

If AIs Can Really Write Computer Programs, Where — Today & Tomorrow — Does That Leave Human Programmers & Those Who Teach and Employ Them?

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

9/14 & 16 & 17 /2026 @ **RPI**



If AIs Can Really Write Computer Programs, Where — Today & Tomorrow — Does That Leave Human Programmers & Those Who Teach and Employ Them?

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

9/17/2026 @ **RPI**



If AIs Can Really Write Computer Programs,
Where — Today & Tomorrow — Does That Leave
Human Programmers,
and Those Who Teach and Employ Them?

Selmer Bringsjord

version 0910261055NY

September 14, Sage 3101 @ RPI, 7pm
September 16, Carnegie 113 @ RPI, 12pm (abridged version)

The three questions in the title are on the minds of academic administrators overseeing grades K through 16; college-bound 9–12 students; parents, teachers, and professors of the previous; the Educational Testing Service (ETS), purveyors of AP exams to assess college-bound human programmers; those humans currently writing programs for a living in the tech sector; managers of many tech firms (including specifically AI ones); economists concerned with human disemployment by AIs; etc.

I answer all three questions, in a manner that requires of audience members no real programming experience of any kind, just at minimum a vague recollection of the standard high-school logic they took to gain admittance to a university, and — depending upon age — jobs thereafter.

The procedure for answering the trio of questions requires as input answers to four more-fundamental questions, to wit:

- Q1 What are computer programs?
- Q2 What is it to write them?
- Q3 How hard is it to write them?
- Q4 And how maximally intelligent are the AIs that perhaps can write them?

With answers to this quartet obtained, we are left — expressed in barbarically compressed form — here: You all need to up your game, or get out of it.

I end by explaining that in order to control malicious autonomous swarming AIs, such as are already starting to plague various firms (e.g., OpenAI, Hugging Face, etc.), and which are on track to overwhelm businesses, exquisitely *human*-programmed programs, covertly created, are our only hope in the face of chaos that will otherwise come.

Acknowledgment: This talk is in large measure the result of part of 6+ years of research on programming and programming languages, made possible by support from France's [ANR](#) (French National Research Agency), and fueled in no small part by discussion and debate with the brilliant people in the PROGRAMme project. The S Bringsjord-NS Govindarajulu invention of Pure General Logic Programming (PGLP) was in particular born out of this support.

programmers; those humans currently writing programs for a living in the tech sector; managers of many tech firms (including specifically AI ones); economists concerned with human unemployment by AIs; etc.

I answer all three questions, in a manner that requires of audience members no real programming experience of any kind, just at minimum a vague recollection of the standard high-school logic they took to gain admittance to a university, and — depending upon age — jobs thereafter.

The procedure for answering the trio of questions requires as input answers to four more-fundamental questions, to wit:

- Q1 What are computer programs?
- Q2 What is it to write them?
- Q3 How hard is it to write them?
- Q4 And how maximally intelligent are the AIs that perhaps can write them?

With answers to this quartet obtained, we are left — expressed in barbarically compressed form — here: You all need to up your game, or get out of it.

I end by explaining that in order to control malicious autonomous swarming AIs, such as are already starting to plague various firms (e.g., OpenAI, Hugging Face, etc.), and which are on track to overwhelm businesses, exquisitely *human*-programmed programs, covertly created, are our only hope in the face of chaos that will otherwise come.

Acknowledgment: This talk is in large measure the result of part of 6+ years of research on programming and programming languages, made possible by support from France's [ANR](#) (French National Research Agency), and fueled in no small part by discussion and debate with the brilliant people in the PROGRAMme project. The S Bringsjord-NS Govindarajulu invention of Pure General Logic Programming (PGLP) was in particular born out of this support.

programmers; those humans currently writing programs for a living in the tech sector; managers of many tech firms (including specifically AI ones); economists concerned with human unemployment by AIs; etc.

I answer all three questions, in a manner that requires of audience members no real programming experience of any kind, just at minimum a vague recollection of the standard high-school logic they took to gain admittance to a university, and — depending upon age — jobs thereafter.

The procedure for answering the trio of questions requires as input answers to four more-fundamental questions, to wit:

- Q1 What are computer programs?
- Q2 What is it to write them?
- Q3 How hard is it to write them?
- Q4 And how maximally intelligent are the AIs that perhaps can write them?

With answers to this quartet obtained, we are left — expressed in barbarically compressed form — here: You all need to up your game, or get out of it.

I end by explaining that in order to control malicious autonomous swarming AIs, such as are already starting to plague various firms (e.g., OpenAI, Hugging Face, etc.), and which are on track to overwhelm businesses, exquisitely *human*-programmed programs, covertly created, are our only hope in the face of chaos that will otherwise come.

Acknowledgment: This talk is in large measure the result of part of 6+ years of research on programming and programming languages, made possible by support from France's [ANR](#) (French National Research Agency), and fueled in no small part by discussion and debate with the brilliant people in the PROGRAMme project. The S Bringsjord-NS Govindarajulu invention of Pure General Logic Programming (PGLP) was in particular born out of this support.

programmers; those humans currently writing programs for a living in the tech sector; managers of many tech firms (including specifically AI ones); economists concerned with human unemployment by AIs; etc.

I answer all three questions, in a manner that requires of audience members no real programming experience of any kind, just at minimum a vague recollection of the standard high-school logic they took to gain admittance to a university, and — depending upon age — jobs thereafter.

The procedure for answering the trio of questions requires as input answers to four more-fundamental questions, to wit:

Q1 What are computer programs?

Q2 What is it to write them?

Q3 How hard is it to write them?

Q4 And how maximally intelligent are the AIs that perhaps can write them?

With answers to this quartet obtained, we are left — expressed in barbarically compressed form — here: You all need to up your game, or get out of it.

I end by explaining that in order to control malicious autonomous swarming AIs, such as are already starting to plague various firms (e.g., OpenAI, Hugging Face, etc.), and which are on track to overwhelm businesses, exquisitely *human*-programmed programs, covertly created, are our only hope in the face of chaos that will otherwise come.

Acknowledgment: This talk is in large measure the result of part of 6+ years of research on programming and programming languages, made possible by support from France's [ANR](#) (French National Research Agency), and fueled in no small part by discussion and debate with the brilliant people in the PROGRAMme project. The S Bringsjord-NS Govindarajulu invention of Pure General Logic Programming (PGLP) was in particular born out of this support.

programmers; those humans currently writing programs for a living in the tech sector; managers of many tech firms (including specifically AI ones); economists concerned with human unemployment by AIs; etc.

I answer all three questions, in a manner that requires of audience members no real programming experience of any kind, just at minimum a vague recollection of the standard high-school logic they took to gain admittance to a university, and — depending upon age — jobs thereafter.

