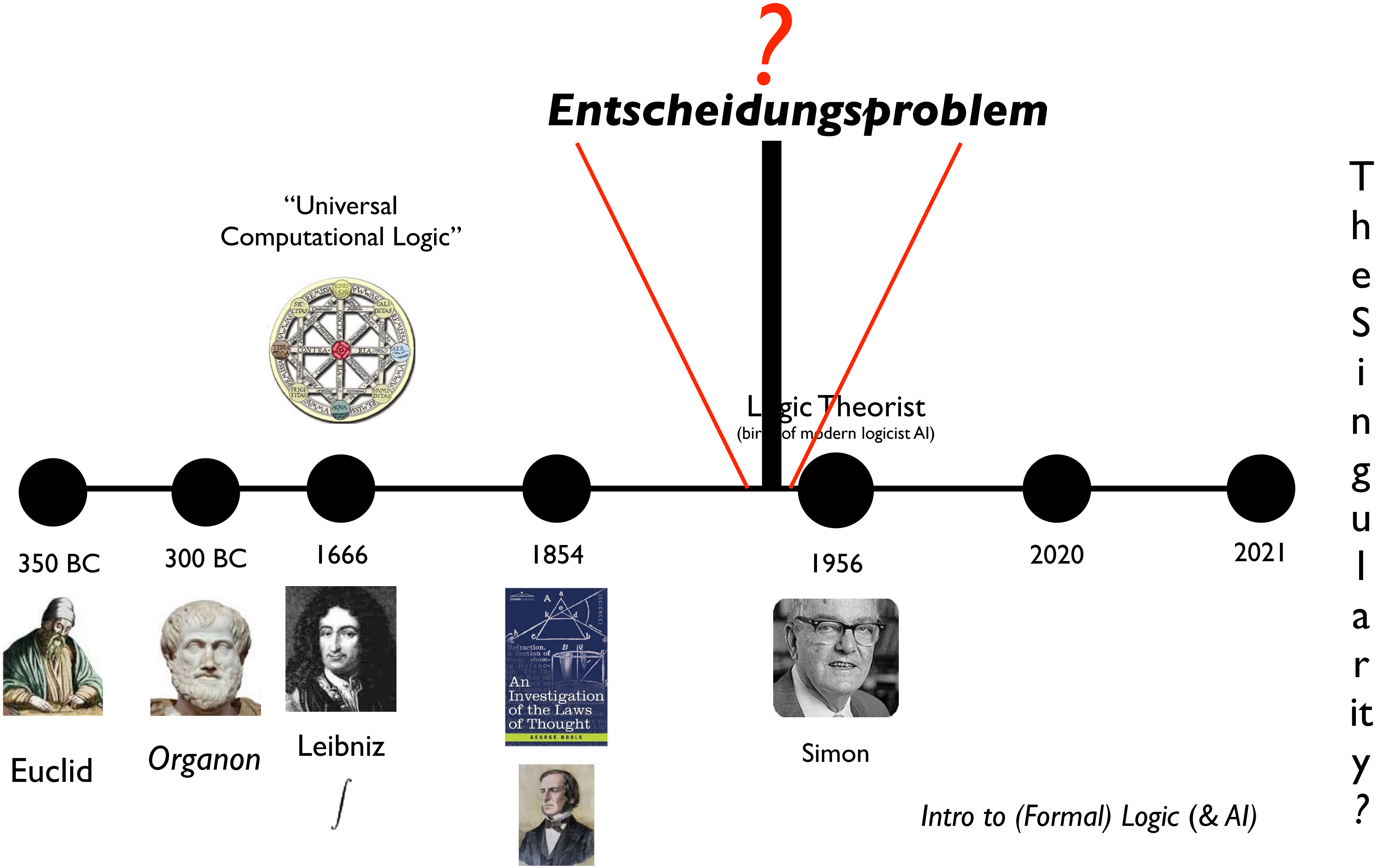
Introducing (= Part 1) … Pure General Logic Programming (PGLP), and HyperLog™

Selmer Bringsjord

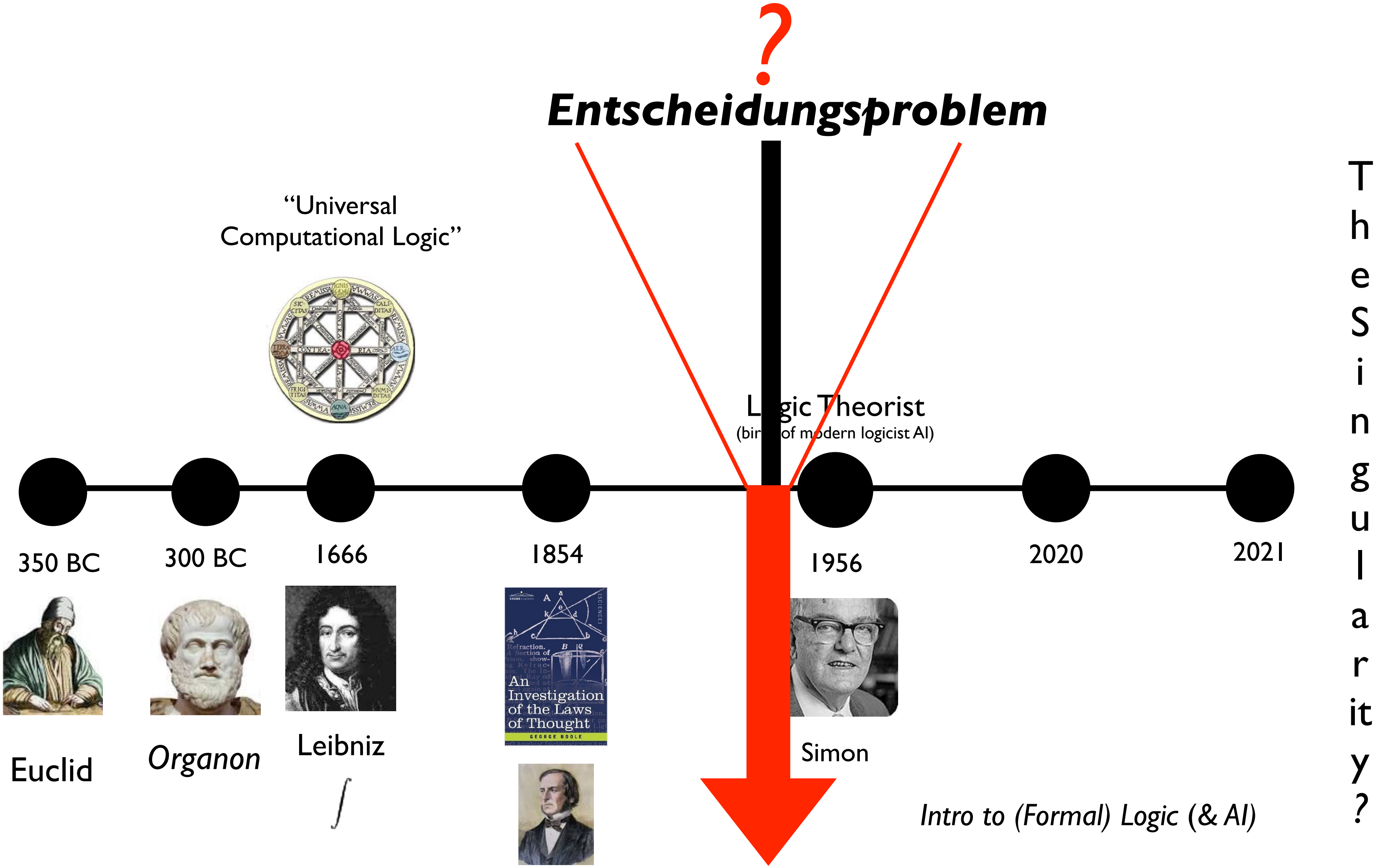
Rensselaer AI & Reasoning (RAIR) Lab
Department of Cognitive Science
Department of Computer Science
Lally School of Management & Technology
Rensselaer Polytechnic Institute (RPI)
Troy, New York 12180 USA

