

Logic in Action

Chapter 6: Logic and Action

`http://www.logicinaction.org/`

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:

- 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:

- **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**

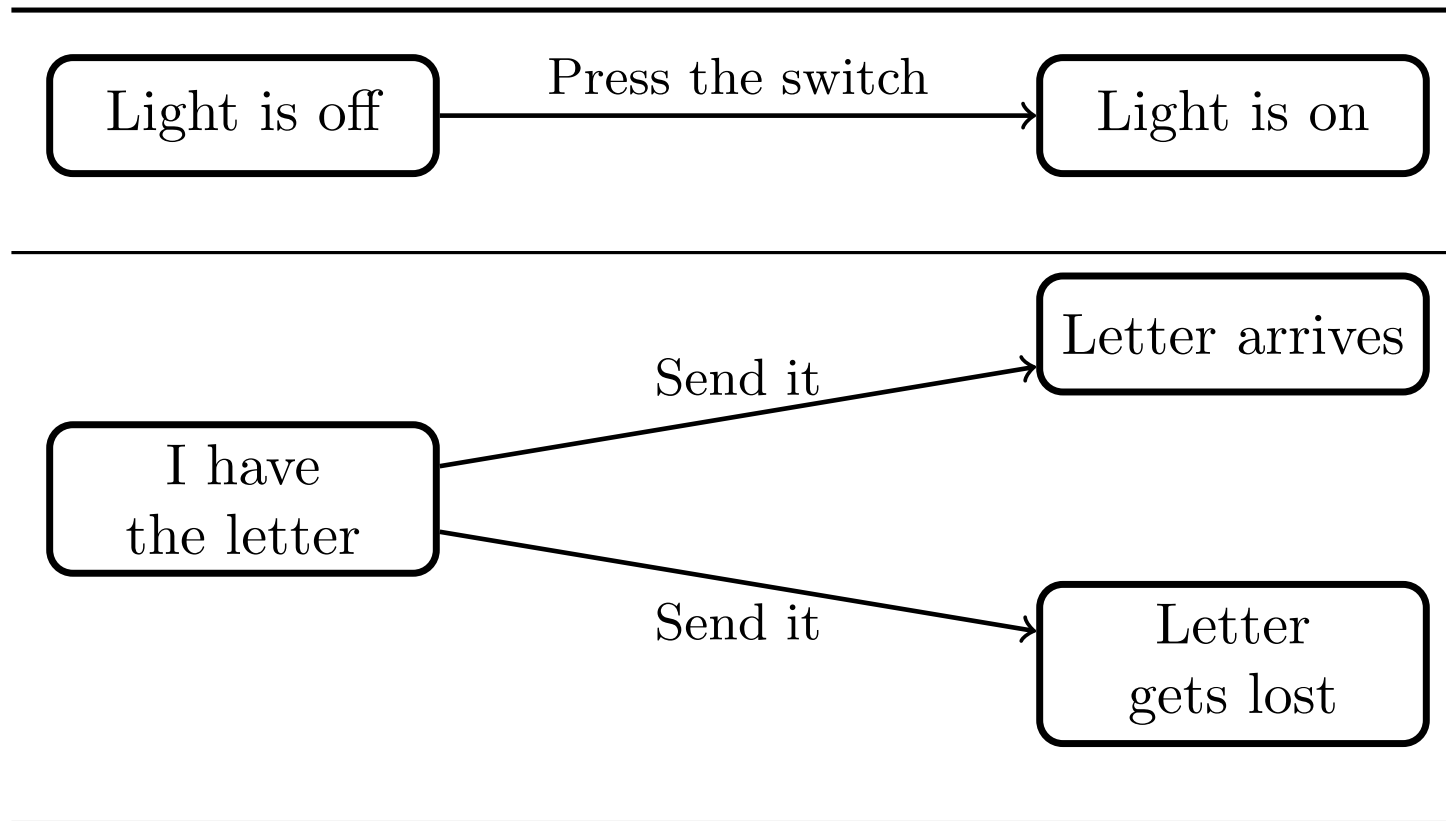
This can be defined as the sequence of ' A ' and then **WHILE (not P)** do **A**.

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 '.