The procedure for answering the trio of questions requires as input answers to four more-fundamental questions, to wit:

Q1 What are computer programs?

Q2 What is it to write them?

Q3 How hard is it to write them?

Q4 And how maximally intelligent are the AIs that perhaps can write them?

With answers to this quartet obtained, we are left — expressed in barbarically compressed form — here: You all need to up your game, or get out of it.

I end by explaining that in order to control malicious autonomous swarming AIs, such as are already starting to plague various firms (e.g., OpenAI, Hugging Face, etc.), and which are on track to overwhelm businesses, exquisitely *human*-programmed programs, covertly created, are our only hope in the face of chaos that will otherwise come.

Acknowledgment: This talk is in large measure the result of part of 6+ years of research on programming and programming languages, made possible by support from France's [ANR](#) (French National Research Agency), and fueled in no small part by discussion and debate with the brilliant people in the PROGRAMme project. The S Bringsjord-NS Govindarajulu invention of Pure General Logic Programming (PGLP) was in particular born out of this support.

programmers; those humans currently writing programs for a living in the tech sector; managers of many tech firms (including specifically AI ones); economists concerned with human unemployment by AIs; etc.

I answer all three questions, in a manner that requires of audience members no real programming experience of any kind, just at minimum a vague recollection of the standard high-school logic they took to gain admittance to a university, and — depending upon age — jobs thereafter.

The procedure for answering the trio of questions requires as input answers to four more-fundamental questions, to wit:

Q1 What are computer programs?

Q2 What is it to write them?

Q3 How hard is it to write them?

Q4 And how maximally intelligent are the AIs that perhaps can write them?

With answers to this quartet obtained, we are left — expressed in barbarically compressed form — here: You all need to up your game, or get out of it.

I end by explaining that in order to control malicious autonomous swarming AIs, such as are already starting to plague various firms (e.g., OpenAI, Hugging Face, etc.), and which are on track to overwhelm businesses, exquisitely *human*-programmed programs, covertly created, are our only hope in the face of chaos that will otherwise come.

Acknowledgment: This talk is in large measure the result of part of 6+ years of research on programming and programming languages, made possible by support from France's [ANR](#) (French National Research Agency), and fueled in no small part by discussion and debate with the brilliant people in the PROGRAMme project. The S Bringsjord-NS Govindarajulu invention of Pure General Logic Programming (PGLP) was in particular born out of this support.

programmers; those humans currently writing programs for a living in the tech sector; managers of many tech firms (including specifically AI ones); economists concerned with human disemployment by AIs; etc.

I answer all three questions, in a manner that requires of audience members no real programming experience of any kind, just at minimum a vague recollection of the standard high-school logic they took to gain admittance to a university, and — depending upon age — jobs thereafter.

The procedure for answering the trio of questions requires as input answers to four more-fundamental questions, to wit:

Q1 What are computer programs?

Q2 What is it to write them?

Q3 How hard is it to write them?

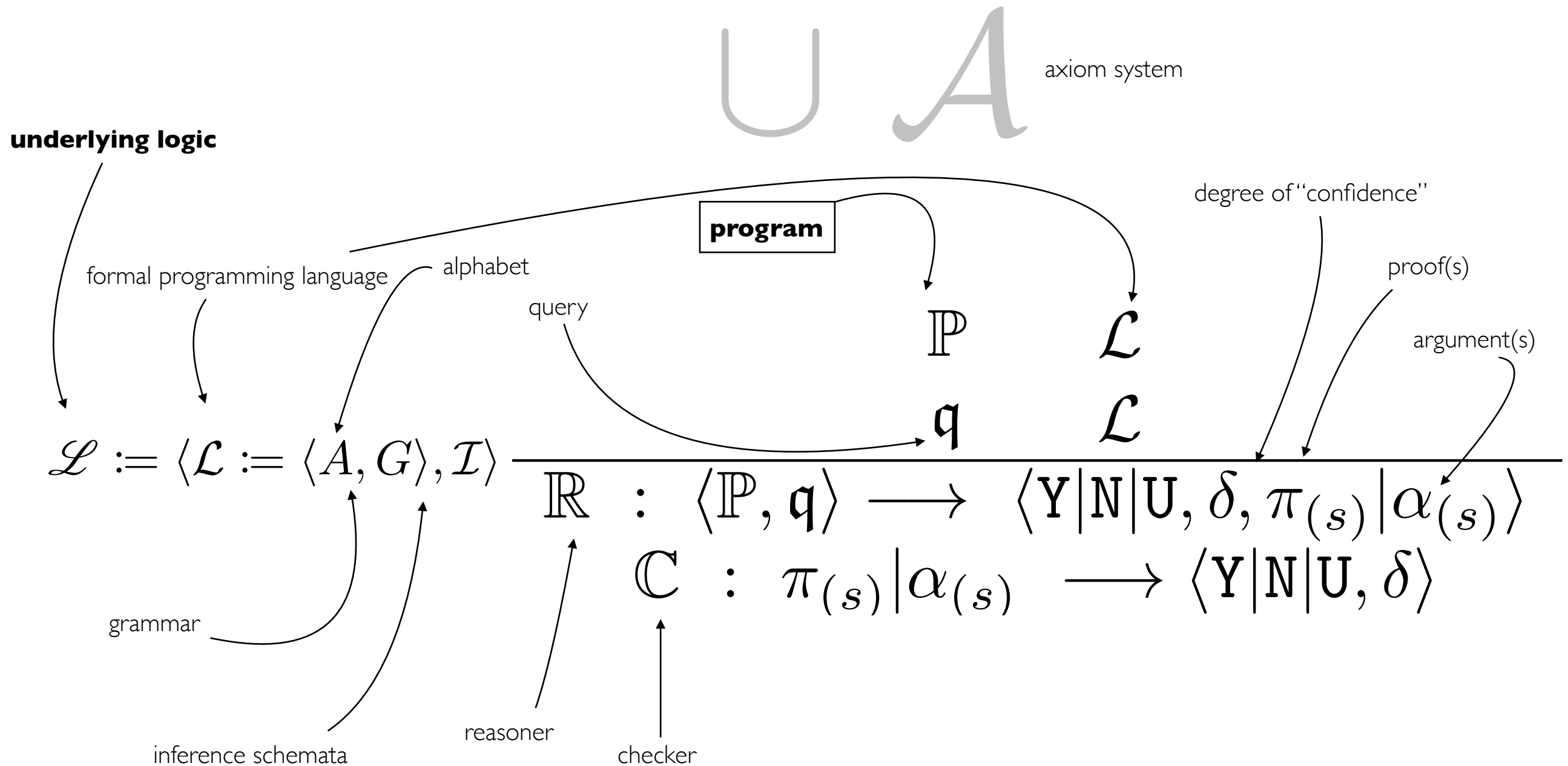
Q4 And how maximally intelligent are the AIs that perhaps can write them?