IFLAI1
2/22/2021

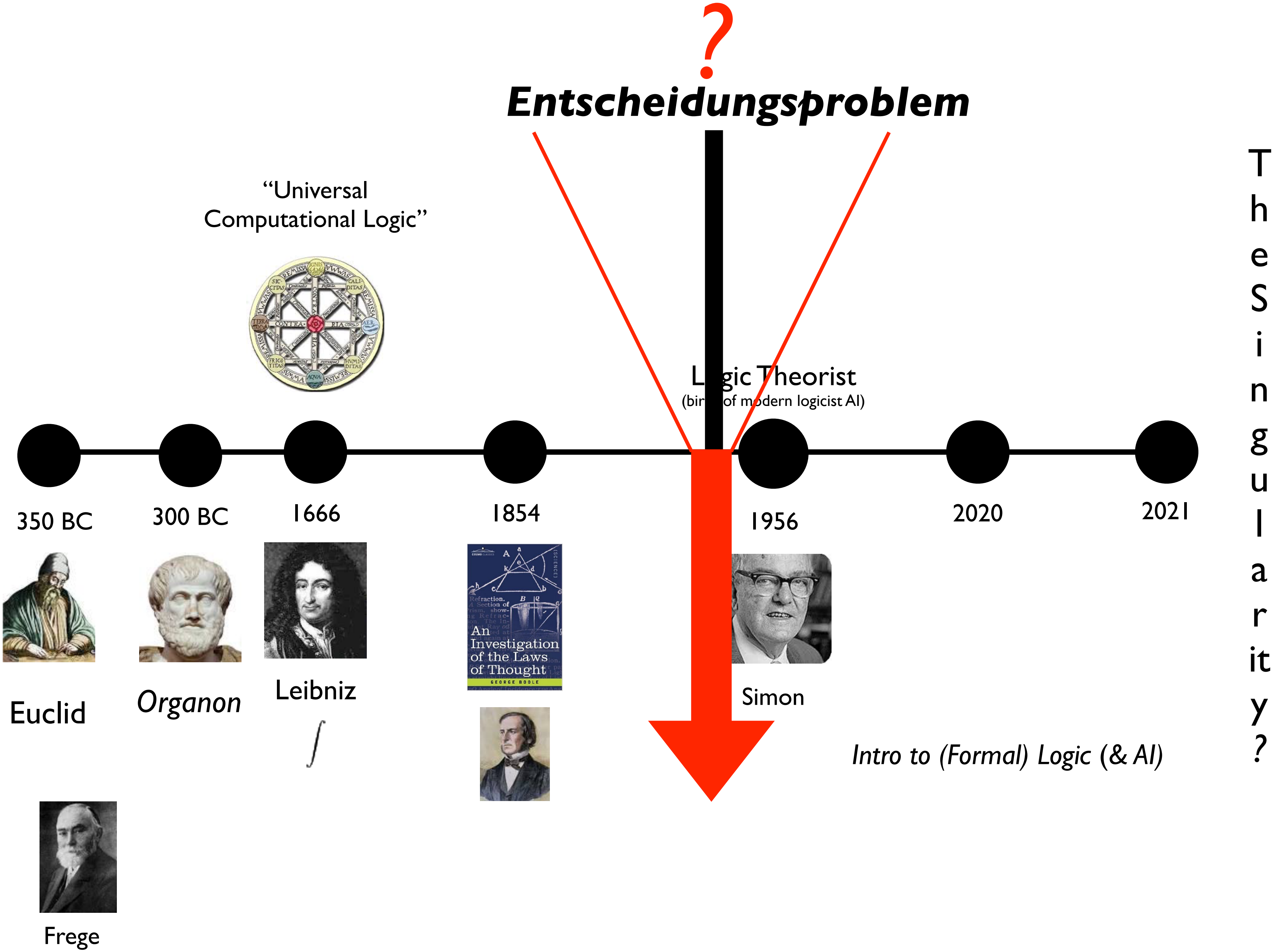
RAIR
Rensselaer AI and Reasoning Lab

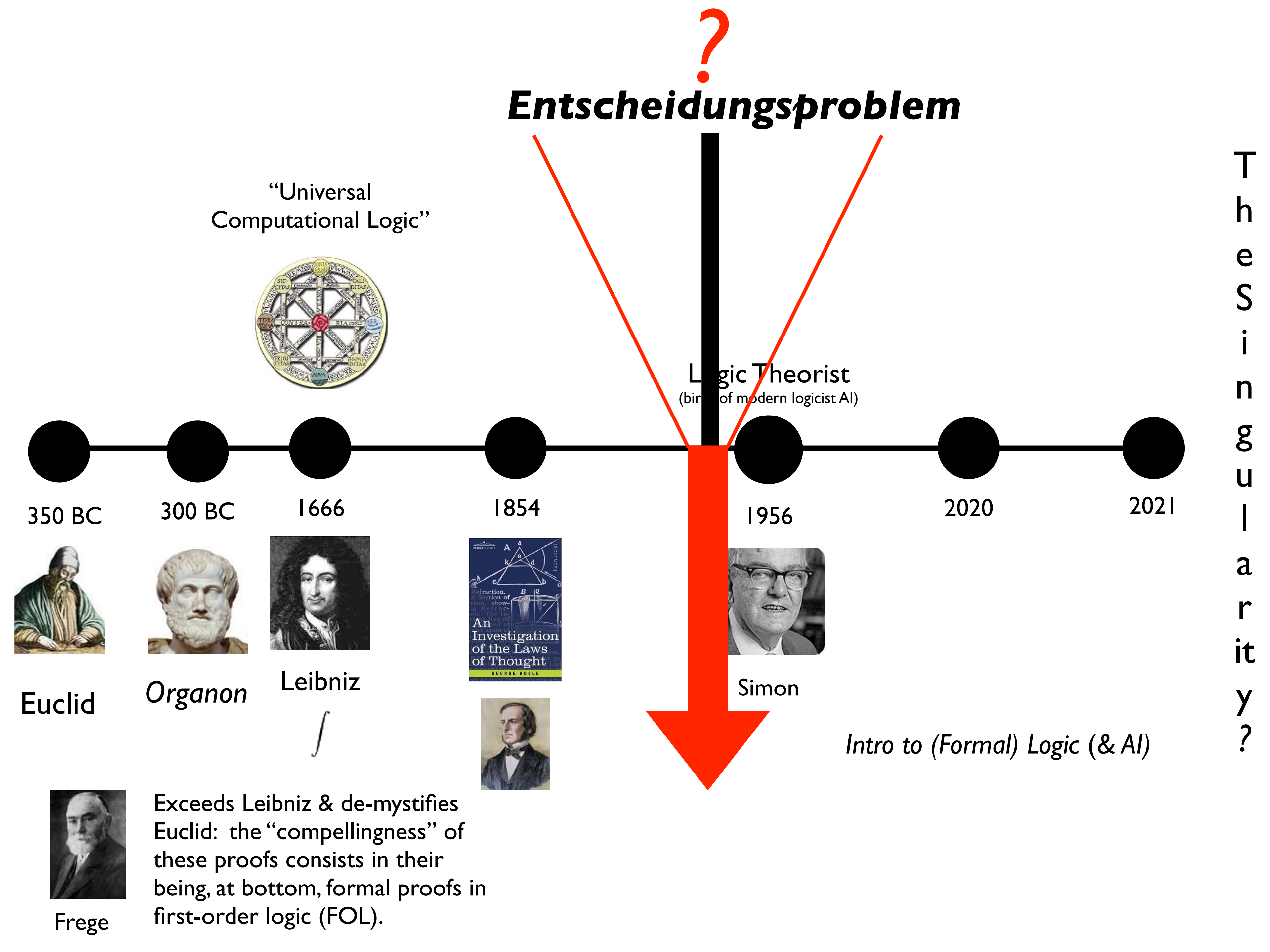


?
Entscheidungsproblem
"Universal Computational Logic"
Logic Theorist
(bir of modern logicist AI)
350 BC
300 BC
1666
1854
1956
2020
2021
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
GEORGE BOOLE
Simon
The Singularity?
Intro to (Formal) Logic (& AI)



?
Entscheidungsproblem
"Universal Computational Logic"
L gic Theorist
(bir of m dern logicist AI)
350 BC
300 BC
1666
1854
1956
2020
2021
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
Simon
Intro to (Formal) Logic (& AI)
The Singularity?

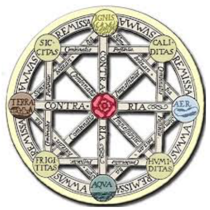

?
Entscheidungsproblem
“Universal
Computational Logic”
L gic Theorist
(bir of m dern logicist AI)
350 BC
300 BC
1666
1854
1956
2020
2021
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
GEORGE BOOLE
Simon
Frege
Intro to (Formal) Logic (& AI)
The Singularity?


?
Entscheidungsproblem
"Universal Computational Logic"
Logic Theorist
(birth of modern logicist AI)
350 BC
300 BC
1666
1854
1956
2020
2021
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
GEORGE BOOLE
Simon
Intro to (Formal) Logic (& AI)
The Singularity?
Frege
Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

?

Entscheidungsproblem

"Universal Computational Logic"

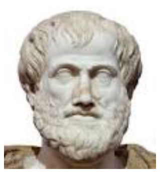


Logic Theorist

(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2020 | 2021

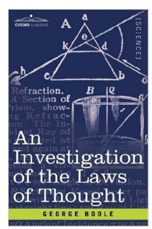
Euclid



Organon

Leibniz

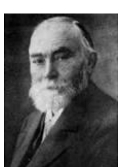
$\int$



Simon

Intro to (Formal) Logic (& AI)

The Singularity?



Frege

Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

Church

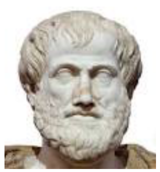
?
Entscheidungsproblem

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2020 | 2021

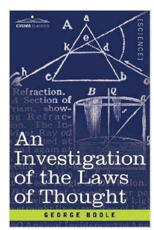
Euclid



Organon

Leibniz

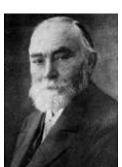
$\int$



Simon

Intro to (Formal) Logic (& AI)

The Singularity?

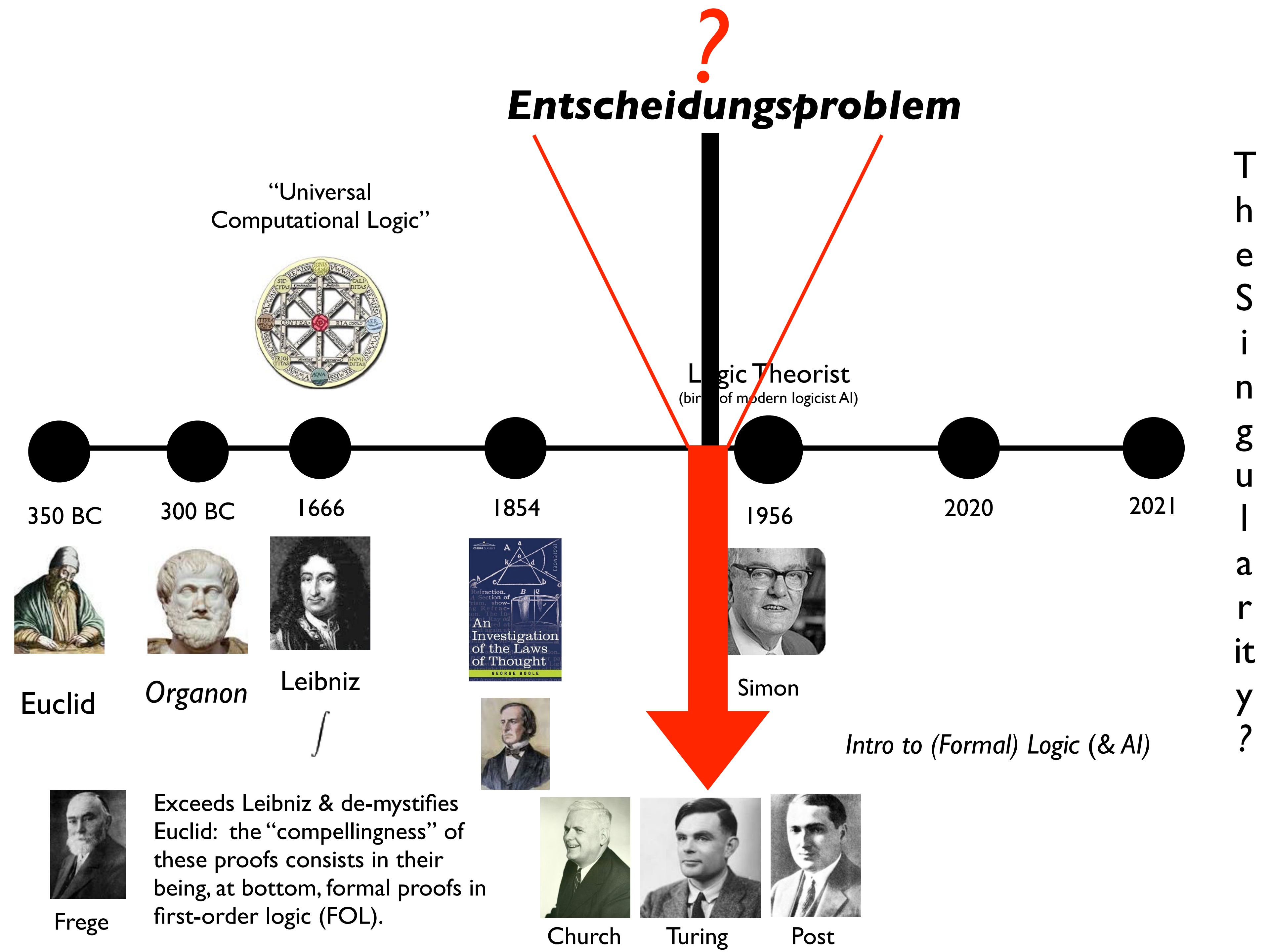


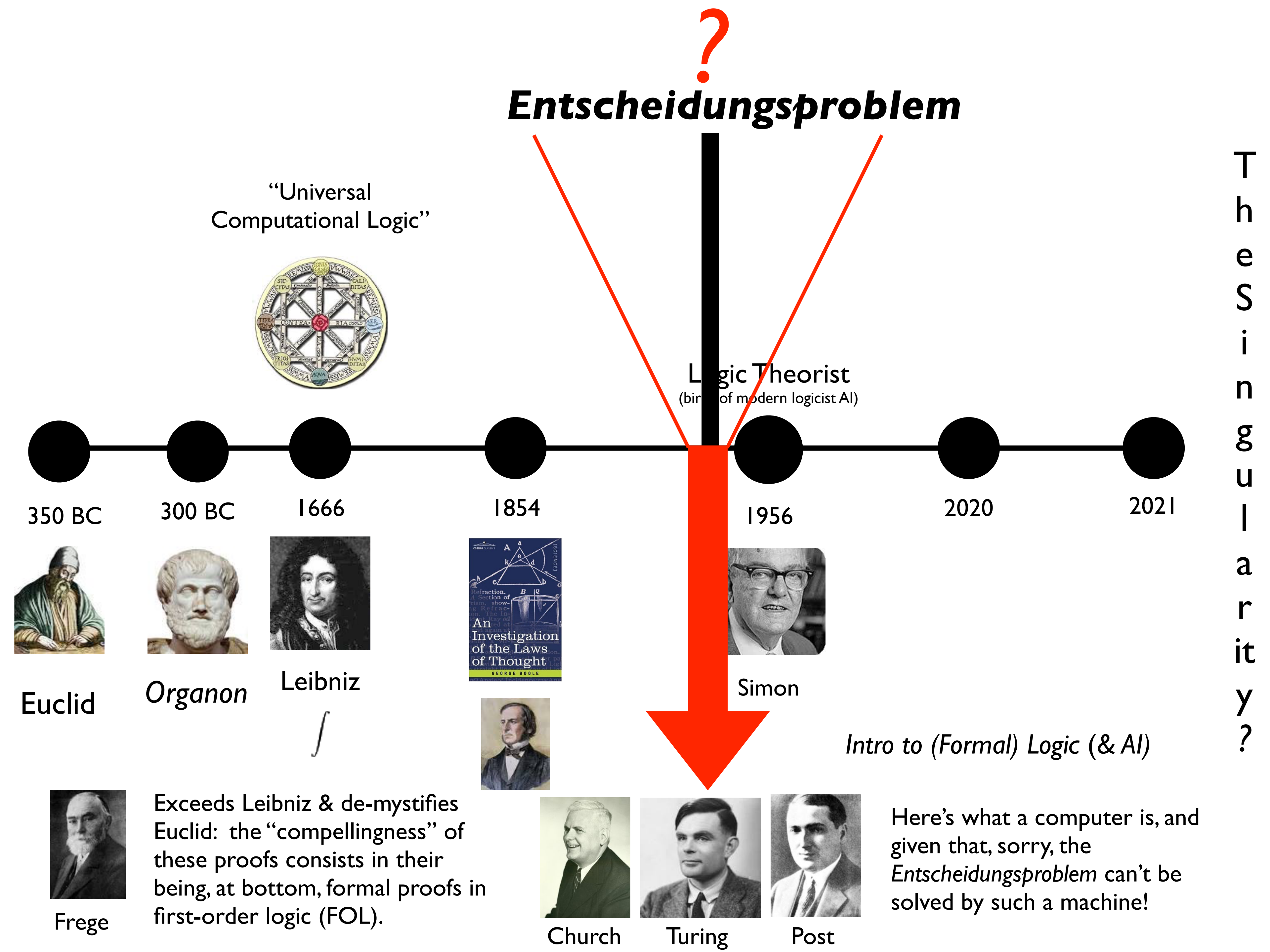
Frege

Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

Church

Turing


?
Entscheidungsproblem
"Universal Computational Logic"
Logic Theorist
(birth of modern logicist AI)
350 BC
300 BC
1666
1854
1956
2020
2021
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
Simon
Intro to (Formal) Logic (& AI)
The Singularity?
Frege
Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).
Church
Turing
Post

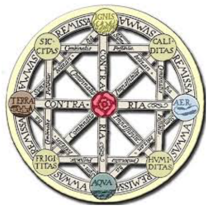

?
Entscheidungsproblem
"Universal
Computational Logic"
Logic Theorist
(birth of modern logicist AI)
350 BC
300 BC
1666
1854
1956
2020
2021
The Singularity?
An Investigation of the Laws of Thought
GEORGE BOOLE
Euclid
Organon
Leibniz
∫
Simon
Intro to (Formal) Logic (& AI)
Frege
Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).
Church
Turing
Post
Here's what a computer is, and given that, sorry, the *Entscheidungsproblem* can't be solved by such a machine!

New for Today:
Functional = Church;
Procedural = Turing.
Where is logic-based/logicist computation/programming?

?

Entscheidungsproblem

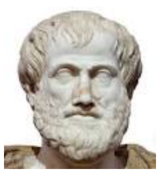
"Universal Computational Logic"



Logic Theorist
(birth of modern logicist AI)

350 BC — 300 BC — 1666 — 1854 — 1956 — 2020 — 2021

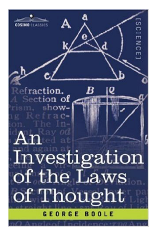
Euclid



Organon

Leibniz

$\int$



Simon

Intro to (Formal) Logic (& AI)

The Singularity?

Frege

Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

Church

Turing

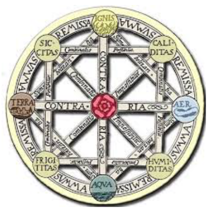
Post

Here's what a computer is, and given that, sorry, the *Entscheidungsproblem* can't be solved by such a machine!