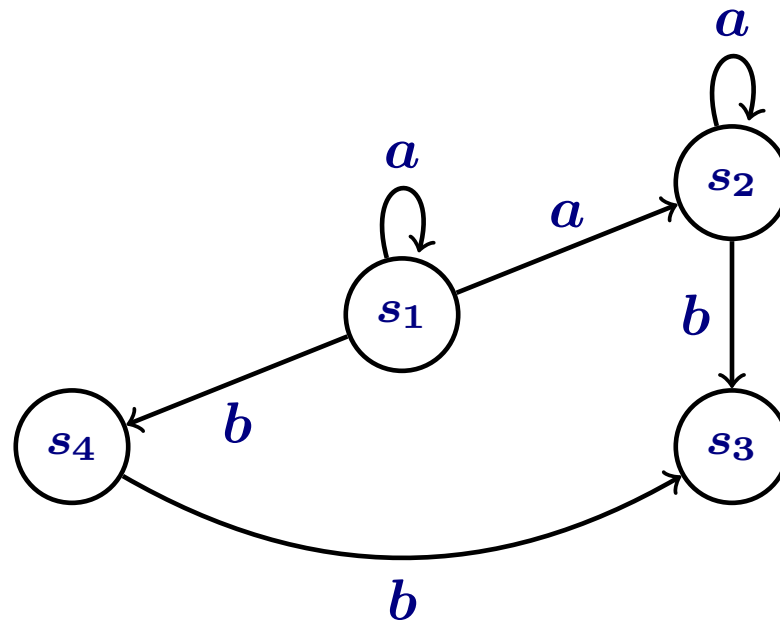
Representing actions abstractly (1)

We can see actions as transitions between states:



Representing actions abstractly (2)

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



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

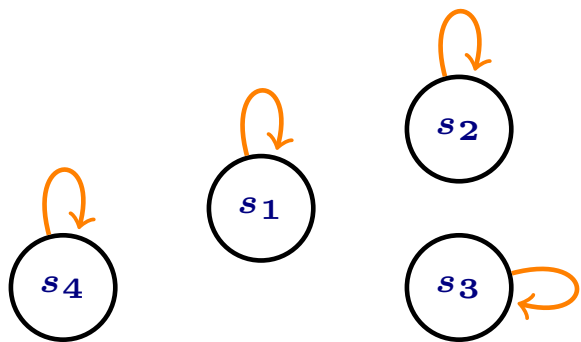
$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

Operations on relations (1)

Let S be a domain $\{s_1, s_2, \dots\}$.

- Identity relation.

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



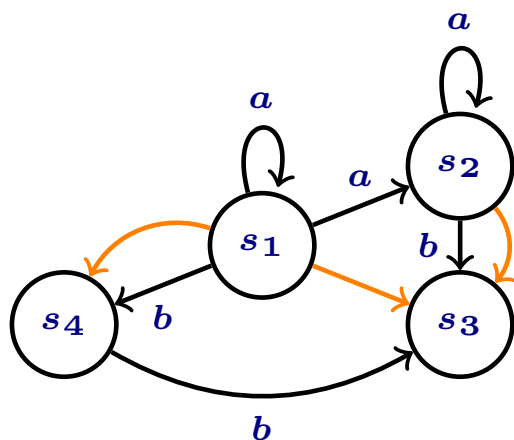
$$I = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}$$

Operations on relations (2)

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

- Composition.

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



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

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

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

In particular, for any relation R_a , we have

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

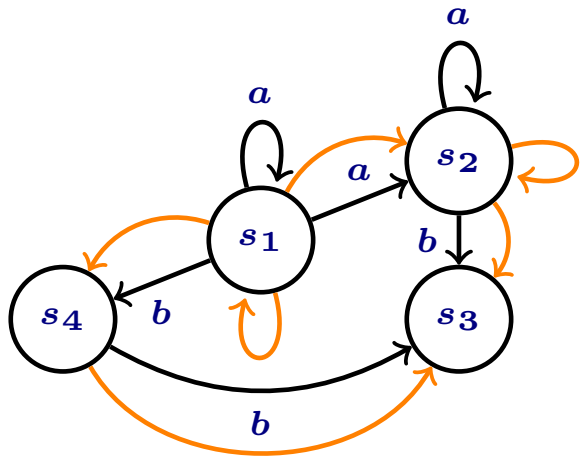
and so on.