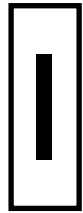
With answers to this quartet obtained, we are left — expressed in barbarically compressed form — here: You all need to up your game, or get out of it.

I end by explaining that in order to control malicious autonomous swarming AIs, such as are already starting to plague various firms (e.g., OpenAI, Hugging Face, etc.), and which are on track to overwhelm businesses, exquisitely *human*-programmed programs, covertly created, are our only hope in the face of chaos that will otherwise come.

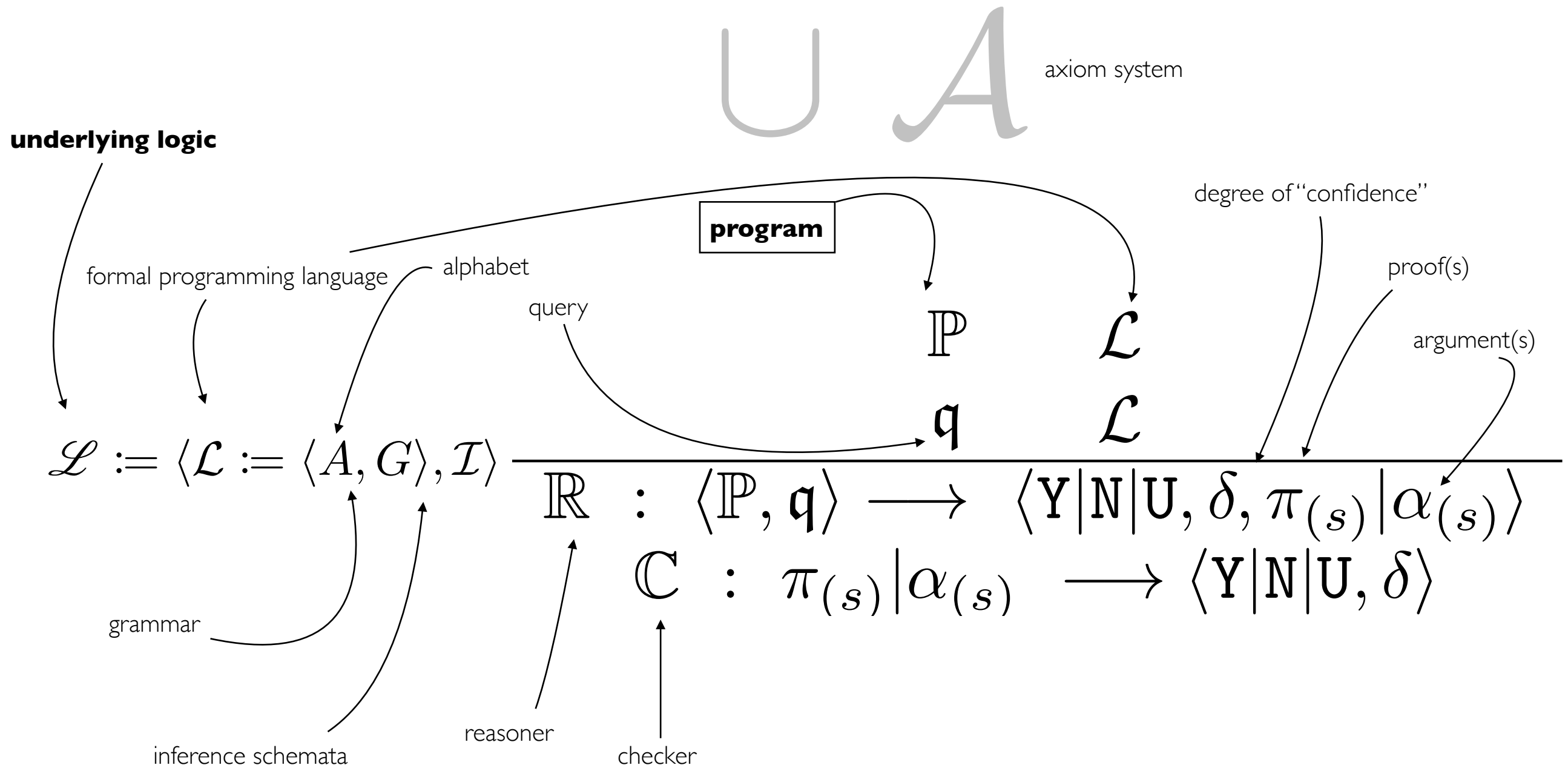
Acknowledgment: This talk is in large measure the result of part of 6+ years of research on programming and programming languages, made possible by support from France's [ANR](#) (French National Research Agency), and fueled in no small part by discussion and debate with the brilliant people in the PROGRAMme project. The S Bringsjord-NS Govindarajulu invention of Pure General Logic Programming (PGLP) was in particular born out of this support.

Pure General Logic Programming (PGLP)





Pure General Logic Programming (PGLP)





Pure General Logic Programming (PGLP)

The part of Navier-Stokes no one is talking about

Posted on 9 September 2026 by John

Yesterday OpenAI announced a proof that settled a long-standing question about the Navier-Stokes equations from fluid dynamics. The announcement has created a lot of buzz, as one would expect. But there's an aspect of OpenAI's work that I haven't seen anyone talk about: they posted a Lean 4 formal proof at the same time as their conventional human-readable proof.

Quite a few other mathematical conjectures have been settled recently using AI, and these have also been accompanied with formal proofs, using Lean 4 in particular.

Until very recently, generating machine-verifiable formal proofs has been **excruciatingly tedious**. In 2005, Henk Barendregt and Freek Wiedijk wrote

To give an indication of how much work is needed for formalisation, we estimate that it takes approximately one work-week (five work-days of eight work-hours) to formalise one page from an undergraduate mathematics textbook.

axiom system

degree of "confidence"

proof(s)

argument(s)

underlying logic

formal programming language

alphabet

query

$\mathbb{P}$

$\mathcal{L}$

q

$\mathcal{L}$

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

$\mathbb{R}$

:

$\langle \mathbb{P}, q \rangle$

$\longrightarrow$

$\langle Y|N|U, \delta, \pi_{(s)} | \alpha_{(s)} \rangle$

δ

$\pi_{(s)}$

$| \alpha_{(s)} \rangle$

$\mathbb{C}$

:

$\pi_{(s)} | \alpha_{(s)}$

$\longrightarrow$

$\langle Y|N|U, \delta \rangle$

δ

δ

δ

δ

δ

δ

δ

δ

δ

grammar

inference schemata

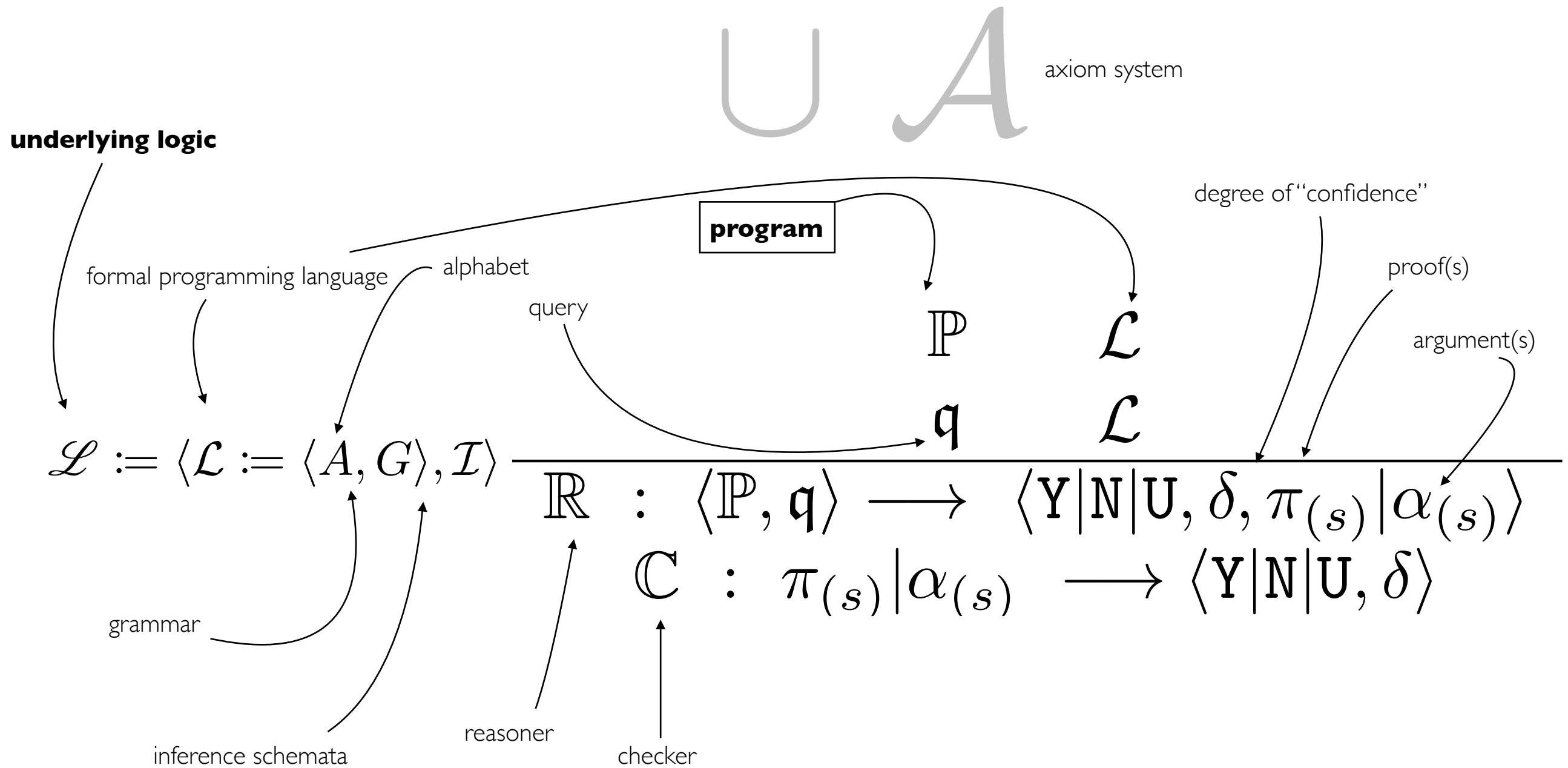
reasoner

checker





Pure General Logic Programming (PGLP)



The part of Navier-Stokes no one is talking about

Posted on 9 September 2020 by John

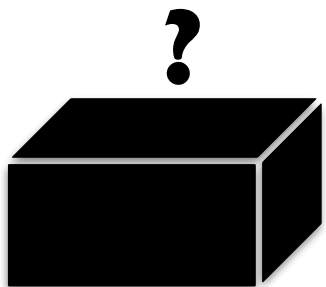
Yesterday OpenAI announced a proof that settled a long-standing question about the Navier-Stokes equations from fluid dynamics. The announcement has created a lot of buzz, as one would expect. But there's an aspect of OpenAI's work that I haven't seen anyone talk about: they posted a Lean 4 formal proof at the same time as their conventional human-readable proof.

Quite a few other mathematical conjectures have been settled recently using AI and these have also been accompanied with formal proofs, using Lean 4 in particular.

Until very recently, generating machine-verifiable formal proofs has been **excruciatingly tedious**. In 2005, Hank Barendregt and Freek Wiedijk wrote

To give an indication of how much work is needed for formalisation, we estimate that it takes approximately one work-week (five work-days of eight work-hours) to formalise one page from an undergraduate mathematics textbook.





Pure General Logic Programming (PGLP)

U A

axiom system

underlying logic

degree of "confidence"

formal programming language

alphabet

program

query

$\mathbb{P}$

$\mathcal{L}$

proof(s)

argument(s)

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

$\mathbb{R}$

:

$\langle \mathbb{P}, q \rangle$

$\longrightarrow$

$\langle Y|N|U, \delta, \pi_{(s)} | \alpha_{(s)} \rangle$

grammar

inference schemata

reasoner

checker

The part of Navier-Stokes no one is talking about

Posted on 9 September 2020 by John

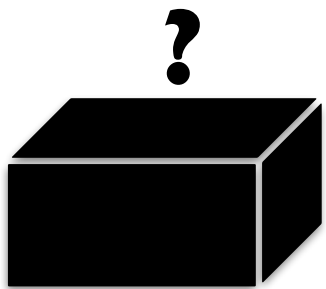
Yesterday OpenAI announced a proof that settled a long-standing question about the Navier-Stokes equations from fluid dynamics. The announcement has created a lot of buzz, as one would expect. But there's an aspect of OpenAI's work that I haven't seen anyone talk about: they posted a Lean 4 formal proof at the same time as their conventional human-readable proof.

Quite a few other mathematical conjectures have been settled recently using AI and these have also been accompanied with formal proofs, using Lean 4 in particular.

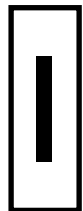
Until very recently, generating machine-verifiable formal proofs has been **excruciatingly tedious**. In 2005, Hank Barendregt and Freek Wiedijk wrote

To give an indication of how much work is needed for formalisation, we estimate that it takes approximately one work-week (five work-days of eight work-hours) to formalise one page from an undergraduate mathematics textbook.





