Logic in Action
Chapter 6: Logic and Action

http://www.logicinaction.org/
Many different kinds of actions:
Many different kinds of actions:

- *She turns the light off,*
Actions in General

Actions

Many different kinds of actions:

- *She turns the light off,*
- *You put the milk in the fridge,*
Many different kinds of actions:

- *She turns the light off,*
- *You put the milk in the fridge,*
- *The apple falls to the ground,*
Many different kinds of actions:

- She turns the light off,
- You put the milk in the fridge,
- The apple falls to the ground,
- I submit the application form when it is completed,
Many different kinds of actions:

- *She turns the light off,*
- *You put the milk in the fridge,*
- *The apple falls to the ground,*
- *I submit the application form when it is completed,*
- *He asks a question only when he knows the answer,*
Actions in General

Actions

Many different kinds of actions:

- She turns the light off,
- You put the milk in the fridge,
- The apple falls to the ground,
- I submit the application form when it is completed,
- He asks a question only when he knows the answer,
- They do nothing.
The effect of an action

Actions can be characterized in terms of their result:
The effect of an action

Actions can be characterized in terms of their result:

- After *she turns the light off*, there will be dark.
The effect of an action

Actions can be characterized in terms of their result:

- After she turns the light off, there will be dark.
- After you put the milk in the fridge, it will be cold.
Actions in General

Actions in General

The effect of an action

Actions can be characterized in terms of their result:

- After *she turns the light off*, there will be dark.
- After *you put the milk in the fridge*, it will be cold.
- Once *the apple falls to the ground*, it will start to rot.
The effect of an action

Actions can be characterized in terms of their result:

- After *she turns the light off*, there will be dark.
- After *you put the milk in the fridge*, it will be cold.
- Once *the apple falls to the ground*, it will start to rot.
- Usually, after *I submit the application form*, the Jury will receive it, but sometimes it may get lost.
Actions in General

The effect of an action

Actions can be characterized in terms of their result:

- After *she turns the light off*, there will be dark.
- After *you put the milk in the fridge*, it will be cold.
- Once *the apple falls to the ground*, it will start to rot.
- Usually, after *I submit the application form*, the Jury will receive it, but sometimes it may get lost.
- After the teacher *asked a question*, the students were completely silent.
The effect of an action

Actions can be characterized in terms of their result:

- After *she turns the light off*, there will be dark.
- After *you put the milk in the fridge*, it will be cold.
- Once *the apple falls to the ground*, it will start to rot.
- Usually, after *I submit the application form*, the Jury will receive it, but sometimes it may get lost.
- After the teacher *asked a question*, the students were completely silent.
- After *they do nothing*, everything stays the same.
Operations over actions

Actions can be combined in several ways:
Operations over actions

Actions can be combined in several ways:

- **Sequence.** Execute one action after another:

  \[
  \text{Pour the mixture over the potatoes, and then cover pan with foil.}
  \]
Operations over actions

Actions can be combined in several ways:

- **Sequence.** Execute one action after another:
  
  *Pour the mixture over the potatoes, and then cover pan with foil.*

- **Choice.** Choose between actions:
  
  *Pick one of the boxes.*
Operations over actions

Actions can be combined in several ways:

- **Sequence.** Execute one action after another:
  
  *Pour the mixture over the potatoes, and then cover pan with foil.*

- **Choice.** Choose between actions:
  
  *Pick one of the boxes.*

- **Repetition.** Perform the same action several times:
  
  *Press the door until you hear a ‘click’.*
Operations over actions

Actions can be combined in several ways:

- **Sequence.** Execute one action after another:

  *Pour the mixture over the potatoes, and then cover pan with foil.*

- **Choice.** Choose between actions:

  *Pick one of the boxes.*

- **Repetition.** Perform the same action several times:

  *Press the door until you hear a ‘click’.*

- **Test.** Verify whether a given condition holds:

  *Check if the bulb is broken.*
Operations over actions

Actions can be combined in several ways:

- **Sequence.** Execute one action after another:
  
  *Pour the mixture over the potatoes, and then cover pan with foil.*

- **Choice.** Choose between actions:
  
  *Pick one of the boxes.*

- **Repetition.** Perform the same action several times:
  
  *Press the door until you hear a ‘click’.*

- **Test.** Verify whether a given condition holds:
  
  *Check if the bulb is broken.*

- **Converse.** Undo an executed action:
  
  *Close the window you just opened.*
Example: programming languages

Consider three famous control structures:

1. **WHILE** \( P \) **do** \( A \)
   
   This can be defined as the repetition of a test for '\( P \)' and the execution of '\( A \)', followed by a test for 'not \( A \)'.

2. **REPEAT** **A** **UNTIL** \( P \)
   
   This can be defined as the sequence of '\( A \)' and then **WHILE** (not \( P \)) **do** \( A \).

3. **IF** \( P \) **THEN** **A** **ELSE** **B**
   
   This can be defined as a choice between a test for '\( P \)' and then '\( A \)', or a test for 'not \( P \)' and then '\( B \)'.
Example: programming languages

Consider three famous control structures:

1. **WHILE P do A**

   This can be defined as the repetition of a test for 'P' and the execution of 'A', followed by a test for 'not A'.

2. **REPEAT A UNTIL P**

   This can be defined as the sequence of 'A' and then **WHILE (not P) do A**.

3. **IF P THEN A ELSE B**

   This can be defined as a choice between a test for 'P' and then 'A', or a test for 'not P' and then 'B'.

(http://www.logicinaction.org/)
Example: programming languages

Consider three famous control structures:

1. **WHILE P do A**
   This can be defined as the repetition of a test for ‘P’ and the execution of ‘A’, followed by a test for ‘not A’.

2. **REPEAT A UNTIL P**
   This can be defined as the sequence of ‘A’ and then **WHILE (not P) do A**.

3. **IF P THEN A ELSE B**
   This can be defined as a choice between a test for ‘P’ and then ‘A’, or a test for ‘not P’ and then ‘B’.
Example: programming languages

Consider three famous control structures:

1. **WHILE P do A**
   This can be defined as the repetition of a test for ‘P’ and the execution of ‘A’, followed by a test for ‘not A’.

2. **REPEAT A UNTIL P**
Example: programming languages

Consider three famous control structures:

1. **WHILE P do A**
   This can be defined as the repetition of a test for ‘P’ and the execution of ‘A’, followed by a test for ‘not A’.

2. **REPEAT A UNTIL P**
   This can be defined as the sequence of ‘A’ and then **WHILE (not P) do A**.
Example: programming languages

Consider three famous control structures:

1. **WHILE P do A**
   This can be defined as the repetition of a test for ‘P’ and the execution of ‘A’, followed by a test for ‘not A’.

2. **REPEAT A UNTIL P**
   This can be defined as the sequence of ‘A’ and then **WHILE (not P) do A**.

3. **IF P THEN A ELSE B**
Consider three famous control structures:

1. **WHILE P do A**
   This can be defined as the repetition of a test for ‘P’ and the execution of ‘A’, followed by a test for ‘not A’.

2. **REPEAT A UNTIL P**
   This can be defined as the sequence of ‘A’ and then **WHILE (not P) do A**.

