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:

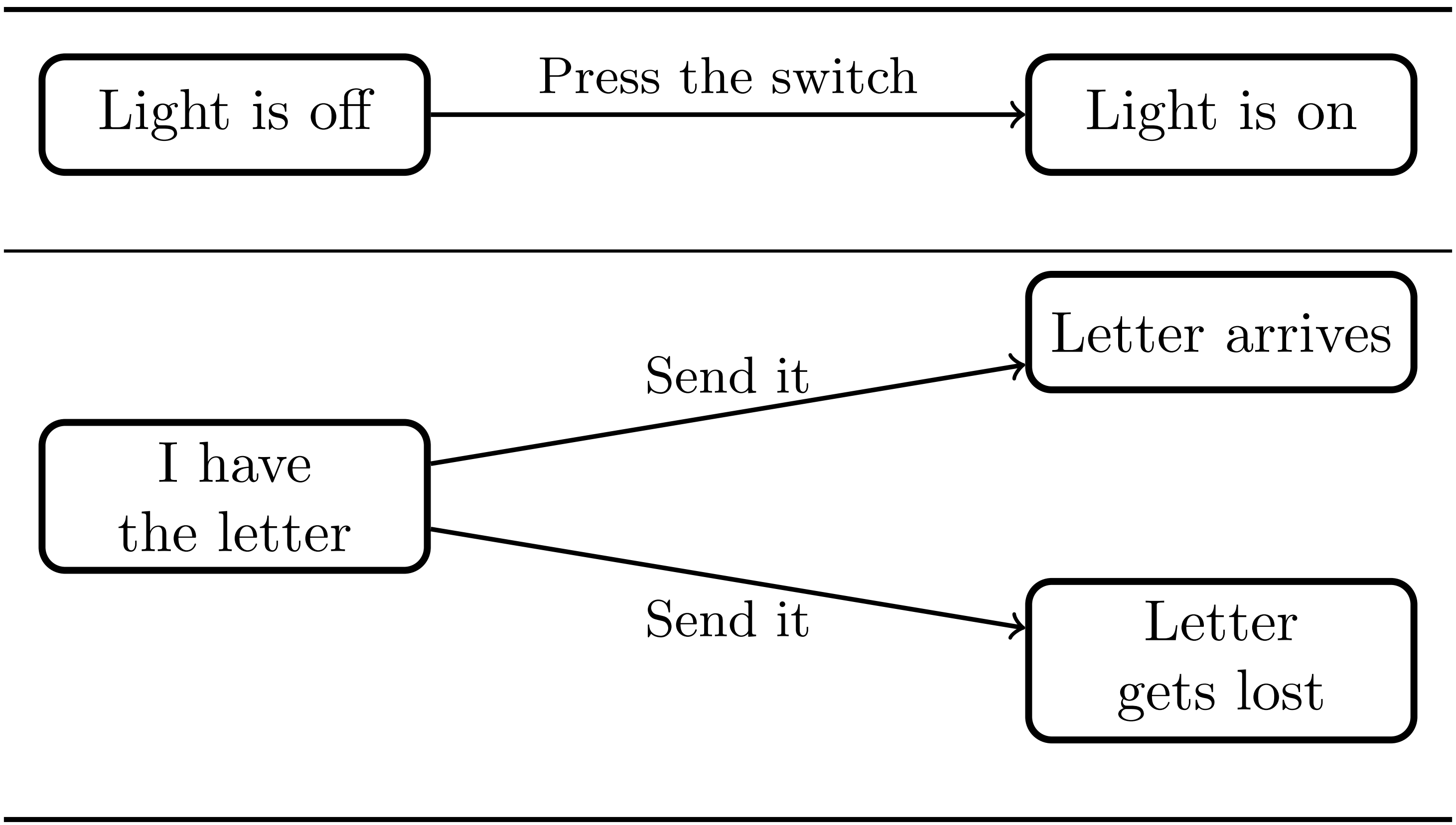
1. **WHILE P do A**
   This can be defined as the repetition of a test for 'P' and the execution of 'A', followed by a test for 'not A'.