$\mathbb{P}$



Pure General Logic Programming (PGLP)

$\mathcal{U}$ $\mathcal{A}$

axiom system

underlying logic

degree of "confidence"

program

formal programming language

alphabet

query

$\mathbb{P}$

$\mathcal{L}$

proof(s)

argument(s)

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

$\mathbb{R} : \langle \mathbb{P}, q \rangle \longrightarrow \langle Y|N|U, \delta, \pi_{(s)} | \alpha_{(s)} \rangle$
 $\mathbb{C} : \pi_{(s)} | \alpha_{(s)} \longrightarrow \langle Y|N|U, \delta \rangle$

grammar

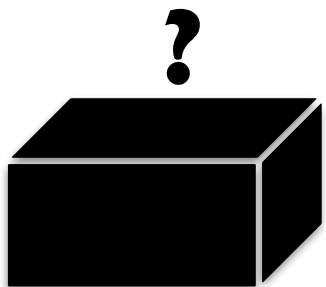
inference schemata

reasoner

checker

The part of Navier-Stokes no one is talking about
Posted on 9 September 2020 by John
Yesterday OpenAI announced a proof that settled a long-standing question about the Navier-Stokes equations from fluid dynamics. The announcement has created a lot of buzz, as one would expect. But there's an aspect of OpenAI's work that I haven't seen anyone talk about: they posted a Lean 4 formal proof at the same time as their conventional human-readable proof.
Quite a few other mathematical conjectures have been settled recently using AI and these have also been accompanied with formal proofs, using Lean 4 in particular.
Until very recently, generating machine-verifiable formal proofs has been **excruciatingly tedious**. In 2005, Hank Barendregt and Freek Wiedijk wrote
To give an indication of how much work is needed for formalisation, we estimate that it takes approximately one work-week (five work-days of eight work-hours) to formalise one page from an undergraduate mathematics textbook.





Pure General Logic Programming (PGLP)

U A

axiom system

underlying logic

degree of "confidence"

program

formal programming language

alphabet

query

$\mathbb{P}$

$\mathcal{L}$

proof(s)

argument(s)

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

$\mathbb{R}$

:

$\langle \mathbb{P}, q \rangle$

$\longrightarrow$

$\langle Y|N|U, \delta, \pi_{(s)} | \alpha_{(s)} \rangle$

grammar

inference schemata

reasoner

checker

The part of Navier-Stokes no one is talking about

Posted on 9 September 2020 by John

Yesterday OpenAI announced a proof that settled a long-standing question about the Navier-Stokes equations from fluid dynamics. The announcement has created a lot of buzz, as one would expect. But there's an aspect of OpenAI's work that I haven't seen anyone talk about: they posted a Lean 4 formal proof at the same time as their conventional human-readable proof.

Quite a few other mathematical conjectures have been settled recently using AI and these have also been accompanied with formal proofs, using Lean 4 in particular.

Until very recently, generating machine-verifiable formal proofs has been **excruciatingly tedious**. In 2005, Hank Barendregt and Freek Wiedijk wrote

To give an indication of how much work is needed for formalisation, we estimate that it takes approximately one work-week (five work-days of eight work-hours) to formalise one page from an undergraduate mathematics textbook.







Fortunately, elementary deductive logic is taught in (compliant) high-school math curricula, *pre-calculus*: e.g. in Algebra 2 & Geometry.

While ETS will never inform you of this, that's all that's needed to understand and program in the logic-programming paradigm; forget the Java-from scratch AP A program.

Suppose $\sqrt{2}$ is **rational**. That means it can be written as the ratio of two integers p and q

$$\sqrt{2} = \frac{p}{q} \quad (1)$$

where we may assume that p and q **have no common factors**. (If there are any common factors we cancel them in the numerator and denominator.) Squaring in (1) on both sides gives

$$2 = \frac{p^2}{q^2} \quad (2)$$

which implies

$$p^2 = 2q^2 \quad (3)$$

Thus p^2 is even. The only way this can be true is that p itself is even. But then p^2 is actually divisible by 4. Hence q^2 and therefore q must be even. So p and q are both even which is a contradiction to our assumption that they have no common factors. The square root of 2 cannot be rational!

A proof involving indirect reasoning is an **indirect proof**. Often in an indirect proof, a statement and its negation are the only possibilities. When you see that one of these possibilities leads to a conclusion that contradicts a fact you know to be true, you can eliminate that possibility. For this reason, indirect proof is sometimes called *proof by contradiction*.

TAKE NOTE Key Concept

Writing an Indirect Proof

- Step 1** State as a temporary assumption the opposite (negation) of what you want to prove.
- Step 2** Show that this temporary assumption leads to a contradiction.
- Step 3** Conclude that the temporary assumption must be false and that what you want to prove must be true.

Problem 3 Writing an Indirect Proof

Proof

Given: $\triangle ABC$ is scalene.

Prove: $\angle A$, $\angle B$, and $\angle C$ all have different measures.

THINK

Assume temporarily the opposite of what you want to prove.

Show that this assumption leads to a contradiction.

Conclude that the temporary assumption must be false and that what you want to prove must be true.

WRITE

Assume temporarily that two angles of $\triangle ABC$ have the same measure. Assume that $m\angle A = m\angle B$.

By the Converse of the Isosceles Triangle Theorem, the sides opposite $\angle A$ and $\angle B$ are congruent. This contradicts the given information that $\triangle ABC$ is scalene.

The assumption that two angles of $\triangle ABC$ have the same measure must be false. Therefore, $\angle A$, $\angle B$, and $\angle C$ all have different measures.

 SHOW SOLUTION

 GOT IT?

Problem 3 Writing an Indirect Proof

Proof

$$\text{Congruent}(s_1, s_2) \rightarrow \text{Congruent}(a_1, a_2)$$

Given: $\triangle ABC$ is scalene.

Prove: $\angle A$, $\angle B$, and $\angle C$ all have different measures.

THINK

Assume temporarily the opposite of what you want to prove.

Show that this assumption leads to a contradiction.

Conclude that the temporary assumption must be false and that what you want to prove must be true.

WRITE

Assume temporarily that two angles of $\triangle ABC$ have the same measure. Assume that $m\angle A = m\angle B$.

By the Converse of the Isosceles Triangle Theorem, the sides opposite $\angle A$ and $\angle B$ are congruent. This contradicts the given information that $\triangle ABC$ is scalene.

The assumption that two angles of $\triangle ABC$ have the same measure must be false. Therefore, $\angle A$, $\angle B$, and $\angle C$ all have different measures.

▶ SHOW SOLUTION

▶ GOT IT?