3. **IF P THEN A ELSE B**
   This can be defined as a choice between a test for ‘P’ and then ‘A’, or a test for ‘not P’ and then ‘B’.

Example: programming languages

(http://www.logicinaction.org/)
We can see actions as transitions between states:
Representing actions abstractly (1)

We can see actions as transitions between states:

- Light is off
- Light is on
We can see actions as transitions between states:

Light is off \[\rightarrow\] Press the switch \[\rightarrow\] Light is on
We can see actions as transitions between states:

- Light is off → Press the switch → Light is on
- I have the letter → Letter arrives
- Letter gets lost
We can see actions as transitions between states:

- Light is off \[\rightarrow\] Press the switch \[\rightarrow\] Light is on
- I have the letter \[\rightarrow\] Send it \[\rightarrow\] Letter arrives
- Send it \[\rightarrow\] Letter gets lost

(\[\text{http://www.logicinaction.org/}\])
Representing actions abstractly (2)

More precisely, if we consider a set of states $S = \{s_1, s_2, \ldots\}$,
More precisely, if we consider a set of states $S = \{s_1, s_2, \ldots\}$,
Representing actions abstractly (2)

More precisely, if we consider a set of states \( S = \{s_1, s_2, \ldots\} \), then we can represent actions as binary relations on \( S \).

\[
S_1 \rightarrow S_2 \rightarrow S_3 \\
S_1 \rightarrow S_4
\]
More precisely, if we consider a set of states $S = \{s_1, s_2, \ldots\}$, then we can represent actions as binary relations on $S$. 
Representing actions abstractly (2)

More precisely, if we consider a set of states \( S = \{s_1, s_2, \ldots\} \), then we can represent actions as binary relations on \( S \).

\[
R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}
\]

\[
R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}
\]
Let $S$ be a domain $\{s_1, s_2, \ldots \}$. 
Let $S$ be a domain $\{s_1, s_2, \ldots\}$.

- **Identity relation.**

  $$I := \{(s, s) \mid s \in S\}$$
Let $S$ be a domain $\{s_1, s_2, \ldots\}$.

- **Identity relation.**

  $$I := \{(s, s) \mid s \in S\}$$

  $$I = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}$$
Let $S$ be a domain $\{s_1, s_2, \ldots\}$.

- **Identity relation.**

  $I := \{(s, s) \mid s \in S\}$

  \[ I = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\} \]
Operations on relations (2)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

$R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}$

$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$
Operations on relations (2)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

- **Composition.**

  $$R_a \circ R_b := \{(s, s') \mid \text{there is } s'' \in S \text{ such that } R_a s s'' \text{ and } R_b s'' s'\}$$

\[R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}\]
\[R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}\]
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

- **Composition.**

\[
R_a \circ R_b := \{(s, s') \mid \text{there is } s'' \in S \text{ such that } R_a s s'' \text{ and } R_b s'' s'\}
\]

\[
R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}
\]

\[
R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}
\]

\[
R_a \circ R_b = \{(s_1, s_4), (s_1, s_3), (s_2, s_3)\}
\]
Operations on relations (2)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- **Composition.**

$$R_a \circ R_b := \{(s, s') \mid \text{there is } s'' \in S \text{ such that } R_a s s'' \text{ and } R_b s'' s'\}$$

Let $R_a$ and $R_b$ be defined as:

$$R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}$$

$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

Then $R_a \circ R_b = \{(s_1, s_4), (s_1, s_3), (s_2, s_3)\}$
Operations on relations (2)

Let $S$ be a domain $\{s_1, s_2, \ldots \}$, and $R_a$, $R_b$ be binary relations on $S$.

- **Composition.**

  $$R_a \circ R_b := \{(s, s') \mid \text{there is } s'' \in S \text{ such that } R_a s s'' \text{ and } R_b s'' s'\}$$

In particular, for any relation $R_a$, we have

$$R_a^0 := I, \quad R_a^1 := R_a \circ R_a^0, \quad R_a^2 := R_a \circ R_a^1, \quad R_a^3 := R_a \circ R_a^2,$$

and so on.
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

\[
R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}
\]
\[
R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}
\]
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

- **Union.**

$$R_a \cup R_b := \{(s, s') \mid R_a ss' \text{ or } R_b ss'\}$$

$R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}$

$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

- **Union.**

  \[
  R_a \cup R_b := \{(s, s') \mid R_a ss' \text{ or } R_b ss'\}
  \]

\[\text{Diagram:} \quad \begin{array}{c}
\text{Diagram:} \\
R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\} \\
R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \\
R_a \cup R_b = \{(s_1, s_1), (s_1, s_2), (s_2, s_2) \\
(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \end{array}\]
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

- **Union.**

$$R_a \cup R_b := \{(s, s') \mid R_a s s' \text{ or } R_b s s'\}$$

$$R_a := \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}$$

$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

$$R_a \cup R_b = \{(s_1, s_1), (s_1, s_2), (s_2, s_2), (s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$
Operations on relations (4)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$
Operations on relations (4)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- Repetition zero or more times.

\[ R_a^* := \{(s, s') | R_a^n ss' \text{ for some } n \in \mathbb{N}\} \]

\[ R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \]
Operations on relations (4)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- Repetition zero or more times.

\[ R^*_a := \{(s, s') \mid R^n_a ss' \text{ for some } n \in \mathbb{N}\} \]

\[ R^*_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \]

\[ R^0_b = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\} \]
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

- Repetition zero or more times.

\[ R_a^* := \{(s, s') \mid R_a^n ss' \text{ for some } n \in \mathbb{N}\} \]

\[ R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \]

\[ R_b^0 = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\} \]
Operations on relations (4)

Let $S$ be a domain $\{s_1, s_2, \ldots \}$, and $R_a$, $R_b$ be binary relations on $S$.

- Repetition zero or more times.

$$R_a^* := \{(s, s') \mid R_a^n ss' \text{ for some } n \in \mathbb{N}\}$$

$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

$$R_b^0 = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}$$

$$R_b^1 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$
Operations on relations (4)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

- Repetition zero or more times.

\[
R_a^* := \{(s, s') | R_n^a ss' \text{ for some } n \in \mathbb{N}\}
\]

\[
R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}
\]

\[
R_b^0 = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}
\]

\[
R_b^1 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}
\]
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- Repetition zero or more times.

$$R_a^* := \{(s, s') \mid R_a^n ss' \text{ for some } n \in \mathbb{N}\}$$

\[
\begin{align*}
R_b & := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \\
R_b^0 & = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\} \\
R_b^1 & = \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \\
R_b^2 & = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}
\end{align*}
\]
Let $S$ be a domain $\{s_1, s_2, \ldots \}$, and $R_a$, $R_b$ be binary relations on $S$.

- Repetition zero or more times.

$$R_a^* := \{(s, s') \mid R_a^n ss' \text{ for some } n \in \mathbb{N}\}$$

$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

$$R_b^0 = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}$$

$$R_b^1 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

$$R_b^2 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$$
Operations on relations (4)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- Repetition zero or more times.

\[ R_a^* := \{(s, s') \mid R_a^n ss' \text{ for some } n \in \mathbb{N}\} \]

\[ R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \]
\[ R_b^0 = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\} \]
\[ R_b^1 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \]
\[ R_b^2 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\} \]
\[ R_b^3 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\} \]
Operations on relations (4)

Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a$, $R_b$ be binary relations on $S$.