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'.

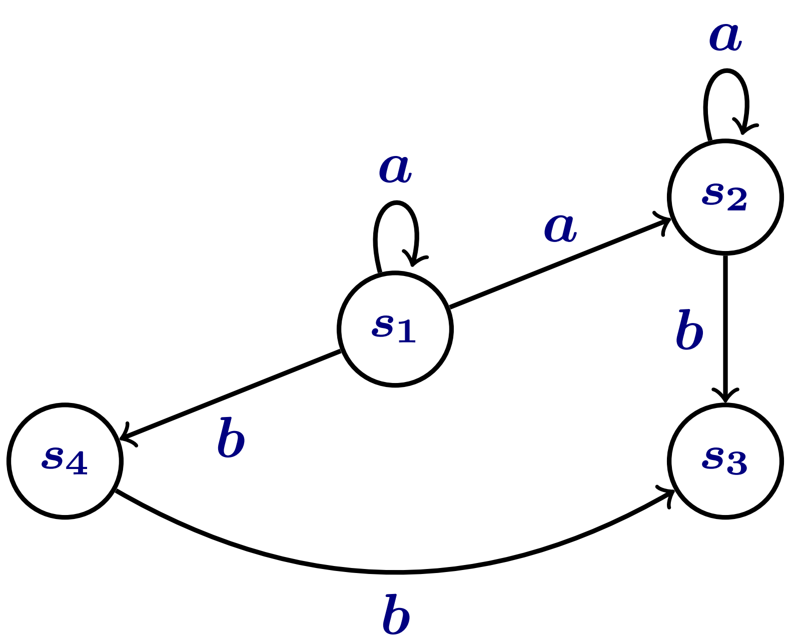
Representing actions abstractly (1)

We can see actions as transitions between states:



Representing actions abstractly (2)

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



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

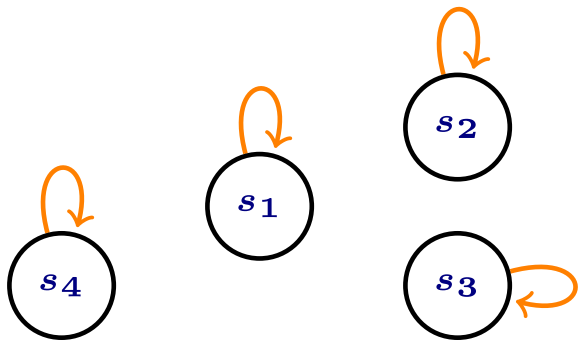
$$R_b := \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\}$$

Operations on relations (1)

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

- **Identity relation.**

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



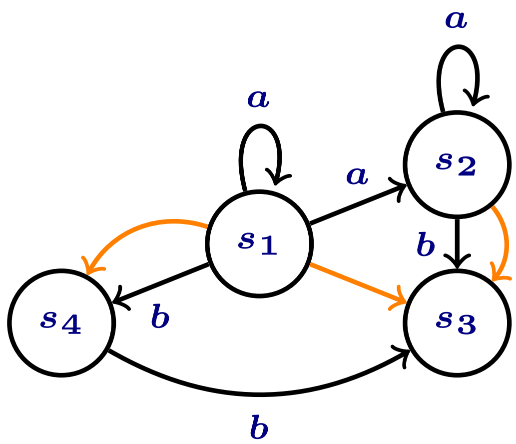
$$I = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\}$$

Operations on relations (2)

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

- **Composition.**

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



$$\begin{aligned} R_a &:= \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\} \\ R_b &:= \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \\ R_a \circ R_b &= \{(s_1, s_4), (s_1, s_3), (s_2, s_3)\} \end{aligned}$$

