Western Norway
University of
Applied Sciences

Verified software through Dynamic Logic

Ph.D. trial lecture

Patrick Stünkel
Bergen
9.2.2022

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

DNB-kunder belastet dobbelt igjen – omfanget er stort

DNB-kunder opplever torsdag morgen tekniske problemer som gjør at de har blitt trukket dobbelt beløp ved betaling.

Sverre Holm-Nilsen
Journalist

Johan B Sættem
@johansaettem
Journalist

Kilde: **NTB / NRK**

Publisert 23. mai 2019 kl. 07:15
Oppdatert 23. mai 2019 kl. 08:31

Artikkelen er mer enn ett år gammel.

PROBLEMER IGJEN: En rekke kunder har opplevd problemer med DNB de siste dagene. DNB sier at de har jobbet gjennom natta, men at problemene fortsetter torsdag morgen.
FOTO: INTS KALNINS / REUTERS

(NRK.no)

Boeing discovers new software problem in 737 Max

The US aerospace giant said it is "keeping our customers and suppliers informed" about the new software issue. Aviation regulators have grounded the 737 Max across the globe after two deadly crashes.

© picture-alliance/dpa/AP Images/T. S. Warren

(dw.com)

DNB med transaksjonsproblem – kunder får ikke lønn og feriepenger

En datafeil hos banken fører til at mange kunder ikke har fått utbetalt lønn og feriepenger fra DNB. Flere kunder reagerer kraftig, og Finanstilsynet følger med.

Torkil Stoltz
Journalist

Publisert 16. juni kl. 11:04
Oppdatert 16. juni kl. 16:29

Bankkunder i DNB får ikke utbetalt lønn og feriepenger på grunn av en feil i bankens systemer.

(NRK.no)

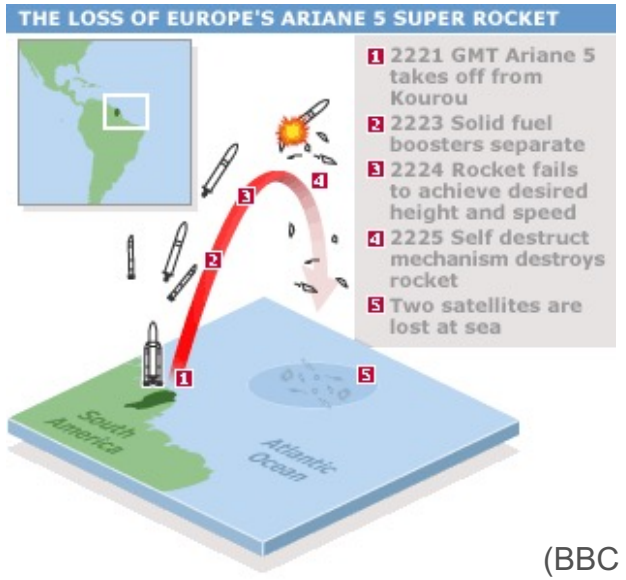
NYHETER | PUBLISERT 27.03.2019

Operasjoner utsatt i Helse Vest på grunn av dataproblemer

(sykepleien.no)

The Ariane-5 incident (1996)

Ariane 5 rocket launch explosion (youtube.com)



(BBC news)

execution

```java
public static void main(String[] args) {
      // small values
      double v1f = 42.0;
      System.out.println("Small Value as 64bit Floating Point No: " +  v1f);
      short v1i = (short) v1f;
      System.out.println("Small Value as 16bit Integer No: " + v1i);
      // big values
      double v2f = ( 987 * 213 * Math.pow(2, 10) ) / 3;
      System.out.println("Big Value as 64bit Floating Point No: " + v2f);
      short v2i = (short) v2f;
      System.out.println("Big Value as 16bit Integer No: " + v2i);
}
```

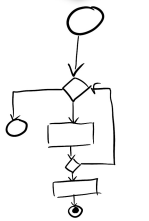
```
<terminated> Ariane5 [Java Application] /Users/past/Documents/dev/java/jdk-14
Small Value as 64bit Floating Point No: 42.0
Small Value as 16bit Integer No: 42
Big Value as 64bit Floating Point No: 7.1758848E7
Big Value as 16bit Integer No: -3072
```

How to write *correct* software?

Problem:

E.g. ”Write a program the calculates the greatest common divisor of two given numbers!”

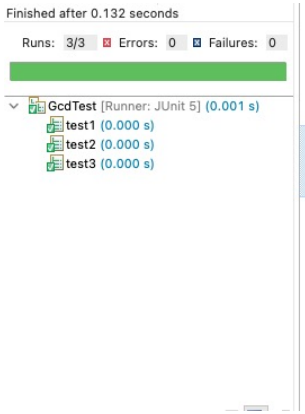
Solution:



```
public static int gcd(int a, int b) {
    if (a % b == 0) {
        return b;
    } else {
        return gcd(b, a % b);
    }
}
```

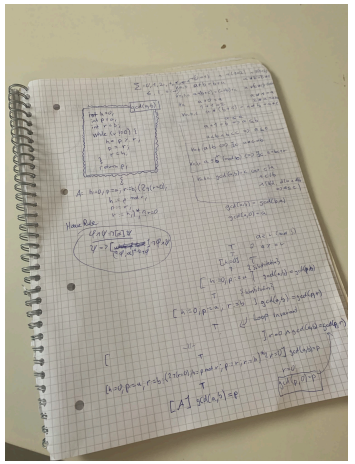
Verification:

Run Testcases

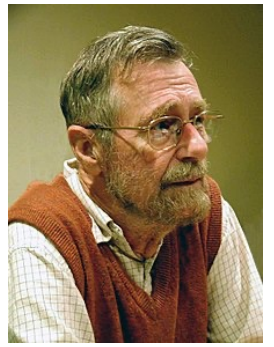



```
import static org.hamcrest.CoreMatchers.equalTo;
import org.junit.Test;
import static no.hvl.past.traillecture.Gcd.gcd;

public class GcdTest {

    @Test
    public void test1() {
        assertThat(gcd(15,12), equalTo(3));
    }

    @Test
    public void test2() {
        assertThat(gcd(25, 15), equalTo(5));
    }

    @Test
    public void test3() {
        assertThat(gcd(13, 9), equalTo(1));
    }

}
```

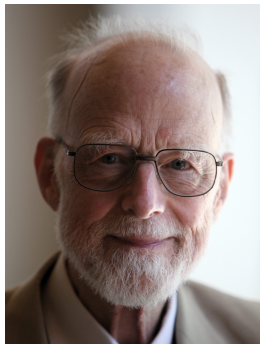
Prove Correctness



How to write correct software?



Edsger Dijkstra
(1930-2002)



”Tony” Hoare
(1934-)

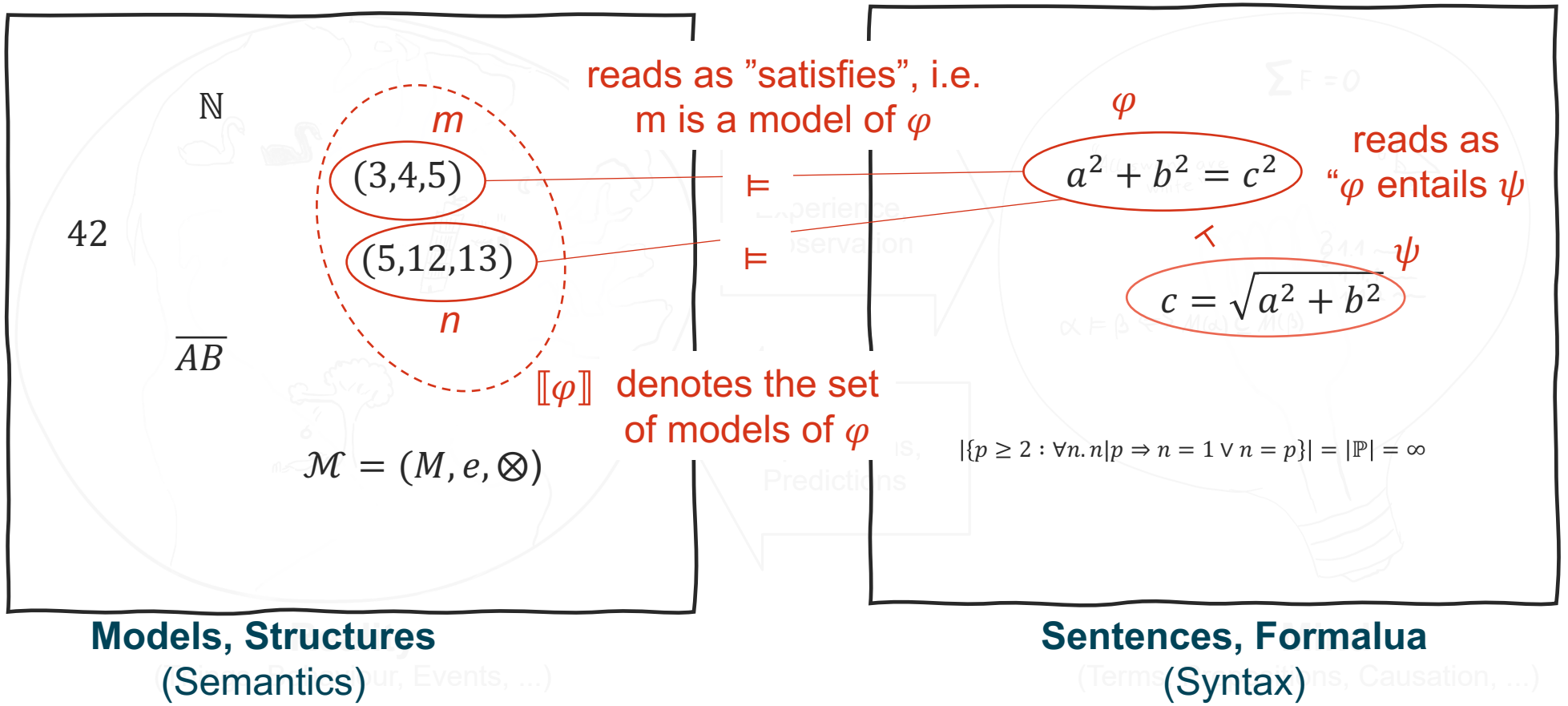
- There is a whole research discipline behind this: Formal Methods!
- Starting with
  - *A Discipline of Programming* by Dijkstra
  - *An Axiomatic Basis for Computer Programming* by Hoare
- Conferences: FM, FME, SEFM, ICFM
- Tools/Technologies: VDM, Z, B method, CASL, Maude, Spin, Alloy, Upaal, CPNtools, ...
- Two major branches
  - Formal Specification
  - **Formal Verification (← today)**