Repetition zero or more times.

$R_a^* := \{(s, s') \mid R_a^n ss' \text{ for some } n \in \mathbb{N}\}$

$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$

$R_b^0 = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}$

$R_b^1 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$

$R_b^2 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$

$R_b^3 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$

$\vdots$
Let $S$ be a domain $\{s_1, s_2, \ldots \}$, and $R_a, R_b$ be binary relations on $S$.

- Repetition zero or more times.

$$R_a^* := \{(s, s') \mid R_a^n ss' \text{ for some } n \in \mathbb{N}\}$$

$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

$$R_b^0 = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}$$

$$R_b^1 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

$$R_b^2 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$$

$$R_b^3 = \{(s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$$

$$\vdots$$

$$R_b^* = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4),$$

$$\ldots, (s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$$
Let \( S \) be a domain \( \{s_1, s_2, \ldots\} \), and \( R_a, R_b \) be binary relations on \( S \).

\[
R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}
\]
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- **Converse.**

$$\tilde{R}_a := \{(s', s) \mid R_a ss'\}$$

$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- **Converse.**

\[ \tilde{R}_a := \{(s', s) \mid R_a ss'\} \]

\[ R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \]

\[ \tilde{R}_b = \{(s_4, s_1), (s_3, s_2), (s_3, s_4)\} \]
Let $S$ be a domain $\{s_1, s_2, \ldots\}$, and $R_a, R_b$ be binary relations on $S$.

- **Converse.**

\[ \tilde{R}_a := \{(s', s) \mid R_\alpha ss'\} \]

$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$

$\tilde{R}_b = \{(s_4, s_1), (s_3, s_2), (s_3, s_4)\}$
Syntax (1)

The language of propositional dynamic logic (PDL) has two components, formulas $\varphi$ and actions $\alpha$.

- **Formulas** are built via the following rules.
Syntax (1)

The language of **propositional dynamic logic (PDL)** has two components, formulas $\varphi$ and actions $\alpha$.

- **Formulas** are built via the following rules.
  - Every basic proposition is a formula
    
    $p, \quad q, \quad r, \quad \ldots$
The language of **propositional dynamic logic (PDL)** has two components, **formulas** $\varphi$ and **actions** $\alpha$.

- **Formulas** are built via the following rules.
  - Every basic proposition is a formula
    
    $p, \quad q, \quad r, \quad \ldots$
  
  - If $\varphi$ and $\psi$ are formulas, then the following are formulas:
    
    $\neg \varphi, \quad \varphi \land \psi, \quad \varphi \lor \psi, \quad \varphi \rightarrow \psi, \quad \varphi \leftrightarrow \psi$
Syntax (1)

The language of **propositional dynamic logic (PDL)** has two components, **formulas** $\varphi$ and **actions** $\alpha$.

- **Formulas** are built via the following rules.
  - Every basic proposition is a formula
    
    $$p, \; q, \; r, \; \ldots$$
  
  - If $\varphi$ and $\psi$ are formulas, then the following are formulas:
    
    $$\neg \varphi, \; \varphi \land \psi, \; \varphi \lor \psi, \; \varphi \rightarrow \psi, \; \varphi \leftrightarrow \psi$$
  
  - If $\varphi$ is a formula and $\alpha$ an action, then the following is a formula:
    
    $$\langle \alpha \rangle \varphi$$
Syntax (2)

The language of **propositional dynamic logic (PDL)** has two components, **formulas** $\varphi$ and **actions** $\alpha$.

- **Actions** are built via the following rules.
The language of **propositional dynamic logic (PDL)** has two components, *formulas* $\varphi$ and *actions* $\alpha$.

- **Actions** are built via the following rules.
  - Every basic action is a action
    
    $$a, \ b, \ c, \ ...$$
The language of **propositional dynamic logic (PDL)** has two components, **formulas** $\varphi$ and **actions** $\alpha$.

- **Actions** are built via the following rules.
  - Every basic action is a action
    
    $$a, \; b, \; c, \; \ldots$$
  - If $\alpha$ and $\beta$ are actions, then the following are actions:
    
    $$\alpha;\beta, \; \alpha \cup \beta, \; \alpha^*$$
The language of propositional dynamic logic (PDL) has two components, formulas $\varphi$ and actions $\alpha$.

- **Actions** are built via the following rules.
  - Every basic action is a action
    
    $a, \ b, \ c, \ \ldots$

  - If $\alpha$ and $\beta$ are actions, then the following are actions:
    
    $\alpha; \beta, \ \alpha \cup \beta, \ \alpha^*$

  - If $\varphi$ is a formula, then the following is an action:
    
    $?\varphi$
Intuitions and abbreviations

\[ \alpha; \beta \]
\[ \alpha \cup \beta \]
\[ \alpha^* \]
\[ ?\varphi \]
\[ \langle \alpha \rangle \varphi \]
Intuitions and abbreviations

- \( \alpha; \beta \): **sequential composition**: execute \( \alpha \) and then \( \beta \).
- \( \alpha \cup \beta \)  
- \( \alpha^* \)  
- \( ? \varphi \)  
- \( \langle \alpha \rangle \varphi \)
Intuitions and abbreviations

- \( \alpha; \beta \) **sequential composition**: execute \( \alpha \) and then \( \beta \).
- \( \alpha \cup \beta \) **non-deterministic choice**: execute \( \alpha \) or \( \beta \).
- \( \alpha^* \)
- \( ? \varphi \)
- \( \langle \alpha \rangle \varphi \)
Intuitions and abbreviations

\[ \alpha; \beta \] \textbf{sequential composition}: execute } \alpha \text{ and then } \beta.  
\[ \alpha \cup \beta \] \textbf{non-deterministic choice}: execute } \alpha \text{ or } \beta. 
\[ \alpha^* \] \textbf{repetition}: execute } \alpha \text{ zero, one, or any finite number of times.}
\[ \exists \varphi \] 
\[ \langle \alpha \rangle \varphi \]
Intuitions and abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha; \beta$</td>
<td><strong>sequential composition</strong>: execute $\alpha$ and then $\beta$.</td>
</tr>
<tr>
<td>$\alpha \cup \beta$</td>
<td><strong>non-deterministic choice</strong>: execute $\alpha$ or $\beta$.</td>
</tr>
<tr>
<td>$\alpha^*$</td>
<td><strong>repetition</strong>: execute $\alpha$ zero, one, or any <strong>finite</strong> number of times.</td>
</tr>
<tr>
<td>$?\varphi$</td>
<td><strong>test</strong>: check whether $\varphi$ is true or not.</td>
</tr>
</tbody>
</table>

$\langle \alpha \rangle \varphi$

We abbreviate $p \lor \neg p$ as $\top$.
We abbreviate $\neg \top$ as $\bot$.
We abbreviate $\neg \langle \alpha \rangle \neg \varphi$ as $[\alpha] \varphi$.