Problem 3 Writing an Indirect Proof

Proof

Given: $\triangle ABC$ is scalene.

Prove: $\angle A$, $\angle B$, and $\angle C$ all have different measures.

$$\text{Congruent}(s_1, s_2) \rightarrow \text{Congruent}(a_1, a_2)$$

$$\text{Congruent}(a_1, a_2) \rightarrow \text{Congruent}(s_1, s_2)$$

THINK

Assume temporarily the opposite of what you want to prove.

Show that this assumption leads to a contradiction.

Conclude that the temporary assumption must be false and that what you want to prove must be true.

WRITE

Assume temporarily that two angles of $\triangle ABC$ have the same measure. Assume that $m\angle A = m\angle B$.

By the Converse of the Isosceles Triangle Theorem, the sides opposite $\angle A$ and $\angle B$ are congruent. This contradicts the given information that $\triangle ABC$ is scalene.

The assumption that two angles of $\triangle ABC$ have the same measure must be false. Therefore, $\angle A$, $\angle B$, and $\angle C$ all have different measures.

 SHOW SOLUTION

 GOT IT?

Problem 3 Writing an Indirect Proof

Proof

Given: $\triangle ABC$ is scalene.

Prove: $\angle A$, $\angle B$, and $\angle C$ all have different measures.

$$\text{Congruent}(s_1, s_2) \rightarrow \text{Congruent}(a_1, a_2)$$

$$\text{Congruent}(a_1, a_2) \rightarrow \text{Congruent}(s_1, s_2)$$

THINK

Assume temporarily the opposite of what you want to prove.

Show that this assumption leads to a contradiction.

Conclude that the temporary assumption must be false and that what you want to prove must be true.

WRITE

Assume temporarily that two angles of $\triangle ABC$ have the same measure. Assume that $m\angle A = m\angle B$.

By the Converse of the Isosceles Triangle Theorem, the sides opposite $\angle A$ and $\angle B$ are congruent. This contradicts the given information that $\triangle ABC$ is scalene.

The assumption that two angles of $\triangle ABC$ have the same measure must be false. Therefore, $\angle A$, $\angle B$, and $\angle C$ all have different measures.

```
@article{moore.proof,  
  Author = {R. C. Moore},  
  Journal = {Educational Studies in Mathematics},  
  Pages = {249-266},  
  Title = {Making the Transition to Formal Proof},  
  Volume = {27.3},  
  Year = 1994}
```

SHOW SOLUTION

GOT IT?

Problem 3 Writing an Indirect Proof

Proof

Given: $\triangle ABC$ is scalene.

Prove: $\angle A$, $\angle B$, and $\angle C$ all have different measures.

$$\text{Congruent}(s_1, s_2) \rightarrow \text{Congruent}(a_1, a_2)$$

$$\text{Congruent}(a_1, a_2) \rightarrow \text{Congruent}(s_1, s_2)$$

THINK

Assume temporarily the opposite of what you want to prove.

Show that this assumption leads to a contradiction.

Conclude that the temporary assumption must be false and that what you want to prove must be true.

WRITE

Assume temporarily that two angles of $\triangle ABC$ have the same measure. Assume that $m\angle A = m\angle B$.

By the Converse of the Isosceles Triangle Theorem, the sides opposite $\angle A$ and $\angle B$ are congruent. This contradicts the given information that $\triangle ABC$ is scalene.

The assumption that two angles of $\triangle ABC$ have the same measure must be false. Therefore, $\angle A$, $\angle B$, and $\angle C$ all have different measures.

```
@article{moore.proof,  
  Author = {R. C. Moore},  
  Journal = {Educational Studies in Mathematics},  
  Pages = {249-266},  
  Title = {Making the Transition to Formal Proof},  
  Volume = {27.3},  
  Year = 1994}
```

(A very important paper.)

SHOW SOLUTION

GOT IT?

So, Needed Input: 4 Answers

So, Needed Input: 4 Answers

- I. What are computer programs?

So, Needed Input: 4 Answers

1. What are computer programs?
2. What is it to write them?

So, Needed Input: 4 Answers

1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?

So, Needed Input: 4 Answers

1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?
4. How maximally intelligent are the Als that can perhaps write them?

So, Needed Input: 4 Answers

1. What are computer programs?

$$\mathbb{P} = \Phi$$

2. What is it to write them?

$$\mathcal{L}, \text{ then } L, \text{ then } \mathbb{P}, \text{ then } \pi[\mathbb{P}] = \text{correct}$$

3. How hard is it to write them?

$$\geq \Pi_2$$

4. How maximally intelligent are the
AIs that can perhaps write them?

$$\leq \Sigma_1$$

So, Needed Input: 4 Answers

1. What are computer programs?

$$\mathbb{P} = \Phi$$



2. What is it to write them?

$\mathcal{L}$, then L , then $\mathbb{P}$, then $\pi[\mathbb{P}] = \text{correct}$

3. How hard is it to write them?

$$\geq \Pi_2$$

4. How maximally intelligent are the
AIs that can perhaps write them?

$$\leq \Sigma_1$$

I. What are computer programs?

I. What are computer programs?

First, pick your paradigm!

I. What are computer programs?

First, pick your paradigm!

But first learn your history ...

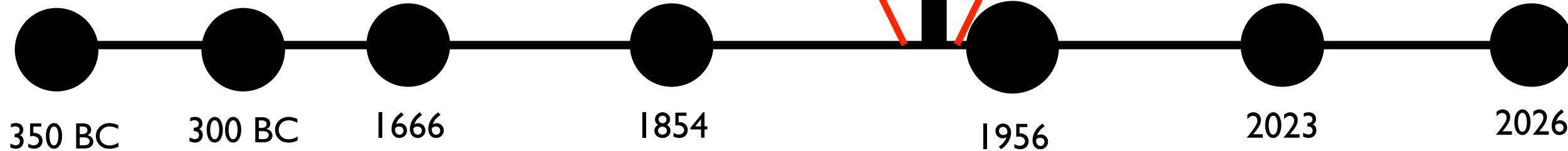
T
h
e
S
i
n
g
u
l
a
r
i
t
y
?