Agenda

- Motivation
- **Background: Logic**
- From Propositional (PL) to Dynamic Logic (DL):
  - Starting with «Propositions only», then...
  - ... adding dynamics: Modal Logic, ...
  - ... adding actions: Propositional Dynamic Logic, ...
  - ... finally, adding internal structure: Dynamic Logic.
- Dynamic Logic in Action
- Wrap-Up
  - Related (Logic) Work & Outlook
  - Summary
  - Bibliographic Notes

Logic: Concepts

- Logic ≙ Mathematics ∩ Philosophy
- *«The study of truth, valid arguments and correct reasoning»*



Logic: Ingredients & Properties

- Every «logic» needs at least
  - a definition of **models** (semantics)
  - a definition of **sentences** (i.e. a language)
  - a **satisfaction**-relationship ($\models$) between models and sentences,
- Optionally, it may further comprise an «**entailment**» ($\vdash$) relation between sentences (i.e. a reasoning system), which is called
  - *sound* if and only if (abbreviated: "*iff.*"):

$$\varphi \vdash \psi \text{ iff. } [\![\varphi]\!] \subseteq [\![\psi]\!]$$

  - *complete* if and only if every valid sentence is entailed

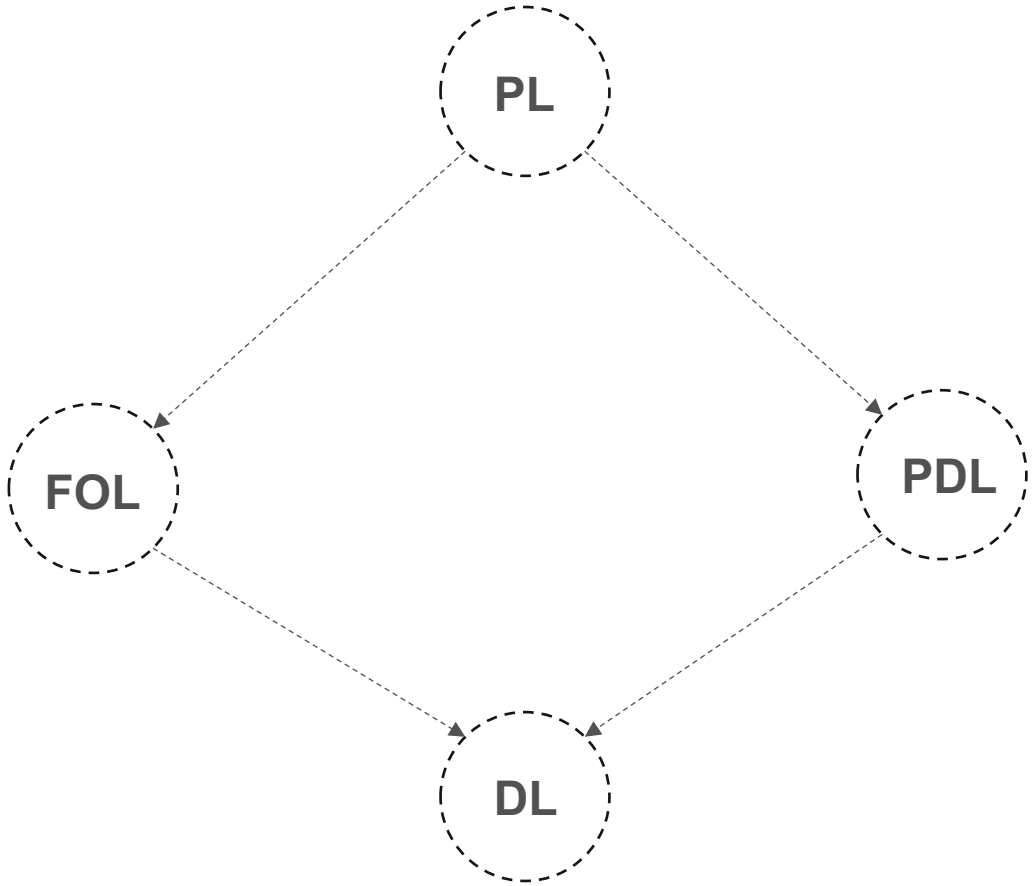
    (a sentence is valid if every possible model satisfies it, a.k.a. *Tautology*)

Overview

- Motivation
- Background: Logic
- **From Propositional (PL) to Dynamic Logic (DL):**
  - **Starting with «Propositions only», then...**
  - **... adding dynamics: Modal Logic, ...**
  - **... adding actions: Propositional Dynamic Logic, ...**
  - **... finally, adding internal structure: Dynamic Logic.**
- Dynamic Logic in Action
- Wrap-Up
  - Related (Logic) Work & Outlook
  - Summary
  - Bibliographic Notes

Slogan:
”Dynamic Logic is a combination of first-order predicate logic, modal logic & the algebra of regular actions”

The Destination: Dynamic Logic



Propositional Logic (Syntax)

- Let Φ_o be a set of **propositional symbols**:

E.g. $\{P \triangleq "There\ is\ pizza\ today", \mathrm{R} \triangleq "It\ is\ raining\ today", S \triangleq "At\ least\ 30\ students\ show\ up",$
$\mathrm{W} \triangleq "I\ am\ soaking\ wet", \mathrm{U} \triangleq "I\ brought\ my\ umbrella", T \triangleq "It\ is\ storming\ outside"\}$

- The language of propositional **formulae** Φ is defined via the following rules:

$\Phi ::= \top \mid \bot$

$\Phi ::= \mathrm{P} \mid \mathrm{R} \mid \ldots$ (for all symbols in Φ_o)

$\Phi ::= \neg\Phi$

$\Phi ::= \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \Rightarrow \Phi \mid \Phi \Leftrightarrow \Phi$

- Examples of grammatically *correct* sentences: $P \wedge \big(R \vee \neg(U \wedge T)\big), \neg\neg W, \neg\mathrm{R} \Rightarrow \bot, \ldots$
- Examples of grammatically *incorrect* sentences: $PUT, R\neg \wedge W, U \wedge\vee\wedge T, \ldots$

Propositional Logic (Semantics)

- A **model** in propositional logic is a valuation, assigning either $true$ or $false$ to every propositional symbol («truth table»)

P	S
$false$	$false$
$false$	$true$
$true$	$false$
$true$	$true$

- To check whether a formula is valid, one may enumerate all possible models and check that it evaluates to true for each of them (this is called *model checking*)

  E.g.: $(A \vee \neg(B \wedge A \wedge C)) \Leftrightarrow (A \Rightarrow D) \wedge (D \Leftrightarrow (E \wedge (B \vee \neg(C \wedge \neg A)))) \wedge (B \Leftrightarrow ((C \vee E) \wedge (B \vee C))$ valid ?

- **Exercise:** How many possibly models are there for our example with 6 prop. symbols?

Propositional Logic (Entailment)

- Instead of resorting to models, one may as well stay on the syntactical level!

$$\varphi \vdash \psi \text{ iff. } \varphi \Rightarrow \psi \text{ is valid}$$

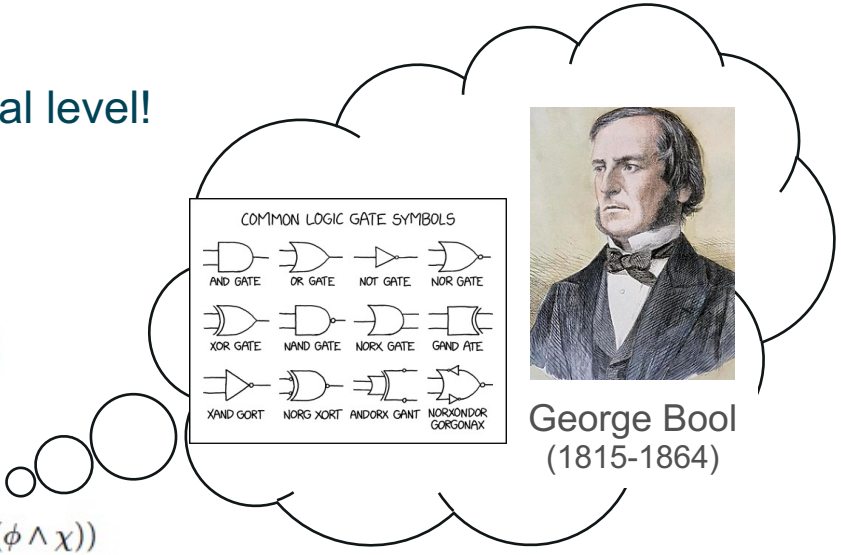
- Valid sentences (Tautologies) in propositional logic:

E.g.: $\phi \vee \neg\phi$

$$\phi \wedge \psi \Leftrightarrow \psi \wedge \phi$$
$$\phi \vee \psi \Leftrightarrow \psi \vee \phi$$
$$\phi \wedge (\psi \wedge \chi) \Leftrightarrow (\phi \wedge \psi) \wedge \chi$$
$$\phi \vee (\psi \vee \chi) \Leftrightarrow (\phi \vee \psi) \vee \chi$$

$$\neg(\neg\phi) \Leftrightarrow \phi$$
$$(\phi \Rightarrow \psi) \Leftrightarrow (\neg\psi \Rightarrow \neg\phi)$$
$$(\phi \Rightarrow \psi) \Leftrightarrow (\neg\phi \vee \psi)$$
$$(\phi \Leftrightarrow \psi) \Leftrightarrow ((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi))$$

$$\neg(\phi \wedge \psi) \Leftrightarrow (\neg\phi \vee \neg\psi)$$
$$\neg(\phi \vee \psi) \Leftrightarrow (\neg\phi \wedge \neg\psi)$$
$$(\phi \wedge (\psi \vee \chi)) \Leftrightarrow ((\phi \wedge \psi) \vee (\phi \wedge \chi))$$
$$(\phi \vee (\psi \wedge \chi)) \Leftrightarrow ((\phi \vee \psi) \wedge (\phi \vee \chi))$$