New for Today:
Functional = Church;
Procedural = Turing.
Where is logic-based/logicist computation/programming?

Entscheidungsproblem ?

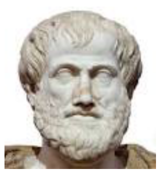
"Universal Computational Logic"



Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2020 | 2021

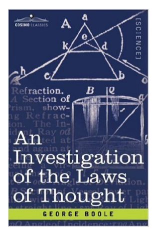
Euclid



Organon

Leibniz

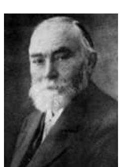
∫



Simon

Intro to (Formal) Logic (& AI)

The Singularity?



Frege

Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

Church

Turing

Post

Here's what a computer is, and given that, sorry, the *Entscheidungsproblem* can't be solved by such a machine!

New for Today:
Functional = Church;
Procedural = Turing.
Where is logic-based/logicist computation/programming?

?
Entscheidungsproblem

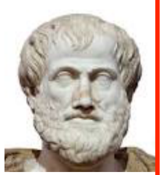
"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

T h e S i n g u l a r i t y ?

350 BC | 300 BC | 1666 | 1854 | 1956 | 2020 | 2021

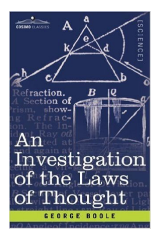
Euclid



Organon

Leibniz

$\int$



Simon

Intro to (Formal) Logic (& AI)

Frege

Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

Church

Turing

Post

Here's what a computer is, and given that, sorry, the *Entscheidungsproblem* can't be solved by such a machine!

New for Today:
Functional = Church;
Procedural = Turing.
Where is logic-based/logicist computation/programming?

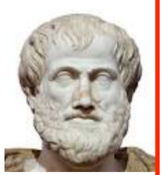
?
Entscheidungsproblem

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2020 | 2021

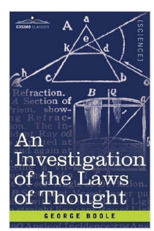
Euclid



Organon

Leibniz

$\int$



Simon

Intro to (Formal) Logic (& AI)

The Singularity?

Frege

Exceeds Leibniz & de-mystifies Euclid: the "compellingness" of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

Church

Turing

Post

Here's what a computer is, and given that, sorry, the *Entscheidungsproblem* can't be solved by such a machine!

New for Today:
Functional = Church;
Procedural = Turing.
Where is logic-based/logicist computation/programming?

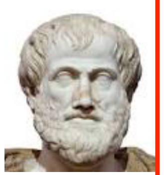
“Universal Computational Logic”

?
Entscheidungsproblem

Logic Theorist
(birth of modern logicist AI)

350 BC — 300 BC — 1666 — 1854 — 1956 — 2020 — 2021

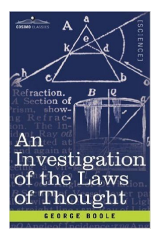
Euclid



Organon

Leibniz

$\int$



Simon

Intro to (Formal) Logic (& AI)

T h e S i n g u l a r i t y ?

Frege

Exceeds Leibniz & de-mystifies Euclid: the “compellingness” of these proofs consists in their being, at bottom, formal proofs in first-order logic (FOL).

Church

Turing

Post

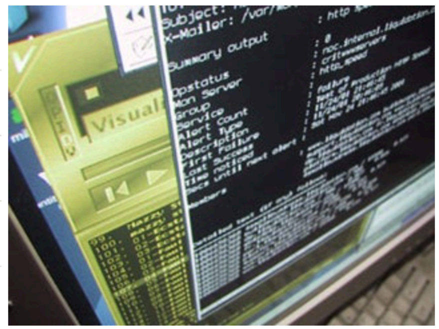
Here’s what a computer is, and given that, sorry, the *Entscheidungsproblem* can’t be solved by such a machine!

FIND COURSES | For Educators | Give Now | About | Search

Home » Courses » Electrical Engineering and Computer Science » Programming Languages

Programming Languages

- COURSE HOME
- SYLLABUS
- CALENDAR
- ASSIGNMENTS
- EXAMS
- TOOLS
- DOWNLOAD COURSE MATERIALS



Programming computer screen. (Photo courtesy of openphoto.net.)

Instructor(s)
Prof. Michael Ernst

MIT Course Number
6.821

As Taught In
Fall 2002

Level
Graduate

CITE THIS COURSE

Course Features

› Assignments: programming (no examples) › Exams (no solutions)

Course Description

6.821 teaches the principles of functional, imperative, and logic programming languages. Topics covered include: meta-circular interpreters, semantics (operational and denotational), type systems (polymorphism, inference, and abstract types), object oriented programming, modules, and multiprocessing. The course involves substantial programming assignments and problem sets as well as a significant amount of reading. The course uses the Scheme+ programming language for all of its assignments.

SYLLABUS

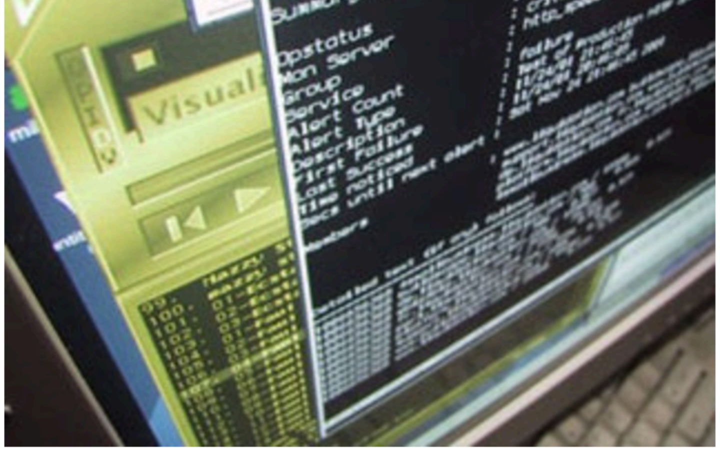
CALENDAR

ASSIGNMENTS

EXAMS

TOOLS

DOWNLOAD COURSE MATERIALS



Programming computer screen. (Photo courtesy of openphoto.net.)

MIT Course Number
6.821

As Taught In
Fall 2002

Level
Graduate

CITE THIS COURSE

Course Features

- Assignments: programming (no examples)
- Exams (no solutions)

Course Description

6.821 teaches the principles of functional, imperative, and logic programming languages. Topics covered include: meta-circular interpreters, semantics (operational and denotational), type systems (polymorphism, inference, and abstract types), object oriented programming, modules, and multiprocessing. The course involves substantial programming assignments and problem sets as well as a significant amount of reading. The course uses the Scheme+ programming language for all of its assignments.

SYLLABUS

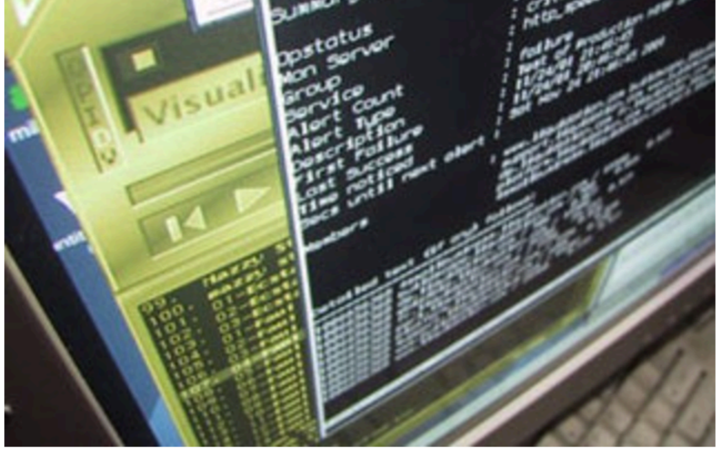
CALENDAR

ASSIGNMENTS

EXAMS

TOOLS

DOWNLOAD COURSE MATERIALS



Programming computer screen. (Photo courtesy of openphoto.net.)

MIT Course Number
6.821

As Taught In
Fall 2002

Level
Graduate

CITE THIS COURSE

Course Features

- Assignments: programming (no examples)
- Exams (no solutions)

Course Description

6.821 teaches the principles of functional, imperative, and logic programming languages. Topics covered include: meta-circular interpreters, semantics (operational and denotational), type systems (polymorphism, inference, and abstract types), object oriented programming, modules, and multiprocessing. The course involves substantial programming assignments and problem sets as well as a significant amount of reading. The course uses the Scheme+ programming language for all of its assignments.

Syllabus
Programming Languages CSCI-4430

Meetings: Webex, TF 2:30-4:20pm
Website: http://www.cs.rpi.edu/~milanova/csci4430

I. Brief Course Description

This course is a study of important concepts in programming languages. Topics include programming language syntax and semantics, types and parameter passing, and programming paradigms (logic-oriented, functional, von Neumann, object-oriented).

Prerequisite: Introduction to Algorithms (CSCI 2300) and Principles of Software (CSCI 2600)

Mailing list: proglang@cs.lists.rpi.edu. Email goes to Milanova, Kuzmin, and Hulbert. Use this list for administrative questions, including homework extension requests, quiz and exam makeup requests, extra time scheduling, and so on.

II. Learning Outcomes

The goal of this course is to teach students how to analyze programming languages. Students will become more productive programmers, will be able to learn new programming languages with ease, and will be able to choose the most suitable programming language for a given problem.

Concretely, students who successfully complete the course should be able to 1) explain programming language syntax and semantics, 2) implement a front-end for a programming language, 3) explain the concepts of scoping, data abstraction, types, control abstraction, and parameter passing, which are essential building blocks of programming languages, and 4) demonstrate competence across a spectrum of programming language paradigms by writing programs in Prolog, Scheme, and Haskell.

III. Required Textbook

Programming Language Pragmatics, Fourth Edition, by Michael Scott, Morgan Kaufmann, 2015.

IV. Class Work and Policies

Quizzes

There are 9 quizzes that should be completed and submitted individually. We will drop the lowest quiz grade and only 8 will count towards the final grade. Quizzes will be administered on Submitty at the beginning of our regularly scheduled class time. We will be offering alternative times for quizzes and exams. If you are unable to attend during regularly scheduled class hours, you must request an alternative time. Email course staff at proglang@cs.lists.rpi.edu by September 10 outlining the reasons why you will be attending at an alternative time (e.g., you reside in a different time zone). We will assign an alternative time and you will be taking the quizzes during this time slot on the date of the quiz. Note that once assigned, you cannot change the quiz time slot.

Syllab
Programming Langu

Meetings: Webex, TF 2:30-4:20pm
Website: http://www.cs.rpi.edu/~milanova/csci4430

I. Brief Course Description

This course is a study of important concepts in programming languages. Topics include programming
(logic-oriented, functional, von Neumann, object-oriented).

Prerequisite: Introduction to Algorithms (CSCI 2300) and Principles of Software (CSCI 2600)

Mailing list: proglang@cs.lists.rpi.edu. Email goes to Milanova, Kuzmin, and Hulbert. Use this list for a
requests, extra time scheduling, and so on.

II. Learning Outcomes

The goal of this course is to teach students how to analyze programming languages. Students will bec
ease, and will be able to choose the most suitable programming language for a given problem.

Concretely, students who successfully complete the course should be able to 1) explain programming
explain the concepts of scoping, data abstraction, types, control abstraction, and parameter passing, v
across a spectrum of programming language paradigms by writing programs in Prolog, Scheme, and I

III. Required Textbook

Programming Language Pragmatics, Fourth Edition, by Michael Scott, Morgan Kaufmann, 2015.

IV. Class Work and Policies

Syllab
Programming Langu

Meetings: Webex, TF 2:30-4:20pm
Website: http://www.cs.rpi.edu/~milanova/csci4430

I. Brief Course Description

This course is a study of important concepts in programming languages. Topics include programming
(logic-oriented, functional, von Neumann, object-oriented).

Prerequisite: Introduction to Algorithms (CSCI 2300) and Principles of Software (CSCI 2600)

Mailing list: proglang@cs.lists.rpi.edu. Email goes to Milanova, Kuzmin, and Hulbert. Use this list for a
requests, extra time scheduling, and so on.

II. Learning Outcomes

The goal of this course is to teach students how to analyze programming languages. Students will bec
ease, and will be able to choose the most suitable programming language for a given problem.

Concretely, students who successfully complete the course should be able to 1) explain programming
explain the concepts of scoping, data abstraction, types, control abstraction, and parameter passing, w
across a spectrum of programming language paradigms by writing programs in Prolog, Scheme, and I

III. Required Textbook

Programming Language Pragmatics, Fourth Edition, by Michael Scott, Morgan Kaufmann, 2015.

IV. Class Work and Policies

There are *Two* Logicist Branches; B1:

There are *Two* Logicist Branches; B1:

Frege, 1893:
"Aha! Currying! I recast multiple-arity operations with functions into a unary affair!"

There are *Two* Logicist Branches; B1:

Frege, 1893:
"Aha! Currying! I recast multiple-arity operations with functions into a unary affair!"

Schönfinkel, 1920's:
"Aha! I can do this stuff using combinatory logic!"

There are *Two* Logicist Branches; B1:

Frege, 1893:
"Aha! Currying! I recast multiple-arity operations with functions into a unary affair!"

Schönfinkel, 1920's:
"Aha! I can do this stuff using combinatory logic!"

Church, 1920's & 30's:
"Aha! The lambda calculus!

There are *Two* Logicist Branches; B1:

Frege, 1893:
"Aha! Currying! I recast multiple-arity operations with functions into a unary affair!"

Schönfinkel, 1920's:
"Aha! I can do this stuff using combinatory logic!"

Church, 1920's & 30's:
"Aha! The lambda calculus!

...

There are *Two* Logicist Branches; B1:

Frege, 1893:
"Aha! Currying! I recast multiple-arity operations with functions into a unary affair!"

Schönfinkel, 1920's:
"Aha! I can do this stuff using combinatory logic!"