In particular, for any relation R_a, we have

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

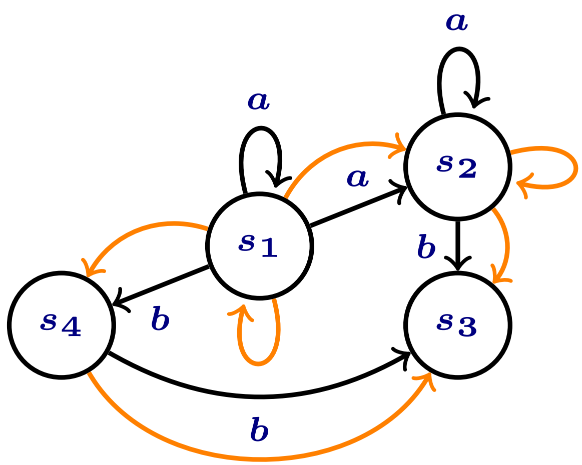
and so on.

Operations on relations (3)

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

- **Union.**

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



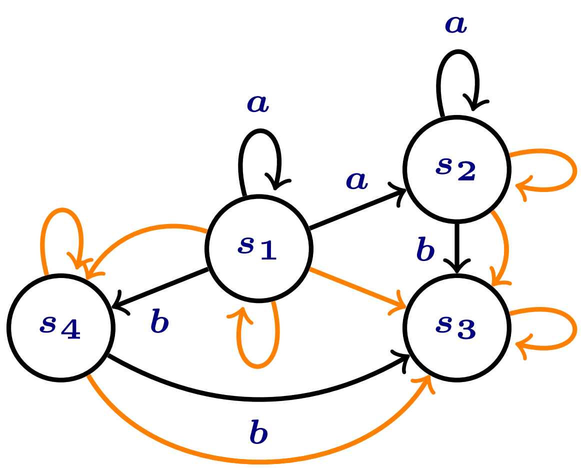
$$\begin{aligned} R_a &:= \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\} \\ R_b &:= \{(s_1, s_4), (s_2, s_3), (s_4, s_3)\} \\ R_a \cup R_b &= \{(s_1, s_1), (s_1, s_2), (s_2, s_2) \\ &\quad (s_1, s_4), (s_2, s_3), (s_4, s_3)\} \end{aligned}$$

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.**

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



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

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

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

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

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

$$\vdots$$

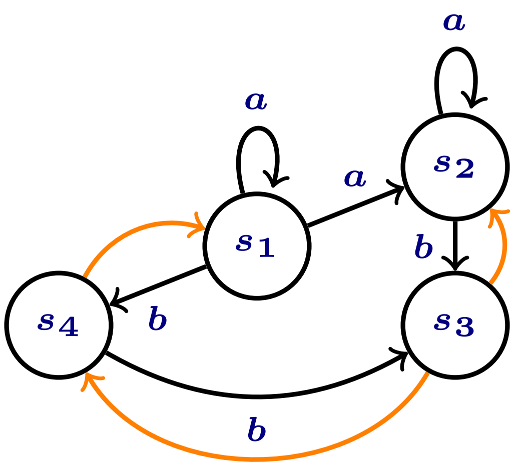
$$R_b^* = \{(s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4), (s_1, s_4), (s_2, s_3), (s_4, s_3), (s_1, s_3)\}$$

Operations on relations (5)

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

- **Converse.**

$$\breve{R_a} := \{(s', s) \mid R_a s s'\}$$



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

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

$$p, \quad q, \quad r, \quad \ldots$$

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

$$\neg\varphi, \quad \varphi \land \psi, \quad \varphi \lor \psi, \quad \varphi \rightarrow \psi, \quad \varphi \leftrightarrow \psi$$

  - 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** $\boldsymbol{\alpha}$.