George Bool
(1815-1864)

- When both $\varphi \vdash \psi$ and $\psi \vdash \varphi$, then φ and ψ are called **equivalent** (written $\varphi \equiv \psi$)

$$\varphi \equiv \psi \text{ iff. } \varphi \Leftrightarrow \psi \text{ is valid}$$

- Two main **inference rules** in propositional logic:
  - (*Modus Ponens*) Knowing that φ and $\varphi \Rightarrow \psi$ are satisfied, infer that ψ is satisfied
  - (*And Elimination*) Knowing that $\varphi \wedge \psi$ is satisfied, infer that φ is satisfied

Propositional Logic (Summary)

- **Sentences**:
  are Built from standard logical connectives ($\wedge$,$\vee$,$\neg$, ...) over propositional symbols Φ_o
- **Models**
  are valuations: $\mathrm{m}: \Phi_o \rightarrow \{true, false\}$
- **Satisfaction**
  is given by the ”standard” interpretation of truth tables

- Propositional Logic has an inference system that is sound and complete!

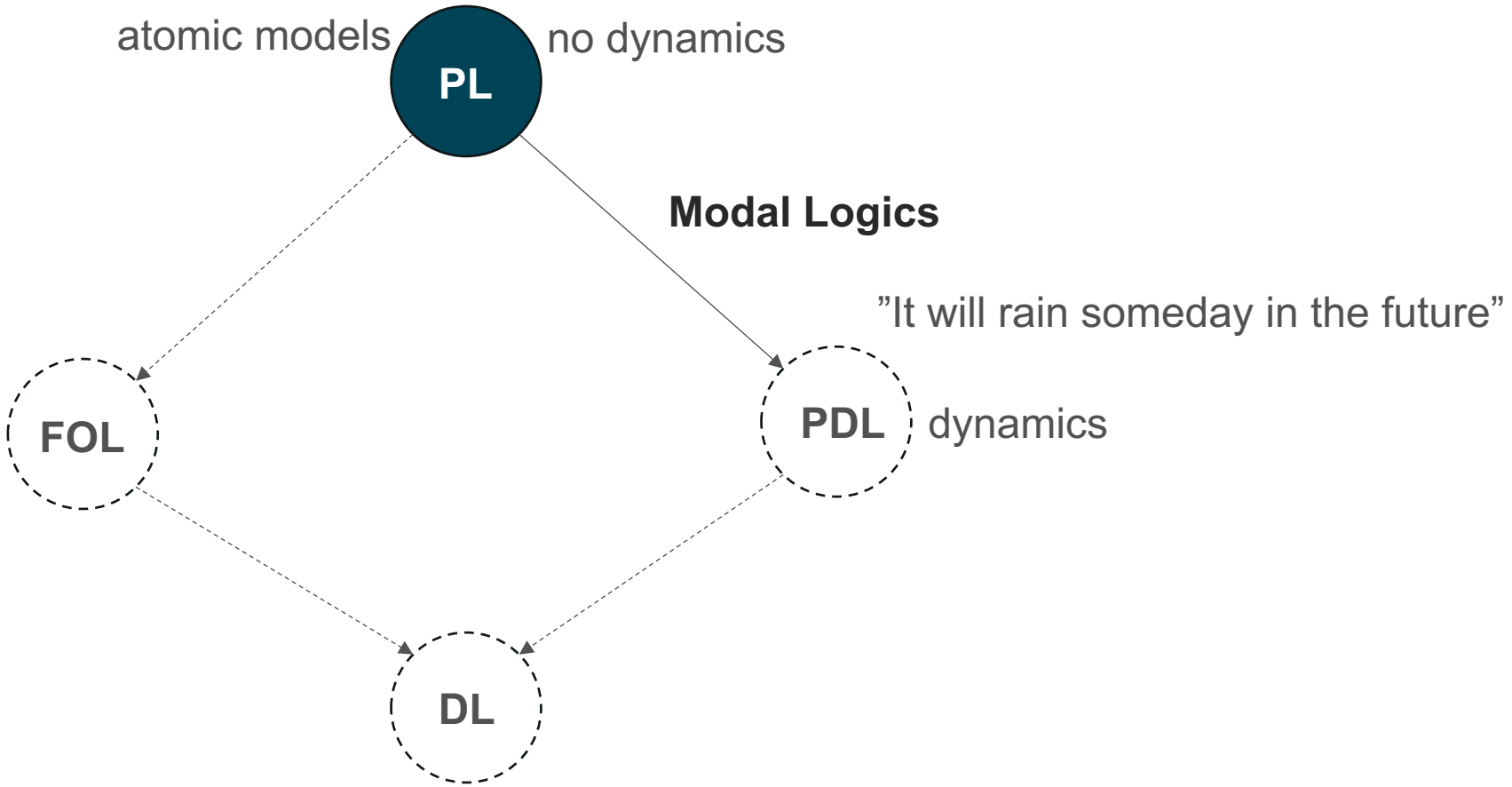
$\phi \vee \neg\phi$	(Excluded middle)
$\phi \wedge \psi \Leftrightarrow \psi \wedge \phi$	(Commutativity of $\wedge$)
$\phi \vee \psi \Leftrightarrow \psi \vee \phi$	(Commutativity of $\vee$)
$\phi \wedge (\psi \wedge \chi) \Leftrightarrow (\phi \wedge \psi) \wedge \chi$	(Associativity of $\wedge$)
$\phi \vee (\psi \vee \chi) \Leftrightarrow (\phi \vee \psi) \vee \chi$	(Associativity of $\vee$)
$\neg(\neg\phi) \Leftrightarrow \phi$	(Double negation elimiation)
$(\phi \Rightarrow \psi) \Leftrightarrow (\neg\psi \Rightarrow \neg\phi)$	(Contraposition)
$(\phi \Rightarrow \psi) \Leftrightarrow (\neg\phi \vee \psi)$	(Implication elimination)
$(\phi \Leftrightarrow \psi) \Leftrightarrow ((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi))$	(Biconditional elimination)
$\neg(\phi \wedge \psi) \Leftrightarrow (\neg\phi \vee \neg\psi)$	(De Morgan)
$\neg(\phi \vee \psi) \Leftrightarrow (\neg\phi \wedge \neg\psi)$	(De Morgan)
$(\phi \wedge (\psi \vee \chi)) \Leftrightarrow ((\phi \wedge \psi) \vee (\phi \wedge \chi))$	(Distributivity of $\wedge$ over $\vee$)
$(\phi \vee (\psi \wedge \chi)) \Leftrightarrow ((\phi \vee \psi) \wedge (\phi \vee \chi))$	(Distributivity of $\vee$ over $\wedge$)

Axioms

$\dfrac{\phi, \phi \Rightarrow \psi}{\psi}$	(Modus Ponens)
$\dfrac{\phi \wedge \psi}{\phi}$	(And Elimination)

Inference Rules

The Destination: Dynamic Logic



Modal Logic: The general idea

- Distinguish different «modes» of truth!

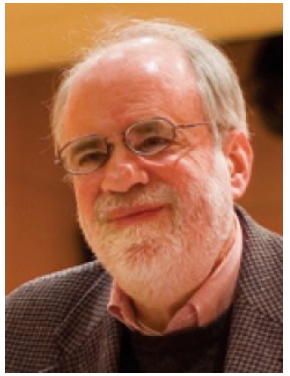
Possibility
(Firstness)

Actuality
(Secondness)

Necessity
(Thirdness)

- Thus: New *types* of sentences via connectors ◊ and □
  - ◊φ means «φ possibly holds»
  - □φ means «φ necessarily holds»

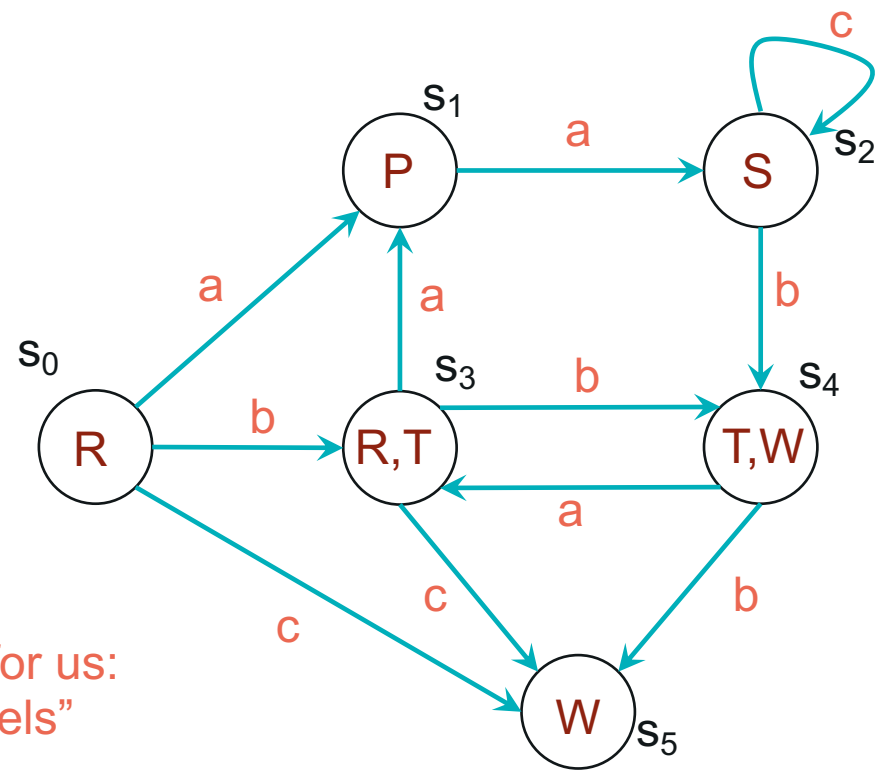
Models in Modal Logic



Saul Kripke
(1940-)

