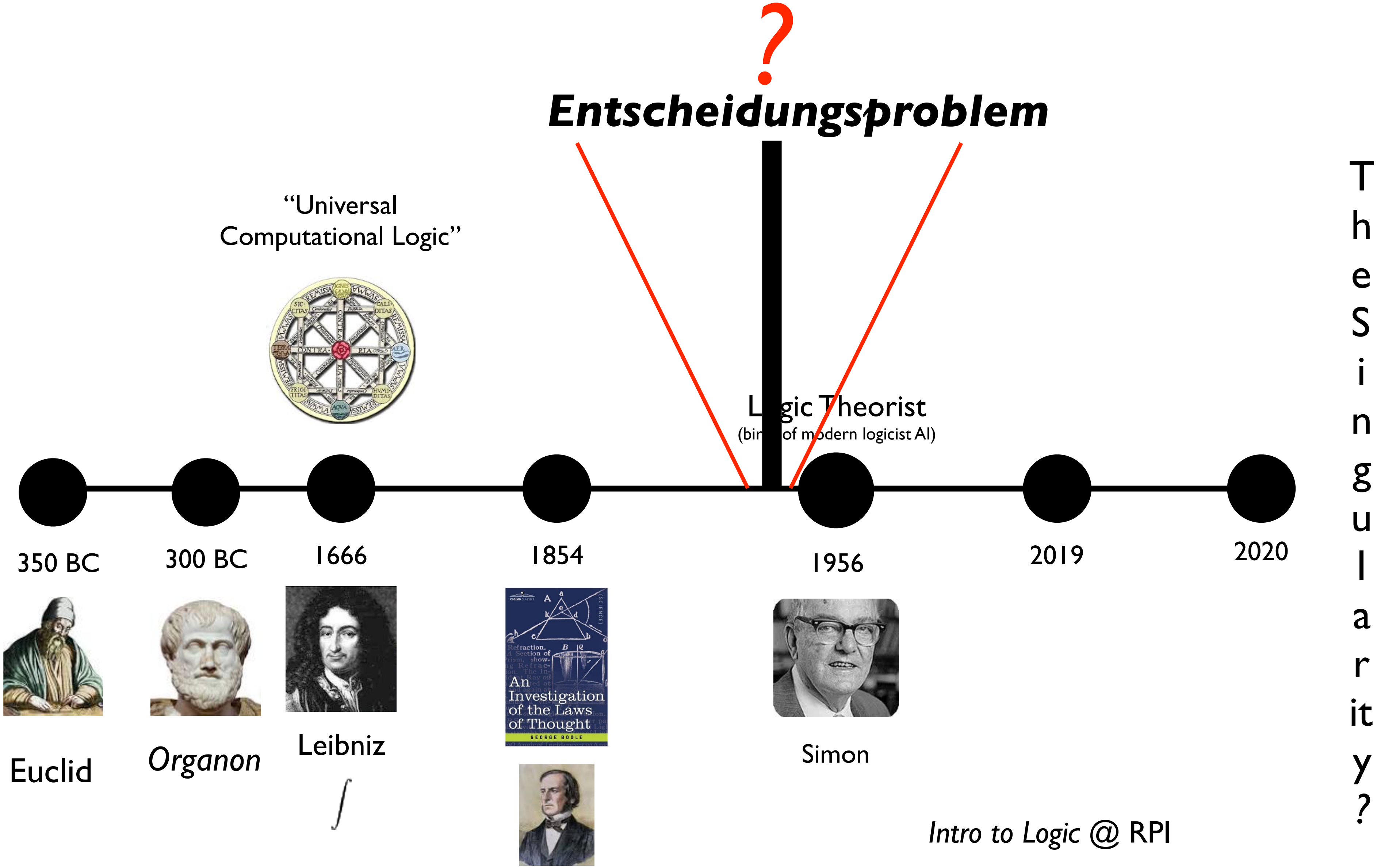
Pure General Logic Programming
(Intro/Overview)

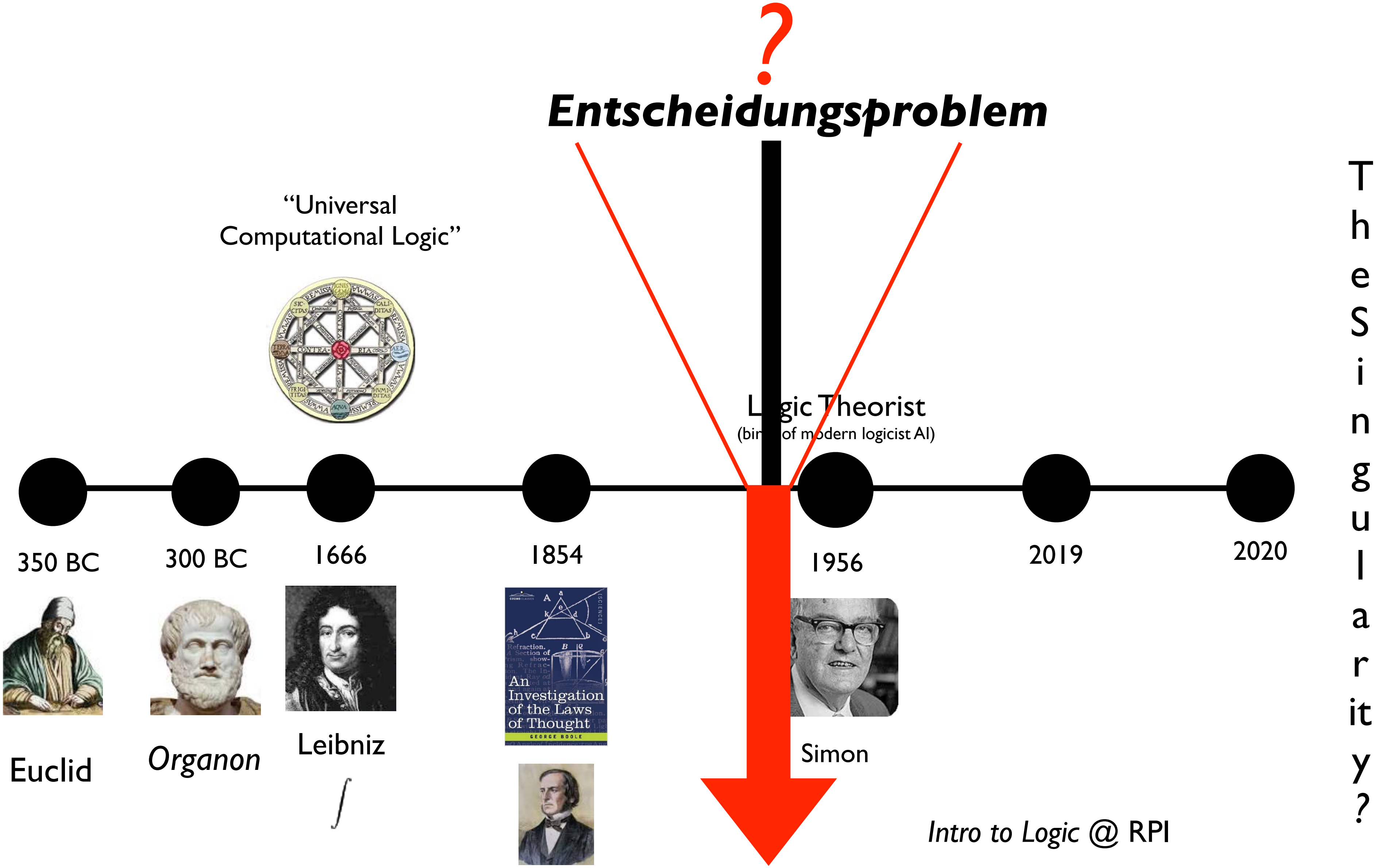
Selmer Bringsjord

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

Intro to Logic
2/10/2020

RAIR
Rensselaer AI and Reasoning Lab


?
Entscheidungsproblem
"Universal Computational Logic"
Logic Theorist
(birth of modern logicist AI)
350 BC
300 BC
1666
1854
1956
2019
2020
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
George Boole
Simon
Intro to Logic @ RPI
The Singularity?

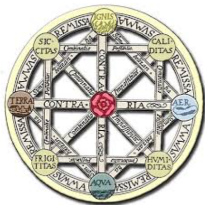


?
Entscheidungsproblem
“Universal Computational Logic”
Logic Theorist
(birth of modern logicist AI)
350 BC
300 BC
1666
1854
1956
2019
2020
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
GEORGE BOOLE
Simon
Intro to Logic @ RPI
The Singularity?

?

Entscheidungsproblem

"Universal Computational Logic"

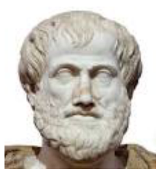


Logic Theorist

(birth of modern logicist AI)

350 BC	300 BC	1666	1854	1956	2019	2020

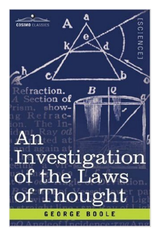
Euclid



Organon

Leibniz

∫



Simon

Intro to Logic @ RPI

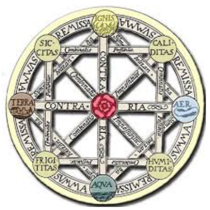
Frege

The Singularity?

?

Entscheidungsproblem

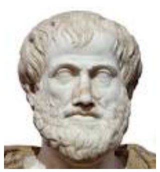
"Universal Computational Logic"



Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2019 | 2020

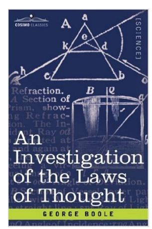
Euclid



Organon

Leibniz

$\int$



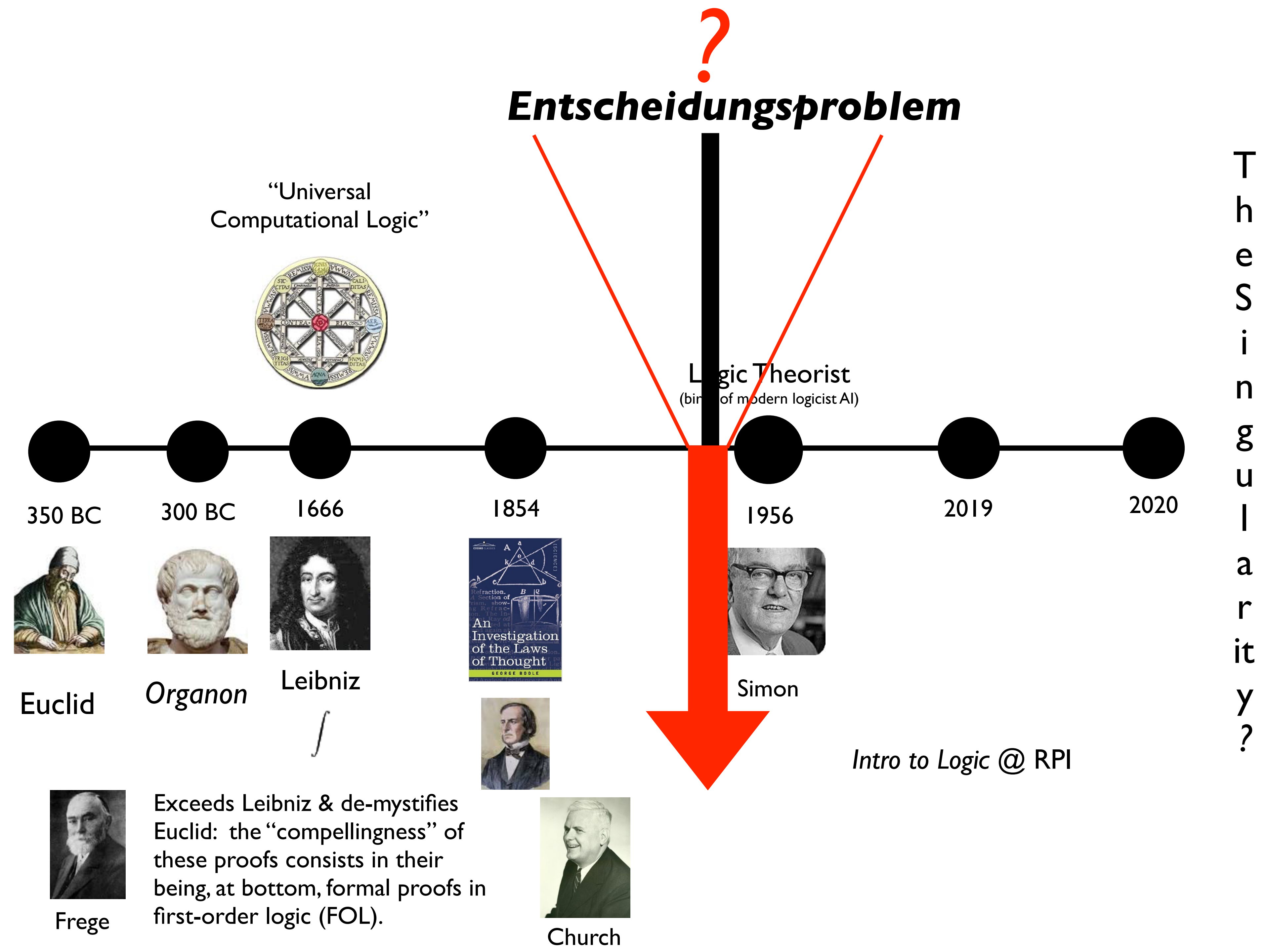
Simon

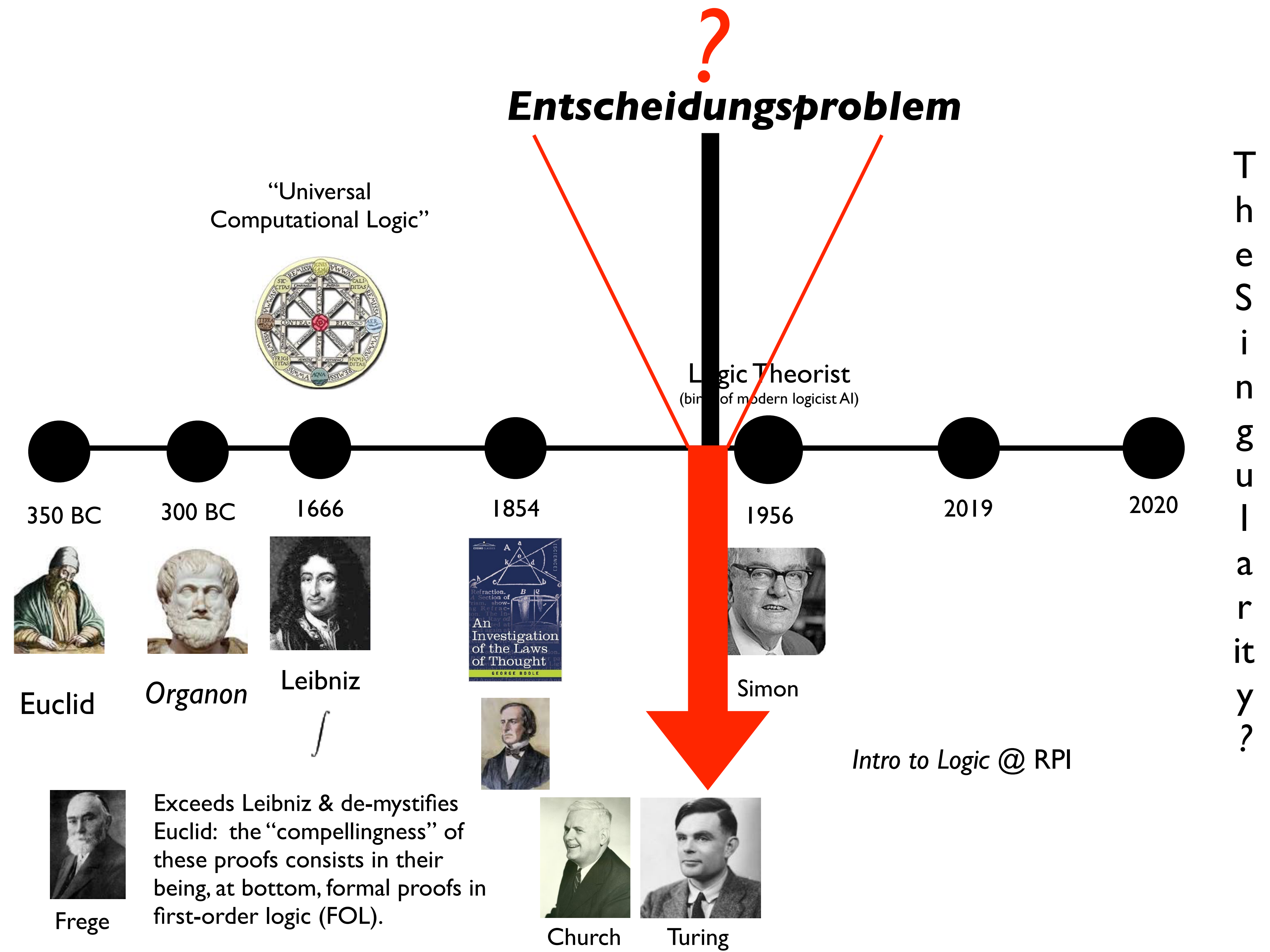
Intro to Logic @ RPI

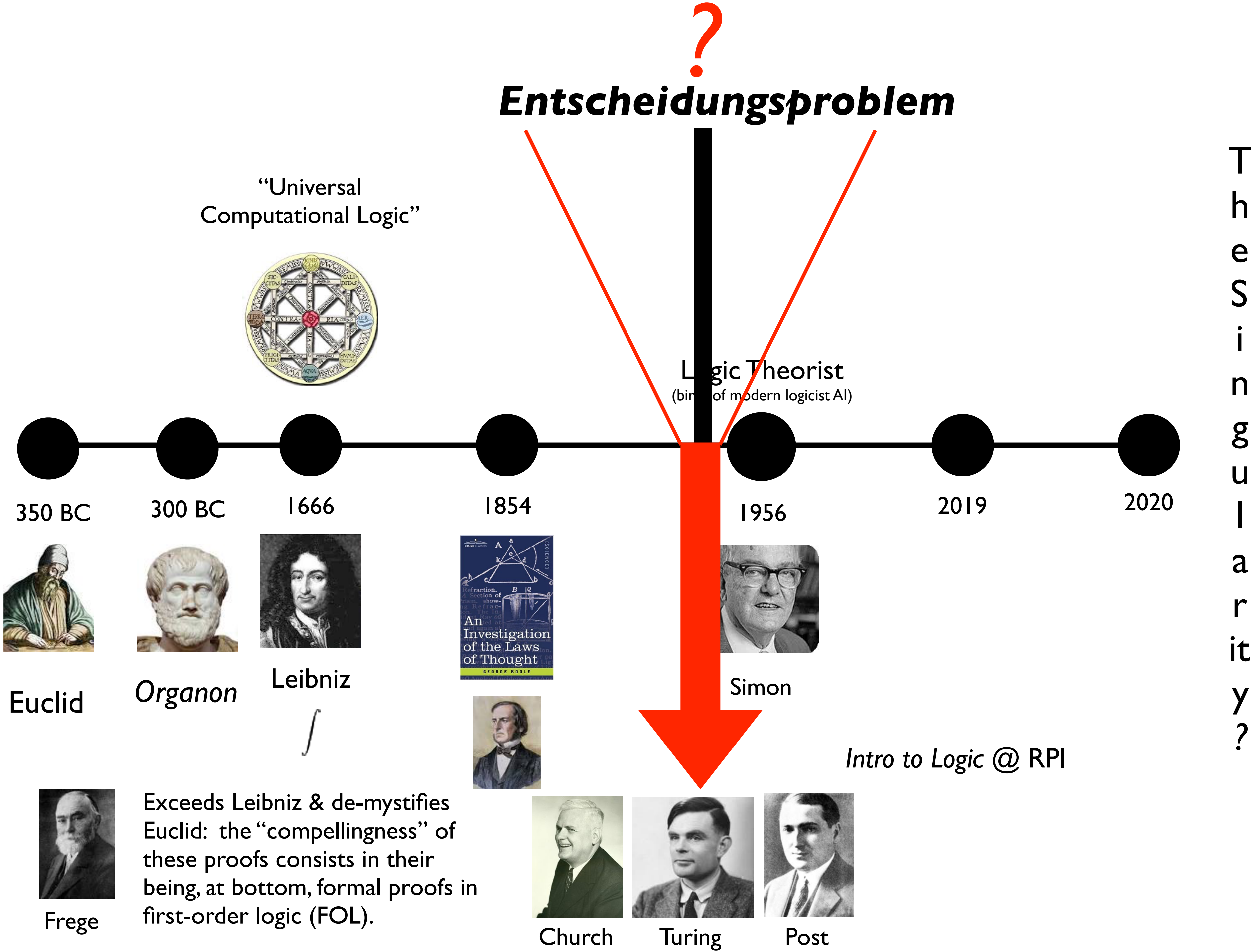
The Singularity?

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
2019
2020
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
GEORGE BOOLE
Simon
Intro to Logic @ RPI
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"
L gic Theorist
(bir of m dern logicist AI)
350 BC
300 BC
1666
1854
1956
2019
2020
Euclid
Organon
Leibniz
∫
An Investigation of the Laws of Thought
GEORGE BOOLE
Simon
Intro to Logic @ RPI
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
2019
2020
Euclid
Organon
Leibniz
An Investigation of the Laws of Thought
GEORGE BOOLE
Simon
Intro to Logic @ RPI
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
The Singularity?

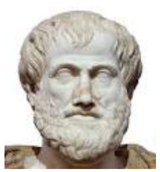
Entscheidungsproblem ?

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

350 BC	300 BC	1666	1854	1956	2019	2020

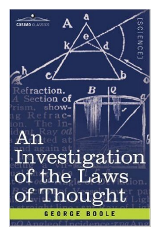
Euclid



Organon

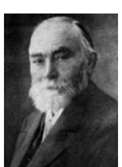
Leibniz

$\int$



Simon

Intro to Logic @ RPI



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!

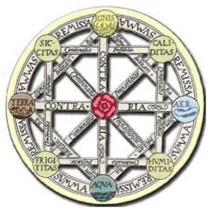
The Singularity?

New: Functional = Church; Procedural = Turing. Where is logicist computation?

?

Entscheidungsproblem

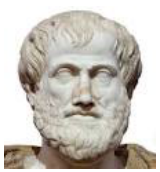
"Universal Computational Logic"



Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2019 | 2020

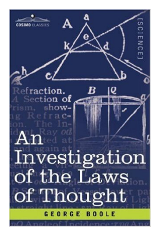
Euclid



Organon

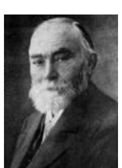
Leibniz

$\int$



Simon

Intro to Logic @ RPI



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!

The Singularity?

New: Functional = Church; Procedural = Turing. Where is logicist computation?

?

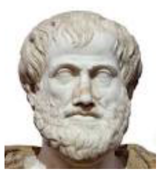
Entscheidungsproblem

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2019 | 2020

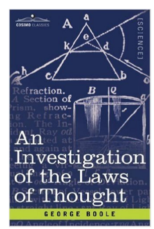
Euclid



Organon

Leibniz

$\int$



Simon

The Singularity?

Intro to Logic @ RPI

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: Functional = Church; Procedural = Turing. Where is logicist computation?

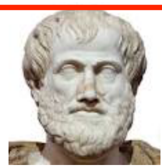
Entscheidungsproblem ?

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2019 | 2020

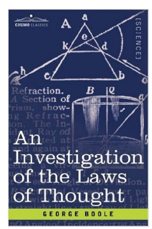
Euclid



Organon

Leibniz

$\int$



Simon

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

Intro to Logic @ RPI

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: Functional = Church; Procedural = Turing. Where is logicist computation?

?

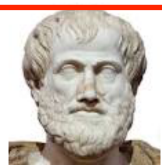
Entscheidungsproblem

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2019 | 2020

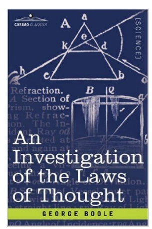
Euclid



Organon

Leibniz

∫



Simon

The Singularity?

Intro to Logic @ RPI

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: Functional = Church; Procedural = Turing. Where is logicist computation?

Entscheidungsproblem ?

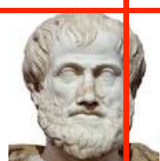
"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

The Singularity?

350 BC	300 BC	1666	1854	1956	2019	2020

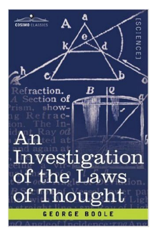
Euclid



Organon

Leibniz

$\int$



Simon

Intro to Logic @ RPI

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: Functional = Church; Procedural = Turing. Where is logicist computation?

?

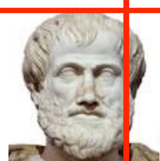
Entscheidungsproblem

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

350 BC | 300 BC | 1666 | 1854 | 1956 | 2019 | 2020

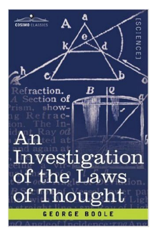
Euclid



Organon

Leibniz

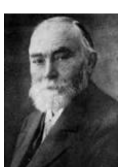
$\int$



Simon

The Singularity?

Intro to Logic @ RPI



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: Functional = Church; Procedural = Turing. Where is logicist computation?

?

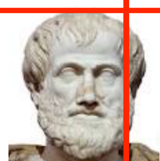
Entscheidungsproblem

"Universal Computational Logic"

Logic Theorist
(birth of modern logicist AI)

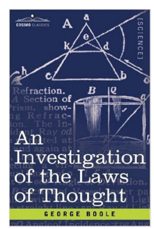
350 BC | 300 BC | 1666 | 1854 | 1956 | 2019 | 2020


Euclid


Organon


Leibniz

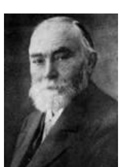
∫




Simon

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

Intro to Logic @ RPI


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!

Two Logicist Branches; B1:

Two Logicist Branches; B1:

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

Two Logicist Branches; B1:

Frege (remember him?), 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!"

Two Logicist Branches; B1:

Frege (remember him?), 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!

Two Logicist Branches; B1:

Frege (remember him?), 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!

...

Two Logicist Branches; B1:

Frege (remember him?), 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

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

Single-Slide Encapsulation …

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

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

answer; could itself be a theory/set of formulae

degree of "confidence"

program

proof(s)

query

argument(s)

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

reasoner

checker

The “British Track” (Procedural) …

The Track

The Track

Computational thinking is … Computer programming is … A computer program is …

The Track

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

ingrained in everyone's lives when words like algorithm and precondition are part of everyone's vocab-

Computational thinking thus has the following characteristics:

Track

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

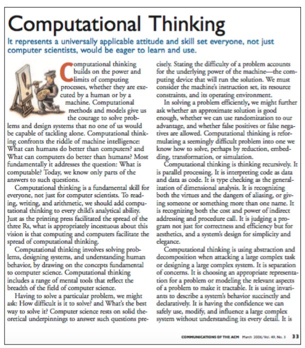
Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

The Track



Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

The Track

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

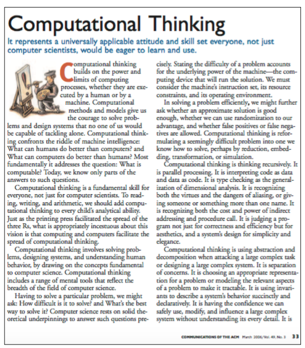
Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

The Track



Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

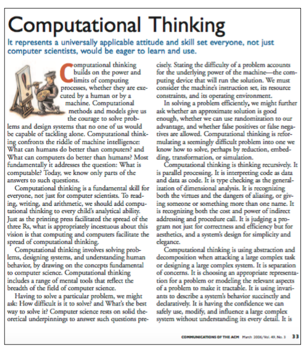
Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

COMMUNICATIONS OF THE ACM March 2006/Vol. 49, No. 3 33

→ Teach computer programming!
(procedural, o-o, functional)

The Track



Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

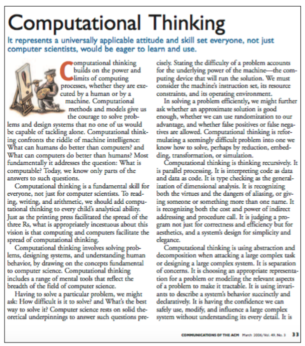
Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

COMMUNICATIONS OF THE ACM March 2006/Vol. 49, No. 3 33

→ Teach computer programming!
(procedural, o-o, functional) →

The Track


Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

COMMUNICATIONS OF THE ACM March 2006/Vol. 49, No. 3 33

Teach computer programming!
(procedural, o-o, functional)

Computer science is the scientific (or STEM) study of:

what problems can be solved,
what tasks can be accomplished,
and what features of the world can be understood . . .

. . . *computationally*, that is, using a language with only:

2 nouns ('0', '1'),
3 verbs ('move', 'print', 'halt'),
3 grammar rules (sequence, selection, repetition),
and nothing else,

and then to provide algorithms to show how this can be done:

efficiently,
practically,
physically,
and ethically.

The Track

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

Teach computer programming!
(procedural, o-o, functional)

Computer science is the scientific (or STEM) study of:

what problems can be solved,
what tasks can be accomplished,
and what features of the world can be understood . . .

. . . *computationally*, that is, using a language with only:

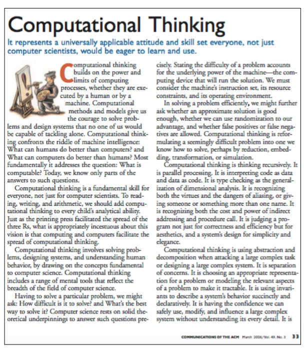
2 nouns ('0', '1'),
3 verbs ('move', 'print', 'halt'),
3 grammar rules (sequence, selection, repetition),
and nothing else,

and then to provide algorithms to show how this can be done:

efficiently,
practically,
physically,
and ethically.

– Rapaport, "phics" book

The Track



Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

COMMUNICATIONS OF THE ACM March 2006/Vol. 49, No. 3 33

→ Teach computer programming!
(procedural, o-o, functional) →

Computer science is the scientific (or STEM) study of:

what problems can be solved,
what tasks can be accomplished,
and what features of the world can be understood . . .

. . . *computationally*, that is, using a language with only:

2 nouns ('0', '1'),
3 verbs ('move', 'print', 'halt'),
3 grammar rules (sequence, selection, repetition),
and nothing else,

and then to provide algorithms to show how this can be done:

efficiently,
practically,
physically,
and ethically.

– Rapaport, "phics" book

What about Turner?!?

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 …"

2. Leibniz’s Universal Calculus Found …

1716

Leibniz

1716

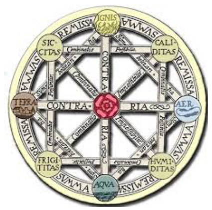
Leibniz

1.5 centuries < Boole!