Church, 1920's & 30's:
"Aha! The lambda calculus!

...

<u>Haskell</u>

There are *Two* Logicist Branches; B1:

Frege, 1893:
"Aha! Currying! I recast multiple-arity operations with functions into a unary affair!"

Schönfinkel, 1920's:
"Aha! I can do this stuff using combinatory logic!"

Church, 1920's & 30's:
"Aha! The lambda calculus!

...

Haskell OCaml, Scheme, …

There are *Two* Logicist Branches; B1:

Frege, 1893:
"Aha! Currying! I recast multiple-arity operations with functions into a unary affair!"

Schönfinkel, 1920's:
"Aha! I can do this stuff using combinatory logic!"

Church, 1920's & 30's:
"Aha! The lambda calculus!

...

Haskell OCaml, Scheme, …
Athena

Two Logicist Branches; B2:

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Simon & Newell @
Dawn of Modern AI: LT & GPS

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Simon & Newell @
Dawn of Modern AI: LT & GPS

...

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Simon & Newell @
Dawn of Modern AI: LT & GPS

…

Prolog?

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Simon & Newell @
Dawn of Modern AI: LT & GPS

…

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Simon & Newell @
Dawn of Modern AI: LT & GPS

…

PGLP

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Simon & Newell @
Dawn of Modern AI: LT & GPS

...

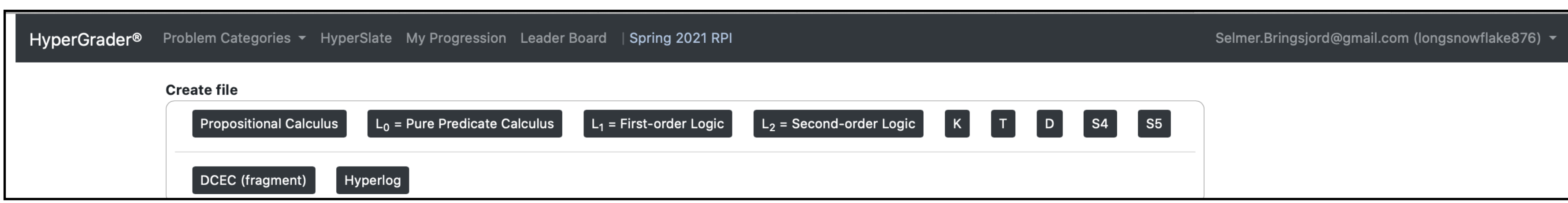
PGLP

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

Simon & Newell @
Dawn of Modern AI: LT & GPS



PGLP

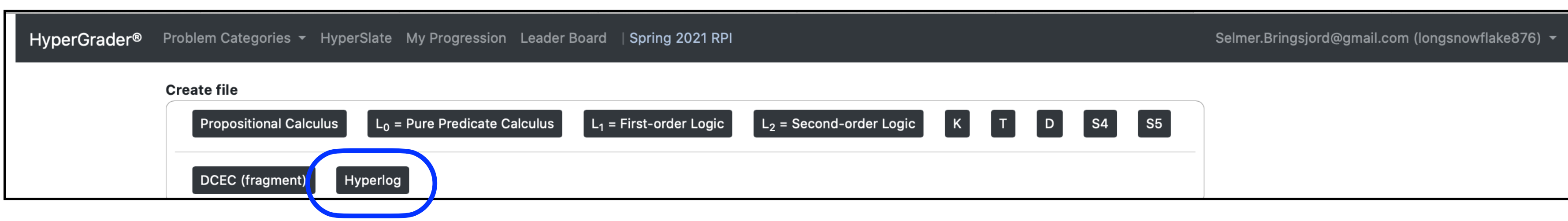
HyperSlate® : HyperLog

Two Logicist Branches; B2:

The AI Branch: Automated Reasoning

Leibniz

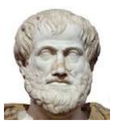
Simon & Newell @
Dawn of Modern AI: LT & GPS



PGLP

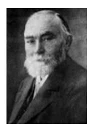
HyperSlate® : HyperLog

HyperLog:
Historico-logico-programming Landscape



First "logic programs"
300 BC

Liebniiz
Dies 1716



Frege
1893

Schöenfinkel
1893

Combinatory Logic

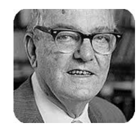
simple type theory

Church

λ-calculus

Turing

Logic Theorist
(birth of modern logicist AI)



Simon

1956

Lisp

Prolog

Fortran

Smalltalk

Athena

ML

Scheme

CL

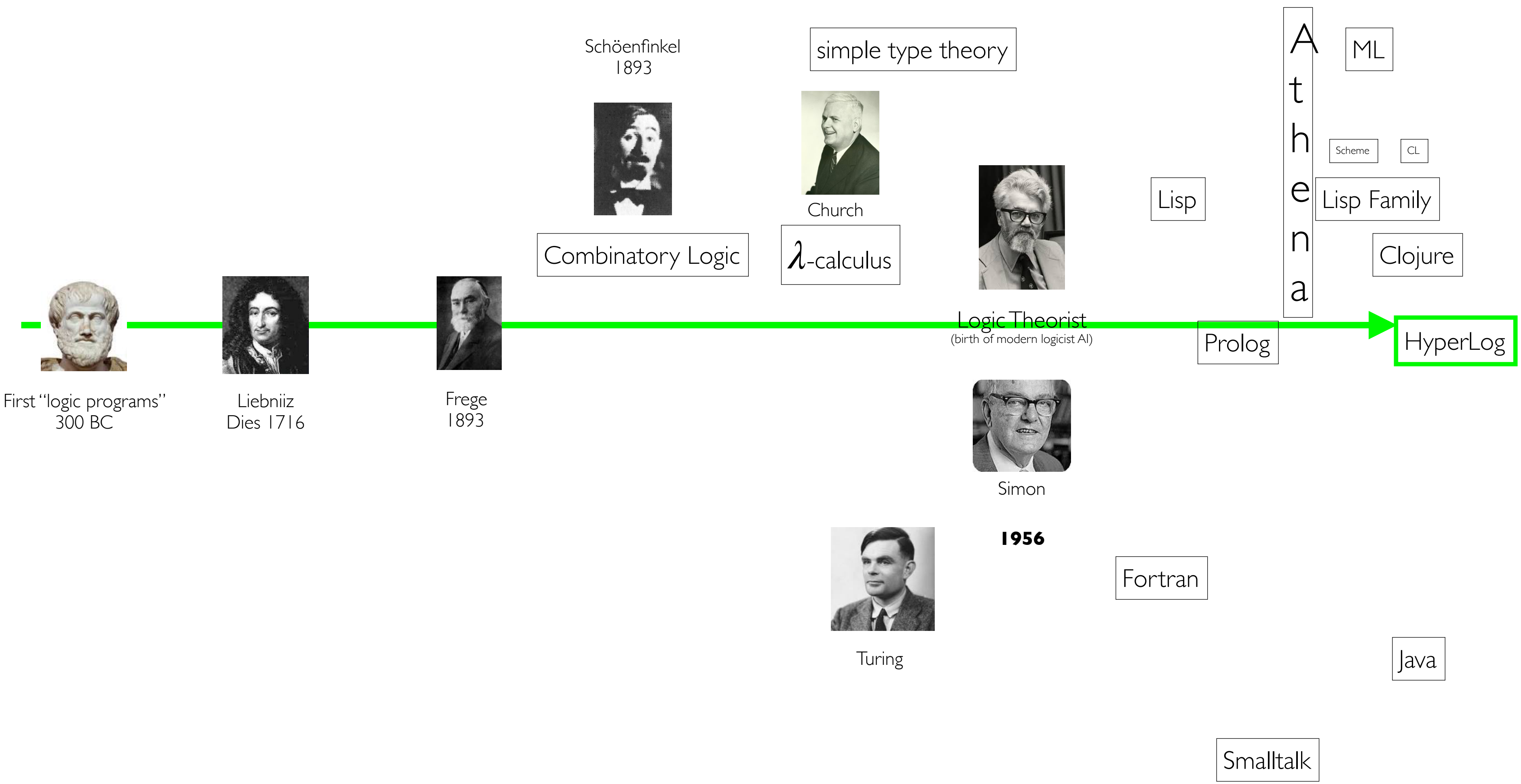
Lisp Family

Clojure

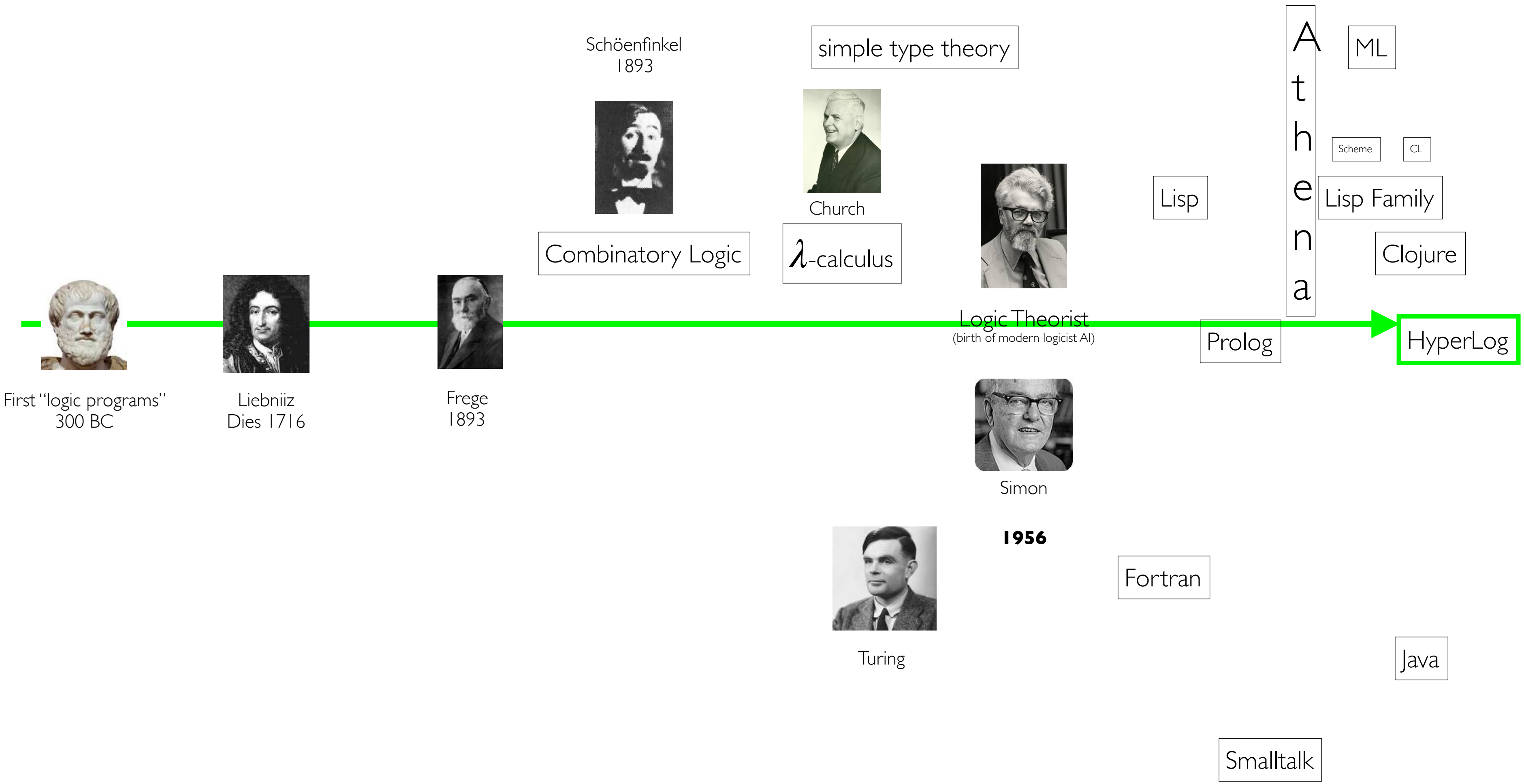
HyperLog

Java

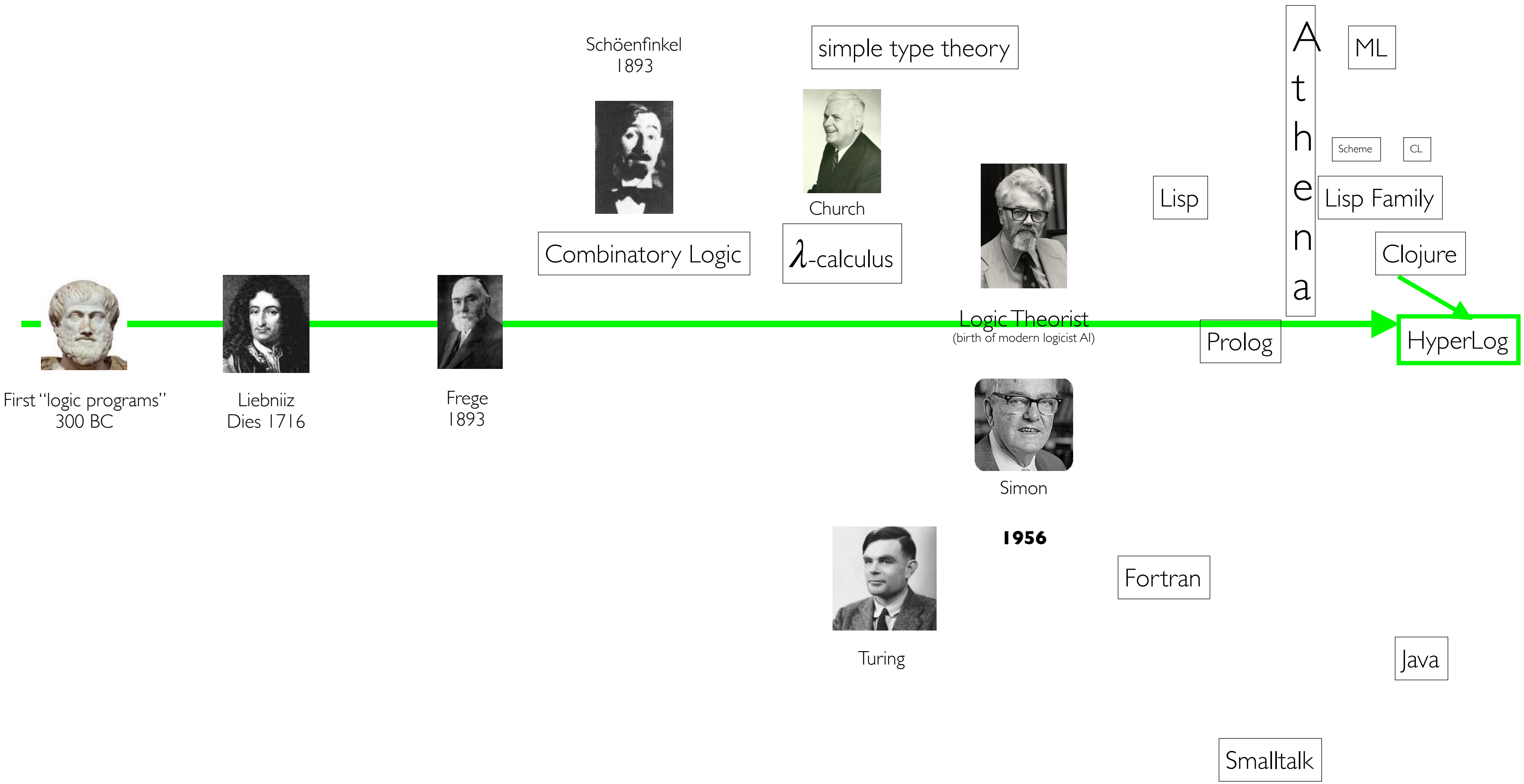
HyperLog:
Historico-logico-programming Landscape



