` machine ensures safety, but does not enforce any sequencing.

To achieve the desired sequencing we will introduce a new machine: `TwoWay`, that *INCLUDES* `SimpleTwoWay` and has three operations `ToRed(dir)`, `ToGreen(dir)`, and `ToAmber(dir)` for changing the lights.

The notion is that the body of each operation will use the operation `ChangeLight(dir, colour)` to change the colour.

The precondition of each operation will constrain the sequencing, and also must ensure that the precondition of `ChangeLight` is satisfied.

# Sequencing

The **SimpleTwoWay** machine ensures safety, but does not enforce any sequencing.

To achieve the desired sequencing we will introduce a new machine: **TwoWay**, that *INCLUDES* **SimpleTwoWay** and has three operations **ToRed**(dir), **ToGreen**(dir), and **ToAmber**(dir) for changing the lights.

The notion is that the body of each operation will use the operation *ChangeLight*(dir, colour) to change the colour.

The precondition of each operation will constrain the sequencing, and also must ensure that the precondition of *ChangeLight* is satisfied.

# Sequencing

The **SimpleTwoWay** machine ensures safety, but does not enforce any sequencing.

To achieve the desired sequencing we will introduce a new machine: **TwoWay**, that *INCLUDES* **SimpleTwoWay** and has three operations **ToRed**(dir), **ToGreen**(dir), and **ToAmber**(dir) for changing the lights.

The notion is that the body of each operation will use the operation *ChangeLight*(dir, colour) to change the colour.

The precondition of each operation will constrain the sequencing, and also must ensure that the precondition of *ChangeLight* is satisfied.

# Sequencing

The **SimpleTwoWay** machine ensures safety, but does not enforce any sequencing.

To achieve the desired sequencing we will introduce a new machine: **TwoWay**, that *INCLUDES* **SimpleTwoWay** and has three operations **ToRed**(dir), **ToGreen**(dir), and **ToAmber**(dir) for changing the lights.

The notion is that the body of each operation will use the operation *ChangeLight*(dir, colour) to change the colour.

The precondition of each operation will constrain the sequencing, and also must ensure that the precondition of *ChangeLight* is satisfied.

# Sequencing

The **SimpleTwoWay** machine ensures safety, but does not enforce any sequencing.

To achieve the desired sequencing we will introduce a new machine: **TwoWay**, that *INCLUDES* **SimpleTwoWay** and has three operations **ToRed**(dir), **ToGreen**(dir), and **ToAmber**(dir) for changing the lights.

The notion is that the body of each operation will use the operation *ChangeLight*(dir, colour) to change the colour.

The precondition of each operation will constrain the sequencing, and also must ensure that the precondition of *ChangeLight* is satisfied.

# The precondition of ToRed

Since the precondition of *ChangeLight* is always satisfied for the colour **Red**, we need to only be concerned with sequencing.

Hence the precondition is

$$\text{lights}(\text{dir}) = \text{Amber}$$

# The precondition of ToRed

Since the precondition of *ChangeLight* is always satisfied for the colour **Red**, we need to only be concerned with sequencing.

Hence the precondition is

$$\text{lights}(\text{dir}) = \text{Amber}$$

# The precondition of ToRed

Since the precondition of *ChangeLight* is always satisfied for the colour **Red**, we need to only be concerned with sequencing.

Hence the precondition is

$$\text{lights}(\text{dir}) = \text{Amber}$$

# The precondition of ToRed

Since the precondition of *ChangeLight* is always satisfied for the colour **Red**, we need to only be concerned with sequencing.

Hence the precondition is

$$\text{lights}(\text{dir}) = \text{Amber}$$

# The precondition of ToGreen

When setting a light to *Green* the precondition of ChangeLight requires

$$\text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

Sequencing requires

$$\text{lights}(\text{dir}) = \text{Red}$$

Thus the precondition is the conjunction

$$\text{lights}(\text{dir}) = \text{Red} \wedge \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

# The precondition of ToGreen

When setting a light to **Green** the precondition of ChangeLight requires

$$\text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

Sequencing requires

$$\text{lights}(\text{dir}) = \text{Red}$$

Thus the precondition is the conjunction

$$\text{lights}(\text{dir}) = \text{Red} \wedge \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

# The precondition of ToGreen

When setting a light to **Green** the precondition of ChangeLight requires

$$\text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

Sequencing requires

$$\text{lights}(\text{dir}) = \text{Red}$$

Thus the precondition is the conjunction

$$\text{lights}(\text{dir}) = \text{Red} \wedge \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

# The precondition of ToGreen

When setting a light to **Green** the precondition of ChangeLight requires

$$\text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

Sequencing requires

$$\text{lights}(\text{dir}) = \text{Red}$$

Thus the precondition is the conjunction

$$\text{lights}(\text{dir}) = \text{Red} \wedge \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

# The precondition of ToGreen

When setting a light to *Green* the precondition of `ChangeLight` requires

$$\text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

Sequencing requires

$$\text{lights}(\text{dir}) = \text{Red}$$

Thus the precondition is the conjunction

$$\text{lights}(\text{dir}) = \text{Red} \wedge \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

# The precondition of ToGreen

When setting a light to **Green** the precondition of ChangeLight requires

$$\text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

Sequencing requires

$$\text{lights}(\text{dir}) = \text{Red}$$

Thus the precondition is the conjunction

$$\text{lights}(\text{dir}) = \text{Red} \wedge \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

# The precondition of ToGreen

When setting a light to *Green* the precondition of `ChangeLight` requires

$$\text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

Sequencing requires

$$\text{lights}(\text{dir}) = \text{Red}$$

Thus the precondition is the conjunction

$$\text{lights}(\text{dir}) = \text{Red} \wedge \text{lights}(\text{OTHER\_DIR}(\text{dir})) = \text{Red}$$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$
- 2  $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 3  $\forall dir. (dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 4  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 5  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