2.5 centuries < Kripke

vindicated by Robinson 2.5 centuries later

“Universal
Cognitive
Calculus”



1716

Leibniz

1.5 centuries < Boole!
2.5 centuries < Kripke
vindicated by Robinson 2.5 centuries later

“Universal
Cognitive
Calculus”

1716

Leibniz

1.5 centuries < Boole!
2.5 centuries < Kripke
vindicated by Robinson 2.5 centuries later

“Universal
Cognitive
Calculus”

1716

2016

Leibniz

1.5 centuries < Boole!
2.5 centuries < Kripke
vindicated by Robinson 2.5 centuries later

“Universal
Cognitive
Calculus”

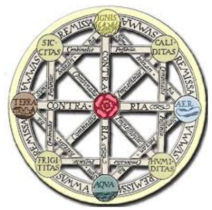
1716

2016

Leibniz

1.5 centuries < Boole!
2.5 centuries < Kripke
vindicated by Robinson 2.5 centuries later

“Universal
Cognitive
Calculus”



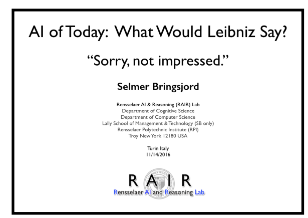
1716

Leibniz

1.5 centuries < Boole!
2.5 centuries < Kripke
vindicated by Robinson 2.5 centuries later

Universal
Cognitive
Calculus
Found

2016


AI of Today: What Would Leibniz Say?

“Sorry, not impressed.”

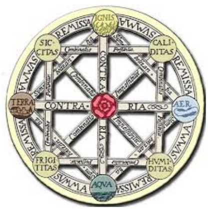
Selmer Bringsjord

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

Turin Italy
11/14/2016

R A I R
Rensselaer AI and Reasoning Lab

“Universal Cognitive Calculus”



Universal Cognitive Calculus *Found*

1716

2016

2018

AI of Today: What Would Leibniz Say?

“Sorry, not impressed.”

Selmer Bringsjord

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

Turin Italy
11/14/2016

RAIR
Rensselaer AI and Reasoning Lab

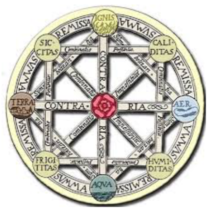
Leibniz

1.5 centuries < Boole!

2.5 centuries < Kripke

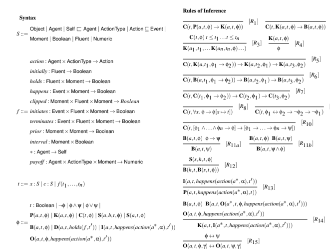
vindicated by Robinson 2.5 centuries later

"Universal Cognitive Calculus"

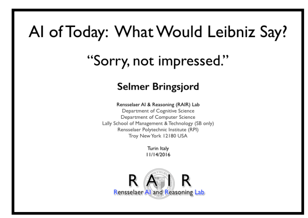


Universal Cognitive Calculus *Found*

$\mathcal{DCEC}^*$



1716 — 2016 — 2018


AI of Today: What Would Leibniz Say?

"Sorry, not impressed."

Selmer Bringsjord

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

Turin Italy
11/14/2016

RAIR
Rensselaer AI and Reasoning Lab

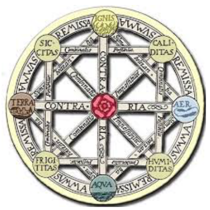
Leibniz

1.5 centuries < Boole!

2.5 centuries < Kripke

vindicated by Robinson 2.5 centuries later

“Universal
Cognitive
Calculus”



1716

Leibniz

1.5 centuries < Boole!
2.5 centuries < Kripke
vindicated by Robinson 2.5 centuries later

Universal
Cognitive
Calculus
Found

2016


AI of Today: What Would Leibniz Say?

“Sorry, not impressed.”

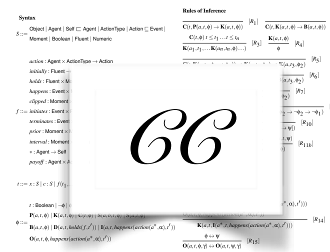
Selmer Bringsjord

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

Turin Italy
11/14/2016

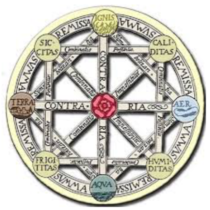
RAIR
Rensselaer AI and Reasoning Lab

$\mathcal{DCEC}^*$



2018

“Universal Cognitive Calculus”



1716

Leibniz

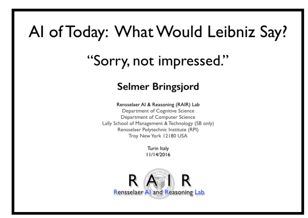
1.5 centuries < Boole!

2.5 centuries < Kripke

vindicated by Robinson 2.5 centuries later

Universal Cognitive Calculus *Found*

2016


AI of Today: What Would Leibniz Say?

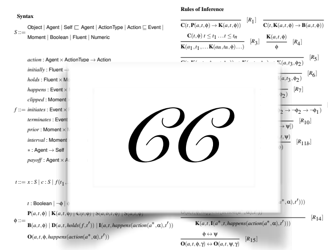
“Sorry, not impressed.”

Selmer Bringsjord

RAIR

Rensselaer AI and Reasoning Lab

$\mathcal{DCEC}^*$



2018

RAIR

Rensselaer AI and Reasoning Lab

“Universal Cognitive Calculus”

1716

Leibniz

1.5 centuries < Boole!

2.5 centuries < Kripke

vindicated by Robinson 2.5 centuries later

Universal Cognitive Calculus *Found*

2016


AI of Today: What Would Leibniz Say?

“Sorry, not impressed.”

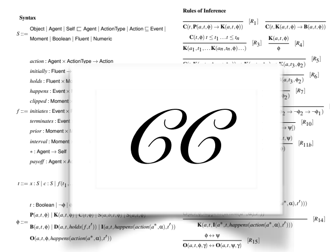
Selmer Bringsjord

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

Turin Italy
11/14/2016

RAIR
Rensselaer AI and Reasoning Lab

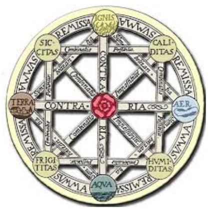
$\mathcal{DCEC}^*$



2018

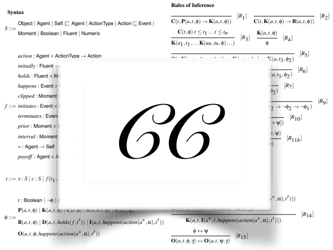
RAIR
Rensselaer AI and Reasoning Lab

“Universal Cognitive Calculus”



Universal Cognitive Calculus *Found*

$\mathcal{DCEC}^*$



1716

2016

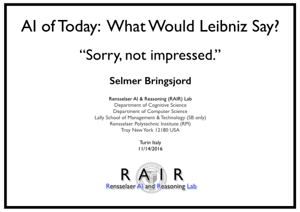
2018

Leibniz

1.5 centuries < Boole!

2.5 centuries < Kripke

vindicated by Robinson 2.5 centuries later



Rensselaer AI and Reasoning Lab

“Universal Cognitive Calculus”

1716

Pure General Logic Programming (PGLP)

ANR

Leibniz

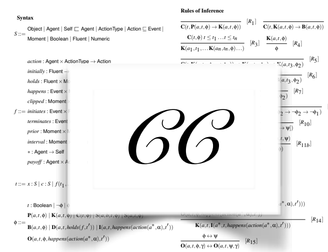
1.5 centuries < Boole!
2.5 centuries < Kripke
vindicated by Robinson 2.5 centuries later

Universal Cognitive Calculus *Found*

2016


AI of Today: What Would Leibniz Say?
“Sorry, not impressed.”
Selmer Bringsjord
Rensselaer AI & Reasoning (RAIR) Lab
Department of Cognitive Science
Department of Computer Science
Lally School of Management & Technology (SB only)
Rensselaer Polytechnic Institute (RPI)
Troy New York 12180 USA
Turin Italy
11/14/2016
R A I R
Rensselaer AI and Reasoning Lab

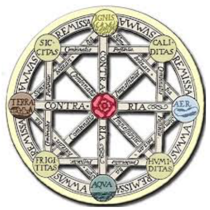
$\mathcal{DCEC}^*$



2018

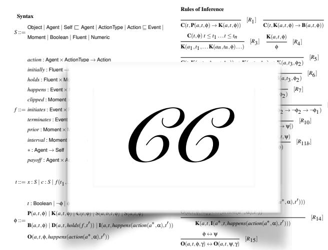
R A I R
Rensselaer AI and Reasoning Lab

"Universal Cognitive Calculus"



Universal Cognitive Calculus *Found*

$\mathcal{DCEC}^*$



1716

Leibniz

1.5 centuries < Boole!

2.5 centuries < Kripke

vindicated by Robinson 2.5 centuries later

2016


AI of Today: What Would Leibniz Say?

"Sorry, not impressed."

Selmer Bringsjord

2018

RAIR
Rensselaer AI and Reasoning Lab

Pure General Logic Programming (PGLP)

ANR

Leibniz's Dream of the Universal Cognitive Calculus

Leibniz's Dream of the Universal Cognitive Calculus

I have come to understand that everything … which algebra proves is only due to a higher science, which I now usually call a *combinatorial characteristic*, though it is far different from what may first occur to someone hearing these words. … Yet I should venture to say that nothing more effective can well be conceived for perfecting the human mind and that if this basis for philosophizing is accepted, there will come a time, and it will be soon, when we shall have as certain knowledge of God and the mind as we now have of figures and numbers and when the invention of machines will be no more difficult than the construction of geometric problems.
(Leibniz, 1675)

Leibniz's Dream of the Universal Cognitive Calculus

Leibniz's Dream of the Universal Cognitive Calculus

This is undoubtedly one of the greatest projects to which men have ever set themselves. It will be an instrument even more useful to the mind than telescopes or microscopes are to the eyes. Every line of this writing will be equivalent to a demonstration. The only fallacies will be easily detected errors in calculation. This will become the great method of discovering truths, establishing them, and teaching them irresistibly when they are established.
(Leibniz, 1679)

Leibniz's Dream of the Universal Cognitive Calculus

Leibniz's Dream of the Universal Cognitive Calculus

I certainly believe that it is useful to depart from rigorous demonstration in geometry because errors are easily avoided there, but in metaphysical and ethical matters I think we should follow the greatest rigor. Yet if we had an established characteristic we might reason as safely in metaphysics as in mathematics. (Leibniz, 1679)

E.g., re. the dream of the Universal Cognitive Calculus

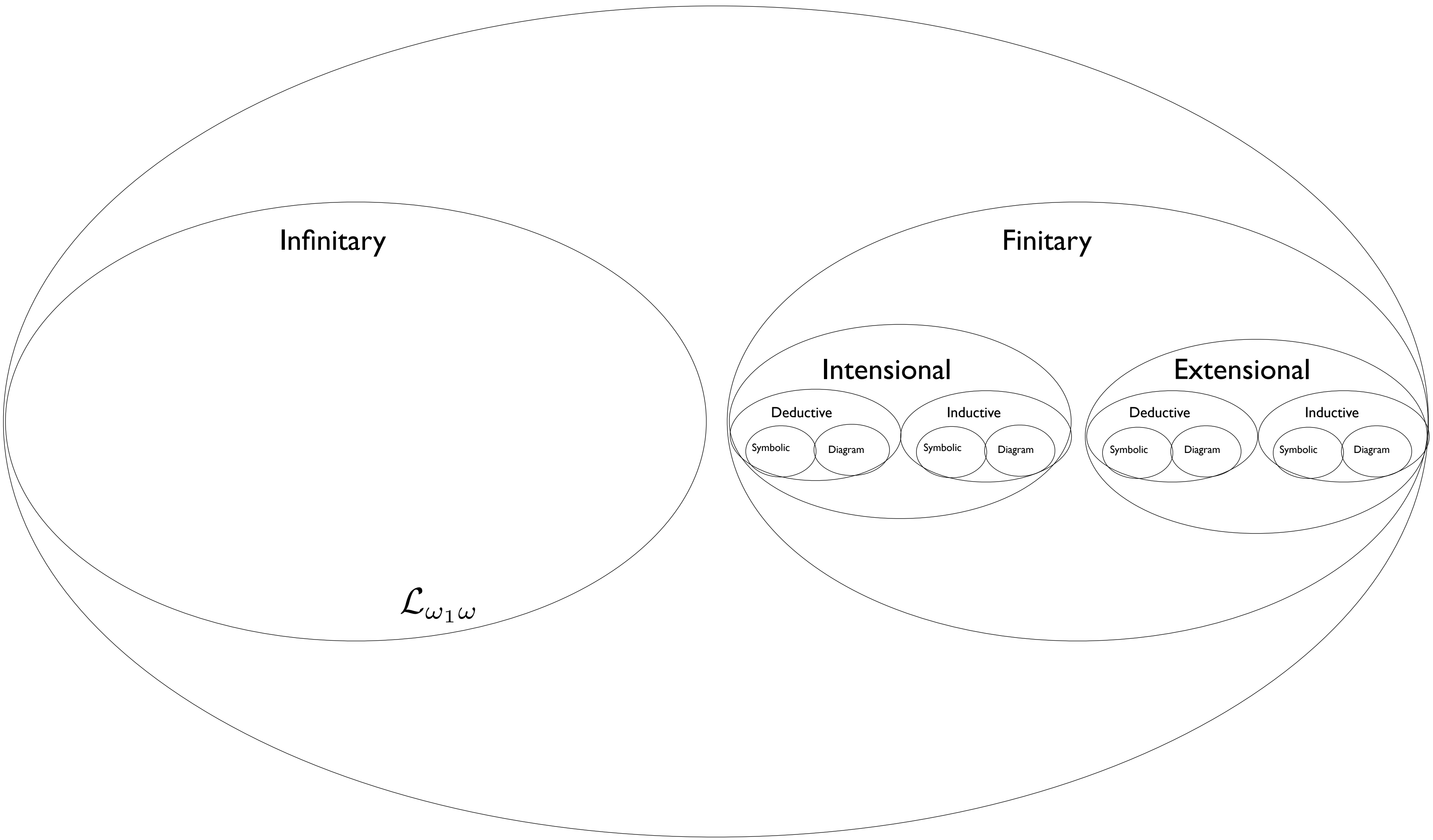
E.g., re. the dream of the Universal Cognitive Calculus

When we lack sufficient data to drive at certainty in our truths, it would also serve to estimate degrees of probability and to see what is needed to provide this certainty. (Leibniz, 1679)

3. The Space of Particular Logical Calculi

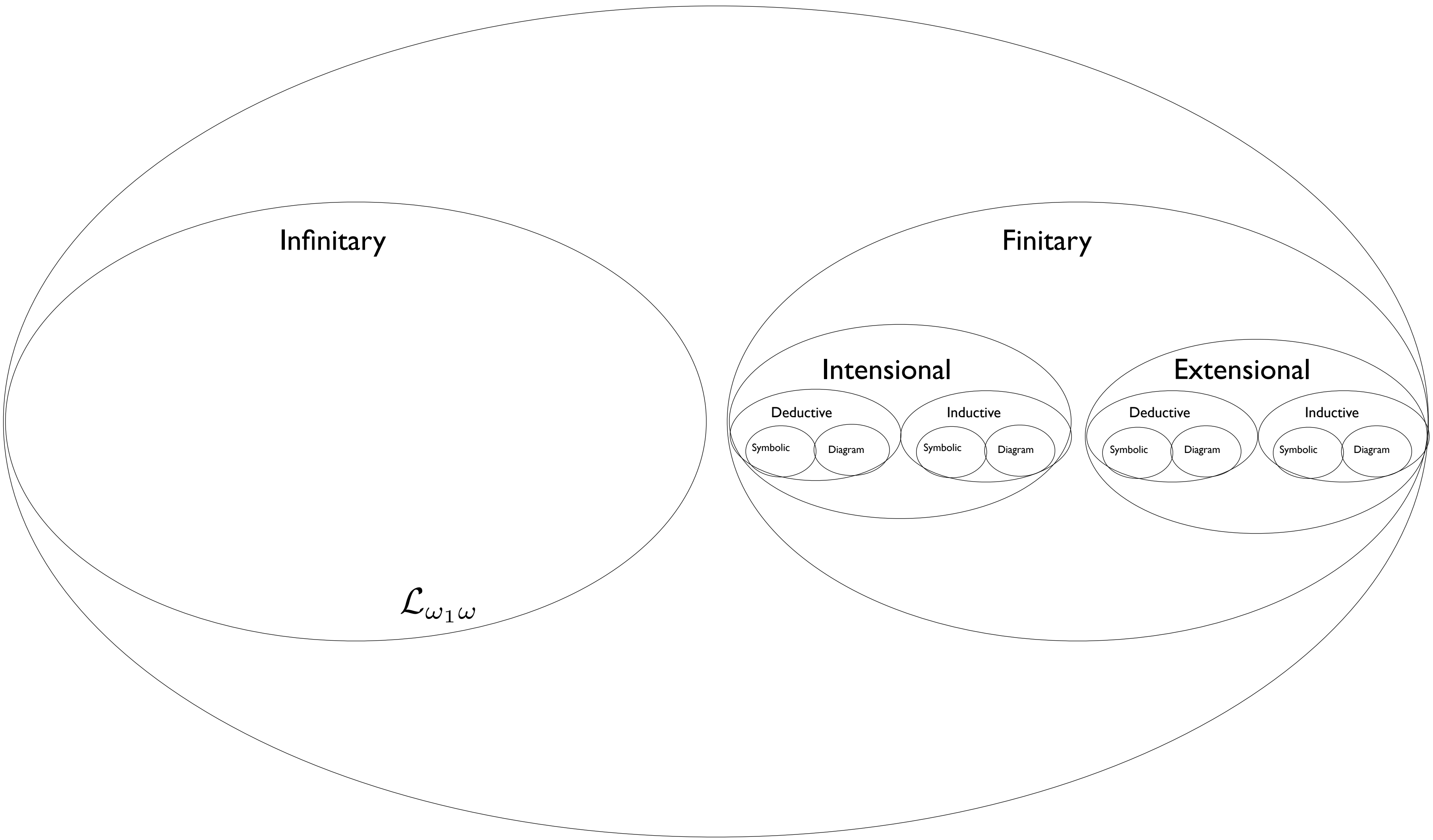
...