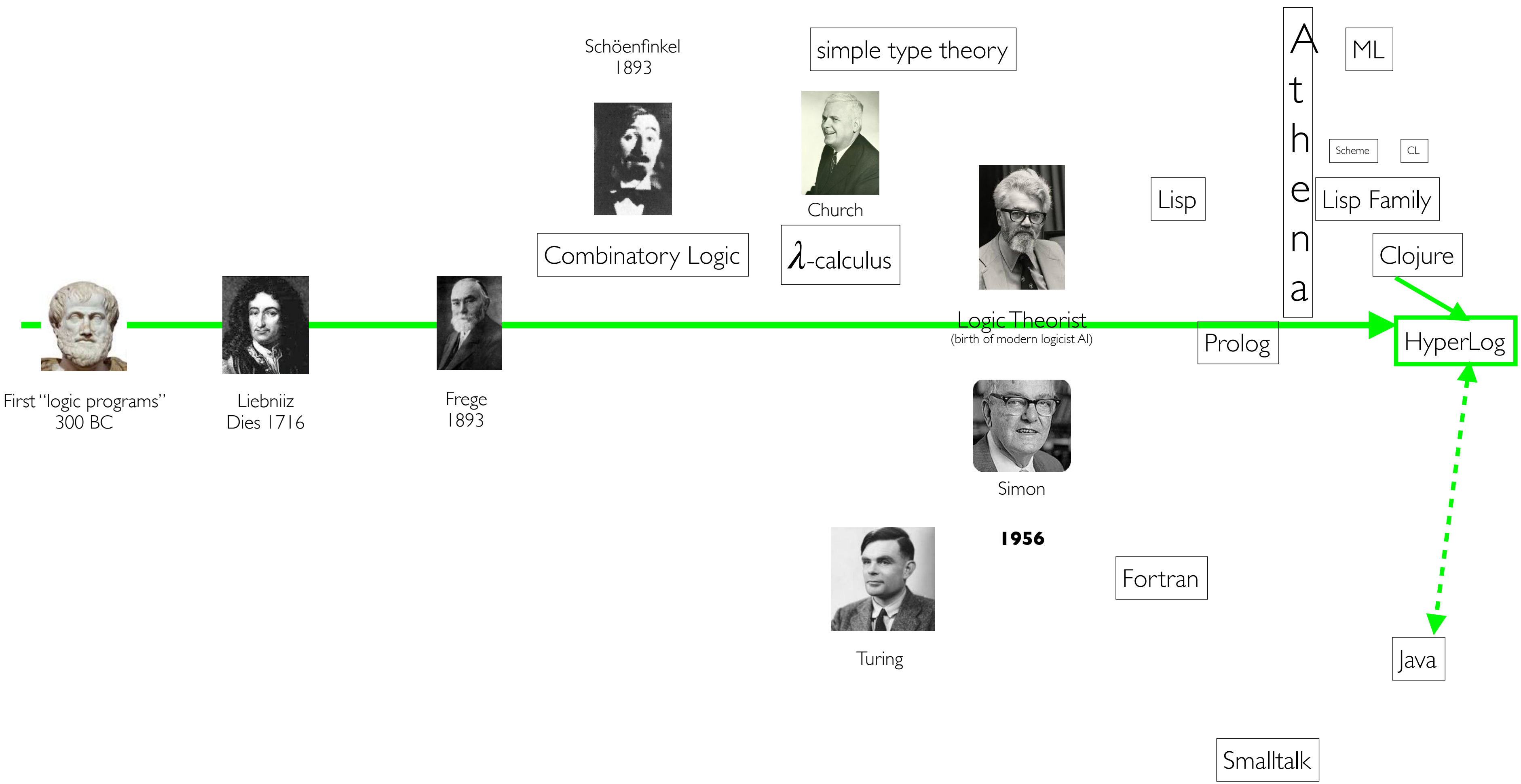
HyperLog: Historico-logico-programming Landscape



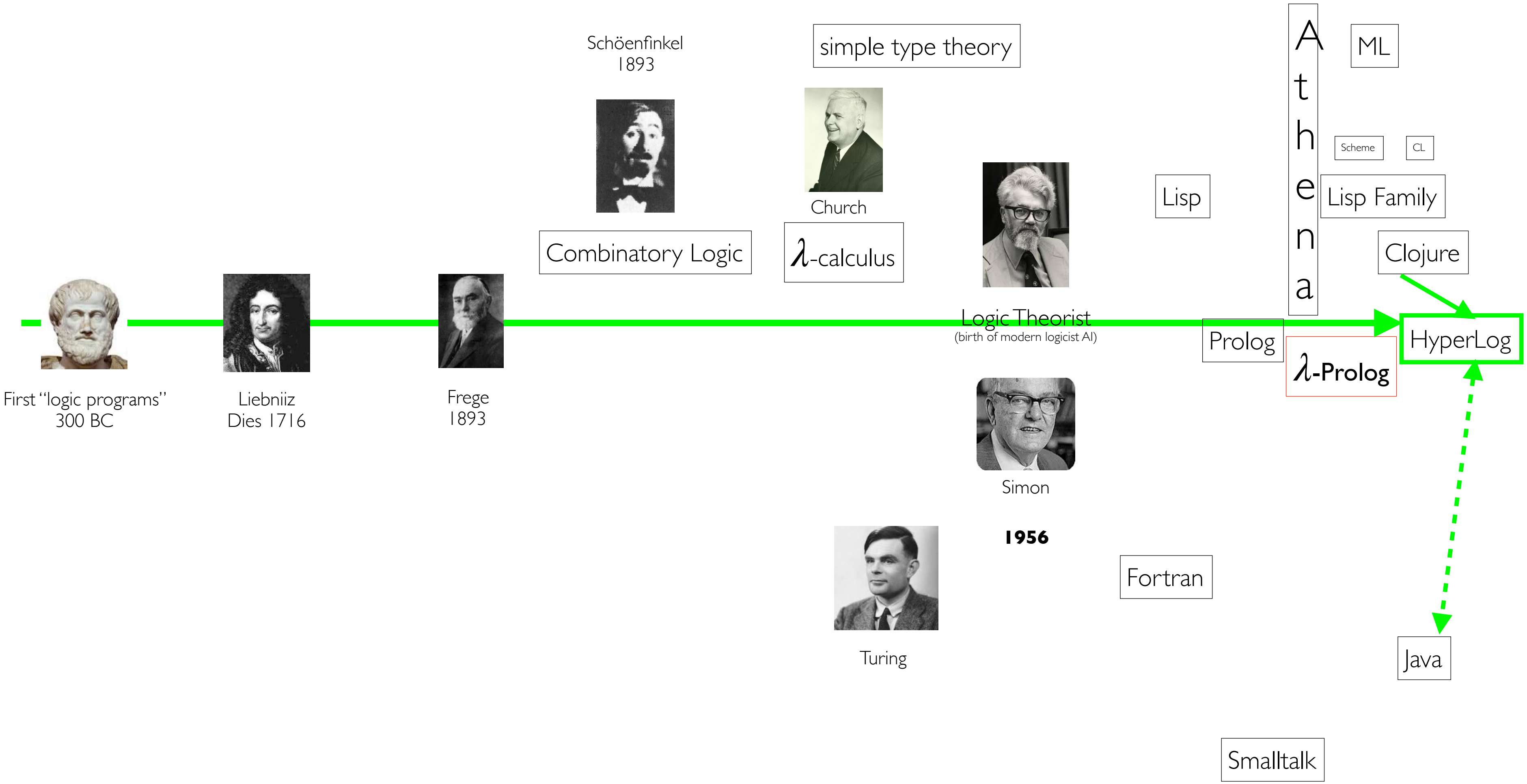
HyperLog:
Historico-logico-programming Landscape



HyperLog:
Historico-logico-programming Landscape



HyperLog:
Historico-logico-programming Landscape



Single-Slide Encapsulation …

$$\mathcal{L} := \langle L, \mathcal{I} \rangle$$

$$\frac{\begin{array}{cc} \mathbb{P} & L \\ \mathfrak{q} & L \end{array}}{\begin{array}{l} \mathbb{R} \ : \ \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta, \pi_{(s)}|\alpha_{(s)} \rangle \\ \mathbb{C} \ : \ \pi_{(s)}|\alpha_{(s)} \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta \rangle \end{array}}$$

program

query

degree of "confidence"

proof(s)

argument(s)

$$\mathcal{L} := \langle L, \mathcal{I} \rangle \qquad \frac{\begin{array}{cc} \mathbb{P} & L \\ \mathfrak{q} & L \end{array}}{\begin{array}{l} \mathbb{R} \;:\; \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta, \pi_{(s)}|\alpha_{(s)} \rangle \\ \quad\;\; \mathbb{C} \;:\; \pi_{(s)}|\alpha_{(s)} \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta \rangle \end{array}}$$

reasoner

checker

1. A Hard Question …

Easy Question

Easy Question

What is pure procedural programming?

Another Easy Question

Another Easy Question

What is pure functional programming?

A *Hard* Question

A *Hard* Question

What is pure *logic* programming?

A *Hard* Question

What is pure *logic* programming?

A *Hard* Question

What is pure *logic* programming?

A: …

B: …

C: …

…

Naveen: "Using automated theorem provers …"

4. Ingredients for Making a PGLP Program

. . .

On the Anatomy of a PGLP Program

On the Anatomy of a PGLP Program

Linguistics

$\vdots \qquad \vdots \qquad \vdots$

L_2^μ meta-level₂ language $(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^μ meta-level₁ language $\exists x \ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

On the Anatomy of a PGLP Program

Linguistics

$\vdots$	$\vdots$	$\vdots$
L_2^{μ}	meta-level$_2$ language	$(\{\phi\} \vdash \psi \land \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$
L_1^{μ}	meta-level$_1$ language	$\exists x \ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

On the Anatomy of a PGLP Program

Linguistics

$$\vdots \qquad \vdots \qquad \vdots$$

L_2^{μ} meta-level$_2$ language $(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^{μ} meta-level$_1$ language $\exists x\ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

Semantics

Reasoning-semantic; wholly inferentialist (after all, what's the semantics of deduction over meta-level$_1$ formulae??).

On the Anatomy of a PGLP Program

Linguistics

$\vdots \quad \vdots \quad \vdots$

L_2^μ meta-level₂ language $(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^μ meta-level₁ language $\exists x \ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

Semantics

Reasoning-semantic; wholly inferentialist (after all, what's the semantics of deduction over meta-level$_1$ formulae??).

On the Anatomy of a PGLP Program

Linguistics

$\vdots \quad \vdots \quad \vdots$

L_2^{μ} meta-level$_2$ language $(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^{μ} meta-level$_1$ language $\exists x\ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

Semantics

Reasoning-semantic; wholly inferentialist (after all, what's the semantics of deduction over meta-level$_1$ formulae??).

$\mathscr{L}$

On the Anatomy of a PGLP Program

Linguistics

$\vdots \qquad \vdots \qquad \vdots$

L_2^μ meta-level$_2$ language $(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^μ meta-level$_1$ language $\exists x \ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

Semantics

Reasoning-semantic; wholly inferentialist (after all, what's the semantics of deduction over meta-level$_1$ formulae??).

$\mathscr{L}$

Selection of language, inference schemata, plus formulae/meta-formulae $= \mathbb{P}_{\mathscr{L}}$

On the Anatomy of a PGLP Program

Linguistics

$\vdots$ $\vdots$ $\vdots$

L_2^μ meta-level$_2$ language $(\{\phi\} \vdash \psi \land \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^μ meta-level$_1$ language $\exists x \; \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

Semantics

Reasoning-semantic; wholly inferentialist (after all, what's the semantics of deduction over meta-level$_1$ formulae??).