› ...thus, models must look a bit differently

A labeled Kripke-structure is defined as follows: $\boldsymbol{K} = (S, T \subseteq S \times A_0 \times S, L: S \to 2^{\Phi_0})$



Important for us:
”Action labels”

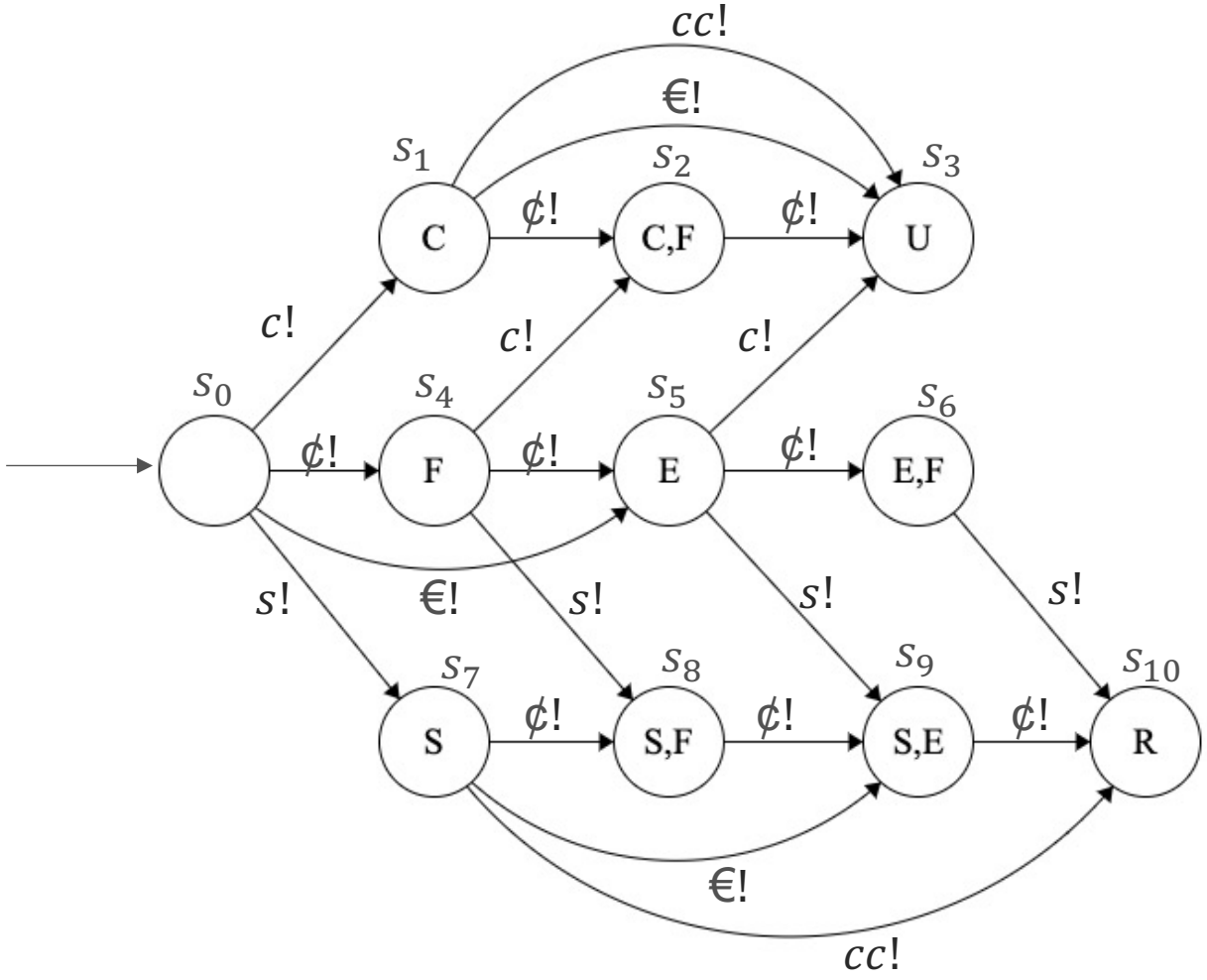
$$S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$$

$$T = \{(s_0, a, s_1), (s_0, b, s_3), (s_0, c, s_5),$$
$$(s_1, a, s_2),$$
$$(s_2, c, s_2), (s_2, b, s_4),$$
$$(s_3, a, s_1), (s_3, b, s_4),$$
$$(s_4, a, s_3), (s_4, b, s_5)\}$$

$$L = \{s_0 \mapsto \{R\}, s_1 \mapsto \{P\}, s_2 \mapsto \{S\},$$
$$s_3 \mapsto \{R, T\}, s_4 \mapsto \{T, W\}, s_5 \mapsto \{W\}\}$$

Action Structures

Example: Vending Machine m



Proposition Symbols Φ_0:
C ≙ "**C**offee selected"
S ≙ "**S**nack selected"
U ≙ "Coffee c**u**p output"
R ≙ "Snack **r**eceived"
E ≙ "1**€** coin inserted"
F = "**50¢** coin inserted"

Atomic Actions A_0:
c! ≙ "select **C**offee"
s! ≙ "select **S**nack"
€! ≙ "insert 1**€** coin"
¢! ≙ "insert 50**¢** coin"
cc! ≙ "swipe **c**redit **c**ard"

What's new

"Action-specific" modal Connectors:

- $(\langle a \rangle \varphi)_{a \in A}$ instead of "just" $\Diamond \varphi$
- $([a]\varphi)_{a \in A}$ instead of "just" $\Box \varphi$

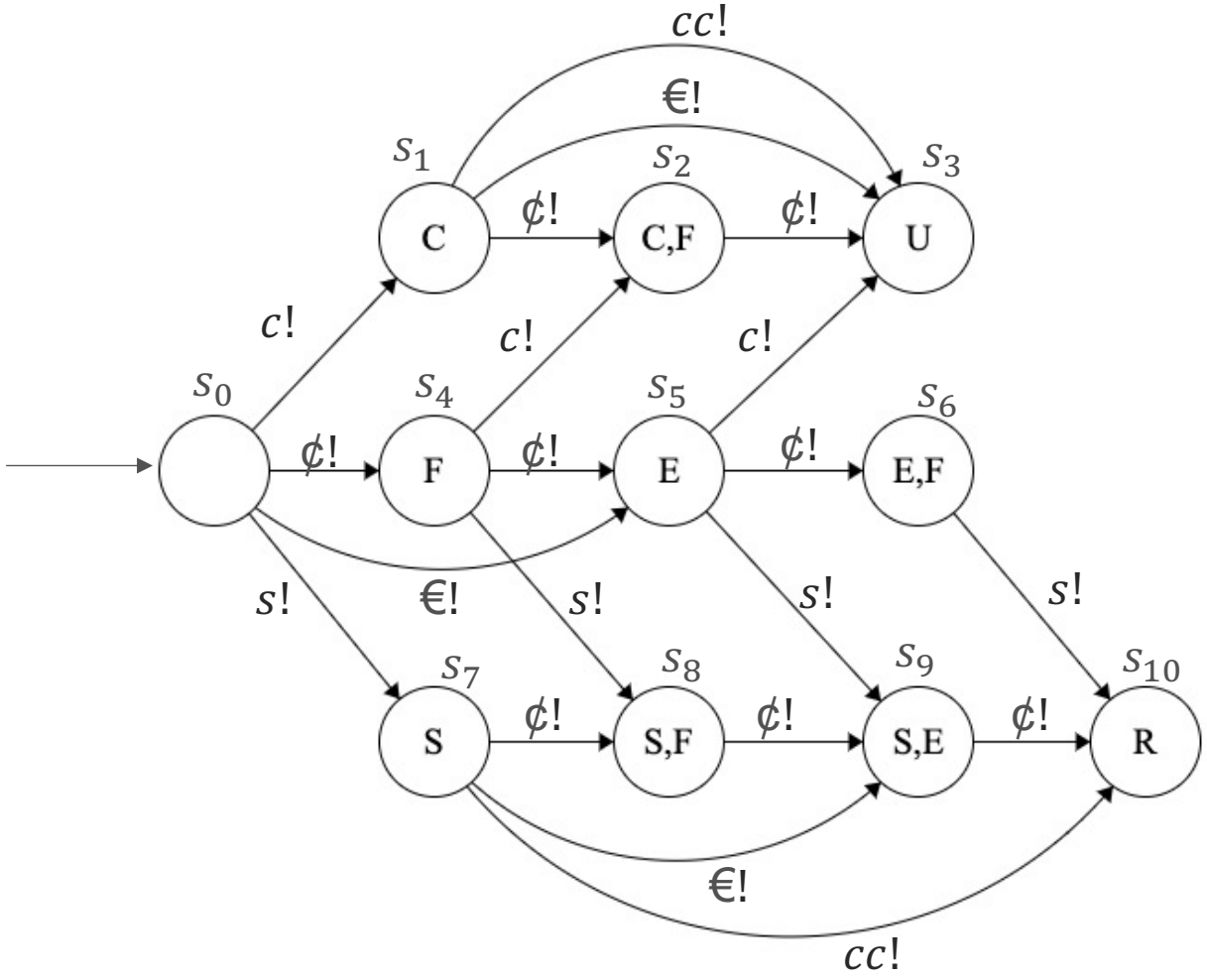
and satisfaction of a formula φ in a model m depend on the current state s, i.e. for $a \in A_0$, $P \in \Phi_0$,$\varphi \in \Phi$:

- $m, s \vDash P$ iff. $P \in L_m(s)$
- $m, s \vDash \langle a \rangle \varphi$ iff. there exists $(s, a, s') \in T_m$ such that $m, s' \vDash \varphi$
- $m, s \vDash [a] \varphi$ iff. for all $(s, a, s') \in T_m$ it holds that $m, s' \vDash \varphi$
- $m, s \vDash \varphi \wedge \psi$ iff. $m, s \vDash \varphi$ and $m, s \vDash \psi$, ... (analogous for the other connectives)

Finally, a formula φ is satisfied in m, iff. it is satisfied in *all* states!