Operations on relations (3)

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

- Union.

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



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

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

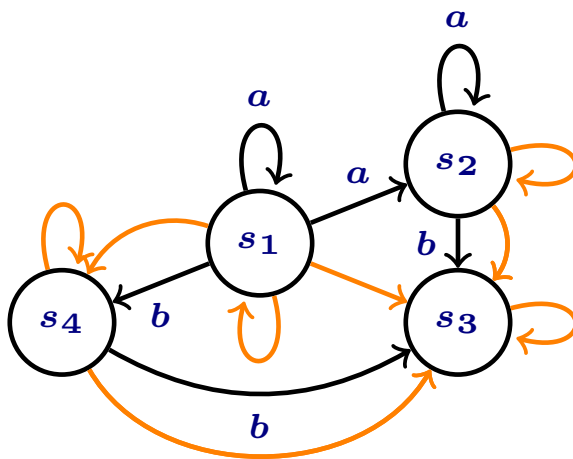
$$R_a \cup R_b = \{(s_1, s_1), (s_1, s_2), (s_2, s_2), (s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

Operations on relations (4)

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

- Repetition zero or more times.

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



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

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

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

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

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

⋮

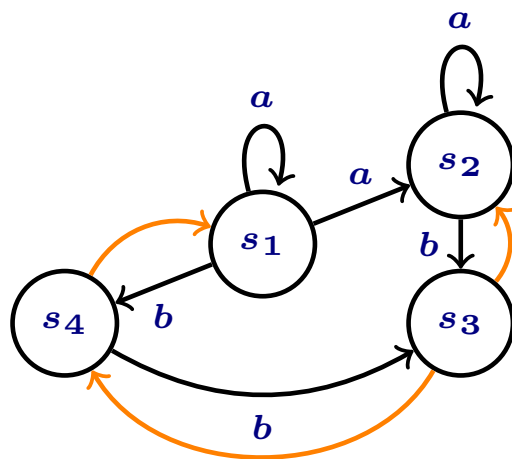
$$R_b^* = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4), (s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$$

Operations on relations (5)

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

- Converse.

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



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

$$\check{R}_b = \{(s_4, s_1), (s_3, s_2), (s_3, s_4)\}$$

Syntax (1)

The language of **propositional dynamic logic** (*PDL*) has two components, **formulas** φ and **actions** α .

- **Formulas** are built via the following rules.

- Every basic proposition is a formula

$$p, \quad q, \quad r, \quad \dots$$

- If φ and ψ are formulas, then the following are formulas:

$$\neg\varphi, \quad \varphi \wedge \psi, \quad \varphi \vee \psi, \quad \varphi \rightarrow \psi, \quad \varphi \leftrightarrow \psi$$

- If φ is a formula and α 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** φ and **actions** α .

- **Actions** are built via the following rules.

- Every basic action is a action

$$a, \quad b, \quad c, \quad \dots$$

- If α and β are actions, then the following are actions:

$$\alpha; \beta, \quad \alpha \cup \beta, \quad \alpha^*$$

- If φ is a formula, then the following is an action:

$$?\varphi$$

Intuitions and abbreviations

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

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

α^* **repetition**: execute α zero, one, or any *finite* number of times.

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

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

We abbreviate $p \vee \neg p$ as \top .

We abbreviate $\neg \top$ as \perp .

We abbreviate $\neg \langle \alpha \rangle \neg \varphi$ as $[\alpha] \varphi$.

$[\alpha] \varphi$ After any execution of α , φ is the case.

Some examples of formulas

$\langle \alpha \rangle \top$ α can be executed.