- **Actions** are built via the following rules.
  - Every basic action is a action

$$\boldsymbol{a}, \quad \boldsymbol{b}, \quad \boldsymbol{c}, \quad \dots$$

  - If $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are actions, then the following are actions:

$$\boldsymbol{\alpha};\boldsymbol{\beta}, \quad \boldsymbol{\alpha} \cup \boldsymbol{\beta}, \quad \boldsymbol{\alpha}^*$$

  - 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 $\bot$.

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

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

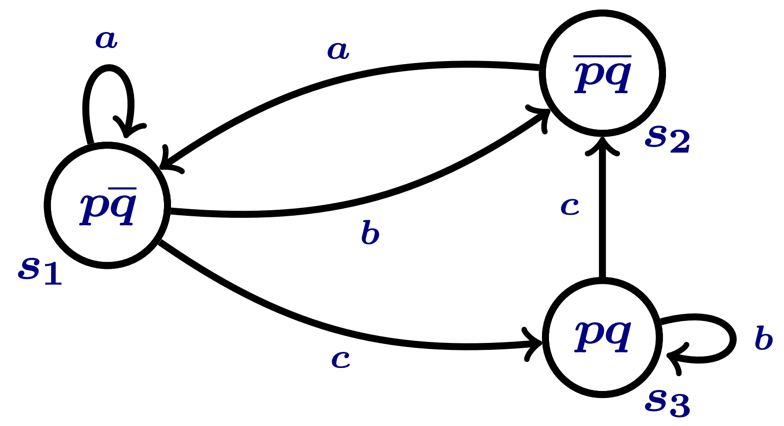
Some examples of formulas

$\langle \alpha \rangle \top$	α can be executed.
$[\alpha] \bot$	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:

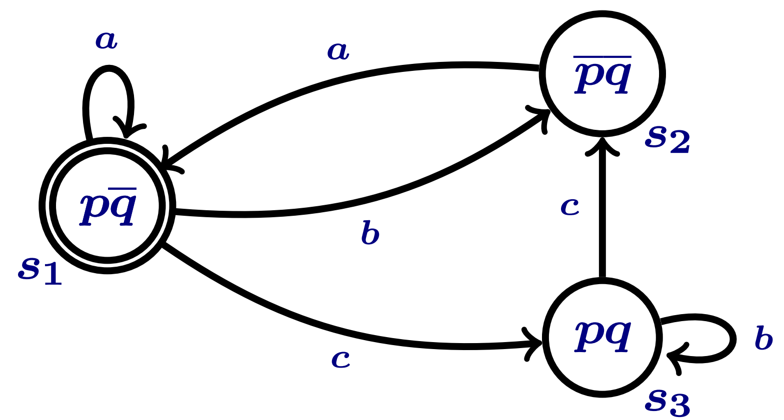
- a non-empty set S of **states**,
- a **valuation function**, V, indicating which atomic propositions are true in each state $s \in S$, and
- an **binary relation** R_a for each basic action a.



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

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

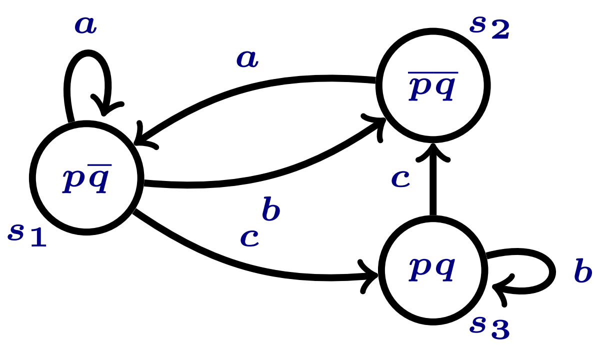
Take a pointed labelled transition system (M, s) with $M = \langle S, R_a, V\rangle$:

$$
\begin{array}{lll}
(M, s) \models p & \text{iff} & p \in V(s) \\
(M, s) \models \neg\varphi & \text{iff} & \text{it is not the case that } (M, s) \models \varphi \\
(M, s) \models \varphi \vee \psi & \text{iff} & (M, s) \models \varphi \ \text{ or } \ (M, s) \models \psi \\
\ldots & \text{iff} & \ldots \\
(M, s) \models \langle\alpha\rangle\,\varphi & \text{iff} & \text{there is a } t \in S \text{ such that } R_\alpha st \text{ and } (M, t) \models \varphi
\end{array}
$$