Action Structures

Example: Vending Machine m



What's new

"Action-specific" modal Connectors:

- $(\langle a \rangle \varphi)_{a \in A}$ instead of "just" $\Diamond \varphi$
- $([a]\varphi)_{a \in A}$ instead of "just" $\Box \varphi$

and satisfaction of a formula φ in a model m depend on the current state s, i.e. for $a \in A_0$, $P \in \Phi_0$,$\varphi \in \Phi$:

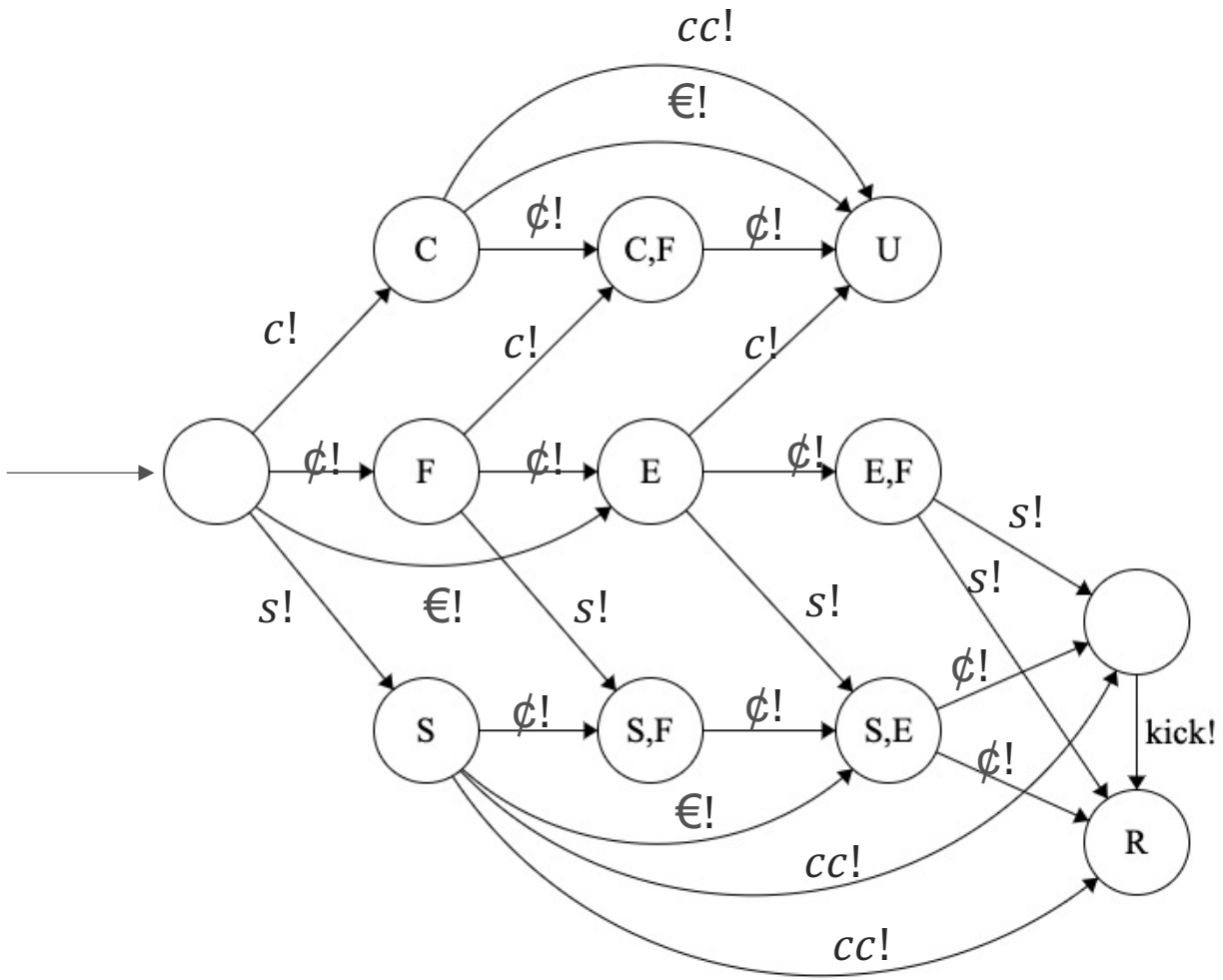
- $m, s \vDash P$ iff. $P \in L_m(s)$
- $m, s \vDash \langle a \rangle \varphi$ iff. there exists $(s, a, s') \in T_m$ such that $m, s' \vDash \varphi$
- $m, s \vDash [a]\varphi$ iff. for all $(s, a, s') \in T_m$ it holds that $m, s' \vDash \varphi$
- $m, s \vDash \varphi \wedge \psi$ iff. $m, s \vDash \varphi$ and $m, s \vDash \psi$, ... (analogous for the other connectives)

Finally, a formula φ is satisfied in m, iff. it is satisfied in *all* states!

Exercise: satisfied or not?

- $m, s_0 \vDash \langle c! \rangle C \wedge [s!]S$
- $m, s_0 \vDash [€!][c!]U$
- $m \vDash [€!](E \vee (U \vee R))$
- $m \vDash \langle €! \rangle (E \vee (U \vee R))$

Non-determinism



Exercise: satisfied or not?

- $m, s_0 \vDash [€!][¢!][s!]R$
- $m, s_0 \vDash [€!][¢!][s!][kick!]R$

Discourse: Relations & their *regular* compositions

- Another way to look at $T \subseteq S \times A_0 \times S \implies$ as a family of relations $T = (R_a)_{a \in A_0}$

- Let's now define combinators for combining actions:
  1. **Sequence** (written $a; b$) is defined via *Relation composition* $\circ$

$$R_b \circ R_a := \{(a,b) \mid \exists c.\, (a,c) \in R_a \wedge (c,b) \in R_b\}$$

  2. **Choice** (written $a \cup b$) is defined via *Union*

$$R_a \cup R_b := \{(a,b) \mid (a,b) \in R_a \vee (a,b) \in R_b\}$$

  3. **Iteration** (written a^*) is defined via *reflexive transitive Closure*

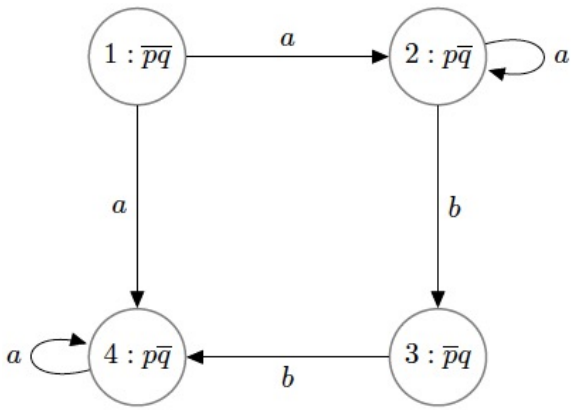
$$R_a^0 := \{(a,a) \mid a \in S\}$$

$$R_a^1 := R_a$$

$$R_a^{n+1} := (R_a^n \circ R_a^{n-1}) \qquad (n>1)$$

$$R_a{}^* := \bigcup_i R_a^i$$

Example



Propositional Dynamic Logic (PDL) - Syntax

- So far, we had:

$$
\begin{aligned}
A_0 &:= \{a, b, c!, €! \ldots\} && \text{((atomic) action labels)} \\
\Phi_0 &:= \{P, R, S, \ldots\} && \text{(propositional symbols)} \\
\Phi^{\mathbf{PL}} &:= \bot \mid \top \mid \Phi_0 \mid \neg\Phi^{\mathbf{PL}} \mid \\
&\quad \Phi^{\mathbf{PL}} \wedge \Phi^{\mathbf{PL}} \mid \Phi^{\mathbf{PL}} \vee \Phi^{\mathbf{PL}} \mid \\
&\quad \Phi^{\mathbf{PL}} \Rightarrow \Phi^{\mathbf{PL}} \mid \Phi^{\mathbf{PL}} \Leftrightarrow \Phi^{\mathbf{PL}} && \text{(propositional Sentences)}
\end{aligned}
$$

- First, add a **4.** action combinator: **test** (written $\varphi?$) with $\varphi \in \Phi$:

$$R_{\varphi?} := \{(s, s) \mid m, s \vDash \varphi\}$$

- Now, define the set of *all actions* A, and

Did you notice?

$$A ::= A_0 \mid A^* \mid A\,;A \mid A \cup A \mid \Phi^{\mathbf{PL}}?$$

- the set of all PDL formulae:

$$\Phi^{\mathbf{PDL}} ::= \Phi^{\mathbf{PL}} \mid \langle A \rangle \Phi^{\mathbf{PDL}} \mid [A]\Phi^{\mathbf{PDL}}$$

There is a “programming language” hidden in PDL!