$\mathscr{L}$

Selection of language, inference schemata, plus formulae/meta-formulae $= \mathbb{P}_{\mathscr{L}}$ + ShadowReasoner

On the Anatomy of a PGLP Program

Linguistics

$\vdots \quad \vdots \quad \vdots$

L_2^{μ} meta-level$_2$ language $(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^{μ} meta-level$_1$ language $\exists x \ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

Semantics

Reasoning-semantic; wholly inferentialist (after all, what's the semantics of deduction over meta-level$_1$ formulae??).

$\mathscr{L}$

+ ShadowReasoner

On the Anatomy of a PGLP Program

Linguistics

$\vdots \qquad \vdots \qquad \vdots$

L_2^μ meta-level₂ language $(\{\phi\} \vdash \psi \land \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$

L_1^μ meta-level₁ language $\exists x \; \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$

L object-level language $\phi \quad \psi \quad \delta$

Inference

A collection of inference schemata. (For economy, see coming Example 1.)

Semantics

Reasoning-semantic; wholly inferentialist (after all, what's the semantics of deduction over meta-level₁ formulae??).

$\mathscr{L}$

$$\mathcal{L} := \langle L, \mathcal{I} \rangle$$

$$\frac{\mathbb{P} \quad L \qquad \mathfrak{q} \quad L}{\begin{array}{l} \mathbb{R} \;:\; \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta, \pi_{(s)}|\alpha_{(s)} \rangle \\ \quad \mathbb{C} \;:\; \pi_{(s)}|\alpha_{(s)} \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta \rangle \end{array}}$$

program

query

degree of "confidence"

proof(s)

argument(s)

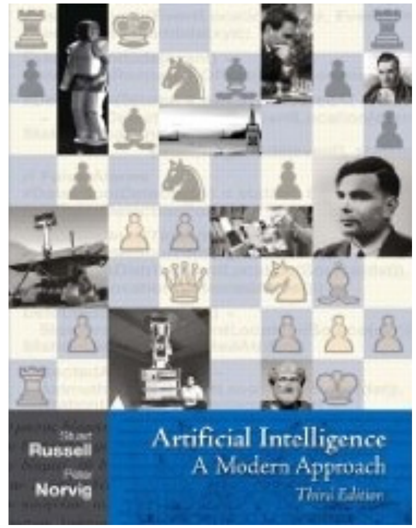
reasoner

checker

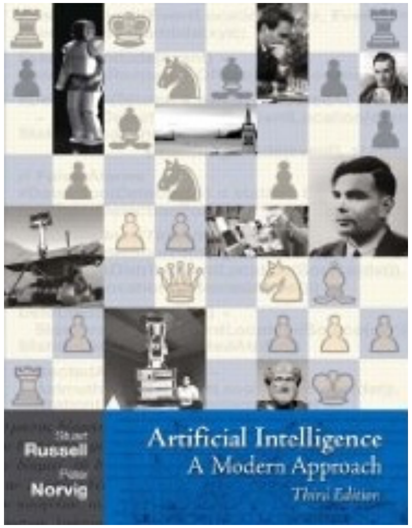
AI today …

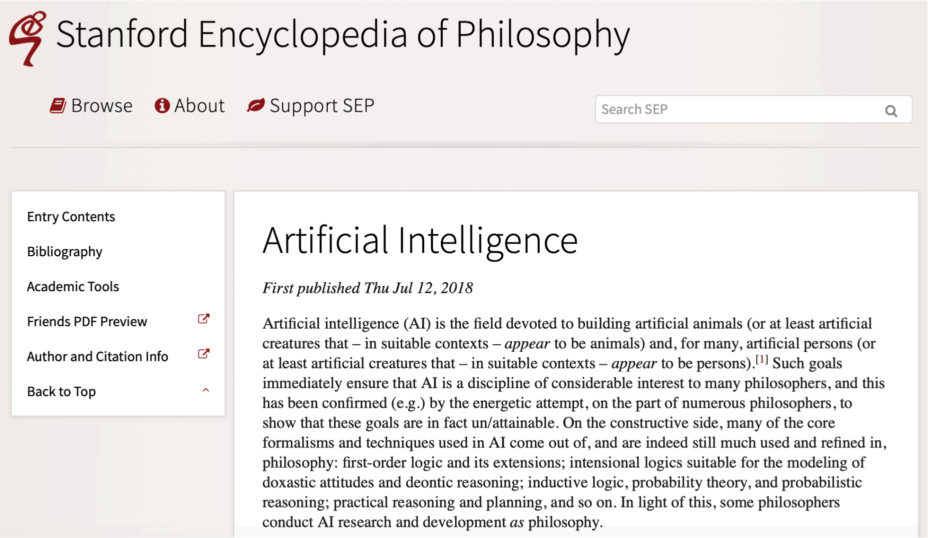
AI today:

AI today:


Stuart
Russell
Peter
Norvig
Artificial Intelligence
A Modern Approach
Third Edition

AI today:





Stanford Encyclopedia of Philosophy

Browse About Support SEP

Search SEP

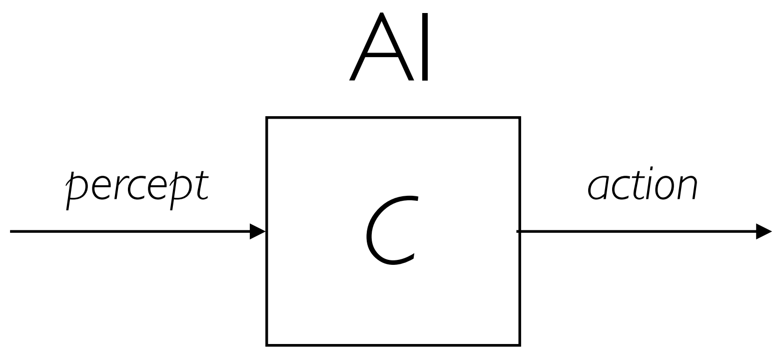
Entry Contents
Bibliography
Academic Tools
Friends PDF Preview
Author and Citation Info
Back to Top

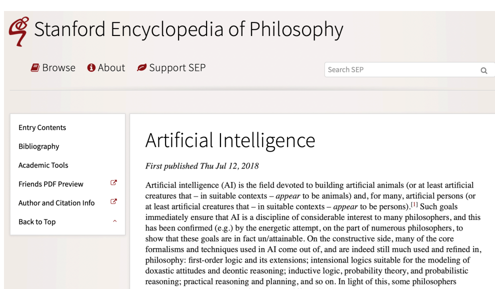
Artificial Intelligence

First published Thu Jul 12, 2018

Artificial intelligence (AI) is the field devoted to building artificial animals (or at least artificial creatures that – in suitable contexts – *appear* to be animals) and, for many, artificial persons (or at least artificial creatures that – in suitable contexts – *appear* to be persons).[1] Such goals immediately ensure that AI is a discipline of considerable interest to many philosophers, and this has been confirmed (e.g.) by the energetic attempt, on the part of numerous philosophers, to show that these goals are in fact un/attainable. On the constructive side, many of the core formalisms and techniques used in AI come out of, and are indeed still much used and refined in, philosophy: first-order logic and its extensions; intensional logics suitable for the modeling of doxastic attitudes and deontic reasoning; inductive logic, probability theory, and probabilistic reasoning; practical reasoning and planning, and so on. In light of this, some philosophers conduct AI research and development *as* philosophy.

AI:




Stanford Encyclopedia of Philosophy

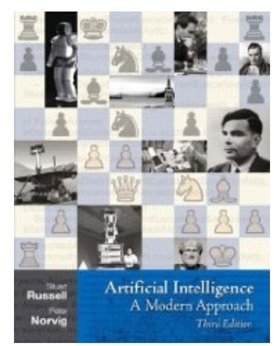
Browse · About · Support SEP · Search SEP

Entry Contents
Bibliography
Academic Tools
Friends PDF Preview
Author and Citation Info
Back to Top

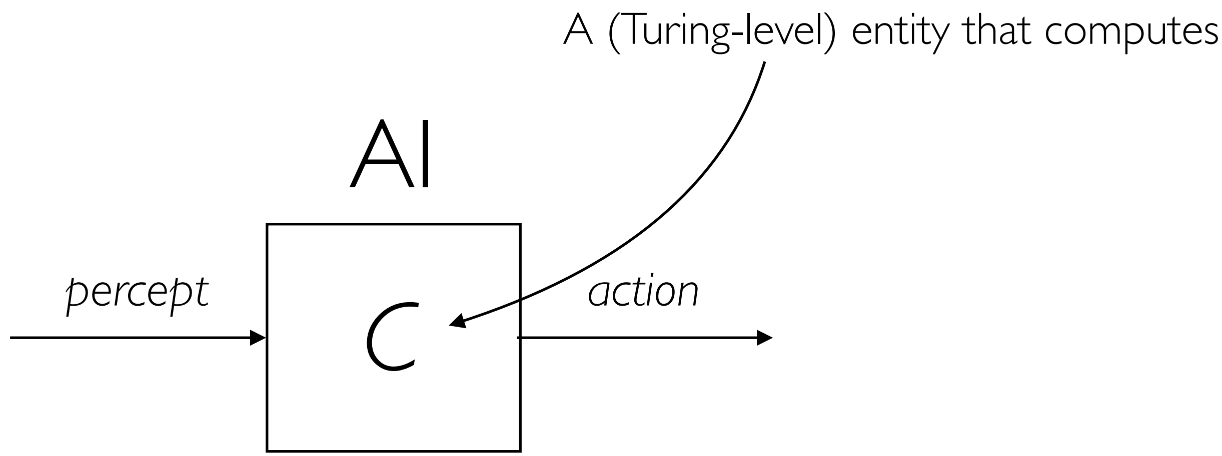
Artificial Intelligence

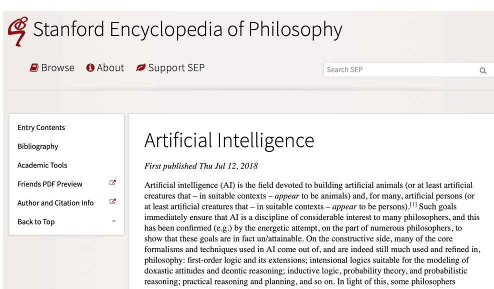
First published Thu Jul 12, 2018

Artificial intelligence (AI) is the field devoted to building artificial animals (or at least artificial creatures that – in suitable contexts – *appear* to be animals) and, for many, artificial persons (or at least artificial creatures that – in suitable contexts – *appear* to be persons).[1] Such goals immediately ensure that AI is a discipline of considerable interest to many philosophers, and this has been confirmed (e.g.) by the energetic attempt, on the part of numerous philosophers, to show that these goals are in fact un/attainable. On the constructive side, many of the core formalisms and techniques used in AI come out of, and are indeed still much used and refined in, philosophy: first-order logic and its extensions; intensional logics suitable for the modeling of doxastic attitudes and deontic reasoning; inductive logic, probability theory, and probabilistic reasoning; practical reasoning and planning, and so on. In light of this, some philosophers conduct AI research and development *as* philosophy.



AI:




Stanford Encyclopedia of Philosophy

Browse About Support SEP

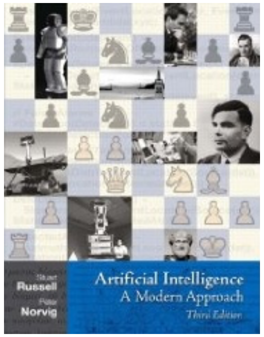
Search SEP

Entry Contents
Bibliography
Academic Tools
Friends PDF Preview
Author and Citation Info
Back to Top

Artificial Intelligence

First published Thu Jul 12, 2018

Artificial intelligence (AI) is the field devoted to building artificial animals (or at least artificial creatures that – in suitable contexts – *appear* to be animals) and, for many, artificial persons (or at least artificial creatures that – in suitable contexts – *appear* to be persons).[1] Such goals immediately ensure that AI is a discipline of considerable interest to many philosophers, and this has been confirmed (e.g.) by the energetic attempt, on the part of numerous philosophers, to show that these goals are in fact un/attainable. On the constructive side, many of the core formalisms and techniques used in AI come out of, and are indeed still much used and refined in, philosophy: first-order logic and its extensions; intensional logics suitable for the modeling of doxastic attitudes and deontic reasoning; inductive logic, probability theory, and probabilistic reasoning; practical reasoning and planning, and so on. In light of this, some philosophers conduct AI research and development *as* philosophy.