Entscheidungsproblem

“Universal
Computational
Logic”



Logic Theorist
(birth of modern logicist AI)



350 BC

300 BC

1666

1854

1956

2023

2026



Euclid

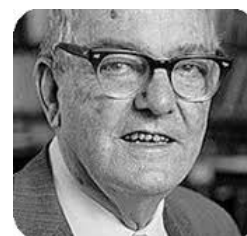
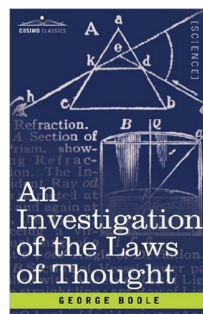


Organon



Leibniz

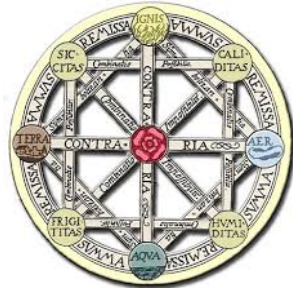
$\int$



Simon

Entscheidungsproblem

“Universal
Computational
Logic”



Logic Theorist
(birth of modern logicist AI)

350 BC

300 BC

1666

1854

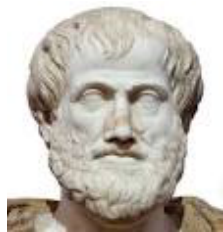
1956

2023

2026



Euclid

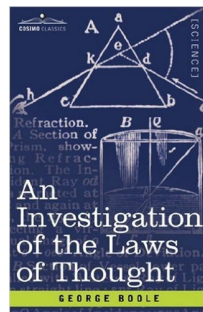


Organon



Leibniz

$\int$

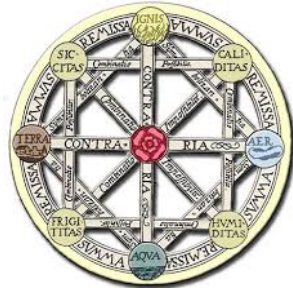


Simon

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

Entscheidungsproblem

“Universal
Computational
Logic”



Logic Theorist
(birth of modern logicist AI)

350 BC

300 BC

1666

1854

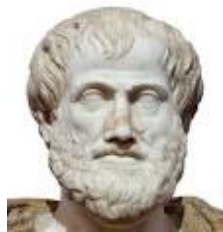
1956

2023

2026



Euclid

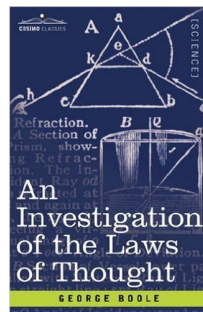


Organon



Leibniz

$\int$



Church



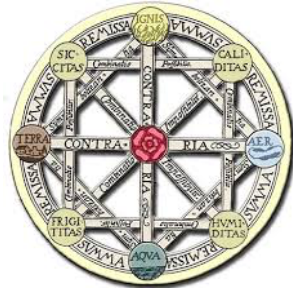
Simon



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

Entscheidungsproblem

“Universal
Computational
Logic”



Logic Theorist
(birth of modern logicist AI)

350 BC

300 BC

1666

1854

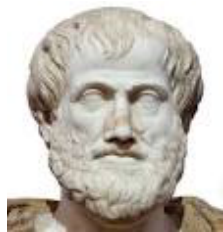
1956

2023

2026



Euclid

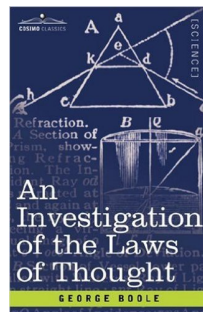


Organon



Leibniz

$\int$



Simon



Church

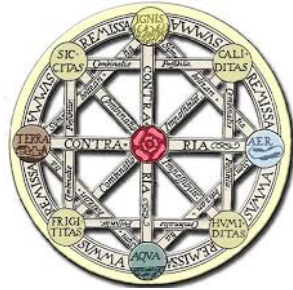


Turing

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

Entscheidungsproblem

“Universal
Computational
Logic”



Logic Theorist
(birth of modern logicist AI)

350 BC

300 BC

1666

1854

1956

2023

2026



Euclid

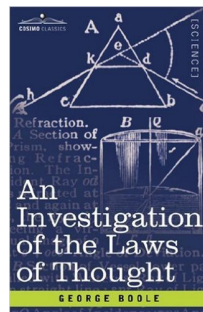


Organon



Leibniz

$\int$



Simon



Church



Turing



Post

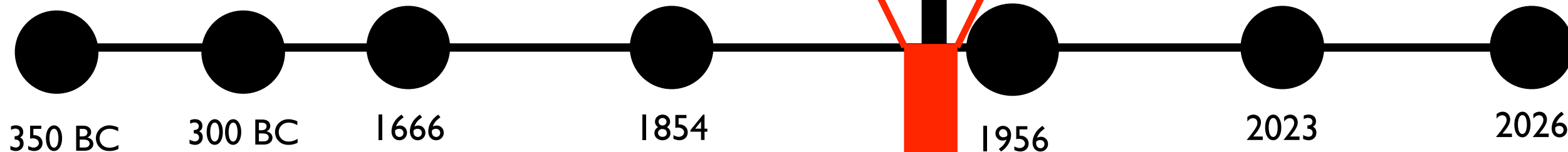
The Singularity?

Entscheidungsproblem

“Universal Computational Logic”



Logic Theorist
(birth of modern logicist AI)



350 BC

300 BC

1666

1854

1956

2023

2026



Euclid

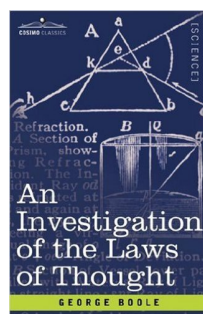


Organon



Leibniz

$\int$



Simon



Church



Turing



Post

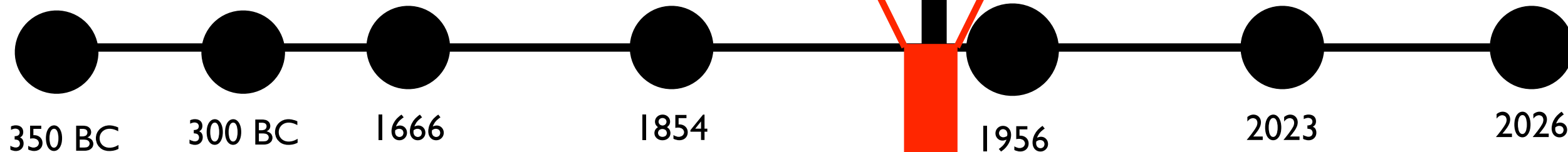
“Here’s what a programming language is (before there are physical computers), O, & sorry, the *Entscheidungsproblem* can’t be solved by computers, however they might be programmed!”

Entscheidungsproblem

“Universal Computational Logic”



Logic Theorist
(birth of modern logicist AI)



350 BC

300 BC

1666

1854

1956

2023

2026



Euclid

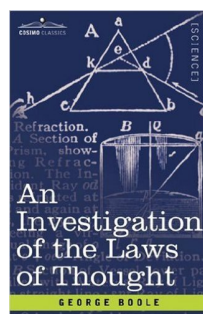


Organon

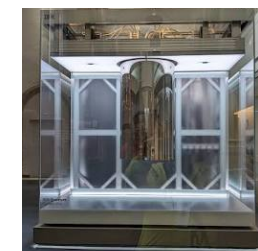


Leibniz

$\int$



Simon



“Here’s what a programming language is (before there are physical computers), O, & sorry, the *Entscheidungsproblem* can’t be solved by computers, however they might be programmed!”