Logical Calculi

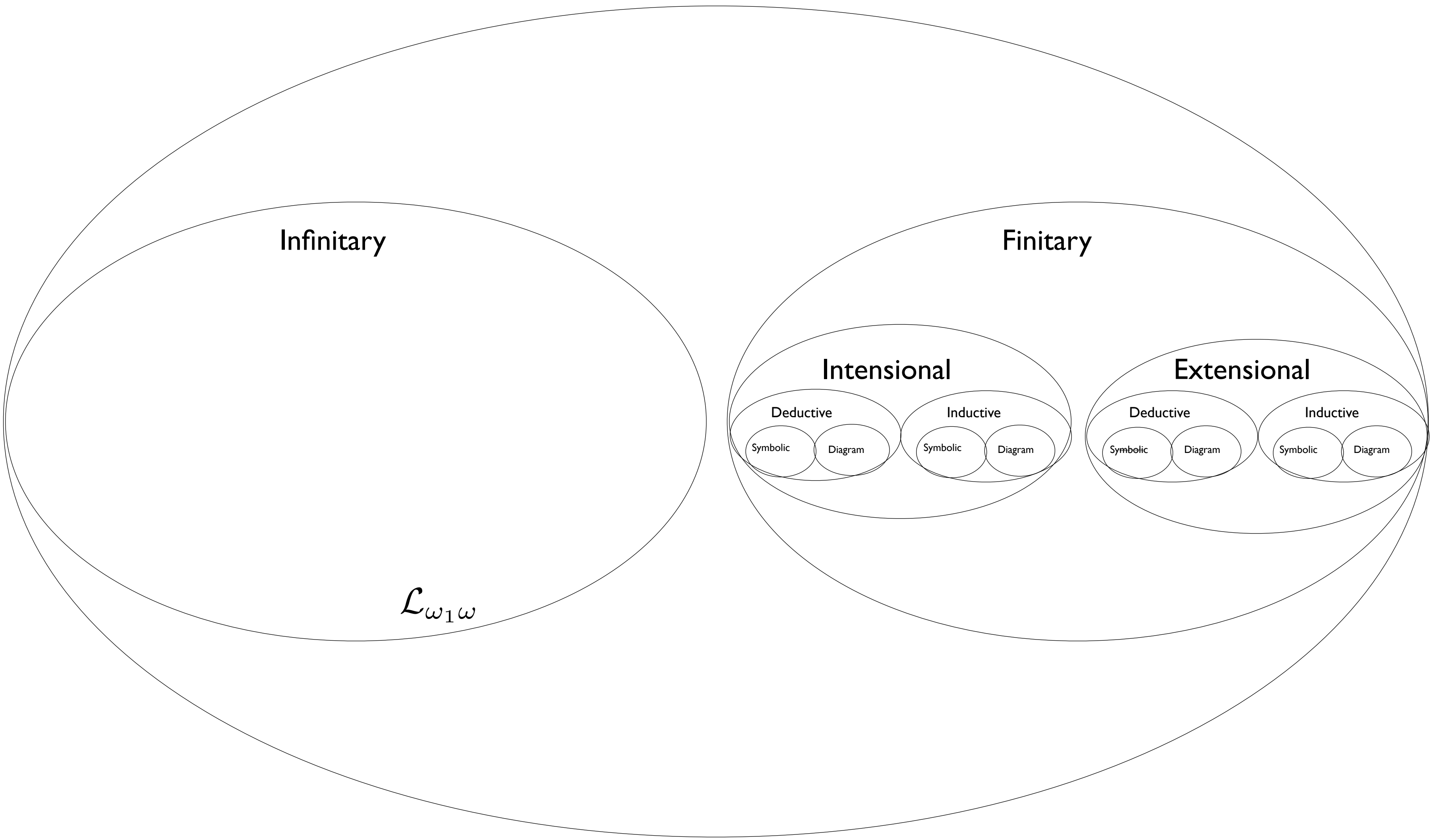


“textbook” AI (≤ FOL)

Logical Calculi

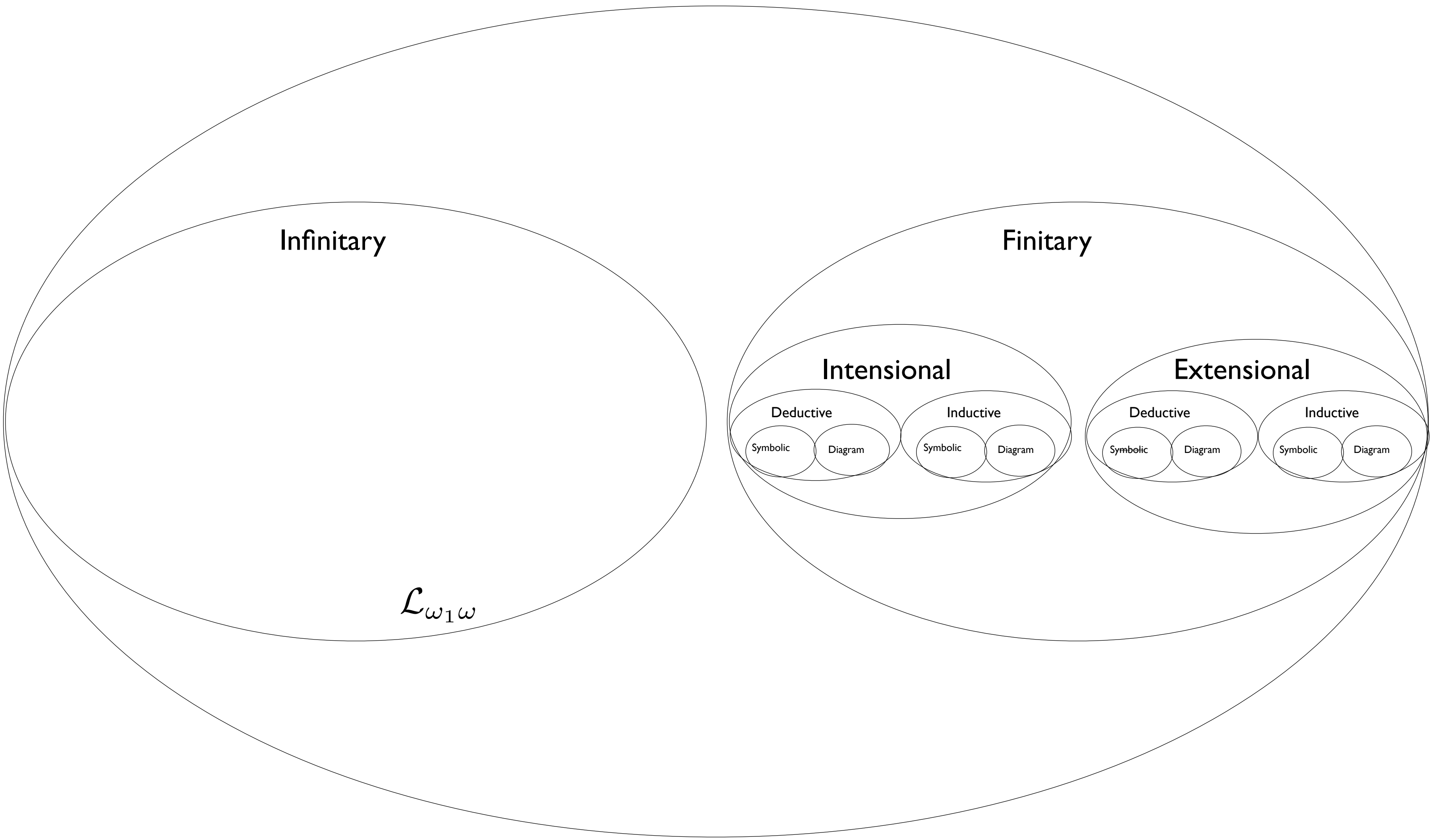


Logical Calculi

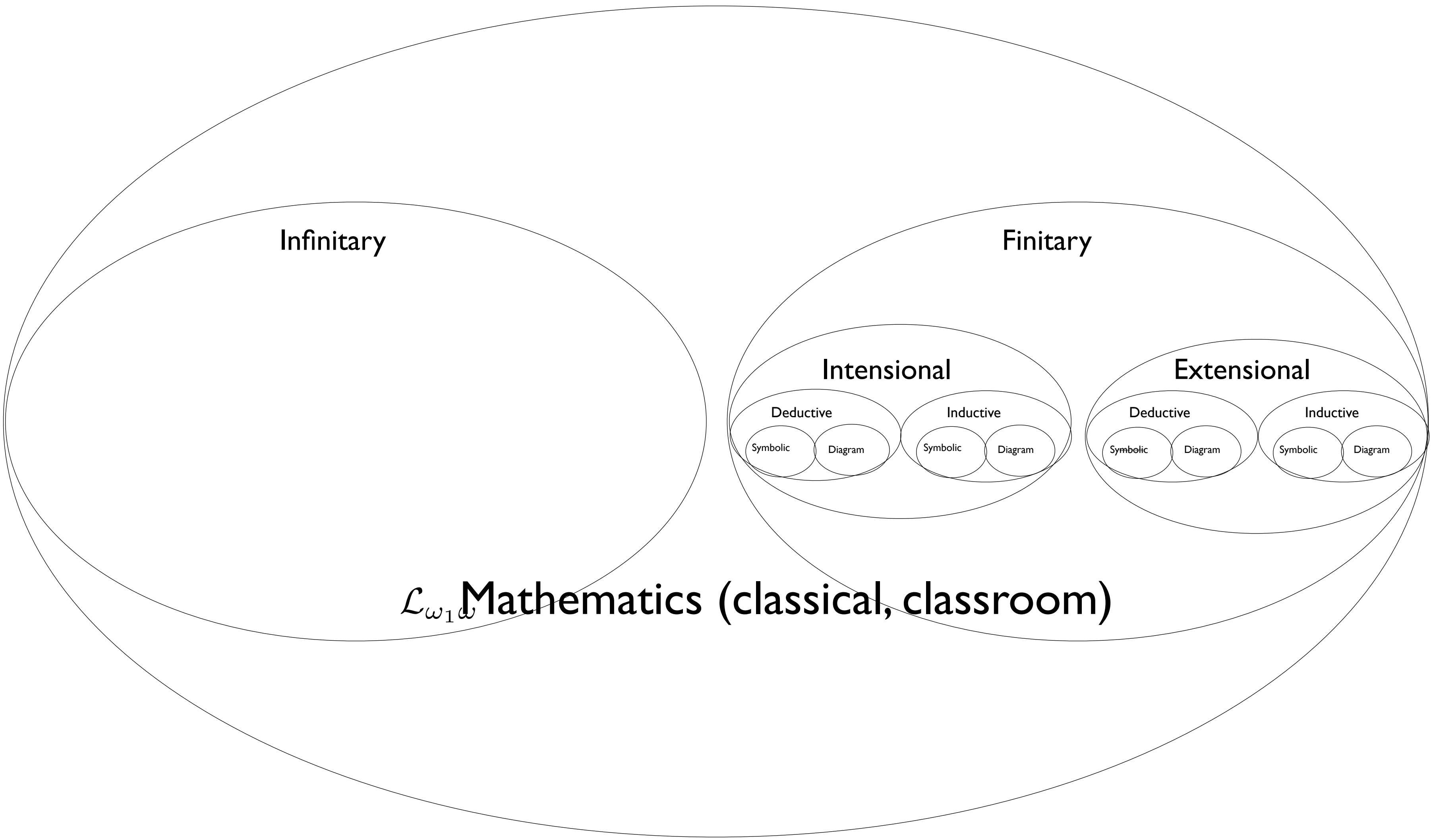


Mathematics (classical, classroom)