After any execution of $\alpha$, $\varphi$ is the case.
Intuitions and abbreviations

\[ \alpha; \beta \quad \text{sequential composition}: \text{execute } \alpha \text{ and then } \beta. \]
\[ \alpha \cup \beta \quad \text{non-deterministic choice}: \text{execute } \alpha \text{ or } \beta. \]
\[ \alpha^* \quad \text{repetition}: \text{execute } \alpha \text{ zero, one, or any finite number of times.} \]
\[ ?\varphi \quad \text{test}: \text{check whether } \varphi \text{ is true or not.} \]
\[ \langle \alpha \rangle \varphi \quad \alpha \text{ can be executed in such a way that, after doing it, } \varphi \text{ is the case.} \]
Intuitions and abbreviations

**α; β**  *sequential composition*: execute $\alpha$ and then $\beta$.

**$\alpha \cup \beta$**  *non-deterministic choice*: execute $\alpha$ or $\beta$.

**$\alpha^*$**  *repetition*: execute $\alpha$ zero, one, or any *finite* number of times.

**?$\varphi$**  *test*: check whether $\varphi$ is true or not.

**$\langle \alpha \rangle \varphi$**  $\alpha$ can be executed in such a way that, after doing it, $\varphi$ is the case.

We abbreviate $p \lor \neg p$ as $\top$.
Intuitions and abbreviations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha;\beta$</td>
<td>sequential composition: execute $\alpha$ and then $\beta$.</td>
</tr>
<tr>
<td>$\alpha \cup \beta$</td>
<td>non-deterministic choice: execute $\alpha$ or $\beta$.</td>
</tr>
<tr>
<td>$\alpha^*$</td>
<td>repetition: execute $\alpha$ zero, one, or any finite number of times.</td>
</tr>
<tr>
<td>$?\varphi$</td>
<td>test: check whether $\varphi$ is true or not.</td>
</tr>
<tr>
<td>$\langle \alpha \rangle \varphi$</td>
<td>$\alpha$ can be executed in such a way that, after doing it, $\varphi$ is the case.</td>
</tr>
</tbody>
</table>

We abbreviate $p \lor \neg p$ as $\top$.
We abbreviate $\neg \top$ as $\bot$. 
Intuitions and abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>α; β</td>
<td><strong>sequential composition</strong>: execute α and then β.</td>
</tr>
<tr>
<td>α ∪ β</td>
<td><strong>non-deterministic choice</strong>: execute α or β.</td>
</tr>
<tr>
<td>α*</td>
<td><strong>repetition</strong>: execute α zero, one, or any finite number of times.</td>
</tr>
<tr>
<td>?φ</td>
<td><strong>test</strong>: check whether φ is true or not.</td>
</tr>
<tr>
<td>⟨α⟩φ</td>
<td>α can be executed in such a way that, after doing it, φ is the case.</td>
</tr>
</tbody>
</table>

We abbreviate \( p \lor \neg p \) as \( \top \).
We abbreviate \( \neg \top \) as \( \bot \).
We abbreviate \( \neg⟨α⟩\negφ \) as \([α]φ\).
## Intuitions and abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha; \beta )</td>
<td>sequential composition: execute ( \alpha ) and then ( \beta ).</td>
<td>( \alpha; \beta )</td>
</tr>
<tr>
<td>( \alpha \cup \beta )</td>
<td>non-deterministic choice: execute ( \alpha ) or ( \beta ).</td>
<td>( \alpha \cup \beta )</td>
</tr>
<tr>
<td>( \alpha^* )</td>
<td>repetition: execute ( \alpha ) zero, one, or any finite number of times.</td>
<td>( \alpha^* )</td>
</tr>
<tr>
<td>( ?\varphi )</td>
<td>test: check whether ( \varphi ) is true or not.</td>
<td>( ?\varphi )</td>
</tr>
<tr>
<td>( \langle \alpha \rangle \varphi )</td>
<td>( \alpha ) can be executed in such a way that, after doing it, ( \varphi ) is the case.</td>
<td>( \langle \alpha \rangle \varphi )</td>
</tr>
</tbody>
</table>

We abbreviate \( p \lor \neg p \) as \( \top \).
We abbreviate \( \neg \top \) as \( \bot \).
We abbreviate \( \neg \langle \alpha \rangle \neg \varphi \) as \( [\alpha] \varphi \).

\[ [\alpha] \varphi \quad \text{After any execution of } \alpha, \varphi \text{ is the case.} \]
Some examples of formulas

\[ \langle \alpha \rangle \top \]
\[ [\alpha] \bot \]
\[ \langle \alpha \rangle \varphi \wedge \neg [\alpha] \varphi \]
Some examples of formulas

\[ \langle \alpha \rangle \top \quad \alpha \text{ can be executed.} \]
\[ [\alpha] \bot \]
\[ \langle \alpha \rangle \varphi \land \neg [\alpha] \varphi \]
Some examples of formulas

\[\langle \alpha \rangle \top\] \; \alpha \text{ can be executed.}

\[\lbrack \alpha \rbrack \bot\] \; \alpha \text{ cannot be executed.}

\[\langle \alpha \rangle \varphi \land \lnot \lbrack \alpha \rbrack \varphi\]
Some examples of formulas

\[ \langle \alpha \rangle \top \quad \alpha \text{ can be executed.} \]

\[ [\alpha] \bot \quad \alpha \text{ cannot be executed.} \]

\[ \langle \alpha \rangle \varphi \land \neg [\alpha] \varphi \quad \alpha \text{ can be executed it at least two different ways.} \]
The models (1)

The structures in which we evaluate PDL formulas, labelled transition systems (LTS), have three components:

\[ M = \langle S, R_a, V \rangle \]
The models (1)

The structures in which we evaluate PDL formulas, labelled transition systems ($LTS$), have three components:

- a non-empty set $S$ of states,

\[ M = \langle S, R_a, V \rangle \]
The models (1)

The structures in which we evaluate PDL formulas, labelled transition systems \((LTS)\), have three components:

- a non-empty set \(S\) of states,
- a valuation function, \(V\), indicating which atomic propositions are true in each state \(s \in S\), and

\[
M = \langle S, V \rangle
\]
The models (1)

The structures in which we evaluate PDL formulas, labelled transition systems ($LTS$), have three components:

- a non-empty set $S$ of states,
- a valuation function, $V$, indicating which atomic propositions are true in each state $s \in S$, and
- an binary relation $R_a$ for each basic action $a$.

\[
M = \langle S, R_a, V \rangle
\]
A labelled transition system with a designate state (the root state) is called a pointed labelled transition system or a process graph.
A labelled transition system with a designate state (the root state) is called a pointed labelled transition system or a process graph.
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R, V \rangle\):
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_a, V \rangle\):

\[(M, s) \models p \quad \text{iff} \quad p \in V(s)\]
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\[(M, s) \models p \quad \text{iff} \quad p \in V(s)\]
\[(M, s) \models \neg \varphi \quad \text{iff} \quad \text{it is not the case that} \quad (M, s) \models \varphi\]
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\[
\begin{align*}
(M, s) &\models p \quad \text{iff} \quad p \in V(s) \\
(M, s) &\models \neg \varphi \quad \text{iff} \quad \text{it is not the case that } (M, s) \models \varphi \\
(M, s) &\models \varphi \lor \psi \quad \text{iff} \quad (M, s) \models \varphi \text{ or } (M, s) \models \psi
\end{align*}
\]
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\((M, s) \models p\)  \iff  \(p \in V(s)\)

\((M, s) \models \neg \varphi\)  \iff  \text{it is not the case that } (M, s) \models \varphi

\((M, s) \models \varphi \lor \psi\)  \iff  (M, s) \models \varphi \text{ or } (M, s) \models \psi

...  \iff  ...

Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\[(M, s) \models p \iff p \in V(s)\]
\[(M, s) \models \neg \varphi \iff \text{it is not the case that } (M, s) \models \varphi\]
\[(M, s) \models \varphi \lor \psi \iff (M, s) \models \varphi \text{ or } (M, s) \models \psi\]
\[\ldots\]
\[(M, s) \models \langle \alpha \rangle \varphi \iff \text{there is a } t \in S \text{ such that } R_\alpha st \text{ and } (M, t) \models \varphi\]
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\[
(M, s) \models p \quad \text{iff} \quad p \in V(s)
\]

\[
(M, s) \models \neg \varphi \quad \text{iff} \quad \text{it is not the case that } (M, s) \models \varphi
\]

\[
(M, s) \models \varphi \lor \psi \quad \text{iff} \quad (M, s) \models \varphi \text{ or } (M, s) \models \psi
\]

\[
\ldots \quad \text{iff} \quad \ldots
\]

\[
(M, s) \models \langle \alpha \rangle \varphi \quad \text{iff} \quad \text{there is a } t \in S \text{ such that } R_\alpha st \text{ and } (M, t) \models \varphi
\]

where the relation \(R_\alpha\) is given, in case \(\alpha\) is not a basic action, by
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\((M, s) \models p\) \text{ iff } p \in V(s)

\((M, s) \models \neg \varphi\) \text{ iff } it is not the case that \((M, s) \models \varphi\)

\((M, s) \models \varphi \lor \psi\) \text{ iff } \((M, s) \models \varphi\) or \((M, s) \models \psi\)

\ldots \text{ iff } \ldots

\((M, s) \models \langle \alpha \rangle \varphi\) \text{ iff } there is a } t \in S \text{ such that } R_\alpha st \text{ and } (M, t) \models \varphi

where the relation \(R_\alpha\) is given, in case \(\alpha\) is not a basic action, by

\[R_\alpha;\beta := R_\alpha \circ R_\beta\]
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\[(M, s) \models p \quad \text{iff} \quad p \in V(s)\]
\[(M, s) \models \neg \varphi \quad \text{iff} \quad \text{it is not the case that } (M, s) \models \varphi\]
\[(M, s) \models \varphi \lor \psi \quad \text{iff} \quad (M, s) \models \varphi \quad \text{or} \quad (M, s) \models \psi\]
\[\ldots \quad \text{iff} \quad \ldots\]
\[(M, s) \models \langle \alpha \rangle \varphi \quad \text{iff} \quad \text{there is a } t \in S \text{ such that } R_\alpha st \text{ and } (M, t) \models \varphi\]

where the relation \(R_\alpha\) is given, in case \(\alpha\) is not a basic action, by

\[R_\alpha;\beta := R_\alpha \circ R_\beta\]
\[R_\alpha \cup \beta := R_\alpha \cup R_\beta\]
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\[(M, s) \models p \quad \text{iff} \quad p \in V(s)\]
\[(M, s) \models \neg \varphi \quad \text{iff} \quad \text{it is not the case that} \ (M, s) \models \varphi\]
\[(M, s) \models \varphi \lor \psi \quad \text{iff} \quad (M, s) \models \varphi \text{ or } (M, s) \models \psi\]
\[\ldots \quad \text{iff} \quad \ldots\]
\[(M, s) \models \langle \alpha \rangle \varphi \quad \text{iff} \quad \text{there is a } t \in S \text{ such that } R_\alpha st \text{ and } (M, t) \models \varphi\]

where the relation \(R_\alpha\) is given, in case \(\alpha\) is not a basic action, by

\[
R_{\alpha;\beta} := R_\alpha \circ R_\beta
\]
\[
R_{\alpha\cup\beta} := R_\alpha \cup R_\beta
\]
\[
R_{\alpha^*} := (R_\alpha)^*
\]
Deciding truth-value of formulas

Take a pointed labelled transition system \((M, s)\) with \(M = \langle S, R_\alpha, V \rangle\):

\[(M, s) \models p \iff p \in V(s)\]
\[(M, s) \models \neg \varphi \iff \text{it is not the case that } (M, s) \models \varphi\]
\[(M, s) \models \varphi \lor \psi \iff (M, s) \models \varphi \text{ or } (M, s) \models \psi\]
\[\ldots \text{iff } \ldots\]
\[(M, s) \models \langle \alpha \rangle \varphi \iff \text{there is a } t \in S \text{ such that } R_\alpha st \text{ and } (M, t) \models \varphi\]

where the relation \(R_\alpha\) is given, in case \(\alpha\) is not a basic action, by

\[R_\alpha;\beta := R_\alpha \circ R_\beta\]
\[R_\alpha \cup \beta := R_\alpha \cup R_\beta\]
\[R_\alpha^* := (R_\alpha)^*\]
\[R?\varphi := \{(s, s) \in S \times S \mid (M, s) \models \varphi\}\]
Example: building complex relations

\[ R_a := \{ (s_1, s_1), (s_2, s_1) \} \]
\[ R_b := \{ (s_1, s_2), (s_3, s_3) \} \]
\[ R_c := \{ (s_1, s_3), (s_3, s_2) \} \]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
Example: building complex relations

\[
R_a := \{(s_1, s_1), (s_2, s_1)\}
\]
\[
R_b := \{(s_1, s_2), (s_3, s_3)\}
\]
\[
R_c := \{(s_1, s_3), (s_3, s_2)\}
\]
\[
R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\}
\]
\[
R_{a \cup c} =
\]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
\[ R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
\[ R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \]
\[ R_{c; c} = \]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_a \cup b = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
\[ R_a \cup c = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \]
\[ R_{c; c} = \{(s_1, s_2)\} \]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
\[ R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \]
\[ R_{c; c} = \{(s_1, s_2)\} \]
\[ R_{b; b} = \]