Resurrection of The Triad

The Triad Resurrected & Rebuilt, & Better

Logic

$\mathcal{L}$

AI

$$\mathcal{L} := \langle L, \mathcal{I} \rangle$$

$$\frac{\begin{array}{cc} \mathbb{P} & L \\ \mathfrak{q} & L \end{array}}{\begin{array}{c} \mathbb{R} \ : \ \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow \ \langle \mathtt{Y} | \mathtt{N} | \mathtt{U}, \delta, \pi_{(s)} | \alpha_{(s)} \rangle \\ \mathbb{C} \ : \ \pi_{(s)} | \alpha_{(s)} \ \longrightarrow \langle \mathtt{Y} | \mathtt{N} | \mathtt{U}, \delta \rangle \end{array}}$$

Pure General Logic Programming

The Triad Resurrected & Rebuilt, & Better

Logic

$\mathcal{L}$

AI

$$\boldsymbol{\mathcal{L}} := \langle L, \mathcal{I} \rangle$$

$$\frac{\begin{array}{cc} \mathbb{P} & L \\ \mathfrak{q} & L \end{array}}{\begin{array}{c} \mathbb{R} \;:\; \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta, \pi_{(s)}|\alpha_{(s)} \rangle \\ \mathbb{C} \;:\; \pi_{(s)}|\alpha_{(s)} \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta \rangle \end{array}}$$

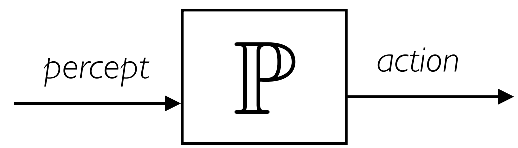
Pure General Logic Programming

The Triad Resurrected & Rebuilt, & Better

Logic

$\mathcal{L}$

AI



$$\boldsymbol{\mathcal{L}} \coloneqq \langle L, \mathcal{I} \rangle \qquad \frac{\begin{array}{cc} \mathbb{P} & L \\ \mathfrak{q} & L \end{array}}{\begin{array}{c} \mathbb{R} \;:\; \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta, \pi_{(s)}|\alpha_{(s)} \rangle \\ \mathbb{C} \;:\; \pi_{(s)}|\alpha_{(s)} \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta \rangle \end{array}}$$

Pure General Logic Programming

What's Part 2 about ? …

PROGRAMMING LANGUAGE PRAGMATICS

IV

FOURTH EDITION

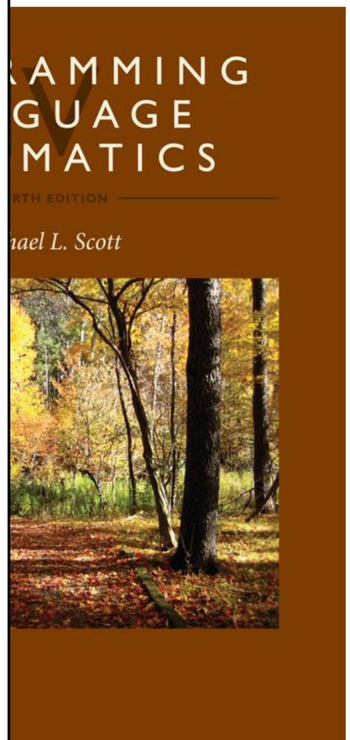
Michael L. Scott

MK
MORGAN KAUFMANN

Alternative Programming Models

As we noted in Chapter 1, programming languages are traditionally though imperfectly classified into various imperative and declarative families. We have had occasion in Parts I and II to mention issues of particular importance to each of the major families. Moreover much of what we have covered—syntax, semantics, naming, types, abstraction—applies uniformly to all. Still, our attention has focused mostly on mainstream imperative languages. In Part III we shift this focus.

Functional and logic languages are the principal nonimperative options. We consider them in Chapters 11 and 12, respectively. In each case we structure our discussion around representative languages: Scheme and OCaml for functional programming, Prolog for logic programming. In Chapter 11 we also cover eager and lazy evaluation, and first-class and higher-order functions. In Chapter 12 we cover issues that make fully automatic, general purpose logic programming difficult, and describe restrictions used in practice to keep the model tractable. Optional sections in both chapters consider mathe-



Models

As we noted in Chapter 1, programming languages are traditionally though imperfectly classified into various imperative and declarative families. We have had occasion in Parts I and II to mention issues of particular importance to each of the major families. Moreover much of what we have covered—syntax, semantics, naming, types, abstraction—applies uniformly to all. Still, our attention has focused mostly on mainstream imperative languages. In Part III we shift this focus.

Functional and logic languages are the principal nonimperative options. We consider them in Chapters 11 and 12, respectively. In each case we structure our discussion around representative languages: Scheme and OCaml for functional programming, Prolog for logic programming. In Chapter 11 we also cover eager and lazy evaluation, and first-class and higher-order functions. In Chapter 12 we cover issues that make fully automatic, general purpose logic programming difficult, and describe restrictions used in practice to keep the model tractable. Optional sections in both chapters consider mathe-

Models

As we noted in Chapter 1
are traditionally though i
various imperative and d
had occasion in Parts I ar
particular importance to
Moreover much of what w
semantics, naming, types
formly to all. Still, our at
on mainstream imperativ
shift this focus.

Functional and logic la
nonimperative options. W
ters 11 and 12, respective
ture our discussion aroun
Scheme and OCaml for f
Prolog for logic program
cover eager and lazy eval
higher-order functions. I
issues that make fully au
logic programming diffic
used in practice to keep t
Optional sections in both

matical foundations: Lambda Calculus for functional programming, Predicate Calculus for logic programming.

The remaining two chapters consider concurrent and scripting models, both of which are increasingly popular, and cut across the imperative/declarative divide. Concurrency is driven by the hardware parallelism of internetworked computers and by the coming explosion in multithreaded processors and chip-level multiprocessors. Scripting is driven by the growth of the World Wide Web and by an increasing emphasis on programmer productivity, which places rapid development and reusability above sheer run-time performance.

Chapter 13 begins with the fundamentals of concurrency, including communication and synchronization, thread creation syntax, and the implementation of threads. The remainder of the chapter is divided between *shared-memory* models, in which threads use explicit or implicit synchronization mechanisms to manage a common set of variables, and (on the companion site) *message-passing* models, in which threads interact only through explicit communication.

The first half of Chapter 14 surveys problem domains in which scripting plays a major role: shell (command) languages, text processing and report generation, mathematics and statistics, the "gluing" together of program components, extension mechanisms for complex applications, and client and server-side Web scripting. The second half considers some of the more important language innovations championed by scripting languages: flexible scoping and naming conventions, string and pattern manipulation (extended regular expressions), and high level data types.

matical foundations: Lambda Calculus for functional programming, Predicate Calculus for logic programming.

The remaining two chapters consider concurrent and scripting models, both of which are increasingly popular, and cut across the imperative/declarative divide. Concurrency is driven by the hardware parallelism of internetworked computers and by the coming explosion in multithreaded processors and chip-level multiprocessors. Scripting is driven by the growth of the World Wide Web and by an increasing emphasis on programmer productivity, which places rapid development and reusability above sheer run-time performance.

Chapter 13 begins with the fundamentals of concurrency, including communication and synchronization, thread creation syntax, and the implementation of threads. The remainder of the chapter is divided between *shared-memory* models, in which threads use explicit or implicit synchronization mechanisms to manage a common set of variables, and (on the companion site) *message-passing* models, in which threads interact only through explicit communication.

The first half of Chapter 14 surveys problem domains in which scripting plays a major role: shell (command) languages, text processing and report generation, mathematics and statistics, the "gluing" together of program components, extension mechanisms for complex applications, and client and server-side Web scripting. The second half considers some of the more important language innovations championed by scripting languages: flexible scoping and naming conventions, string and pattern manipulation (extended regular expressions), and high level data types.

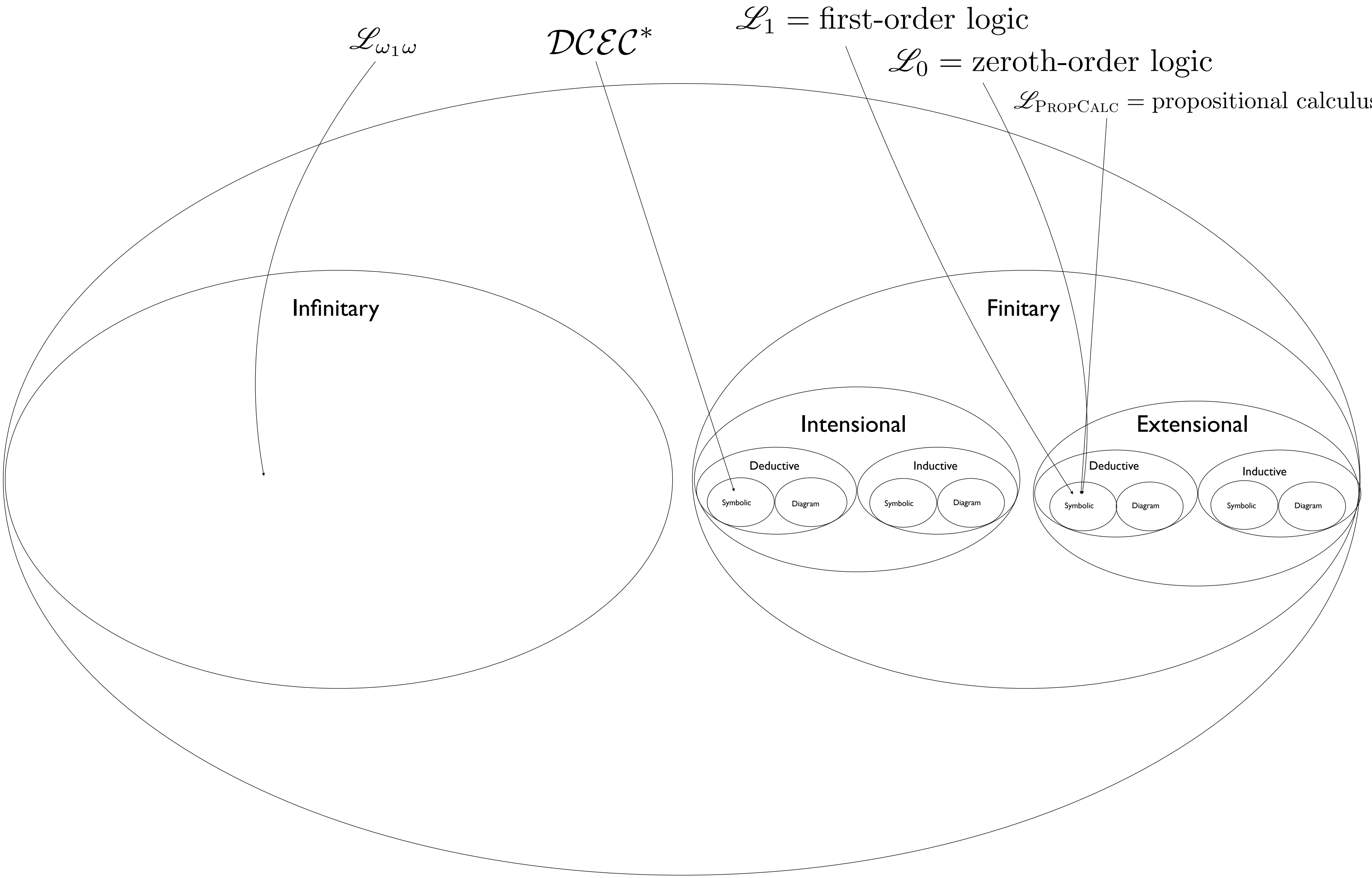
matical foundations: Lambda Calculus for functional programming, Predicate Calculus for logic programming.

The remaining two chapters consider concurrent and scripting models, both of which are increasingly popular, and cut across the imperative/declarative divide. Concurrency is driven by the hardware parallelism of internetworked computers and by the coming explosion in multithreaded processors and chip-level multiprocessors. Scripting is driven by the growth of the World Wide Web and by an increasing emphasis on programmer productivity, which places rapid development and reusability above sheer run-time performance.

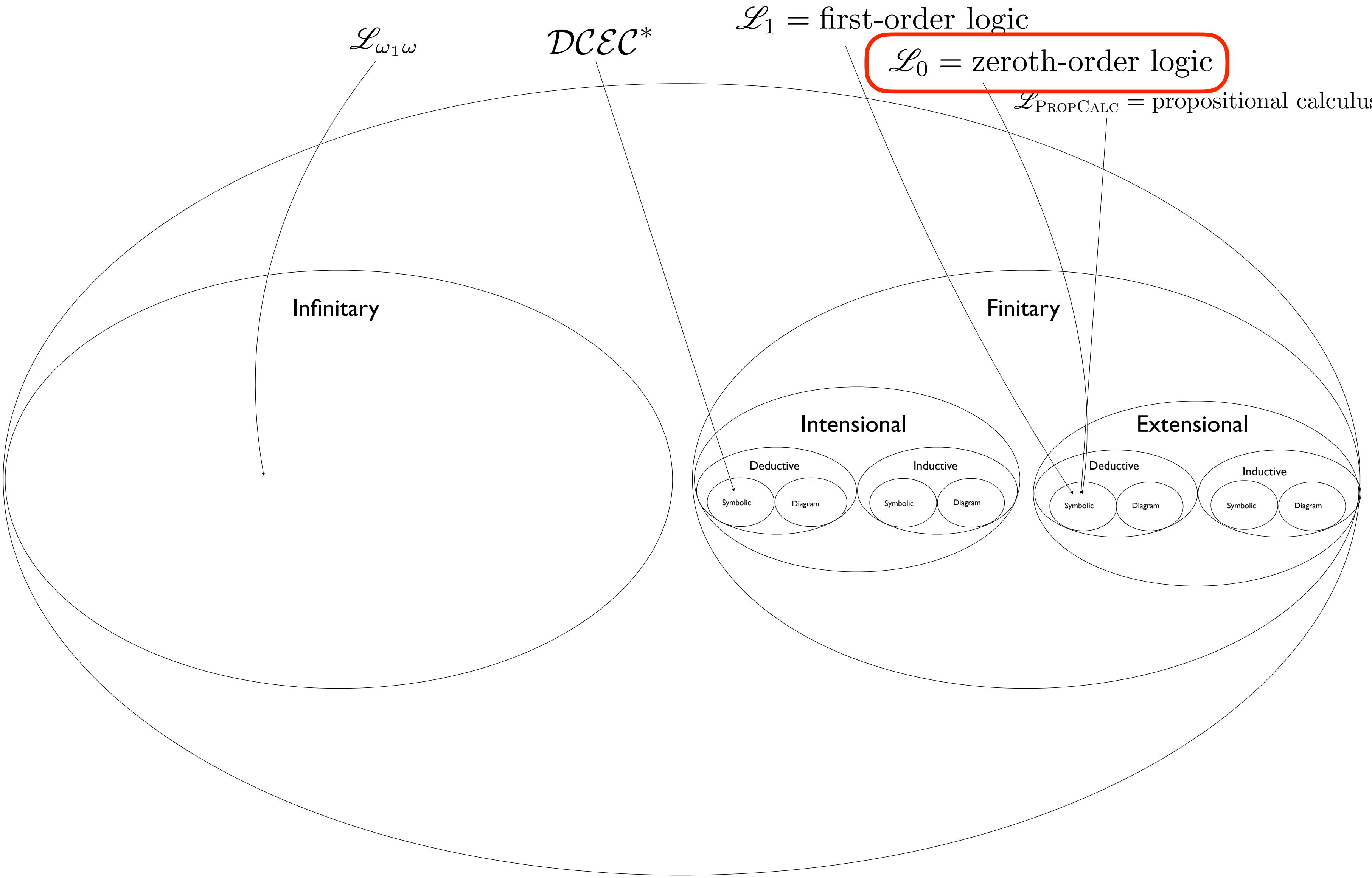
Chapter 13 begins with the fundamentals of concurrency, including communication and synchronization, thread creation syntax, and the implementation of threads. The remainder of the chapter is divided between *shared-memory* models, in which threads use explicit or implicit synchronization mechanisms to manage a common set of variables, and (on the companion site) *message-passing* models, in which threads interact only through explicit communication.