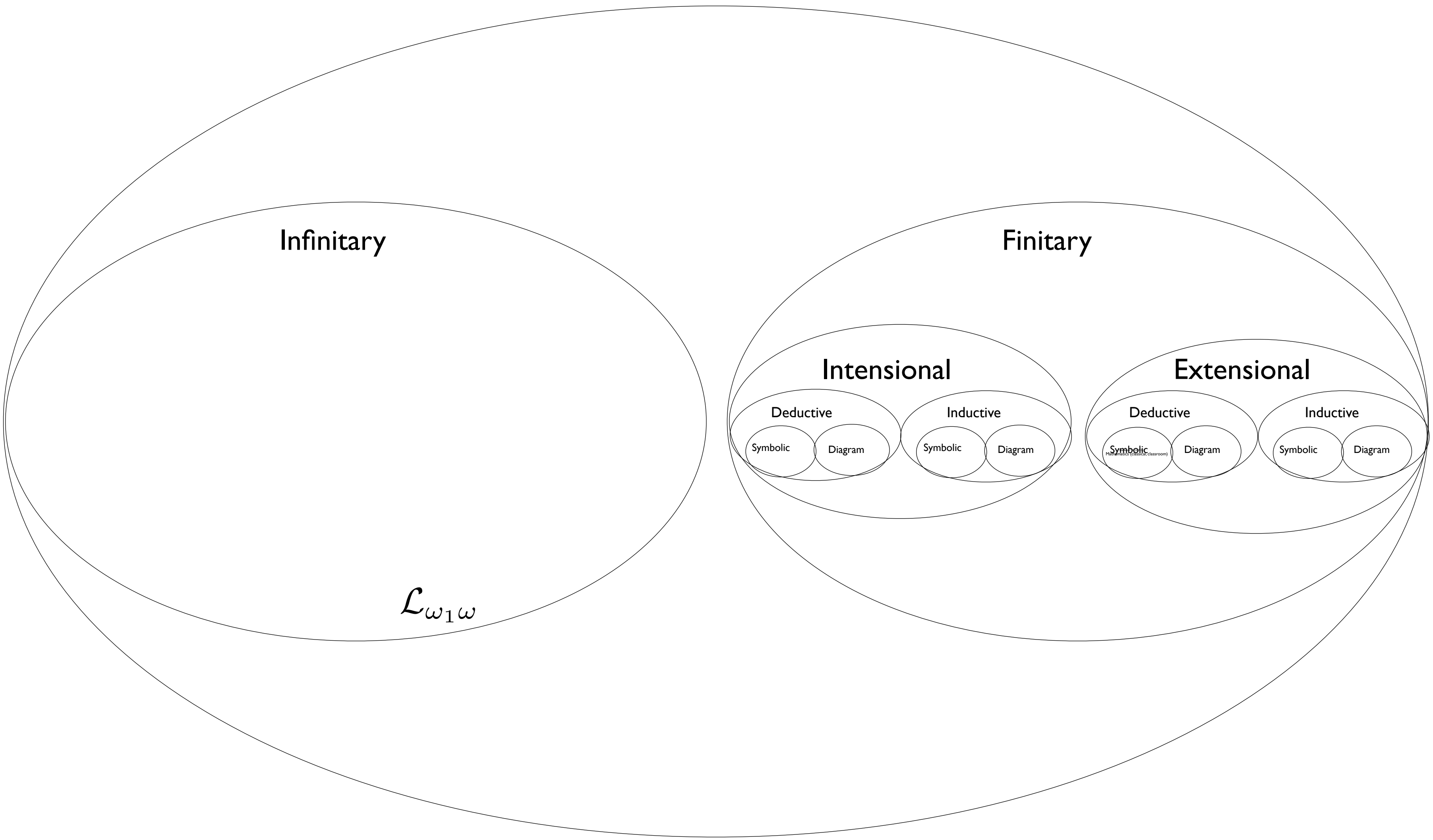
Logical Calculi



Logical Calculi

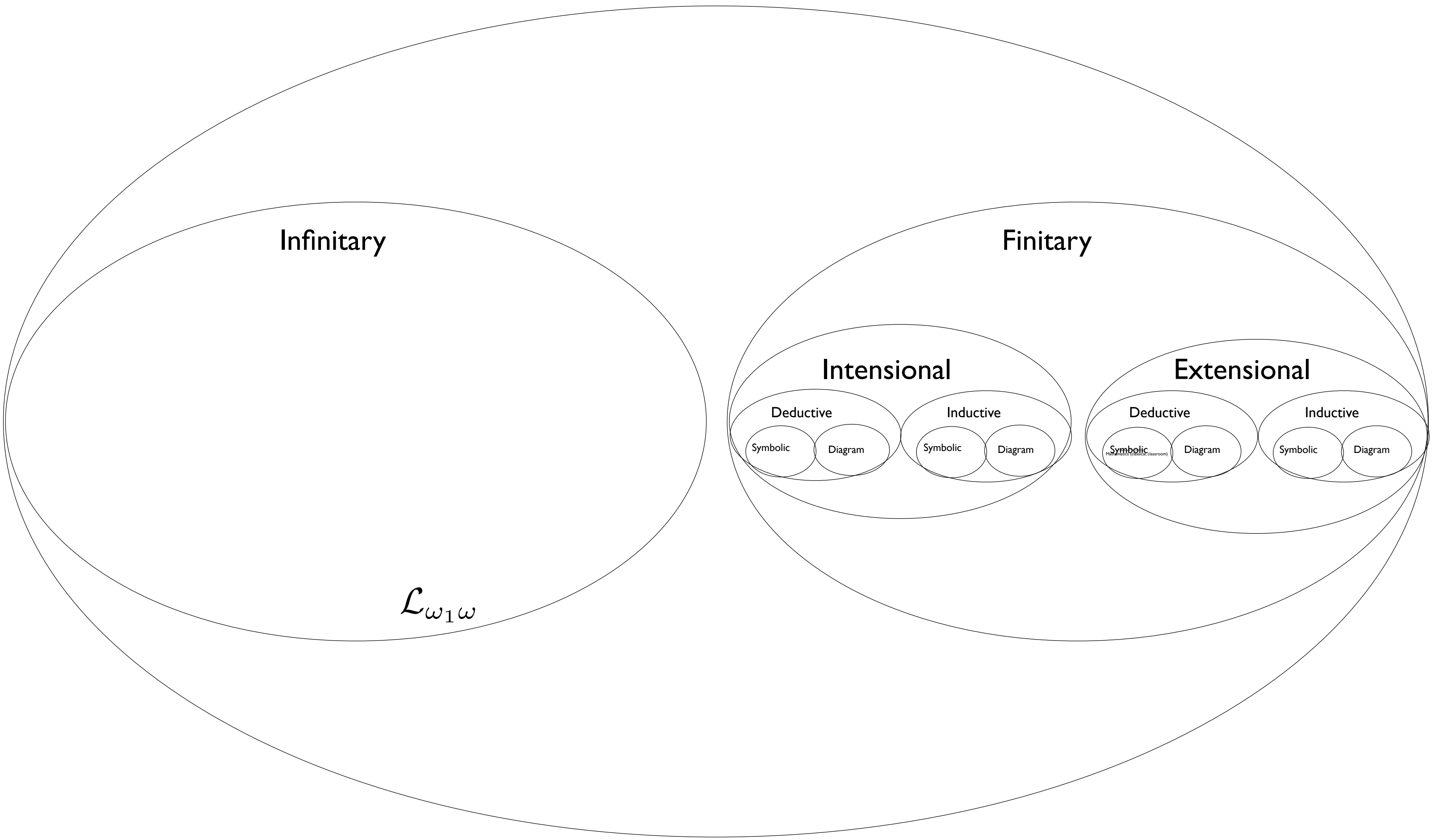


Logical Calculi

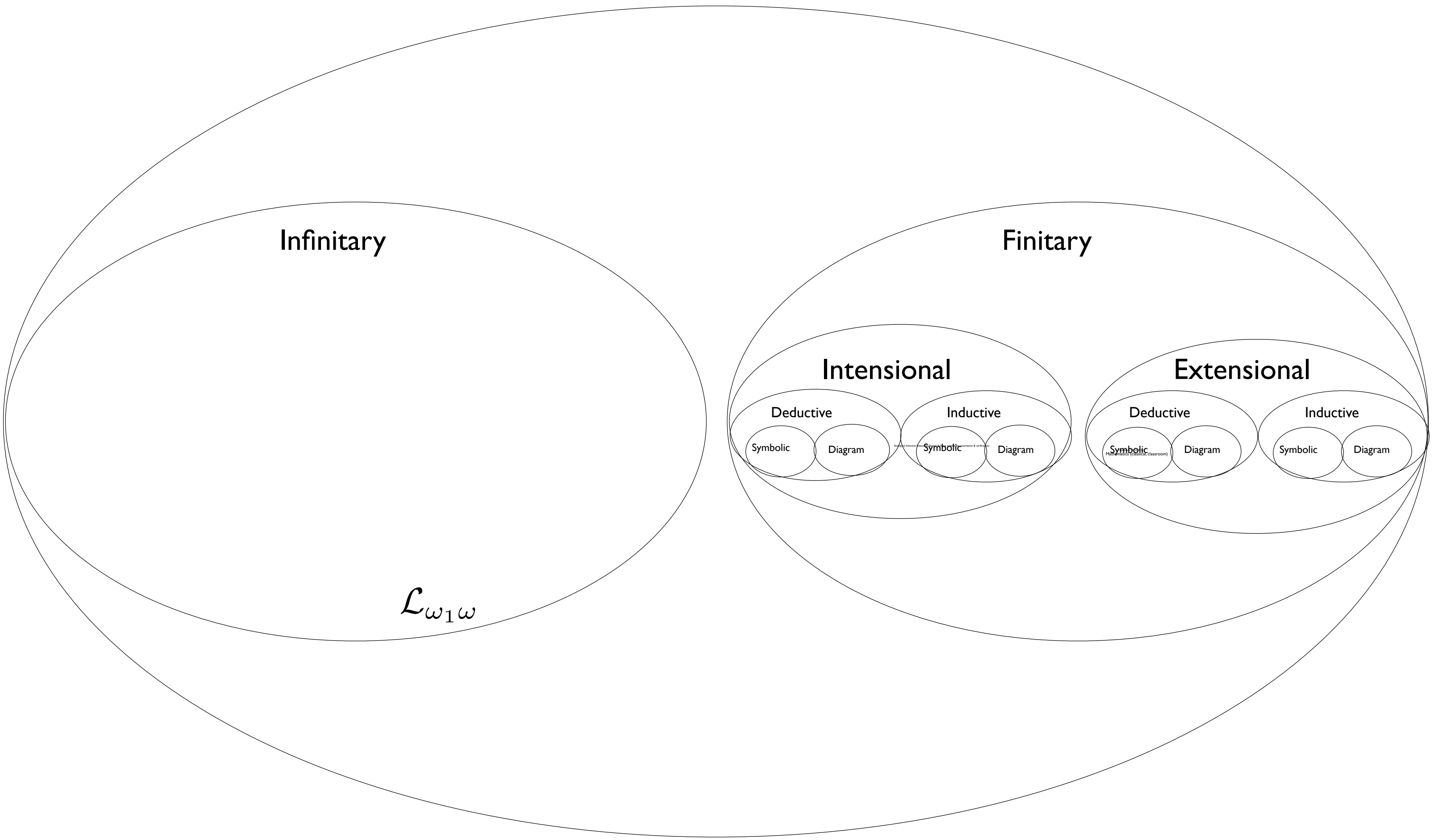


Analogical Inductive Reasoning (formal & hence suitable for automation & verification)

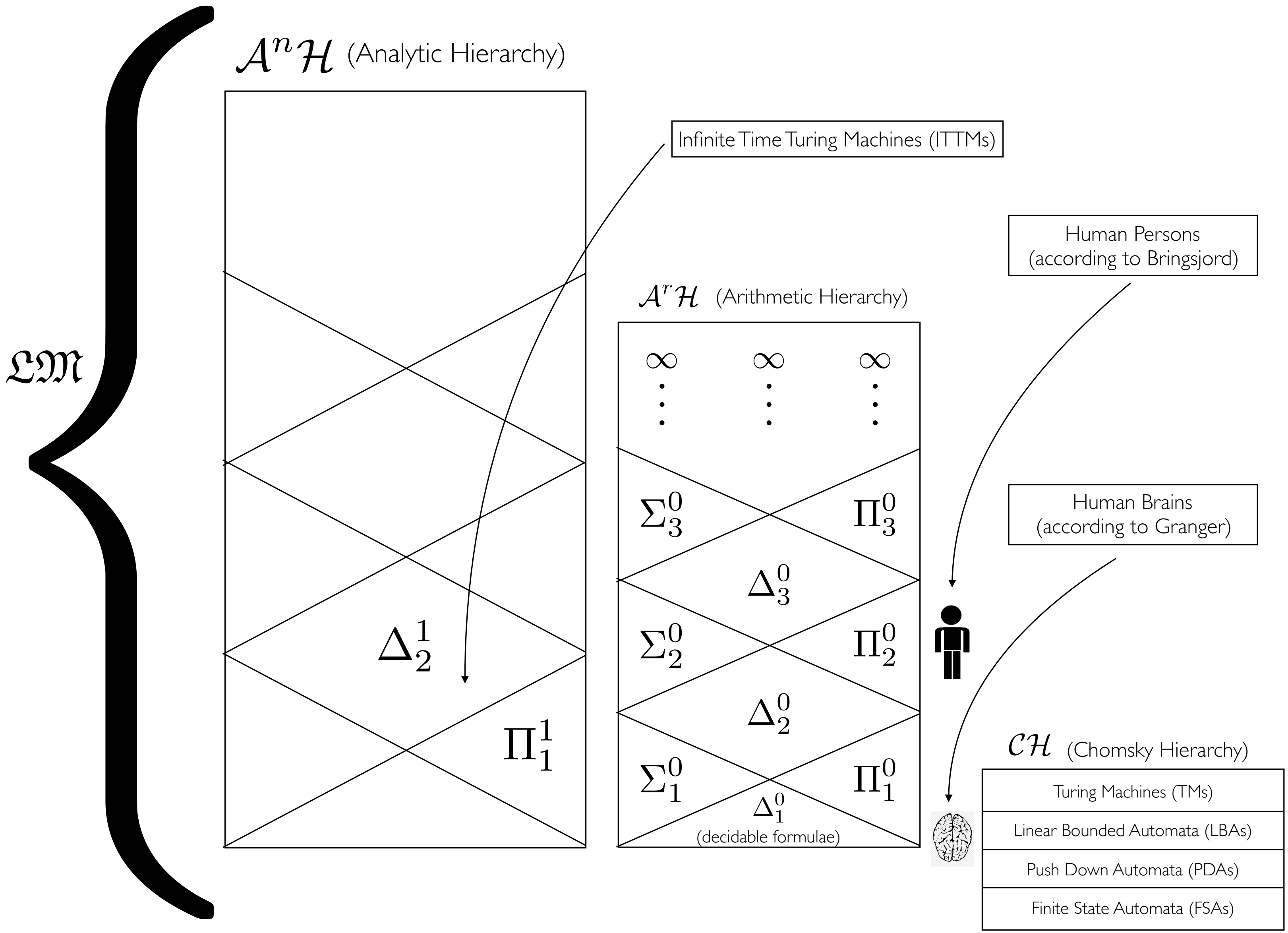
Logical Calculi



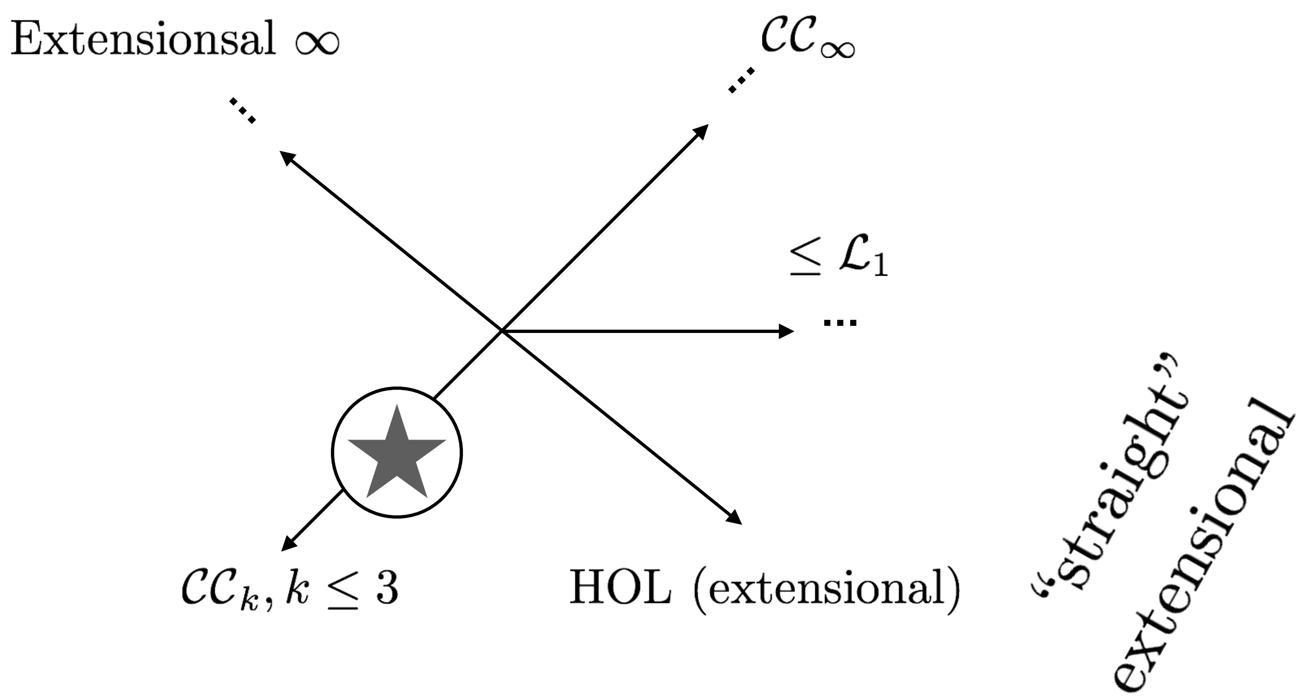
Logical Calculi



Four Hierarchies



Space of Some Logical Calculi in Five Dimensions



4. Ingredients for Making a PGLP Program

...