(http://www.logicinaction.org/)
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
\[ R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \]
\[ R_{c; a} = \{(s_1, s_2)\} \]
\[ R_{b; c} = \{\} \]
Example: building complex relations

\[
R_a := \{(s_1, s_1), (s_2, s_1)\}
\]
\[
R_b := \{(s_1, s_2), (s_3, s_3)\}
\]
\[
R_c := \{(s_1, s_3), (s_3, s_2)\}
\]

\[
R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\}
\]
\[
R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\}
\]
\[
R_{c; c} = \{(s_1, s_2)\}
\]
\[
R_{b; b} = \{\}
\]
\[
R_{\neg(p \lor q)} =
\]
Example: building complex relations

\[
\begin{align*}
R_a & := \{(s_1, s_1), (s_2, s_1)\} \\
R_b & := \{(s_1, s_2), (s_3, s_3)\} \\
R_c & := \{(s_1, s_3), (s_3, s_2)\}
\end{align*}
\]

\[
\begin{align*}
R_{a \cup b} & = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \\
R_{a \cup c} & = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \\
R_{c; c} & = \{(s_1, s_2)\} \\
R_{b; b} & = \{\} \\
R_{R \neg(p \lor q)} & = \{(s_2, s_2)\}
\end{align*}
\]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
\[ R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \]
\[ R_{c; c} = \{(s_1, s_2)\} \]
\[ R_{b; b} = \{\} \]
\[ R_{? \neg (p \lor q)} = \{(s_2, s_2)\} \]
\[ R_{?(p \lor q)} = \]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[ R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \]
\[ R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \]
\[ R_{c; c} = \{(s_1, s_2)\} \]
\[ R_{b; b} = \{\} \]
\[ R_?(\neg (p \lor q)) = \{(s_2, s_2)\} \]
\[ R_?(p \lor q) = \{(s_1, s_1), (s_3, s_3)\} \]
Example: building complex relations

\[ \begin{align*}
R_a & := \{(s_1, s_1), (s_2, s_1)\} \\
R_b & := \{(s_1, s_2), (s_3, s_3)\} \\
R_c & := \{(s_1, s_3), (s_3, s_2)\}
\end{align*} \]
Example: building complex relations

\[
\begin{align*}
R_a & := \{(s_1, s_1), (s_2, s_1)\} \\
R_b & := \{(s_1, s_2), (s_3, s_3)\} \\
R_c & := \{(s_1, s_3), (s_3, s_2)\}
\end{align*}
\]

\[
\begin{align*}
R_{a \cup b} & = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \\
R_{a \cup c} & = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \\
R_{c; c} & = \{(s_1, s_2)\} \\
R_{b; b} & = \{\} \\
R_{\neg (p \lor q)} & = \{(s_2, s_2)\} \\
R_{? (p \lor q)} & = \{(s_1, s_1), (s_3, s_3)\} \\
R_{? \neg (p \lor q); a; ?(p \lor q)} & = \{(s_2, s_1)\}
\end{align*}
\]
Example: building complex relations

\[ R_a := \{(s_1, s_1), (s_2, s_1)\} \]
\[ R_b := \{(s_1, s_2), (s_3, s_3)\} \]
\[ R_c := \{(s_1, s_3), (s_3, s_2)\} \]

\[
\begin{align*}
R_{a \cup b} &= \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \\
R_{a \cup c} &= \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \\
R_{c; c} &= \{(s_1, s_1)\} \\
R_{b; b} &= \{\} \\
R_{\neg(p \lor q)} &= \{(s_2, s_2)\} \\
R_{(p \lor q)} &= \{(s_1, s_1), (s_3, s_3)\} \\
R_{\neg(p \lor q); a; ?(p \lor q)} &= \{(s_2, s_1)\} \\
R_{c; a} &=
\end{align*}
\]
Example: building complex relations

\[
\begin{align*}
R_a & := \{(s_1, s_1), (s_2, s_1)\} \\
R_b & := \{(s_1, s_2), (s_3, s_3)\} \\
R_c & := \{(s_1, s_3), (s_3, s_2)\} \\
\end{align*}
\]
Example: building complex relations

\[
\begin{align*}
R_a & := \{(s_1, s_1), (s_2, s_1)\} \\
R_b & := \{(s_1, s_2), (s_3, s_3)\} \\
R_c & := \{(s_1, s_3), (s_3, s_2)\}
\end{align*}
\]

\[
\begin{align*}
R_a \cup b & = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\} \\
R_a \cup c & = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\} \\
R_c ; c & = \{(s_1, s_2)\} \\
R_b ; b & = \{\} \\
R \neg (p \lor q) & = \{(s_2, s_2)\} \\
R ? (p \lor q) & = \{(s_1, s_1), (s_3, s_3)\} \\
R ? (p \lor q) ; a ; ? (p \lor q) & = \{(s_2, s_1)\} \\
R_c ; a & = \{(s_3, s_1)\} \\
R_{(c ; a)}^{*} & = \\
\end{align*}
\]
Example: building complex relations

\[ R_a := \{ (s_1, s_1), (s_2, s_1) \} \]
\[ R_b := \{ (s_1, s_2), (s_3, s_3) \} \]
\[ R_c := \{ (s_1, s_3), (s_3, s_2) \} \]

\[ R_a \cup b = \{ (s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3) \} \]
\[ R_a \cup c = \{ (s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2) \} \]
\[ R_c;c = \{ (s_1, s_2) \} \]
\[ R_b;b = \{ \} \]
\[ R_?\neg(p \lor q) = \{ (s_2, s_2) \} \]
\[ R_?(p \lor q) = \{ (s_1, s_1), (s_3, s_3) \} \]
\[ R_?\neg(p \lor q);a;?(p \lor q) = \{ (s_2, s_1) \} \]
\[ R_c;a = \{ (s_3, s_1) \} \]
\[ R_{(c;a)^*} = \{ (s_3, s_1), (s_1, s_1), (s_2, s_2), (s_3, s_3) \} \]
Example: building complex relations

\[
\begin{align*}
R_a & := \{(s_1, s_1), (s_2, s_1)\}  \\
R_b & := \{(s_1, s_2), (s_3, s_3)\}  \\
R_c & := \{(s_1, s_3), (s_3, s_2)\}
\end{align*}
\]

\[
\begin{align*}
R_{a \cup b} & = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\}  \\
R_{a \cup c} & = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\}  \\
R_{c; c} & = \{(s_1, s_2)\}  \\
R_{b; b} & = \{\}\n\end{align*}
\]

\[
\begin{align*}
R_{\neg (p \lor q)} & = \{(s_2, s_2)\}  \\
R_? (p \lor q) & = \{(s_1, s_1), (s_3, s_3)\}  \\
R_{\neg (p \lor q); a; ?(p \lor q)} & = \{(s_2, s_1)\}  \\
R_{c; a} & = \{(s_3, s_1)\}  \\
R_{(c; a)^*} & = \{(s_3, s_1), (s_1, s_1), (s_2, s_2), (s_3, s_3)\}
\end{align*}
\]
Example: evaluating formulas

\[ p \land \neg (a \cup b) \]
Example: Evaluating Formulas

\( \langle a \cup b \rangle p \land \neg [a \cup b] p \)  \( \mathcal{M}, s_1 \models \) ?  \( \langle c; a \rangle^* p \)  \( \mathcal{M}, s_3 \models \) ?

\( [b] \perp \)  \( \mathcal{M}, s_1 \models \) ?  \( [?p] p \)  \( \mathcal{M}, s_3 \models \) ?

\( [a \rightarrow [b] \top] \)  \( \mathcal{M}, s_2 \models \) ?