<sup>1</sup>This uses relational image  $r[s] = ran(s \ll r)$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$   
 $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 2  $\forall dir (dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 3  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 4  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

<sup>1</sup>This uses relational image  $r[s] = ran(s \ll r)$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$   
 $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 2  $\forall dir.(dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 3  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 4  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

---

<sup>1</sup>This uses *relational image*  $r[s] = ran(s \triangleleft r)$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$   
 $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 2  $\forall dir.(dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 3  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 4  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

---

<sup>1</sup>This uses *relational image*  $r[s] = ran(s \triangleleft r)$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$   
 $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 2  $\forall dir.(dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 3  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 4  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

---

<sup>1</sup>This uses *relational image*  $r[s] = ran(s \triangleleft r)$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$   
 $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 2  $\forall dir.(dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 3  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 4  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

---

<sup>1</sup>This uses *relational image*  $r[s] = ran(s \triangleleft r)$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$   
 $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 2  $\forall dir.(dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 3  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 4  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

---

<sup>1</sup>This uses *relational image*  $r[s] = ran(s \triangleleft r)$

# Equivalent predicates

This may be expressed by a number of equivalent predicates, when combined with the sequencing predicate  $lights(dir) = Red$ :

- 1  $dir = NorthSouth \Rightarrow lights(EastWest) = Red \wedge$   
 $dir = EastWest \Rightarrow lights(NorthSouth) = Red$
- 2  $\forall dir.(dir \in DIRECTION \Rightarrow lights(dir) = Red)$
- 3  $lights[DIRECTION] = \{Red\}$ <sup>1</sup>
- 4  $ran(lights) = \{Red\}$

It is worth noting that alternatives 2, 3 and 4 if generalized to more directions, are too strong.

---

<sup>1</sup>This uses *relational image*  $r[s] = ran(s \triangleleft r)$

# The precondition of ToAmber

In order to satisfy the precondition of *ChangeLight*, the light in the other direction must be showing **Red**.

But, since the sequencing condition is

$lights(dir) = \text{Green}$ ,

the state invariant implies that

$$lights(OTHER\_DIR(dir)) = \text{Red} .$$

So, only the sequencing condition is required

$$lights(dir) = \text{Green}$$

# The precondition of ToAmber

In order to satisfy the precondition of *ChangeLight*, the light in the other direction must be showing **Red**.

But, since the sequencing condition is

$lights(dir) = \textit{Green}$ ,

the state invariant implies that

$$lights(OTHER\_DIR(dir)) = \textit{Red} .$$

So, only the sequencing condition is required

$$lights(dir) = \textit{Green}$$

# The precondition of ToAmber

In order to satisfy the precondition of *ChangeLight*, the light in the other direction must be showing **Red**.

But, since the sequencing condition is

$lights(dir) = \text{Green}$ ,

the state invariant implies that

$$lights(OTHER\_DIR(dir)) = \text{Red} .$$

So, only the sequencing condition is required

$$lights(dir) = \text{Green}$$

# The precondition of ToAmber

In order to satisfy the precondition of *ChangeLight*, the light in the other direction must be showing **Red**.

But, since the sequencing condition is

$lights(dir) = \text{Green}$ ,

the state invariant implies that

$$lights(OTHER\_DIR(dir)) = \text{Red} .$$

So, only the sequencing condition is required

$$lights(dir) = \text{Green}$$

# The precondition of ToAmber

In order to satisfy the precondition of *ChangeLight*, the light in the other direction must be showing **Red**.

But, since the sequencing condition is

$lights(dir) = \text{Green}$ ,

the state invariant implies that

$$lights(OTHER\_DIR(dir)) = \text{Red} .$$

So, only the sequencing condition is required

$$lights(dir) = \text{Green}$$

# The precondition of ToAmber

In order to satisfy the precondition of *ChangeLight*, the light in the other direction must be showing **Red**.

But, since the sequencing condition is

$lights(dir) = \text{Green}$ ,

the state invariant implies that

$$lights(OTHER\_DIR(dir)) = \text{Red} .$$

So, only the sequencing condition is required

$$lights(dir) = \text{Green}$$

# The precondition of ToAmber

In order to satisfy the precondition of *ChangeLight*, the light in the other direction must be showing **Red**.

But, since the sequencing condition is

$$\mathit{lights}(\mathit{dir}) = \mathit{Green},$$

the state invariant implies that

$$\mathit{lights}(\mathit{OTHER\_DIR}(\mathit{dir})) = \mathit{Red} .$$

So, only the sequencing condition is required

$$\mathit{lights}(\mathit{dir}) = \mathit{Green}$$

# The TwoWay Machine I

**MACHINE** *TwoWay*  
**SEES** *TwoWay\_Ctx*  
**INCLUDES** *SimpleTwoWay*

**OPERATIONS**

# The TwoWay Machine II

**ToRed** ( *dir* )  $\hat{=}$

**PRE**  $dir \in DIRECTION \wedge lights ( dir ) = Amber$

**THEN**  $ChangeLight ( dir , Red )$

**END ;**

**ToGreen** ( *dir* )  $\hat{=}$

**PRE**  $dir \in DIRECTION \wedge lights ( dir ) = Red \wedge$

$lights ( OTHER\_DIR ( dir ) ) = Red$

**THEN**  $ChangeLight ( dir , Green )$

**END ;**

**ToAmber** ( *dir* )  $\hat{=}$

**PRE**  $dir \in DIRECTION \wedge lights ( dir ) = Green$

**THEN**  $ChangeLight ( dir , Amber )$

**END**

**END**