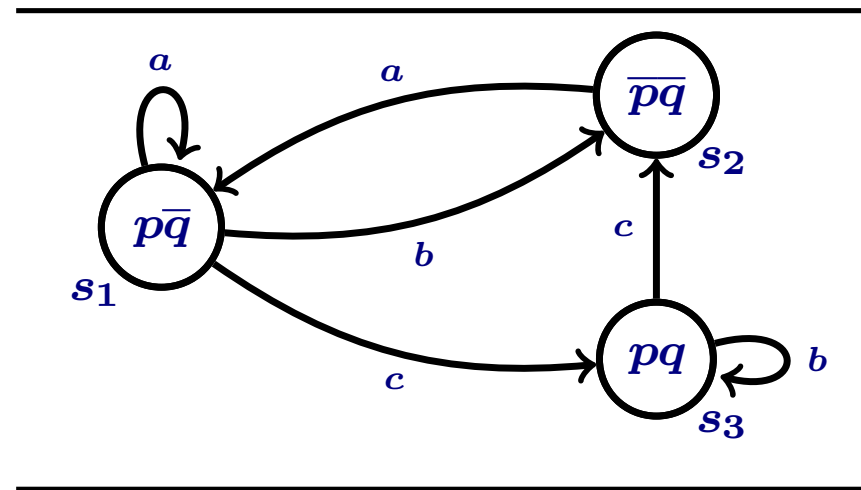
$[\alpha] \perp$ α cannot be executed.

$\langle \alpha \rangle \varphi \wedge \neg [\alpha] \varphi$ α 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:

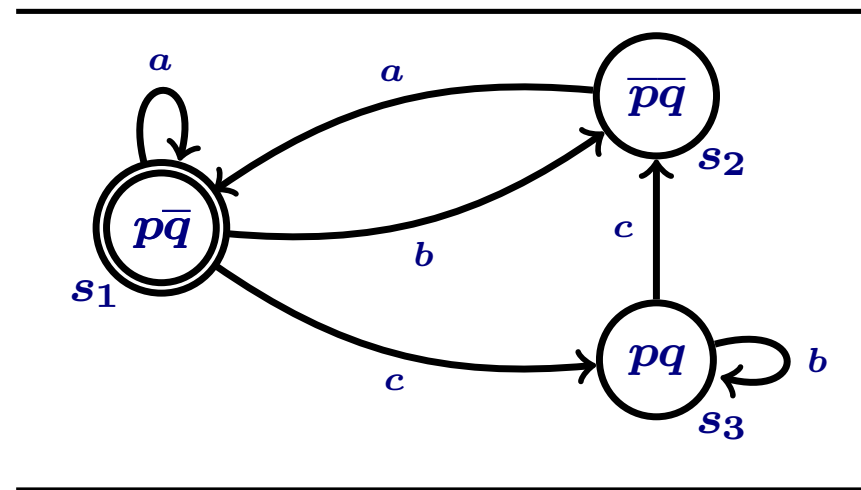
- a non-empty set S of **states**,
- a **valuation function**, V , indicating which atomic propositions are true in each state $s \in S$, and
- an **binary relation** R_a for each basic action a .



$$M = \langle S, R_a, V \rangle$$

The models (2)

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 $M = \langle S, R_\alpha, V \rangle$:

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

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

$$(M, s) \models \varphi \vee \psi \quad \text{iff} \quad (M, s) \models \varphi \text{ or } (M, s) \models \psi$$

$$\dots \quad \text{iff} \quad \dots$$

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

where the relation R_α is given, in case α is not a basic action, by

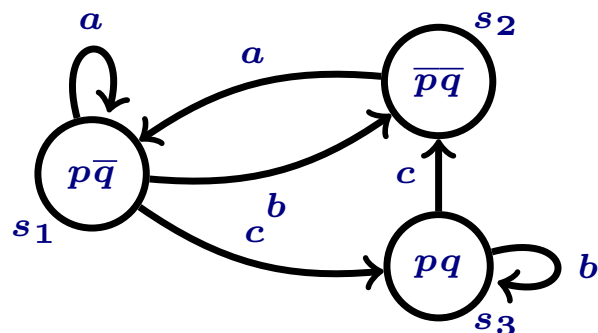
$$R_{\alpha;\beta} := R_\alpha \circ R_\beta$$