- How, would a programmer call the following “actions”? (for $\varphi \in \Phi^{\mathbf{PL}}, \alpha, \beta \in A$)

$(\varphi?;\alpha) \cup ((\neg\varphi)?;\beta)$	**if** φ **then** α **else** β
$(\varphi?;\alpha)^*(\neg\varphi)?$	**while** φ **do** α
$\top?$	**skip**
$\bot?$	**fail**

Exercise: What is the PDL equivalent of ”repeat”

repeat α **until** φ

Propositional Dynamic Logic (PDL) - Entailment

- PDL has an inference procedure that is *sound* and *complete!*
- The axioms are given by:

$\phi \vee \neg\phi$	(Excluded middle)
$\phi \wedge \psi \Leftrightarrow \psi \wedge \phi$	(Commutativity of $\wedge$)
$\phi \vee \psi \Leftrightarrow \psi \vee \phi$	(Commutativity of $\vee$)
$\phi \wedge (\psi \wedge \chi) \Leftrightarrow (\phi \wedge \psi) \wedge \chi$	(Associativity of $\wedge$)
$\phi \vee (\psi \vee \chi) \Leftrightarrow (\phi \vee \psi) \vee \chi$	(Associativity of $\vee$)
$\neg(\neg\phi) \Leftrightarrow \phi$	(Double negation elimiation)
$(\phi \Rightarrow \psi) \Leftrightarrow (\neg\psi \Rightarrow \neg\phi)$	(Contraposition)
$(\phi \Rightarrow \psi) \Leftrightarrow (\neg\phi \vee \psi)$	(Implication elimination)
$(\phi \Leftrightarrow \psi) \Leftrightarrow ((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi))$	(Biconditional elimination)
$\neg(\phi \wedge \psi) \Leftrightarrow (\neg\phi \vee \neg\psi)$	(De Morgan)
$\neg(\phi \vee \psi) \Leftrightarrow (\neg\phi \wedge \neg\psi)$	(De Morgan)
$(\phi \wedge (\psi \vee \chi)) \Leftrightarrow ((\phi \wedge \psi) \vee (\phi \wedge \chi))$	(Distributivity of $\wedge$ over $\vee$)
$(\phi \vee (\psi \wedge \chi)) \Leftrightarrow ((\phi \vee \psi) \wedge (\phi \vee \chi))$	(Distributivity of $\vee$ over $\wedge$)

Propositional Tautologies

+

$[\alpha]\phi \Leftrightarrow \neg\langle\alpha\rangle(\neg\phi)$	(Duality)
$[\alpha](\phi \Rightarrow \psi) \Rightarrow ([\alpha]\phi \Rightarrow [\alpha]\psi)$	("K axiom": $[\alpha]$ distributes over $\Rightarrow$)
$[\alpha](\phi \wedge \psi) \Leftrightarrow [\alpha]\phi \wedge [\alpha]\psi$	(Modal And distribution)
$[\alpha \cup \beta]\phi \Leftrightarrow [\alpha]\phi \wedge [\beta]\phi$	(Choice)
$[\alpha;\beta]\phi \Leftrightarrow [\alpha][\beta]\phi$	(Sequuence)
$[\psi?]\phi \Leftrightarrow (\psi \Rightarrow \phi)$	(Test)
$(\phi \wedge [\alpha][\alpha^*]\phi) \Leftrightarrow [\alpha^*]\phi$	(Mix)
$(\phi \wedge [\alpha^*](\phi \Rightarrow [\alpha]\phi)) \Rightarrow [\alpha^*]\phi$	(Induction)

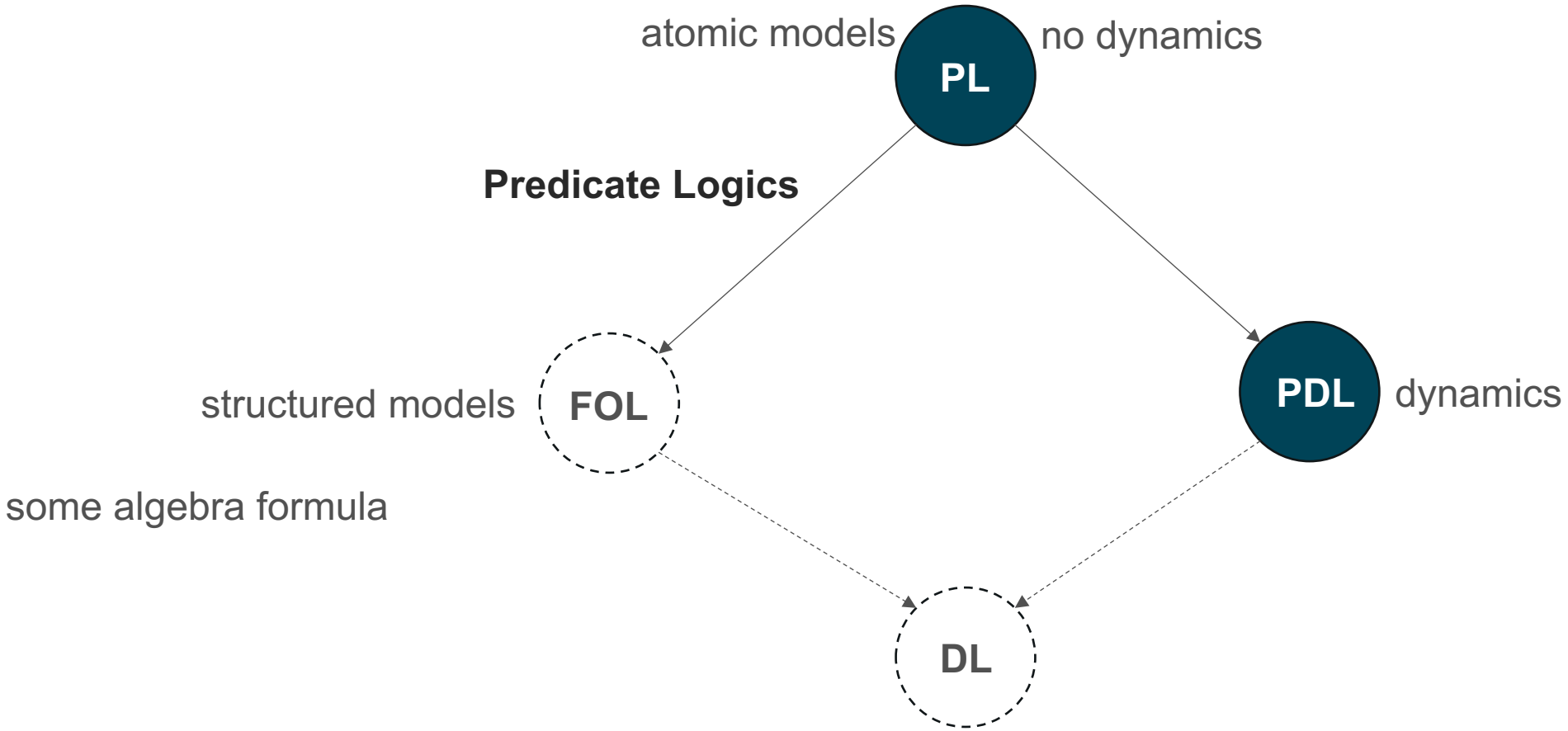
PDL Tautologies

- The inference rules are:

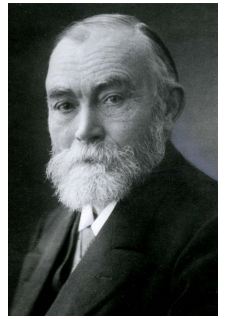
$$\frac{\phi,\ \phi \Rightarrow \psi}{\psi} \quad \text{(Modus Ponens)}$$

$$\frac{\phi}{[\alpha]\phi} \quad \text{(Modal Generalisation)}$$

The Destination: Dynamic Logic

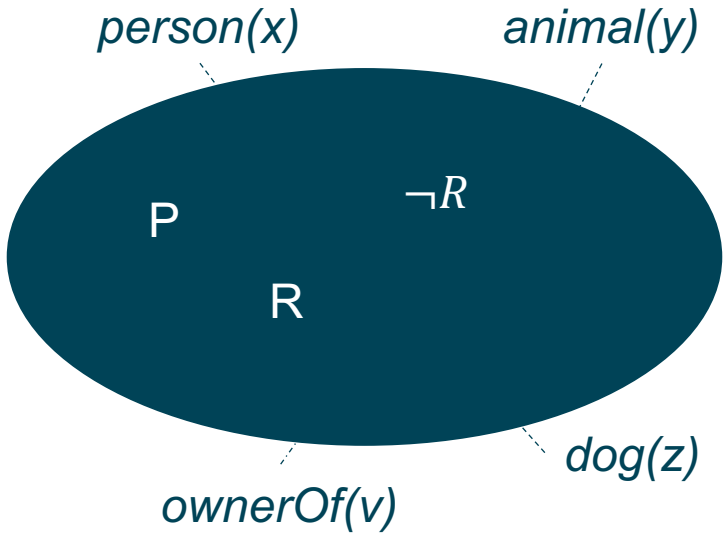


(First-order) predicate Logic (FOL) - idea



Gottlob Frege
(1848-1925)

- So far models (states) have been atomic...