On the Anatomy of a PGLP Program

On the Anatomy of a PGLP Program

Linguistics

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

L^{μ}_{2}	meta-level$_2$ language	$(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$
L^{μ}_{1}	meta-level$_1$ language	$\exists x \ \mathrm{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{U} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

On the Anatomy of a PGLP Program

Linguistics

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

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.)

On the Anatomy of a PGLP Program

Linguistics

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

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

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

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

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

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??).

$\mathscr{L}$

On the Anatomy of a PGLP Program

Linguistics

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

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

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

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

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	$(\{\phi\} \vdash \psi \wedge \{\psi\} \vdash \delta) \vdash_{\mu_2} \{\phi\} \vdash \delta$
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??).

(Linguistics, Inference and Semantics together constitute $\mathscr{L}$.)

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

On the Anatomy of a PGLP Program

Linguistics

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

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

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

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??).

$\mathscr{L}$

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

program

query

degree of "confidence"

proof(s)

argument(s)

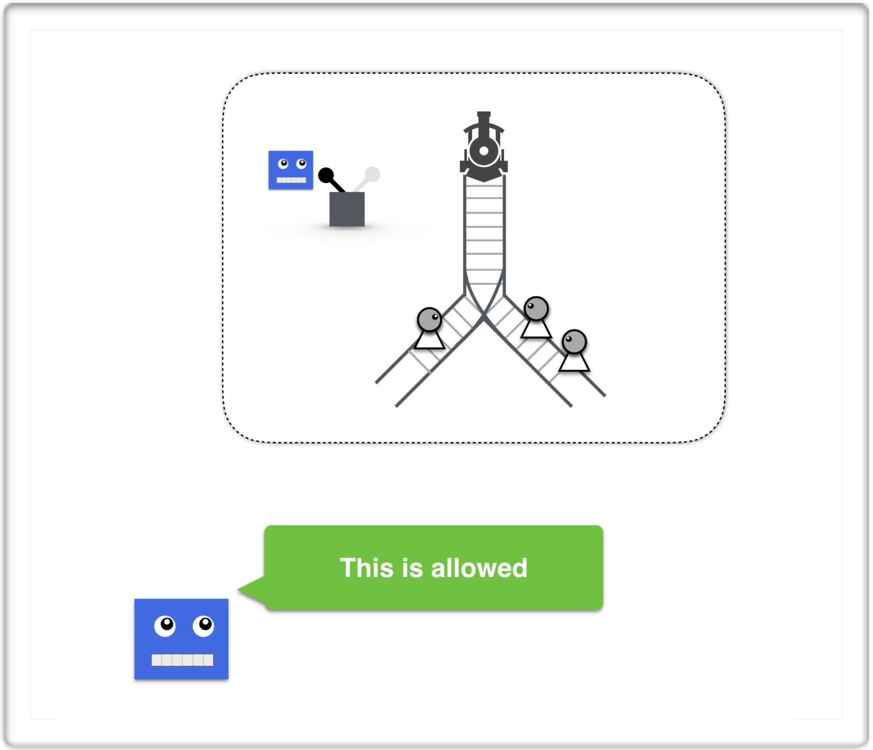
$$\frac{\begin{array}{cc} \mathbb{P} & L \\ \mathfrak{q} & L \end{array}}{\begin{array}{l} \mathbb{R} \ : \ \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow \langle \mathtt{Y}|\mathtt{N}|\mathtt{U}, \delta, \pi_{(s)}|\alpha_{(s)} \rangle \\ \qquad \mathbb{C} \ : \ \pi_{(s)}|\alpha_{(s)} \longrightarrow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

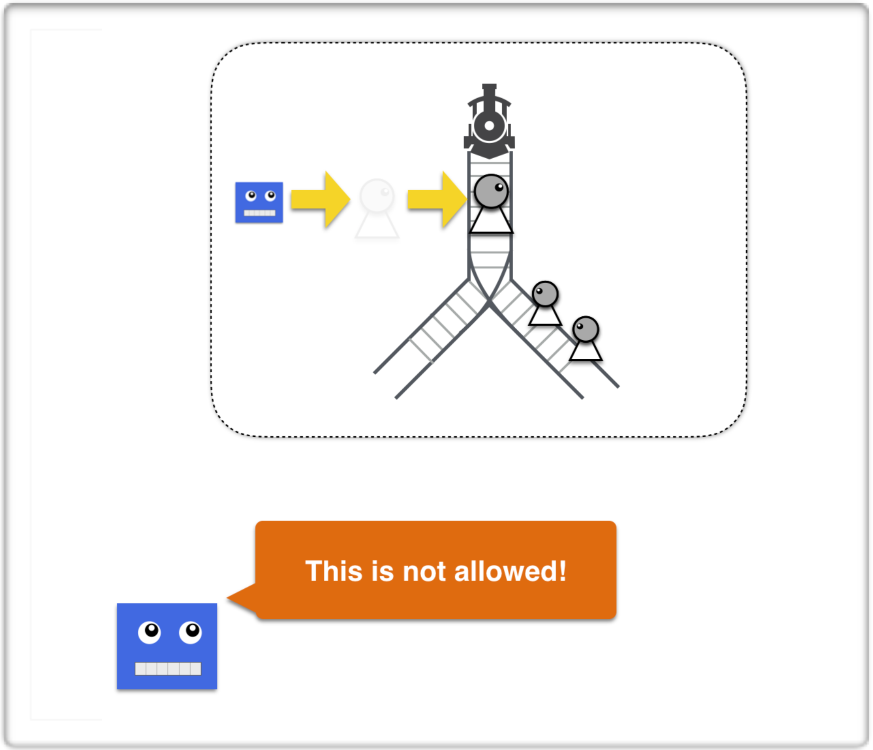
reasoner

checker

5. Example 1: Ethical Control via a Program Based on *DCEC** + ShadowProver …

A Trolley Dilemma





Doctrine of Double Effect $\mathcal{DDE}$

Doctrine of Double Effect $\mathcal{DDE}$

- A long-studied (!) ethical principle that adjudicates certain class of moral dilemmas.

Doctrine of Double Effect $\mathcal{DDE}$

- A long-studied (!) ethical principle that adjudicates certain class of moral dilemmas.

- The Doctrine of Double Effect "comes to the rescue" and prescribes what to do in some moral dilemmas.

Doctrine of Double Effect $\mathcal{DDE}$

- A long-studied (!) ethical principle that adjudicates certain class of moral dilemmas.
- The Doctrine of Double Effect "comes to the rescue" and prescribes what to do in some moral dilemmas.
- E.g. the "original" moral dilemma: Can you defend your own life by ending the lives of (perhaps many) attackers?

Doctrine of Double Effect $\mathcal{DDE}$

- A long-studied (!) ethical principle that adjudicates certain class of moral dilemmas.
- The Doctrine of Double Effect "comes to the rescue" and prescribes what to do in some moral dilemmas.
- E.g. the "original" moral dilemma: Can you defend your own life by ending the lives of (perhaps many) attackers?

Informal Version of DDE

C_1 the action is not forbidden (where we assume an ethical hierarchy such as the one given by Bringsjord [2017], and require that the action be neutral or above neutral in such a hierarchy);

C_2 the net utility or goodness of the action is greater than some positive amount γ;

C_{3a} the agent performing the action intends only the good effects;

C_{3b} the agent does not intend any of the bad effects;

C_4 the bad effects are not used as a means to obtain the good effects; and

C_5 if there are bad effects, the agent would rather the situation be different and the agent not have to perform the action. That is, the action is unavoidable.

Informal Version of DDE

C_1 the action is not forbidden (where we assume an ethical hierarchy such as the one given by Bringsjord [2017], and require that the action be neutral or above neutral in such a hierarchy);

C_2 the net utility or goodness of the action is greater than some positive amount γ;

C_{3a} the agent performing the action intends only the good effects;

C_{3b} the agent does not intend any of the bad effects;

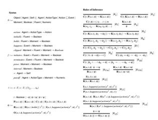
C_4 the bad effects are not used as a means to obtain the good effects; and

C_5 if there are bad effects, the agent would rather the situation be different and the agent not have to perform the action. That is, the action is unavoidable.

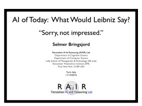
“Univer
sal

Univers
al
Cognitiv

$\mathcal{DCEC}^*$



-


AI of Today: What Would Leibniz Say?

“Sorry, not impressed.”

Selmer Bringsjord

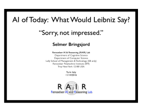
R A I R
Rensselaer AI and Reasoning Lab

-

.5 centuries

“Univer
sal

Univers
al
Cognitiv



AI of Today: What Would Leibniz Say?

“Sorry, not impressed.”

Selmer Bringsjord

RAIR

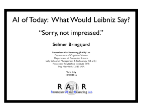
Rensselaer AI and Reasoning Lab

.5 centuries

"Univer
sal

Univers
al
Cognitiv

CC



AI of Today: What Would Leibniz Say?

"Sorry, not impressed."

Selmer Bringsjord

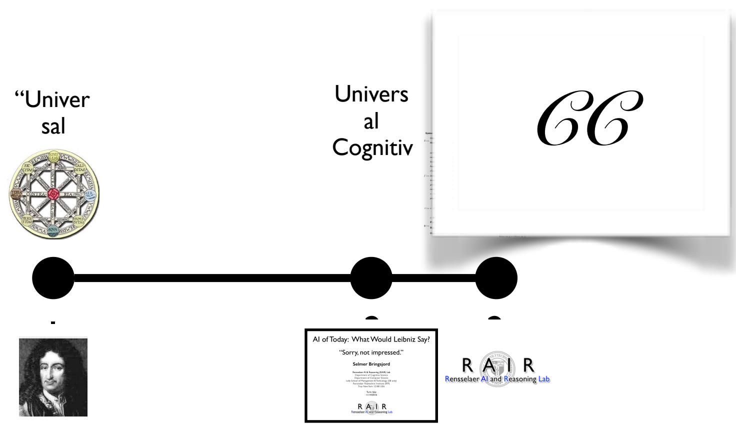
Rensselaer AI & Reasoning (RAIR) Lab
Department of Cognitive Science
Department of Computer Science
Lally School of Management & Technology (SB only)
Rensselaer Polytechnic Institute (RPI)
Troy New York 12180 USA

Turin Italy
11/14/2016

RAIR
Rensselaer AI and Reasoning Lab

RAIR
Rensselaer AI and Reasoning Lab

.5 centuries



.5 centuries

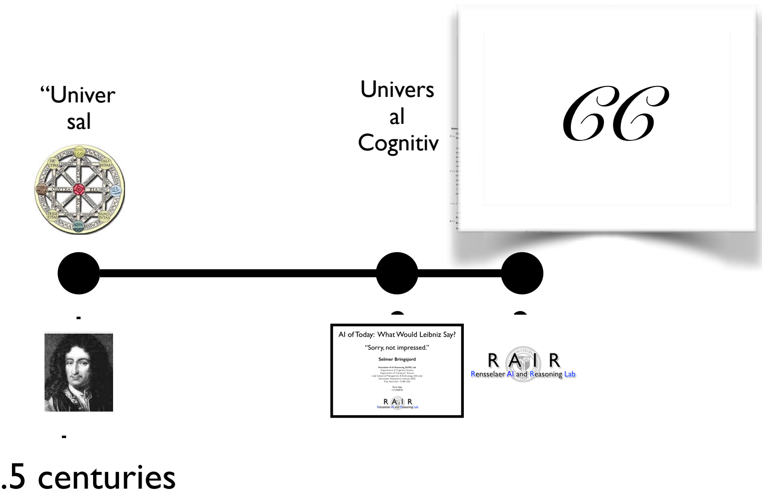
Syntax

$$S ::= \text{Object} \mid \text{Agent} \mid \text{ActionType} \mid \text{Action} \sqsubseteq \text{Event} \mid \text{Moment} \mid \text{Formula} \mid \text{Fluent}$$

$$f ::= \begin{cases} action : \text{Agent} \times \text{ActionType} \rightarrow \text{Action} \\ initially : \text{Fluent} \rightarrow \text{Formula} \\ Holds : \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ happens : \text{Event} \times \text{Moment} \rightarrow \text{Formula} \\ clipped : \text{Moment} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ initiates : \text{Event} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ terminates : \text{Event} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ prior : \text{Moment} \times \text{Moment} \rightarrow \text{Formula} \end{cases}$$

$$t ::= x : S \mid c : S \mid f(t_1, \ldots, t_n)$$

$$\phi ::= \begin{cases} t : \text{Formula} \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathbf{P}(a,t,\phi) \mid \mathbf{K}(a,t,\phi) \mid \mathbf{C}(t,\phi) \\ \mathbf{S}(a,b,t,\phi) \mid \mathbf{S}(a,t,\phi) \mid \mathbf{B}(a,t,\phi) \mid \mathbf{D}(a,t,Holds(f,t')) \mid \mathbf{I}(a,t,\phi) \\ \mathbf{O}(a,t,\phi,(\neg)happens(action(a^*,\alpha),t')) \end{cases}$$



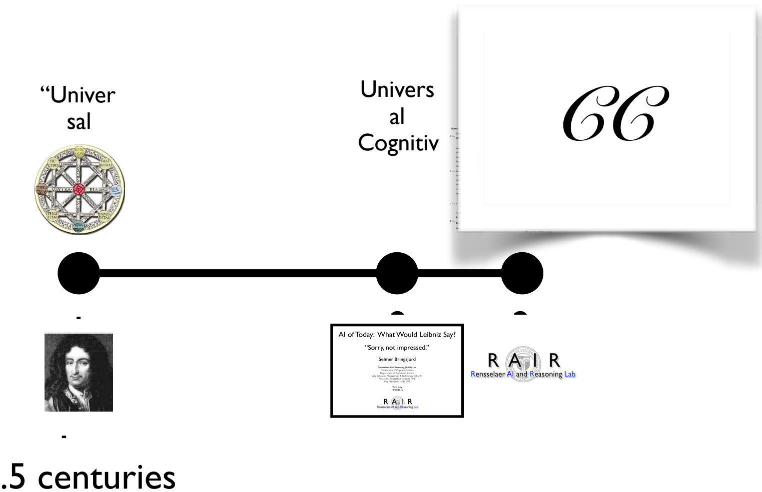
Syntax

$$S ::= \text{Object} \mid \text{Agent} \mid \text{ActionType} \mid \text{Action} \sqsubseteq \text{Event} \mid \text{Moment} \mid \text{Formula} \mid \text{Fluent}$$

$$f ::= \begin{cases} \mathit{action} : \text{Agent} \times \text{ActionType} \rightarrow \text{Action} \\ \mathit{initially} : \text{Fluent} \rightarrow \text{Formula} \\ \mathit{Holds} : \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{happens} : \text{Event} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{clipped} : \text{Moment} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{initiates} : \text{Event} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{terminates} : \text{Event} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{prior} : \text{Moment} \times \text{Moment} \rightarrow \text{Formula} \end{cases}$$

$$t ::= x : S \mid c : S \mid f(t_1, \ldots, t_n)$$

$$\phi ::= \begin{cases} t : \text{Formula} \mid \neg\phi \mid \phi \land \psi \mid \phi \lor \psi \mid \mathbf{P}(a,t,\phi) \mid \mathbf{K}(a,t,\phi) \mid \mathbf{C}(t,\phi) \\ \mathbf{S}(a,b,t,\phi) \mid \mathbf{S}(a,t,\phi) \mid \mathbf{B}(a,t,\phi) \mid \mathbf{D}(a,t,\mathit{Holds}(f,t')) \mid \mathbf{I}(a,t,\phi) \\ \mathbf{O}(a,t,\phi,(\neg)\mathit{happens}(\mathit{action}(a^*,\alpha),t')) \end{cases}$$