The first half of Chapter 14 surveys problem domains in which scripting plays a major role: shell (command) languages, text processing and report generation, mathematics and statistics, the "gluing" together of program components, extension mechanisms for complex applications, and client and server-side Web scripting. The second half considers some of the more important language innovations championed by scripting languages: flexible scoping and naming conventions, string and pattern manipulation (extended regular expressions), and high level data types.

The Universe of Logics



The Universe of Logics



Starter HyperLog®: Datalog

$$\mathscr{L}_0 = \text{zeroth-order logic}$$

Datalog Syntax

$$\begin{array}{lcr}
(\textit{program})\ P & := & \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \\
(\textit{horn-clause formula})\ \phi_i & := & \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha_j \\
(\textit{atomic formula})\ \alpha_i & := & R(t_1, t_2, \ldots t_o) \\
(\textit{terms})\ t & := & x \mid c
\end{array}$$

where x is a variable and c a constant

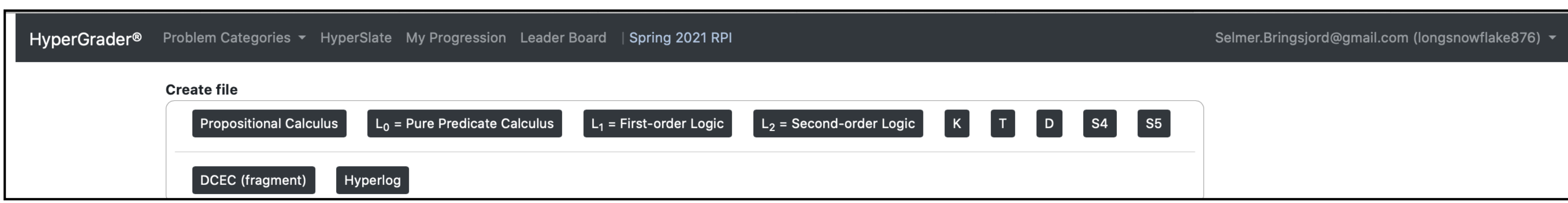
Starter HyperLog®: Datalog

$$\mathscr{L}_0 = \text{zeroth-order logic}$$

Datalog Syntax

$(\textit{program})\ P$	$:=$	$\phi_1 \land \phi_2 \land \ldots \land \phi_n$
$(\textit{horn-clause formula})\ \phi_i$	$:=$	$\alpha_1 \land \ldots \land \alpha_m \rightarrow \alpha_j$
$(\textit{atomic formula})\ \alpha_i$	$:=$	$R(t_1, t_2, \ldots t_o)$
$(\textit{terms})\ t$	$:=$	$x \mid c$

where x is a variable and c a constant



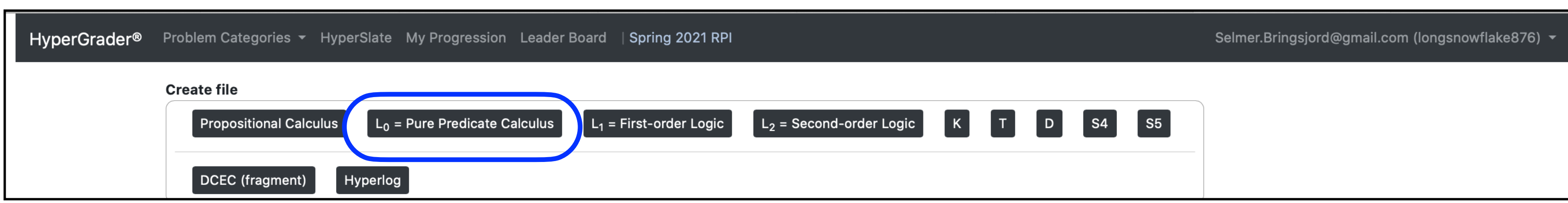
Starter HyperLog®: Datalog

$$\mathscr{L}_0 = \text{zeroth-order logic}$$

Datalog Syntax

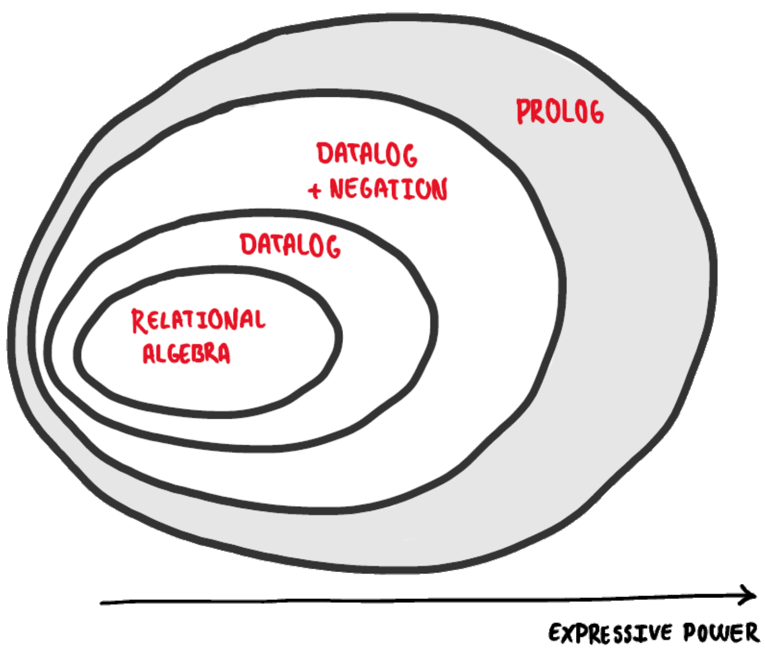
(*program*) P	$:=$	$\phi_1 \land \phi_2 \land \ldots \land \phi_n$
(*horn-clause formula*) ϕ_i	$:=$	$\alpha_1 \land \ldots \land \alpha_m \rightarrow \alpha_j$
(*atomic formula*) α_i	$:=$	$R(t_1, t_2, \ldots t_o)$
(*terms*) t	$:=$	$x \mid c$

where x is a variable and c a constant



Starter HyperLog®: Datalog

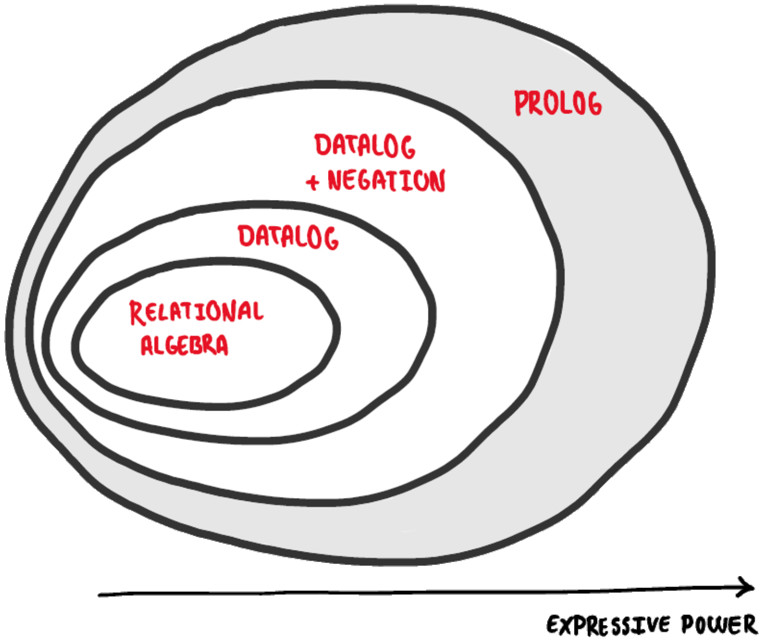
From "Introduction to Datalog," an excellent online piece.



$\mathscr{L}_0$

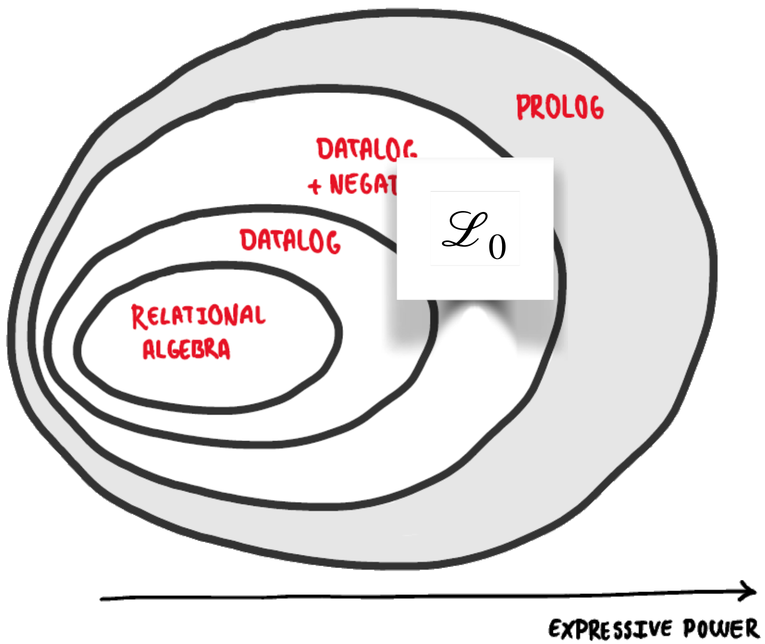
Starter HyperLog®: Datalog

From “Introduction to Datalog,” an excellent online piece.



Starter HyperLog®: Datalog

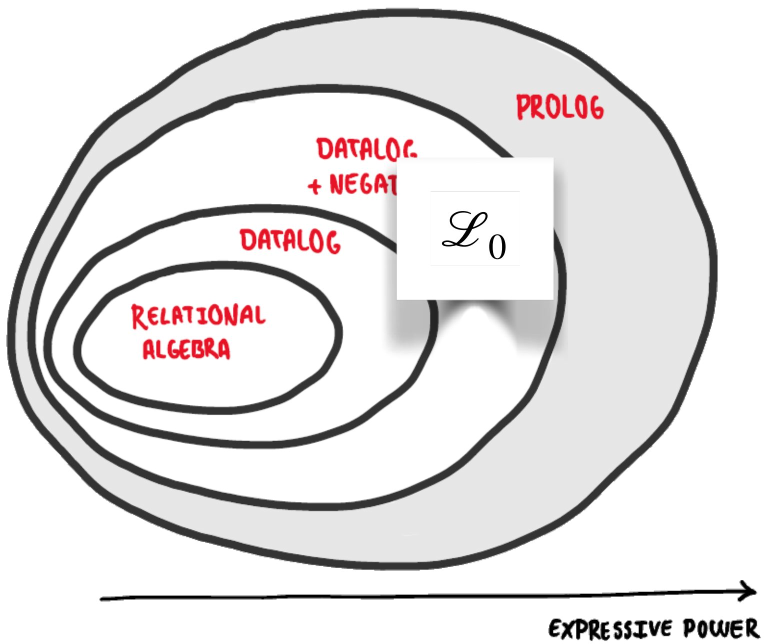
From "Introduction to Datalog," an excellent online piece.



$\mathscr{L}_1$

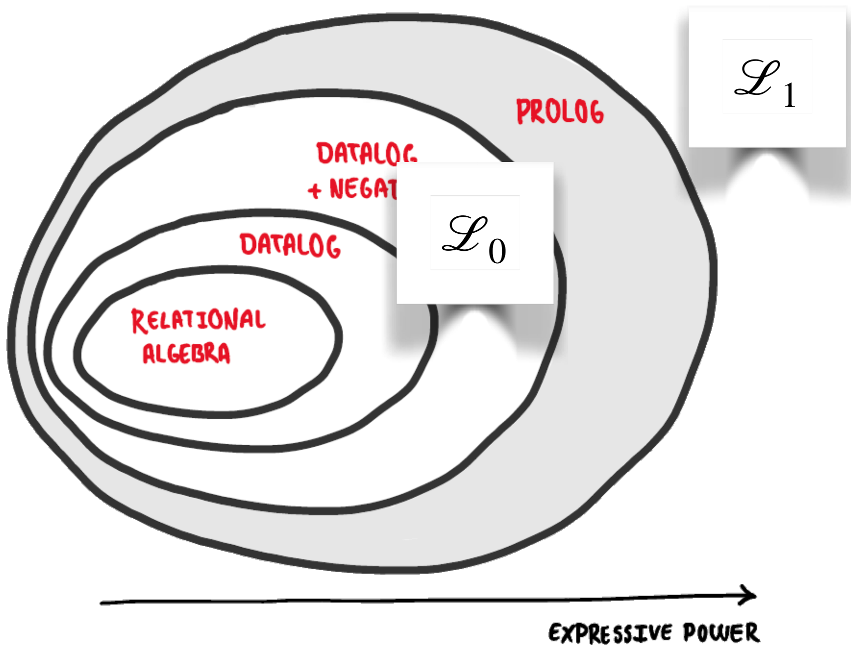
Starter HyperLog®: Datalog

From “Introduction to Datalog,” an excellent online piece.



Starter HyperLog®: Datalog

From "Introduction to Datalog," an excellent online piece.



slutten