- now they have internal structure, which is composed of
  - **objects**
  - **relations**
  - **functions**

(First-order) predicate Logic (FOL) - Syntax

- First, we need a **signature** Σ comprising
  - A set of *variable symbols* V
  - A set of *predicate symbols* P
  - A set of *function symbols* F
  - A function $ar: P \sqcup F \to \mathbb{N}$ assigning an *arity* (number of slots) to each predicate and function symbol

Σ

$V ::= a \mid b \mid c \mid m \mid n \mid p \mid q \mid r$

$P ::= = \mid \leq \mid \mathbf{div}$ } arity = 2

$F ::= 0 \mid 1 \mid 2$ } arity = 0

$::= -$ } arity = 1

$::= + \mid *$ } arity = 2

- The formulae of FOL are defined as follows

$\forall m.\, \forall n.\, (m \;\mathbf{div}\; n \Leftrightarrow \exists q.\, (m * q = n))$

(Terms) $T_\Sigma ::= V \mid f(\overbrace{T_\Sigma, \ldots, T_\Sigma}^{n \text{ times}})$ $f \in F$ and $ar(f) = n$

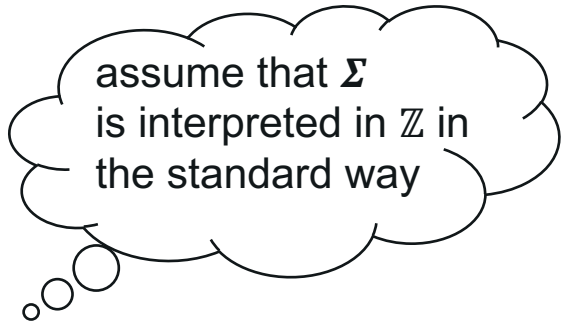
(Atoms) $\Phi^{\mathbf{FOL}} ::= p(\overbrace{T_\Sigma, \ldots, T_\Sigma}^{n \text{ times}})$ $p \in P$ and $ar(p) = n$

(Connectives) $\Phi^{\mathbf{FOL}} ::= \neg\Phi^{\mathbf{FOL}} \mid \Phi^{\mathbf{FOL}} \wedge \Phi^{\mathbf{FOL}} \mid \Phi^{\mathbf{FOL}} \vee \Phi^{\mathbf{FOL}} \mid \Phi^{\mathbf{FOL}} \Rightarrow \Phi^{\mathbf{FOL}} \mid \Phi^{\mathbf{FOL}} \Leftrightarrow \Phi^{\mathbf{FOL}}$

(Quantifiers) $\Phi^{\mathbf{FOL}} ::= \forall V\,.\,\Phi^{\mathbf{F}} \mid \exists V.\,\Phi^{\mathbf{FOL}}$

(First-order) predicate Logic (FOL) - Models

- A FOL model m is an interpretation of the signature, it comprises
  - a set D^m (the interpretation domain),
  - for each function symbol $f \in F$, a function $f^m: \overbrace{D^m \times \cdots \times D^m}^{n \text{ times}} \to D^m$, $ar(f) = n$
  - for each predicate symbol $p \in P$, a relation $p^m \subseteq \overbrace{D^m \times \cdots \times D^m}^{n \text{ times}}$, and $ar(p) = n$
  - an assignment of every free variable (i.e. not used in quantifier) to an element in D^m.
- This gives rise to function $eval_m: T_\Sigma \to D^m$
- Satisfaction is then defined via
  - $m \vDash p(t_1, \ldots, t_n)$ iff. $\big(eval_m(t_1), \ldots, eval(t_n)\big) \in p^m$
  - $m \vDash \forall v.\, \varphi$ iff. for all $x \in D^m$ such that it holds $m \vDash \{v/x\}\varphi$
  - $m \vDash \forall v.\, \varphi$ iff. there exists a $x \in D^m$ such that it holds $m \vDash \{v/x\}\varphi$
  - ...

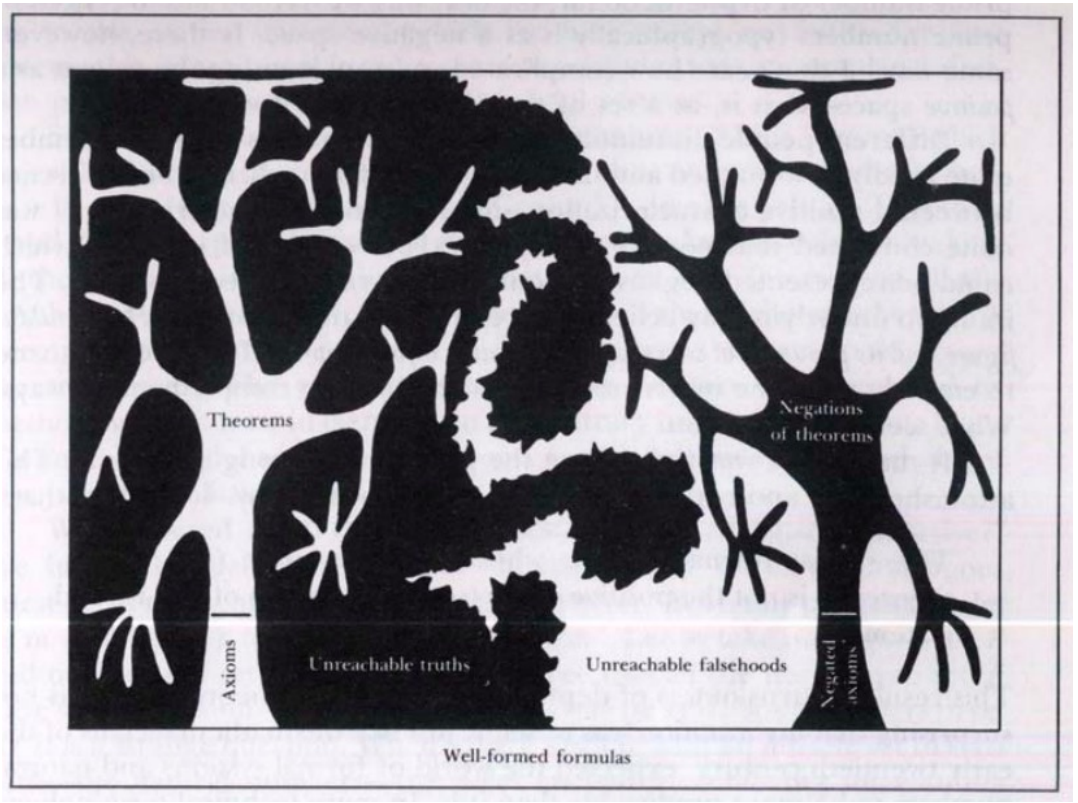


(First-order) predicate Logic (FOL) - Entailment

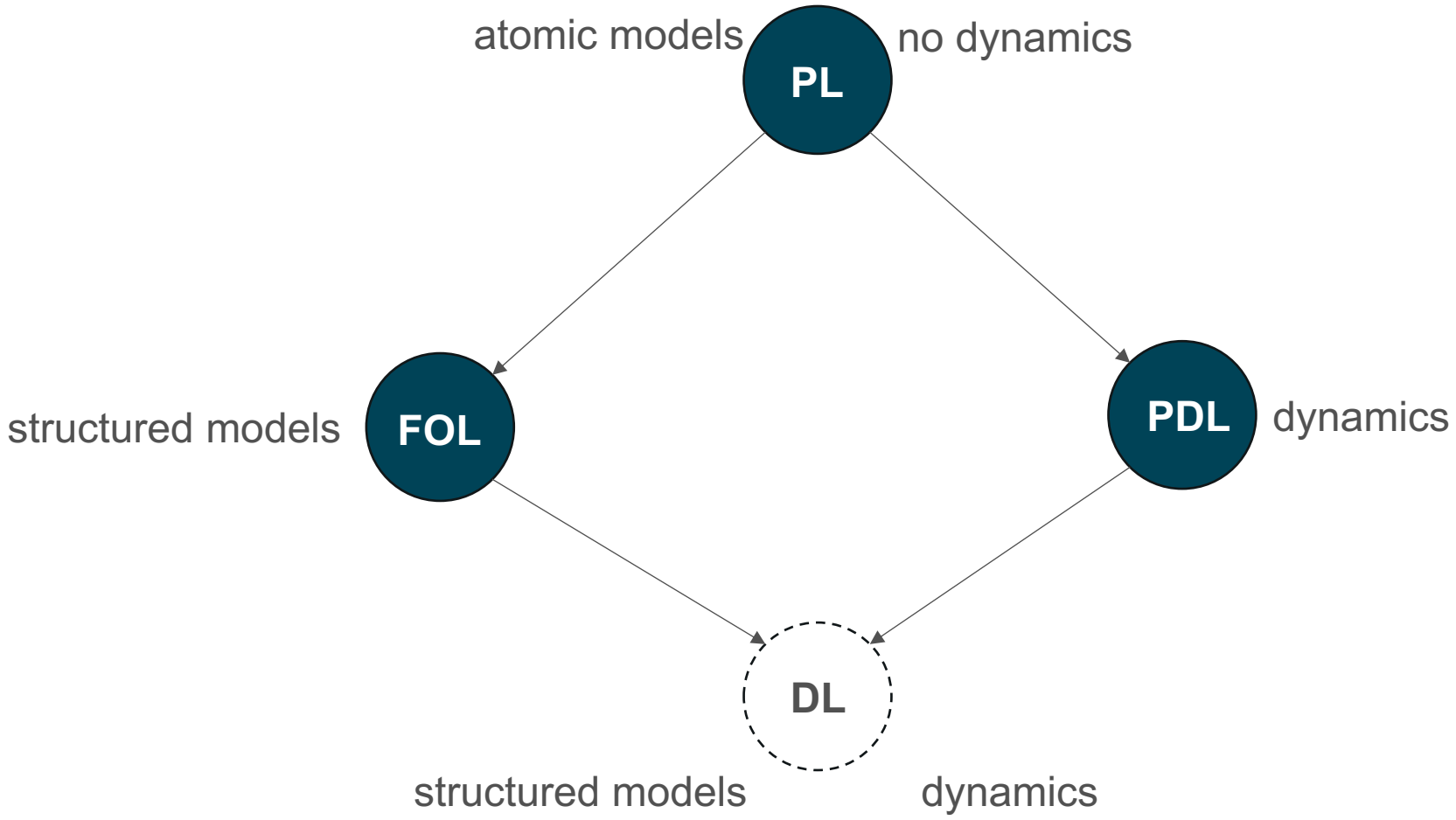
Kurt Gödel
(1906-1978)

- **Attention**: The inference procedure in FOL is only *semi-decidable!*
  - i.e. there are valid sentences that cannot be proven!
  - invalid sentences (:= contradictions) are always found, though!