\( [c^* \top] \)  \( \mathcal{M}, s_2 \models \) ?
Example: evaluating formulas

\[
(M, s_1) \models \langle a \cup b \rangle p \land \neg [a \cup b] p \quad \checkmark \quad (M, s_3) \models [(c; a)^*] p \quad ?
\]

\[
(M, s_1) \models [b] \bot \quad ? \quad (M, s_3) \models [?p] p \quad ?
\]

\[
(M, s_2) \models \langle a \rangle \top \rightarrow \langle b \rangle \top \quad ?
\]

\[
(M, s_2) \models \langle c^* \rangle \top \quad ?
\]
Example: evaluating formulas

\begin{align*}
(M, s_1) &\models \langle a \cup b \rangle p \land \neg[a \cup b] p \quad \checkmark \\
(M, s_1) &\models [b] \bot \\
(M, s_2) &\models \langle a \rangle \top \rightarrow \langle b \rangle \top \\
(M, s_2) &\models \langle c^* \rangle \top
\end{align*}

\begin{align*}
(M, s_3) &\models [(c; a)^*] p \\
(M, s_3) &\not\models [?p] p
\end{align*}
Example: evaluating formulas

\( M \)

\[
\begin{align*}
(M, s_1) & \models \langle a \cup b \rangle p \land \neg[a \cup b] p \quad \checkmark \\
(M, s_1) & \models [b] \bot \quad \times \\
(M, s_2) & \models \langle a \rangle \top \rightarrow \langle b \rangle \top \quad \times \\
(M, s_2) & \models \langle c^* \rangle \top \\
(M, s_3) & \models [(c; a)^*] p \quad ? \\
(M, s_3) & \models [?p] p \quad ?
\end{align*}
\]
Example: evaluating formulas

\[ (M, s_1) \models \langle a \cup b \rangle p \land \neg [a \cup b] p \checkmark \]
\[ (M, s_1) \models [b] \bot \times \]
\[ (M, s_2) \models \langle a \rangle \top \rightarrow \langle b \rangle \top \times \]
\[ (M, s_2) \models \langle c^* \rangle \top \checkmark \]

\[ (M, s_3) \models [(c; a)^*] p \?
\]
\[ (M, s_3) \models [?p] p \?
\]
Example: evaluating formulas

\[
\begin{align*}
(M, s_1) & \models \langle a \cup b \rangle p \land \neg [a \cup b] p & (M, s_3) & \models [(c; a)^*] p \\
(M, s_1) & \models [b] \bot & (M, s_3) & \models [?p] p \\
(M, s_2) & \models \langle a \rangle \top \rightarrow \langle b \rangle \top & (M, s_2) & \models \langle c^* \rangle \top
\end{align*}
\]
Example: evaluating formulas

\[ (M, s_1) \models \langle a \cup b \rangle p \land \neg [a \cup b] p \quad \checkmark \quad (M, s_3) \models [(c; a)^*] p \quad \checkmark \]

\[ (M, s_1) \models [b] \perp \quad \times \quad (M, s_3) \models [?p] p \quad \checkmark \]

\[ (M, s_2) \models \langle a \rangle \top \rightarrow \langle b \rangle \top \quad \times \]

\[ (M, s_2) \models \langle c^* \rangle \top \quad \checkmark \]
The valid formulas of \textit{PDL} can be derived from the following principles:

1. All propositional tautologies.
2. \[ \alpha (\phi \rightarrow \psi) \rightarrow (\alpha \phi \rightarrow \alpha \psi) \] for any action \( \alpha \).
3. Modus ponens (MP): from \( \phi \) and \( \phi \rightarrow \psi \), infer \( \psi \).
4. Necessitation (Nec): from \( \phi \) infer \( \square \alpha \phi \) for any action \( \alpha \).
Axiom system (1)

The valid formulas of $PDL$ can be derived from the following principles:

1. All propositional tautologies.
Axiomatization

Axiom system (1)

The valid formulas of \( PDL \) can be derived from the following principles:

1. All propositional tautologies.

2. \([\alpha] (\varphi \rightarrow \psi) \rightarrow ([\alpha] \varphi \rightarrow [\alpha] \psi)\) for any action \( \alpha \).
Axiom system (1)

The valid formulas of *PDL* can be derived from the following principles:

1. All propositional tautologies.
2. \([\alpha] (\varphi \rightarrow \psi) \rightarrow ([\alpha] \varphi \rightarrow [\alpha] \psi)\) for any action \(\alpha\).
3. **Modus ponens** (MP): from \(\varphi\) and \(\varphi \rightarrow \psi\), infer \(\psi\).
Axiomatization

Axiom system (1)

The valid formulas of *PDL* can be derived from the following principles:

1. All propositional tautologies.
2. \([\alpha] (\varphi \rightarrow \psi) \rightarrow ([\alpha] \varphi \rightarrow [\alpha] \psi)\) for any action \(\alpha\).
3. Modus ponens (MP): from \(\varphi\) and \(\varphi \rightarrow \psi\), infer \(\psi\).
4. Necessitation (Nec): from \(\varphi\) infer \([\alpha] \varphi\) for any action \(\alpha\).
Axiom system (2)

Principles for action operations:

- Test: \[ \psi \leftrightarrow (\psi \rightarrow \phi) \]
- Sequence: \[ [\alpha; \beta] \phi \leftrightarrow [\alpha] [\beta] \phi \]
- Choice: \[ [\alpha \cup \beta] \phi \leftrightarrow ([\alpha] \phi \land [\beta] \phi) \]
- Repetition:
  - Mix: \[ [\alpha^*] \phi \leftrightarrow (\phi \land [\alpha] [\alpha^*] \phi) \]
  - Induction:
    \[ \phi \land [\alpha^*] (\phi \rightarrow [\alpha] \phi) \rightarrow [\alpha^*] \phi \]
Principles for action operations:

- **Test:**

  \[
  \left[?\psi\right] \varphi \leftrightarrow (\psi \rightarrow \varphi)
  \]
Axiom system (2)

Principles for action operations:

- Test:
  
  \[ [?\psi] \varphi \leftrightarrow (\psi \rightarrow \varphi) \]

- Sequence:
  
  \[ [\alpha; \beta] \varphi \leftrightarrow [\alpha] [\beta] \varphi \]
Axiomatization

Axiom system (2)

Principles for action operations:

- Test:
  \[ [?\psi] \varphi \leftrightarrow (\psi \rightarrow \varphi) \]

- Sequence:
  \[ [\alpha; \beta] \varphi \leftrightarrow [\alpha] [\beta] \varphi \]

- Choice:
  \[ [\alpha \cup \beta] \varphi \leftrightarrow ([\alpha] \varphi \land [\beta] \varphi) \]
Axiom system (2)

5 Principles for action operations:

- **Test:**
  \[ [?\psi] \varphi \leftrightarrow (\psi \rightarrow \varphi) \]

- **Sequence:**
  \[ [\alpha; \beta] \varphi \leftrightarrow [\alpha] [\beta] \varphi \]

- **Choice:**
  \[ [\alpha \cup \beta] \varphi \leftrightarrow ([\alpha] \varphi \land [\beta] \varphi) \]

- **Repetition:**
Axiomatization

Axiom system (2)

Principles for action operations:

- **Test:**
  \[ \lbrack \psi \rbrack \varphi \leftrightarrow (\psi \rightarrow \varphi) \]

- **Sequence:**
  \[ \lbrack \alpha; \beta \rbrack \varphi \leftrightarrow \lbrack \alpha \rbrack \lbrack \beta \rbrack \varphi \]

- **Choice:**
  \[ \lbrack \alpha \cup \beta \rbrack \varphi \leftrightarrow (\lbrack \alpha \rbrack \varphi \land \lbrack \beta \rbrack \varphi) \]