$$R_{\alpha \cup \beta} := R_\alpha \cup R_\beta$$

$$R_{\alpha^*} := (R_\alpha)^*$$

$$R_{?\varphi} := \{(s, s) \in S \times S \mid (M, s) \models \varphi\}$$

Example: building complex relations



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

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

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

$$R_{a \cup b} = \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\}$$

$$R_{a \cup c} = \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\}$$

$$R_{c;c} = \{(s_1, s_2)\}$$

$$R_{b;b} = \{\}$$

$$R_{?\neg(p \vee q)} = \{(s_2, s_2)\}$$

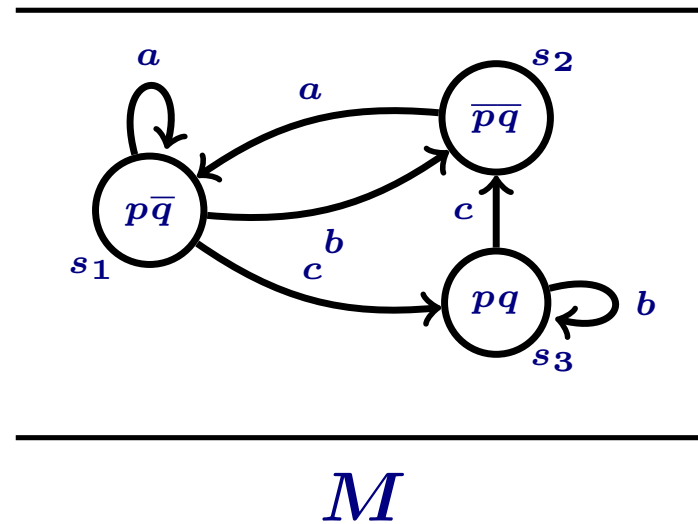
$$R_{?(p \vee q)} = \{(s_1, s_1), (s_3, s_3)\}$$

$$R_{?\neg(p \vee q); a; ?(p \vee q)} = \{(s_2, s_1)\}$$

$$R_{c;a} = \{(s_3, s_1)\}$$

$$R_{(c;a)^*} = \{(s_3, s_1), (s_1, s_1), (s_2, s_2), (s_3, s_3)\}$$

Example: evaluating formulas



$$(M, s_1) \models \langle a \cup b \rangle p \wedge \neg[a \cup b] p \quad \checkmark$$

$$(M, s_1) \models [b] \perp \quad \times$$

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

$$(M, s_2) \models \langle c^* \rangle \top \quad \checkmark$$

$$(M, s_3) \models [(c; a)^*] p \quad \checkmark$$

$$(M, s_3) \models [?p] p \quad \checkmark$$

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 α .
- ③ **Modus ponens** (MP): from φ and $\varphi \rightarrow \psi$, infer ψ .
- ④ **Necessitation** (Nec): from φ infer $[\alpha] \varphi$ for any action α .

Axiom system (2)

5 Principles for action operations:

- Test:

$$[?\psi] \varphi \leftrightarrow (\psi \rightarrow \varphi)$$

- Sequence:

$$[\alpha; \beta] \varphi \leftrightarrow [\alpha] [\beta] \varphi$$

- Choice:

$$[\alpha \cup \beta] \varphi \leftrightarrow ([\alpha] \varphi \wedge [\beta] \varphi)$$

- Repetition:

- Mix:

$$[\alpha^*] \varphi \leftrightarrow (\varphi \wedge [\alpha] [\alpha^*] \varphi)$$

- Induction:

$$\left(\varphi \wedge [\alpha^*] (\varphi \rightarrow [\alpha] \varphi) \right) \rightarrow [\alpha^*] \varphi$$

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

Example

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

From left to right:

- | | | |
|----|---|----------------------|
| 1. | $[(\alpha \cup \beta); \gamma] \varphi$ | Assumption |
| 2. | $[\alpha \cup \beta] [\gamma] \varphi$ | Sequence from step 1 |
| 3. | $[\alpha] [\gamma] \varphi \wedge [\beta] [\gamma] \varphi$ | Choice from step 2 |
| 4. | $[\alpha; \gamma] \varphi \wedge [\beta; \gamma] \varphi$ | 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 φ do α :

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

② REPEAT α UNTIL φ :

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

③ IF φ THEN α ELSE β :

$$(? \varphi; \alpha) \cup (? \neg \varphi; \beta)$$