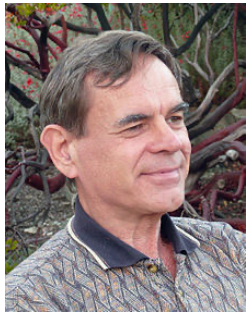
Cut out this one:



The Destination: Dynamic Logic



Putting it all together: Dynamic Logic



Vaughan Pratt
(1944-)

- General ideas:
  - First-order signature and models correspond to base data types (**`int`**, **`string`**, **`float`**)
  - Free variables correspond to program variables
  - Kripke-structure capture the states in a program execution (current variable values)
  - The atomic action is variable assignment
- As a conclusion, I present you the formulae of dynamic logic

(atomic Actions) $A_0 ::= ''\mathrm{V} := T_\Sigma''$

(all Actions) $A ::= A_0 \mid A^* \mid A\,;A \mid A \cup A \mid \Phi^{\mathbf{FOL}}\,?$

$\Phi^{\mathbf{DL}} ::= \Phi^{\mathbf{FOL}} \mid \langle A \rangle \Phi^{\mathbf{DL}} \mid [A]\Phi^{\mathbf{DL}}$

Overview

- Motivation
- Background: Logic
- From Propositional (PL) to Dynamic Logic (DL):
  - Starting with «Propositions only», then...
  - ... adding dynamics: Modal Logic, ...
  - ... adding actions: Propositional Dynamic Logic, ...
  - ... finally, adding internal structure: Dynamic Logic.
- **Dynamic Logic in Action**
- Wrap-Up
  - Related (Logic) Work & Outlook
  - Summary
  - Bibliographic Notes

Dynamic Logic in Action

(coming back to the start)

```
public static int gcdImpl(int a, int b) {
      int h = 0;
      int p = a;
      int r = b;
      while (r != 0) {
            h = p % r;
            p = r;
            r = h;
      }
      return p;
}
```

$$\alpha := h := 0;\ p := a;\ r := b;\ (\neg(r = 0)?;\ h := p \bmod r;\ p := r;\ r := h)^*;\ r = 0?$$

Loop Invariant

$$\frac{\phi \wedge \psi \Rightarrow [\alpha]\phi}{\phi \wedge \psi \Rightarrow [\textbf{while } \psi \textbf{ do } \alpha]\phi} \equiv \frac{\phi \wedge \psi \Rightarrow [\alpha]\phi}{\phi \wedge \psi \Rightarrow [(\psi;\alpha)^*;\neg\psi]\phi}$$

$$[h := 0; p := a; r := b]gcd(a,b) = gcd(p,r)$$

$$\top$$

$$[h := 0; p := a; r := b; (\neg(r = 0)?; h := p \bmod r; p := r; r := h)^*; r = 0?]r = 0 \wedge gcd(a,b) = gcd(p,r)$$

$$\top$$

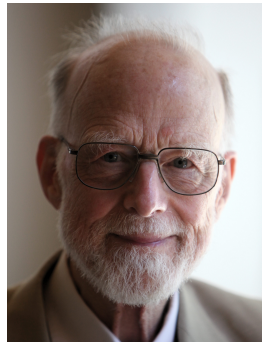
$\forall x.gcd(x,0) = x$

$$[h := 0; p := a; r := b; (\neg(r = 0)?; h := p \bmod r; p := r; r := h)^*; r = 0?]gcd(a,b) = p$$

$$|||$$

$$[\alpha]gcd(a,b) = p$$

Digression: Hoare-Rule & Number theory



”Tony” Hoare
(1934-)

Hoare Rule:

$$\frac{\phi \wedge \psi \Rightarrow [\alpha]\phi}{\phi \wedge \psi \Rightarrow [\textbf{while } \psi \textbf{ do } \alpha]\phi} \equiv \frac{\phi \wedge \psi \Rightarrow [\alpha]\phi}{\phi \wedge \psi \Rightarrow [(\psi;\alpha)^*;\neg\psi]\phi}$$

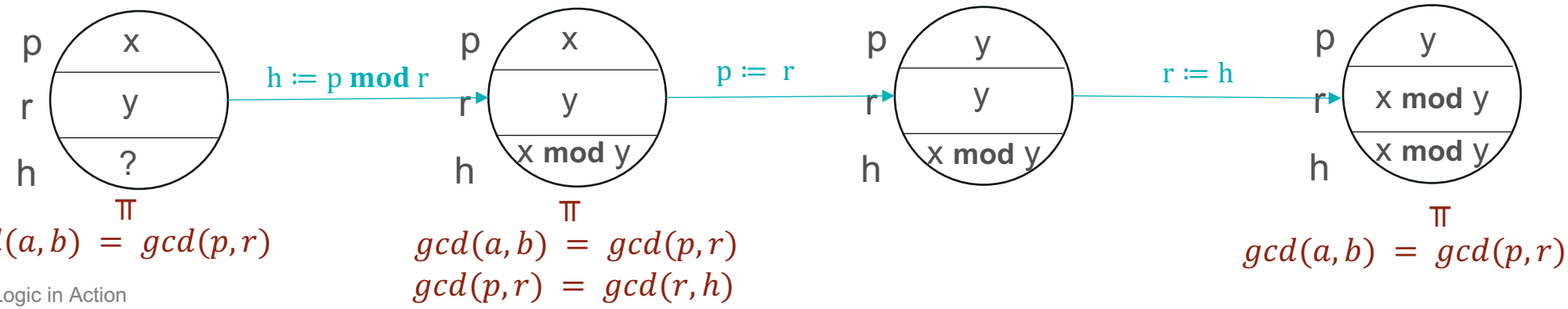
Proof. As an exercise ;-P

Euclid
(~300BC)

Number Theory Lemma

(Euclid) $m = q * n + x \implies gcd(m,n) = gcd(n,x)$

(Def. modulo) $m \bmod n = x \Leftrightarrow \exists q.q * n + x = m$



Dynamic Logic in Action

(coming back to the start)

$$\top$$

$$\top$$

$$[h := 0]\top$$

$$\top$$

$$[h := 0; p := a]gcd(a, b) = gcd(p, b)$$

$$\top$$

$$[h := 0; p := a; r := b]gcd(a, b) = gcd(p, r)$$

$$\top$$

$$[h := 0; p := a; r := b; (\neg(r = 0)?; h := p \bmod r; p := r; r := h)^*; r = 0?]r = 0 \wedge gcd(a, b) = gcd(p, r)$$

$$\top$$

$$[h := 0; p := a; r := b; (\neg(r = 0)?; h := p \bmod r; p := r; r := h)^*; r = 0?]gcd(a, b) = p$$

$$\Vert\Vert$$

$$[\alpha]gcd(a, b) = p$$

Dynamic Logic in Action

- Program verification in DL in general
  - Termination $\langle\alpha\rangle\top$ **May not terminate, though!**
  - Determinism $\langle\alpha\rangle\varphi \Rightarrow [\alpha]\varphi$
  - Equivalence $\langle\alpha\rangle\varphi \Leftrightarrow \langle\beta\rangle\varphi$

Overview

- Motivation
- Background: Logic
- From Propositional (PL) to Dynamic Logic (DL):
  - Starting with «Propositions only», then...
  - ... adding dynamics: Modal Logic, ...
  - ... adding actions: Propositional Dynamic Logic, ...
  - ... finally, adding internal structure: Dynamic Logic.
- Dynamic Logic in Action
- **Wrap-Up**
  - **Related (Logic) Work & Outlook**
  - **Summary**
  - **Bibliographic Notes**

Related (Logic) Work & Outlook

- Original Program Properties were investigated via Hoare Logic

  Hoare Triple $\{\varphi\}\,\alpha\,\{\psi\}$

  - Dynamic Logic provides a more classic approach to the same problem. Hoare triples are expressed in Dynamic Logic as via

    $$\varphi \Rightarrow [\alpha]\psi$$

- Dynamic Logic is well-suited to express positive termination criteria but not so much for defining **Liveness** criteria
  - thus, there are temporal logics, which allow modal connectives such as
    - “eventually”
    - “until”
    - ...

Sentecen that cannot be expressed in Dynamic Logic

Summary

- Takeaways from today:
  - Formal Verification is an approach/research discipline that seeks to apply mathematical methods to verify the correctness of software
  - A «System» for mathematical logic must provide definitions for **models**, sentences, their **satisfaction** relationship ($\models$), and **entailment** ($\vdash$) which allows to infer valid sentences from others
  - Propositional Logic is the «simplest» type of such a system, which comprises **atomic propositions** that can be combined via **logical connectives** ($\neg, \wedge, \vee, \ldots$), models are truth tables
  - Modal Logic extends the definition of models by the concept of **states** (dynamics)
  - There is a subtype of modal logic (called Propositional Dynamic Logic) where state transitions are labelled with **actions** that can be composed via **"regular" connectors** ($;, \cup, *$). The latter gives rise to an internal (W-)programming language
  - First order-logic adds internal structure to models/states comprising **objects**, **relations**, and **functions**. But inference in first-order logic is only semi-decidable
  - **Dynamic Logic** combines first-order logic and the concepts defined in Propositional Dynamic Logic and can be used to **verify properties of programs.**

Bibliographic Notes

- The seminal paper about program verification is [Hoare,1969].
- Pratt, defined Dynamic Logic in [Pratt,1976] in order to express such sentences in a more «classic way» building on another system called “Algorithmic Logic” [Salwicki,1970].
- The more simple Propositional Dynamic Logic was introduced in [Fischer et al. 1970].
- [Harel et al.,2000] gives a general introduction into Dynamic Logic with many examples.
- [Goldblatt,1992] is an introductory textbook into Dynamic and Temporal Logic.

- Hoare, C., 1969, “An axiomatic basis for computer programming”, *Communications of the Association of Computing Machinery*, 12: 576–580.
- Pratt, V., 1976, “Semantical considerations on Floyd-Hoare logic”, in *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*, Los Alamitos, CA: IEEE Computer Society, 109–121.
- Salwicki, A., 1970, “Formalized algorithmic languages”, *Bulletin de l’Academie Polonaise des Sciences, Serie des sciences mathematiques, astronomiques et physiques*, 18: 227–232.
- Fischer, M., and R. Ladner, 1979, “Propositional dynamic logic of regular programs”, *Journal of Computer and System Sciences*, 18: 194–211.
- Harel, D., D. Kozen, and J. Tiuryn, 2000, *Dynamic Logic*, Cambridge, MA: MIT Press.
- Goldblatt, R., 1992a, *Logics of Time and Computation*, Stanford: Center for the Study of Language and Information Publications.

Lecture notes &
exercises due in week 7
;-P

www.corrlang.io/lectures/dynamicLogic

Thank you! Any questions?