- **Repetition:**
  - **Mix:**
    \[ \lbrack \alpha^* \rbrack \varphi \leftrightarrow (\varphi \land \lbrack \alpha \rbrack \lbrack \alpha^* \rbrack \varphi) \]
Axiomatization

Axiom system (2)

5 Principles for action operations:

- **Test:**
  \[ [?\psi] \varphi \leftrightarrow (\psi \rightarrow \varphi) \]

- **Sequence:**
  \[ [\alpha; \beta] \varphi \leftrightarrow [\alpha] [\beta] \varphi \]

- **Choice:**
  \[ [\alpha \cup \beta] \varphi \leftrightarrow ([\alpha] \varphi \land [\beta] \varphi) \]

- **Repetition:**
  - **Mix:**
    \[ [\alpha^*] \varphi \leftrightarrow (\varphi \land [\alpha] [\alpha^*] \varphi) \]
  - **Induction:**
    \[ (\varphi \land [\alpha^*] (\varphi \rightarrow [\alpha] \varphi) ) \rightarrow [\alpha^*] \varphi \]

A formula that can be derived by following these principles in a finite number of steps is called a theorem.
Axiomatization

Axiom system (2)

5 Principles for action operations:

- **Test:**
  \[ [?\psi] \varphi \leftrightarrow (\psi \rightarrow \varphi) \]

- **Sequence:**
  \[ [\alpha; \beta] \varphi \leftrightarrow [\alpha] [\beta] \varphi \]

- **Choice:**
  \[ [\alpha \cup \beta] \varphi \leftrightarrow ([\alpha] \varphi \land [\beta] \varphi) \]

- **Repetition:**
  - **Mix:**
    \[ [\alpha^*] \varphi \leftrightarrow (\varphi \land [\alpha] [\alpha^*] \varphi) \]
  - **Induction:**
    \[ (\varphi \land [\alpha^*] (\varphi \rightarrow [\alpha] \varphi)) \rightarrow [\alpha^*] \varphi \]

A formula that can be derived by following these principles in a finite number of steps is called a **theorem**.
Example

Prove that \([(\alpha \cup \beta); \gamma] \varphi \leftrightarrow ([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi)\) is valid.
Example

Prove that $[(\alpha \cup \beta); \gamma] \varphi \leftrightarrow ([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi)$ is valid.

From left to right:
Example

Prove that \( [(\alpha \cup \beta); \gamma] \varphi \leftrightarrow ([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi) \) is valid.

From left to right:

1. \( [(\alpha \cup \beta); \gamma] \varphi \) Assumption
Example

Prove that \([\alpha \cup \beta); \gamma] \varphi \leftrightarrow ([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi)\) is valid.

From left to right:

1. \([\alpha \cup \beta); \gamma] \varphi\) Assumption
2. \([\alpha \cup \beta] [\gamma] \varphi\) Sequence from step 1

The right to left direction is similar.
Prove that $[(\alpha \cup \beta); \gamma] \varphi \leftrightarrow ([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi)$ is valid.

From left to right:

1. $[(\alpha \cup \beta); \gamma] \varphi$ Assumption
2. $[\alpha \cup \beta] [\gamma] \varphi$ Sequence from step 1
3. $[\alpha] [\gamma] \varphi \land [\beta] [\gamma] \varphi$ Choice from step 2
Example

Prove that 

\[ [(\alpha \cup \beta); \gamma] \varphi \leftrightarrow ([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi) \]

is valid.

From left to right:

1. \( [(\alpha \cup \beta); \gamma] \varphi \) \hspace{1cm} \text{Assumption}
2. \( [\alpha \cup \beta] [\gamma] \varphi \) \hspace{1cm} \text{Sequence from step 1}
3. \( [\alpha] [\gamma] \varphi \land [\beta] [\gamma] \varphi \) \hspace{1cm} \text{Choice from step 2}
4. \( [\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi \) \hspace{1cm} \text{Sequence from step 3}
Example

Prove that \([(\alpha \cup \beta); \gamma] \varphi \leftrightarrow ([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi)\) is valid.

From left to right:

1. \([(\alpha \cup \beta); \gamma] \varphi \quad \text{Assumption}
2. \([\alpha \cup \beta] [\gamma] \varphi \quad \text{Sequence from step 1}
3. \([\alpha] [\gamma] \varphi \land [\beta] [\gamma] \varphi \quad \text{Choice from step 2}
4. \([\alpha; \gamma] \varphi \land [\beta; \gamma] \varphi \quad \text{Sequence from step 3}

The right to left direction is similar.


**PDL as a programming language**

With *PDL* we can define actions representing program control structures.

1. **WHILE** $\phi$ do $\alpha$:
   - $(\phi; \alpha \ast; \neg \phi)$

2. **REPEAT** $\alpha$ **UNTIL** $\phi$:
   - $\alpha; (\neg \phi; \alpha \ast; \phi)$

3. **IF** $\phi$ **THEN** $\alpha$ **ELSE** $\beta$:
   - $(\phi; \alpha) \cup (\neg \phi; \beta)$

(Website: [http://www.logicinaction.org/](http://www.logicinaction.org/))
PDL as a programming language

With PDL we can define actions representing program control structures.

\[
\text{WHILE } \varphi \text{ do } \alpha:
\]
**PDL as a programming language**

With **PDL** we can define actions representing program control structures.

1. **WHILE** \( \varphi \) **do** \( \alpha \): 

\[
(? \varphi; \alpha)^*; ?\neg \varphi
\]
PDL as a programming language

With PDL we can define actions representing program control structures.

1. WHILE $\varphi$ do $\alpha$:

   $(?\varphi; \alpha)^*; ?\neg \varphi$

2. REPEAT $\alpha$ UNTIL $\varphi$:
PDL as a programming language

With PDL we can define actions representing program control structures.

1. **WHILE** \( \varphi \) do \( \alpha \):

\[
(\varphi; \alpha)^*; ?\neg \varphi
\]

2. **REPEAT** \( \alpha \) **UNTIL** \( \varphi \):

\[
\alpha; (?\neg \varphi; \alpha)^*; ?\varphi
\]
**PDL as a programming language**

With **PDL** we can define actions representing program control structures.

1. **WHILE** \( \varphi \) do \( \alpha \):
   
   \((?\varphi; \alpha)^*; ?\neg \varphi\)

2. **REPEAT** \( \alpha \) **UNTIL** \( \varphi \):
   
   \(\alpha; (?\neg \varphi; \alpha)^*; ?\varphi\)

3. **IF** \( \varphi \) **THEN** \( \alpha \) **ELSE** \( \beta \):
**PDL as a programming language**

With PDL we can define actions representing program control structures.

1. **WHILE** \( \varphi \) **do** \( \alpha \):
   
   \( (?\varphi; \alpha)^*; ?\neg \varphi \)

2. **REPEAT** \( \alpha \) **UNTIL** \( \varphi \):
   
   \( \alpha; (?\neg \varphi; \alpha)^*; ?\varphi \)

3. **IF** \( \varphi \) **THEN** \( \alpha \) **ELSE** \( \beta \):
   
   \( (?\varphi; \alpha) \cup (?\neg \varphi; \beta) \)