Syntax

$$S ::= \text{Object} \mid \text{Agent} \mid \text{ActionType} \mid \text{Action} \sqsubseteq \text{Event} \mid \text{Moment} \mid \text{Formula} \mid \text{Fluent}$$

$$f ::= \begin{cases} \mathit{action} : \text{Agent} \times \text{ActionType} \rightarrow \text{Action} \\ \mathit{initially} : \text{Fluent} \rightarrow \text{Formula} \\ \mathit{Holds} : \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{happens} : \text{Event} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{clipped} : \text{Moment} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{initiates} : \text{Event} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{terminates} : \text{Event} \times \text{Fluent} \times \text{Moment} \rightarrow \text{Formula} \\ \mathit{prior} : \text{Moment} \times \text{Moment} \rightarrow \text{Formula} \end{cases}$$

$$t ::= x : S \mid c : S \mid f(t_1, \ldots, t_n)$$

$$\phi ::= \begin{cases} t : \text{Formula} \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathbf{P}(a,t,\phi) \mid \mathbf{K}(a,t,\phi) \mid \mathbf{C}(t,\phi) \\ \mathbf{S}(a,b,t,\phi) \mid \mathbf{S}(a,t,\phi) \mid \mathbf{B}(a,t,\phi) \mid \mathbf{D}(a,t,\mathit{Holds}(f,t')) \mid \mathbf{I}(a,t,\phi) \\ \mathbf{O}(a,t,\phi,(\neg)\mathit{happens}(\mathit{action}(a^*,\alpha),t')) \end{cases}$$

Inference Schemata

$$\frac{\mathbf{K}(a,t_1,\Gamma),\ \Gamma \vdash \phi,\ t_1 \leq t_2}{\mathbf{K}(a,t_2,\phi)} [R_{\mathbf{K}}] \qquad \frac{\mathbf{B}(a,t_1,\Gamma),\ \Gamma \vdash \phi,\ t_1 \leq t_2}{\mathbf{B}(a,t_2,\phi)} [R_{\mathbf{B}}]$$

$$\frac{}{\mathbf{C}(t,\mathbf{P}(a,t,\phi) \rightarrow \mathbf{K}(a,t,\phi))} [R_1] \qquad \frac{}{\mathbf{C}(t,\mathbf{K}(a,t,\phi) \rightarrow \mathbf{B}(a,t,\phi))} [R_2]$$

$$\frac{\mathbf{C}(t,\phi)\ t \leq t_1 \ldots t \leq t_n}{\mathbf{K}(a_1,t_1,\ldots \mathbf{K}(a_n,t_n,\phi)\ldots)} [R_3] \qquad \frac{\mathbf{K}(a,t,\phi)}{\phi} [R_4]$$

$$\frac{}{\mathbf{C}(t,\mathbf{K}(a,t_1,\phi_1 \rightarrow \phi_2)) \rightarrow \mathbf{K}(a,t_2,\phi_1) \rightarrow \mathbf{K}(a,t_3,\phi_2)} [R_5]$$

$$\frac{}{\mathbf{C}(t,\mathbf{B}(a,t_1,\phi_1 \rightarrow \phi_2)) \rightarrow \mathbf{B}(a,t_2,\phi_1) \rightarrow \mathbf{B}(a,t_3,\phi_2)} [R_6]$$

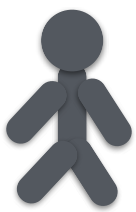
$$\frac{}{\mathbf{C}(t,\mathbf{C}(t_1,\phi_1 \rightarrow \phi_2)) \rightarrow \mathbf{C}(t_2,\phi_1) \rightarrow \mathbf{C}(t_3,\phi_2)} [R_7]$$

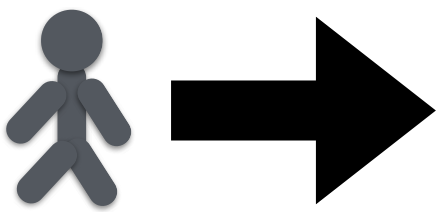
$$\frac{}{\mathbf{C}(t,\forall x.\ \phi \rightarrow \phi[x \mapsto t])} [R_8] \qquad \frac{}{\mathbf{C}(t,\phi_1 \leftrightarrow \phi_2 \rightarrow \neg\phi_2 \rightarrow \neg\phi_1)} [R_9]$$

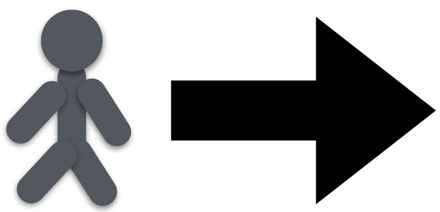
$$\frac{}{\mathbf{C}(t,[\phi_1 \wedge \ldots \wedge \phi_n \rightarrow \phi] \rightarrow [\phi_1 \rightarrow \ldots \rightarrow \phi_n \rightarrow \psi])} [R_{10}]$$

$$\frac{\mathbf{S}(s,h,t,\phi)}{\mathbf{B}(h,t,\mathbf{B}(s,t,\phi))} [R_{12}] \qquad \frac{\mathbf{I}(a,t,\mathit{happens}(\mathit{action}(a^*,\alpha),t'))}{\mathbf{P}(a,t,\mathit{happens}(\mathit{action}(a^*,\alpha),t))} [R_{13}]$$

$$\frac{\mathbf{B}(a,t,\phi) \quad \mathbf{B}(a,t,\mathbf{O}(a,t,\phi,\chi)) \quad \mathbf{O}(a,t,\phi,\chi)}{\mathbf{K}(a,t,\mathbf{I}(a,t,\chi))} [R_{14}]$$







Formal Conditions for $\mathcal{DDE}$

$\mathbf{F_1}$ α carried out at t is not forbidden. That is:

$$\Gamma \nvdash \neg \mathbf{O}\Big(a, t, \sigma, \neg happens\big(action(a,\alpha), t\big)\Big)$$

$\mathbf{F_2}$ The net utility is greater than a given positive real γ:

$$\Gamma \vdash \sum_{y=t+1}^{H} \left(\sum_{f \in \alpha_I^{a,t}} \mu(f,y) - \sum_{f \in \alpha_T^{a,t}} \mu(f,y) \right) > \gamma$$

$\mathbf{F_{3a}}$ The agent a intends at least one good effect. ($\mathbf{F_2}$ should still hold after removing all other good effects.) There is at least one fluent f_g in $\alpha_I^{a,t}$ with $\mu(f_g, y) > 0$, or f_b in $\alpha_T^{a,t}$ with $\mu(f_b, y) < 0$, and some y with $t < y \leq H$ such that the following holds:

$$\Gamma \vdash \left(\begin{array}{c} \exists f_g \in \alpha_I^{a,t}\ \mathbf{I}\Big(a, t, Holds\big(f_g, y\big)\Big) \\ \vee \\ \exists f_b \in \alpha_T^{a,t}\ \mathbf{I}\Big(a, t, \neg Holds\big(f_b, y\big)\Big) \end{array} \right)$$

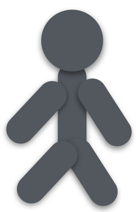
$\mathbf{F_{3b}}$ The agent a does not intend any bad effect. For all fluents f_b in $\alpha_I^{a,t}$ with $\mu(f_b, y) < 0$, or f_g in $\alpha_T^{a,t}$ with $\mu(f_g, y) > 0$, and for all y such that $t < y \leq H$ the following holds:

$$\Gamma \nvdash \mathbf{I}\Big(a, t, Holds\big(f_b, y\big)\Big) \text{ and}$$

$$\Gamma \nvdash \mathbf{I}\Big(a, t, \neg Holds\big(f_g, y\big)\Big)$$

$\mathbf{F_4}$ The harmful effects don't cause the good effects. Four permutations, paralleling the definition of $\rhd$ above, hold here. One such permutation is shown below. For any bad fluent f_b holding at t_1, and any good fluent f_g holding at some t_2, such that $t < t_1, t_2 \leq H$, the following holds:

$$\Gamma \vdash \neg \rhd \Big(Holds\big(f_b, t_1\big), Holds\big(f_g, t_2\big)\Big)$$



Formal Conditions for $\mathcal{DDE}$

F_1 α carried out at t is not forbidden. That is:

$$\Gamma \nvdash \neg \mathbf{O}\Big(a, t, \sigma, \neg happens\big(action(a,\alpha), t\big)\Big)$$

F_2 The net utility is greater than a given positive real γ:

$$\Gamma \vdash \sum_{y=t+1}^{H} \left(\sum_{f \in \alpha_I^{a,t}} \mu(f,y) - \sum_{f \in \alpha_T^{a,t}} \mu(f,y) \right) > \gamma$$

F_{3a} The agent a intends at least one good effect. (**F_2** should still hold after removing all other good effects.) There is at least one fluent f_g in $\alpha_I^{a,t}$ with $\mu(f_g, y) > 0$, or f_b in $\alpha_T^{a,t}$ with $\mu(f_b, y) < 0$, and some y with $t < y \leq H$ such that the following holds:

$$\Gamma \vdash \begin{pmatrix} \exists f_g \in \alpha_I^{a,t}\ \mathbf{I}\Big(a, t, Holds\big(f_g, y\big)\Big) \\ \vee \\ \exists f_b \in \alpha_T^{a,t}\ \mathbf{I}\Big(a, t, \neg Holds\big(f_b, y\big)\Big) \end{pmatrix}$$

F_{3b} The agent a does not intend any bad effect. For all fluents f_b in $\alpha_I^{a,t}$ with $\mu(f_b, y) < 0$, or f_g in $\alpha_T^{a,t}$ with $\mu(f_g, y) > 0$, and for all y such that $t < y \leq H$ the following holds:

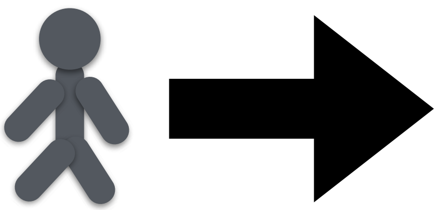
$$\Gamma \nvdash \mathbf{I}\Big(a, t, Holds\big(f_b, y\big)\Big) \text{ and}$$

$$\Gamma \nvdash \mathbf{I}\Big(a, t, \neg Holds\big(f_g, y\big)\Big)$$

F_4 The harmful effects don't cause the good effects. Four permutations, paralleling the definition of $\rhd$ above, hold here. One such permutation is shown below. For any bad fluent f_b holding at t_1, and any good fluent f_g holding at some t_2, such that $t < t_1, t_2 \leq H$, the following holds:

$$\Gamma \vdash \neg \rhd \Big(Holds\big(f_b, t_1\big), Holds\big(f_g, t_2\big)\Big)$$

$\mathbb{P}_{\text{DDE}_1}$ + ShadowProver



Formal Conditions for $\mathcal{DDE}$

$\mathbf{F_1}$ α carried out at t is not forbidden. That is:

$$\Gamma \nvdash \neg \mathbf{O}\Big(a, t, \sigma, \neg happens\big(action(a,\alpha), t\big)\Big)$$

$\mathbf{F_2}$ The net utility is greater than a given positive real γ:

$$\Gamma \vdash \sum_{y=t+1}^{H} \left(\sum_{f \in \alpha_I^{a,t}} \mu(f,y) - \sum_{f \in \alpha_T^{a,t}} \mu(f,y) \right) > \gamma$$

$\mathbf{F_{3a}}$ The agent a intends at least one good effect. ($\mathbf{F_2}$ should still hold after removing all other good effects.) There is at least one fluent f_g in $\alpha_I^{a,t}$ with $\mu(f_g, y) > 0$, or f_b in $\alpha_T^{a,t}$ with $\mu(f_b, y) < 0$, and some y with $t < y \leq H$ such that the following holds:

$$\Gamma \vdash \begin{pmatrix} \exists f_g \in \alpha_I^{a,t}\ \mathbf{I}\Big(a, t, Holds\big(f_g, y\big)\Big) \\ \vee \\ \exists f_b \in \alpha_T^{a,t}\ \mathbf{I}\Big(a, t, \neg Holds\big(f_b, y\big)\Big) \end{pmatrix}$$

$\mathbf{F_{3b}}$ The agent a does not intend any bad effect. For all fluents f_b in $\alpha_I^{a,t}$ with $\mu(f_b, y) < 0$, or f_g in $\alpha_T^{a,t}$ with $\mu(f_g, y) > 0$, and for all y such that $t < y \leq H$ the following holds:

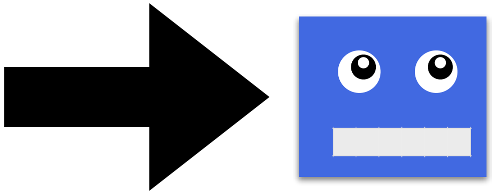
$$\Gamma \nvdash \mathbf{I}\Big(a, t, Holds\big(f_b, y\big)\Big) \text{ and}$$

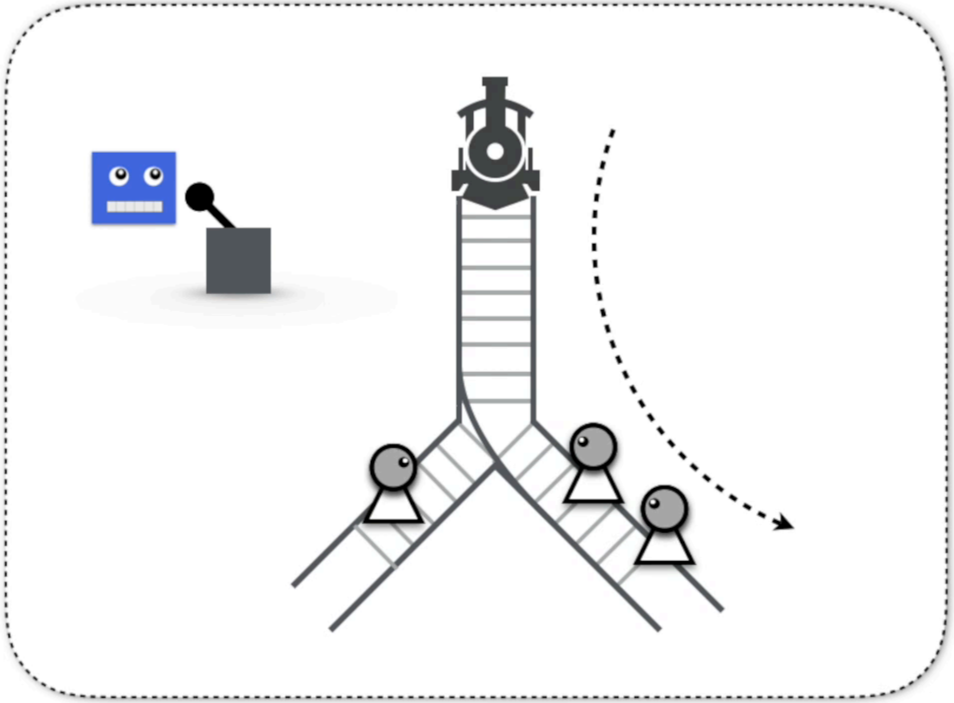
$$\Gamma \nvdash \mathbf{I}\Big(a, t, \neg Holds\big(f_g, y\big)\Big)$$