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

$$
\begin{aligned}
R_{\alpha;\beta} &:= R_\alpha \circ R_\beta \\
R_{\alpha\cup\beta} &:= R_\alpha \cup R_\beta \\
R_{\alpha^*} &:= (R_\alpha)^* \\
R_{?\varphi} &:= \{(s, s) \in S \times S \mid (M, s) \models \varphi\}
\end{aligned}
$$

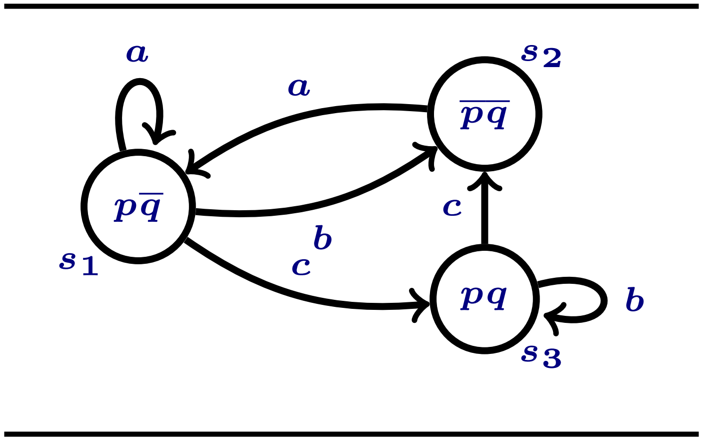
Example: building complex relations



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

$$\begin{aligned}
R_{a\cup b} &= \{(s_1, s_1), (s_2, s_1), (s_1, s_2), (s_3, s_3)\}\\
R_{a\cup c} &= \{(s_1, s_1), (s_2, s_1), (s_1, s_3), (s_3, s_2)\}\\
R_{c;c} &= \{(s_1, s_2)\}\\
R_{b;b} &= \{\ \}\\
R_{?\neg(p\vee q)} &= \{(s_2, s_2)\}\\
R_{?(p\vee q)} &= \{(s_1, s_1), (s_3, s_3)\}\\
R_{?\neg(p\vee q);a;?(p\vee q)} &= \{(s_2, s_1)\}\\
R_{c;a} &= \{(s_3, s_1)\}\\
R_{(c;a)^*} &= \{(s_3, s_1), (s_1, s_1), (s_2, s_2), (s_3, s_3)\}
\end{aligned}$$

Example: evaluating formulas



M

$(M, s_1) \models \langle a \cup b \rangle\, p \land \neg [a \cup b]\, p$ ✓

$(M, s_1) \models [b]\, \bot$ ✗

$(M, s_2) \models \langle a \rangle\, \top \to \langle b \rangle\, \top$ ✗

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

$(M, s_3) \models [(c; a)^*]\, p$ ✓

$(M, s_3) \models [?p]\, p$ ✓

Axiom system (1)

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

1. All propositional tautologies.
2. $[\alpha]\,(\varphi \to \psi) \to ([\alpha]\,\varphi \to [\alpha]\,\psi)$ for any action α.
3. **Modus ponens** (MP): from φ and $\varphi \to \psi$, infer ψ.
4. **Necessitation** (Nec): from φ infer $[\alpha]\,\varphi$ for any action α.

Axiom system (2)

5. **Principles for action operations:**
   - **Test:**
   $$[?\psi]\,\varphi \leftrightarrow (\psi \to \varphi)$$
   - **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:**
     $$\big(\varphi \wedge [\alpha^*]\,(\varphi \to [\alpha]\,\varphi)\big) \to [\alpha^*]\,\varphi$$

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:

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