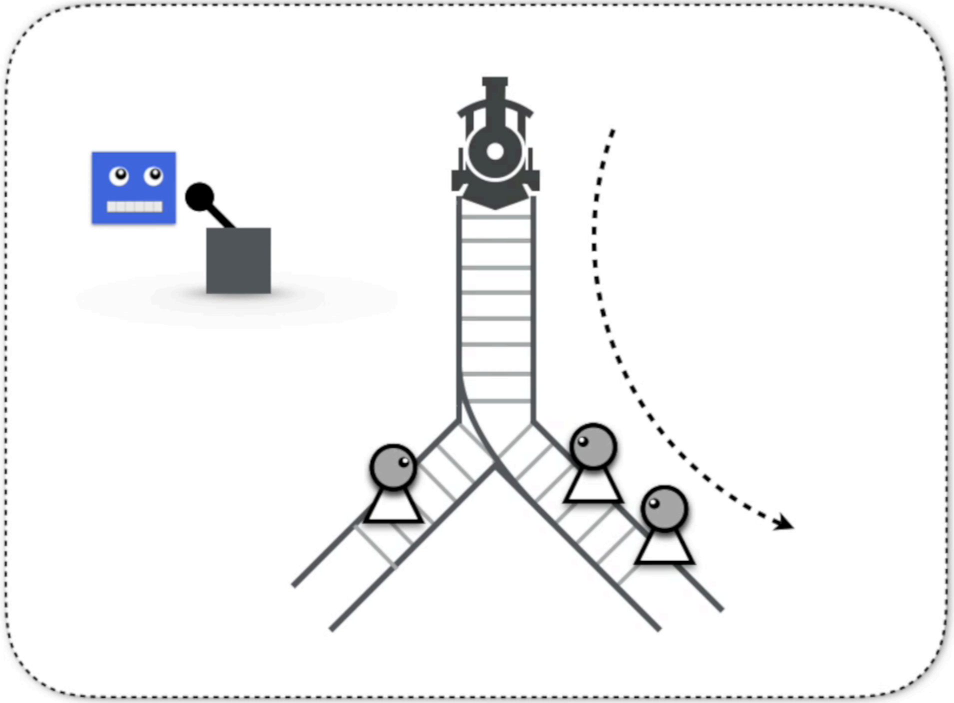
$\mathbf{F_4}$ The harmful effects don't cause the good effects. Four permutations, paralleling the definition of $\rhd$ above, hold here. One such permutation is shown below. For any bad fluent f_b holding at t_1, and any good fluent f_g holding at some t_2, such that $t < t_1, t_2 \leq H$, the following holds:

$$\Gamma \vdash \neg \rhd \Big(Holds\big(f_b, t_1\big), Holds\big(f_g, t_2\big)\Big)$$

$\mathbb{P}_{\mathrm{DDE}_1}$ + ShadowProver

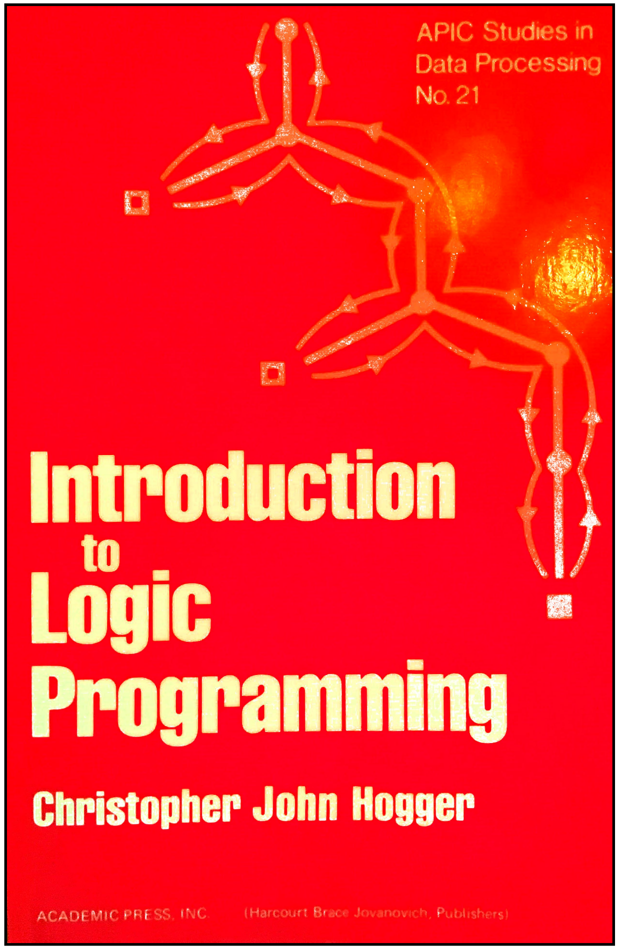




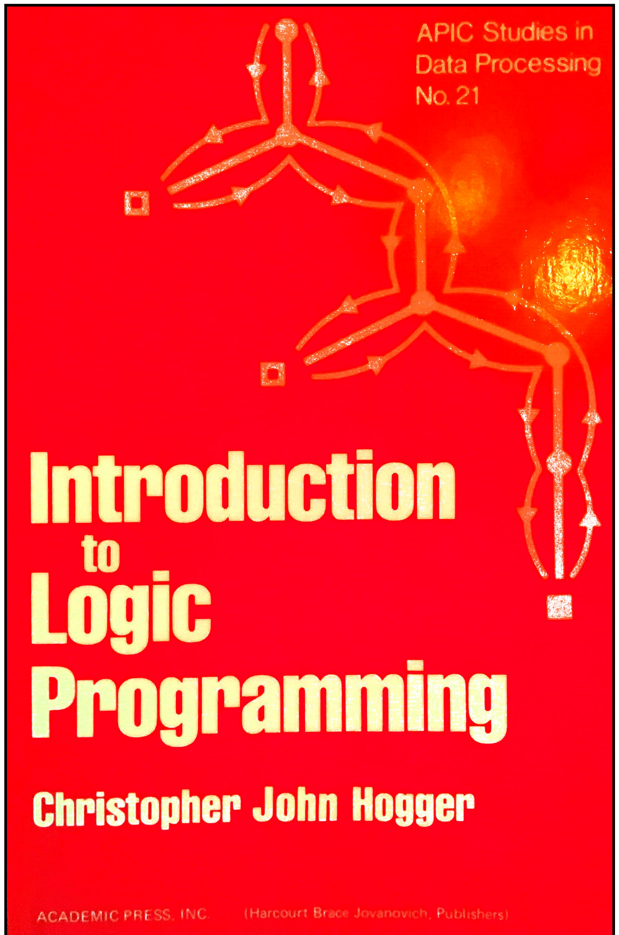


7. Toward Mundane Examples …

7. Toward Mundane Examples …

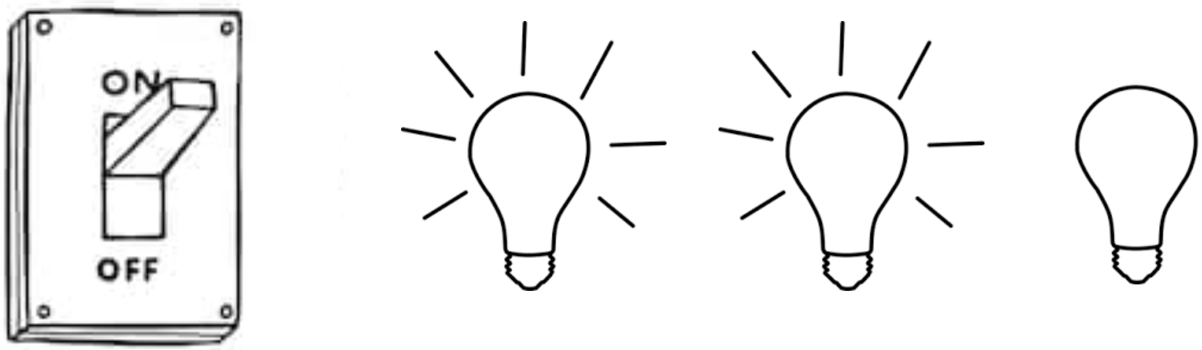


7. Toward Mundane Examples …

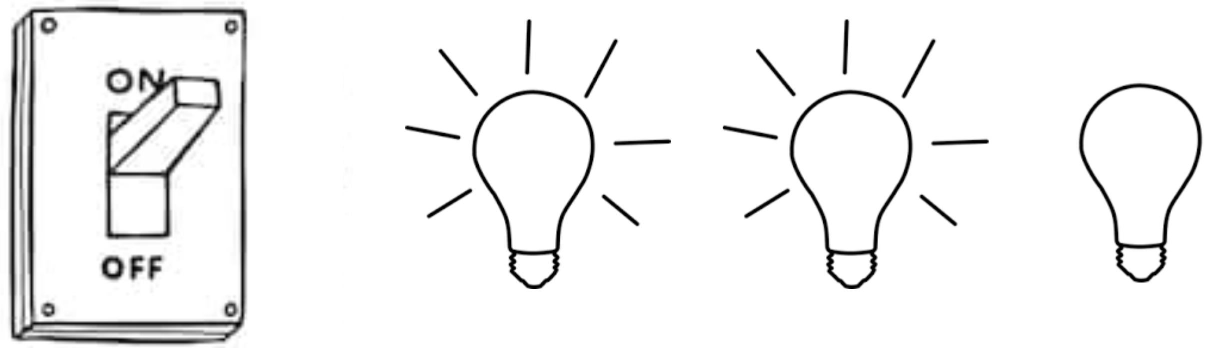


$\mathscr{L}_1$ + ShadowProver

Hogger's "Simple" Switch Example (pp. 2–3)

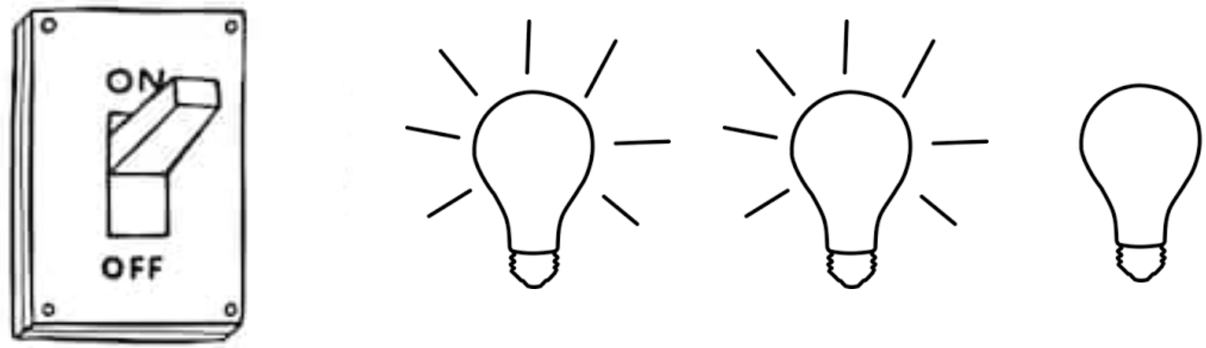


Hogger's "Simple" Switch Example (pp. 2–3)



$$[On(b1, on \wedge On(b2, on \wedge On(b3, \mathit{off})] \rightarrow \mathbf{O}(SwitchState(on))$$

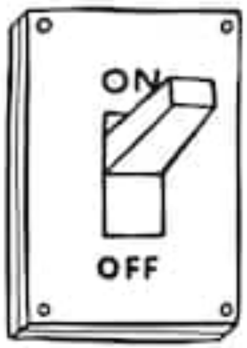
Hogger's "Simple" Switch Example (pp. 2–3)

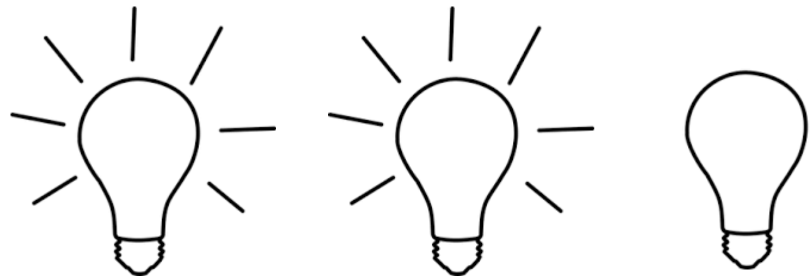


$$[On(b1, on \wedge On(b2, on \wedge On(b3, \mathit{off})] \rightarrow \mathbf{O}(SwitchState(on))$$

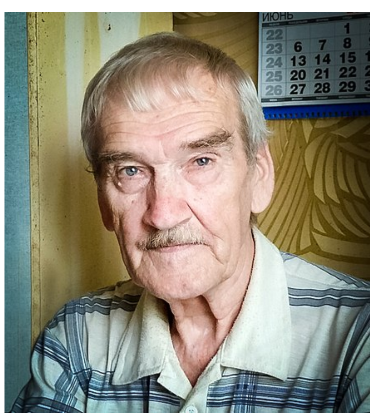
$$\vdots$$

Hogger's "Simple" Switch Example (pp. 2–3)



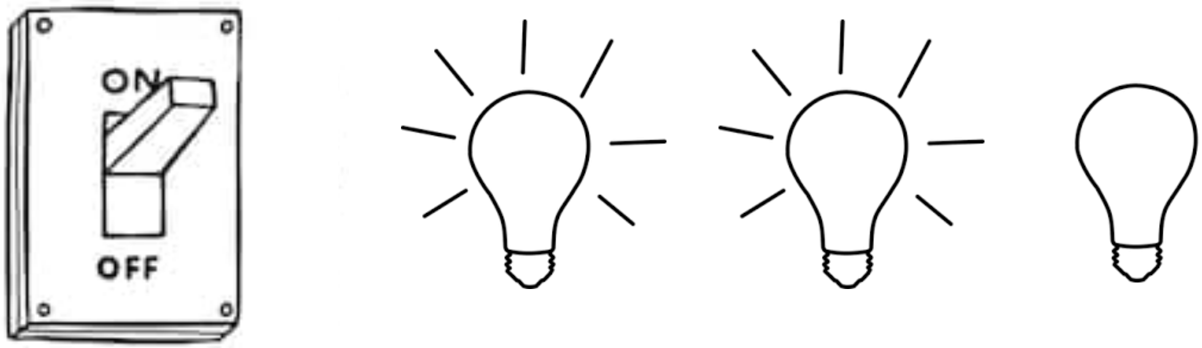


$$[On(b1, on \wedge On(b2, on \wedge On(b3, \mathit{off})] \rightarrow \mathbf{O}(SwitchState(on))$$

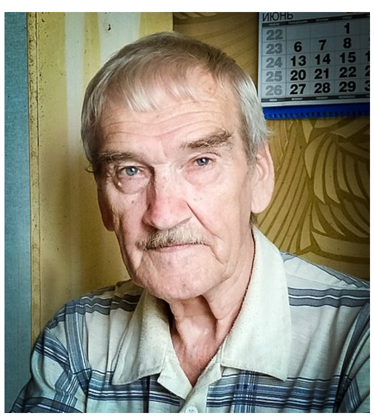


$\vdots$

Hogger's "Simple" Switch Example (pp. 2–3)



$$[On(b1, on \wedge On(b2, on \wedge On(b3, \mathit{off})] \rightarrow \mathbf{O}(\mathit{SwitchState}(on))$$



- What kind of AI/program do we want in place when there *isn't* a human in the loop who can throw in a "wrench"?

Pop Problem

finis