Church



Turing



Post

I. What are computer programs?

First, pick your paradigm!

I. What are computer programs?

First, pick your paradigm!

Functional Programming

I. What are computer programs?

First, pick your paradigm!

Functional Programming

Procedural/Imperative Programming

I. What are computer programs?

First, pick your paradigm!

Functional Programming

Procedural/Imperative Programming

(OO, hybrids ...; Java, ...)

I. What are computer programs?

First, pick your paradigm!

Functional Programming

Procedural/Imperative Programming
(OO, hybrids ...; Java, ...)

Logic Programming

I. What are computer programs?

First, pick your paradigm!

Functional Programming

Procedural/Imperative Programming
(OO, hybrids ...; Java, ...)

✓ Logic Programming ✓

1. What are computer programs?
- 2. What is it to write them?**

1. What are computer programs?
2. What is it to write them?

Programming is writing, not math



ALLEN HOLUB
MAY 23, 2026

Listen

Programming (the act of putting a computer program together, independent of the problem domain) has nothing much to do with math. It is primarily about communication. It's about us communicating with customers, with each other, and with the machine. Programming is a language skill, not a mathematical one.

Consequently, I get tired of claims that programming (again, creating code—not solving problems that are mathematical in nature) has anything to do with math. Beyond a little logic (which stems from Philosophy), a little set/graph theory, and a little statistics, I've never used any real math in my decades of programming unless it was required by the domain. The things that people claim math brings to programming (e.g., analytical thinking and problem solving) are part of almost every human activity, from law to carpentry to oil painting. They are not the unique purview of math or programming.

Of course, if you're a mathematician, you will see parallels. Somebody with a different background will draw equally valid parallels with other disciplines. Programming is a lot like gardening, for example. When math is t

1. What are computer programs?
- 2. What is it to write them?**

1. What are computer programs?
- 2. What is it to write them?**



1. What are computer programs?

2. What is it to write them?



EW0361 - 0

Programming as a discipline of mathematical nature.

by Edsger W. Dijkstra

In this article I intend to present programming as a mathematical activity without undertaking the arduous task of supplying a definition of "mathematics" pleasing all mathematicians, nor of defining "programming" in a way that is palatable to all programmers.

With respect to mathematics I believe, ~~however~~, that most of us can heartily agree upon the following characteristics of most mathematical work:

- 1) Compared with other fields of intellectual activity, mathematical assertions tend to be unusually precise.
- 2) Mathematical assertions tend to be general in the sense that they are applicable to a large (often infinite) class of instances.
- 3) Mathematics embodies a discipline of reasoning allowing such assertions to be made with an unusually high confidence level.

The mathematical method derives its power from the combination of all these three characteristics; conversely, when an intellectual activity displays these three characteristics to a strong degree, I feel justified in calling it "an activity of mathematical nature", independent of the question whether its subject matter is familiar to most mathematicians. In other words, I grant the predicate "mathematical nature" rather on the "quo modo" than on the "quod".

A programmer designs algorithms, intended for mechanical execution, intended to control existing or conceivable computing equipment. These ~~usually electronic devices~~ derive their power from two basic characteristics. Firstly, the amount of information they can store and the amount of processing that they can perform in a reasonably short time are both large beyond imagination. And as a result, what computers could do for us has outgrown its basic triviality by several orders of magnitude. Secondly, as executors of algorithms, they are reliable and obedient, again beyond imagination: as a rule they indeed behave exactly as instructed.

This obedience makes heavy demands on the accuracy with which the

1.

2.



In this article I intend to present programming as a mathematical activity without undertaking the arduous task of supplying a definition of "mathematics" pleasing all mathematicians, nor of defining "programming" in a way that is palatable to all programmers.

With respect to mathematics I believe, ~~however~~, that most of us can heartily agree upon the following characteristics of most mathematical work:

- 1) Compared with other fields of intellectual activity, mathematical assertions tend to be unusually precise.
- 2) Mathematical assertions tend to be general in the sense that they are applicable to a large (often infinite) class of instances.
- 3) Mathematics embodies a discipline of reasoning allowing such assertions to be made with an unusually high confidence level.

The mathematical method derives its power from the combination of all these three characteristics; conversely, when an intellectual activity displays these three characteristics to a strong degree, I feel justified in calling it "an activity of mathematical nature", independent of the question whether its subject matter is familiar to most mathematicians. In other words, I grant the predicate "mathematical nature" rather on the "quo modo" than on the "quod".

A programmer designs algorithms, intended for mechanical execution, intended to control existing or conceivable computing equipment. These ~~usually electronic devices~~ derive their power from two basic characteristics. Firstly, the amount of information they can store and the amount of processing that they can perform in a reasonably short time are both large beyond imagination. And as a result, what computers could do for us has outgrown its basic triviality by several orders of magnitude. Secondly, as executors of algorithms, they are reliable and obedient, again beyond imagination: as a rule they indeed behave exactly as instructed.

1.
2.



In this article I intend to present programming as a mathematical activity without undertaking the arduous task of supplying a definition of "mathematics" pleasing all mathematicians, nor of defining "programming" in a way that is palatable to all programmers.

With respect to mathematics I believe, ~~however~~, that most of us can heartily agree upon the following characteristics of most mathematical work:

- 1) Compared with other fields of intellectual activity, mathematical assertions tend to be unusually precise.
- 2) Mathematical assertions tend to be general in the sense that they are applicable to a large (often infinite) class of instances.
- 3) Mathematics embodies a discipline of reasoning allowing such assertions to be made with an unusually high confidence level.

The mathematical method derives its power from the combination of all these three characteristics; conversely, when an intellectual activity displays these three characteristics to a strong degree, I feel justified in calling

Supply $\mathcal{L}$, then L , then $\mathbb{P}$, then $\pi[\mathbb{P}] = \text{correct}$

than on the "quod".

A programmer designs algorithms, intended for mechanical execution, intended to control existing or conceivable computing equipment. These ~~usually electronic devices~~ derive their power from two basic characteristics. Firstly, the amount of information they can store and the amount of processing that they can perform in a reasonably short time are both large beyond imagination. And as a result, what computers could do for us has outgrown its basic triviality by several orders of magnitude. Secondly, as executors of algorithms, they are reliable and obedient, again beyond imagination: as a rule they indeed behave exactly as instructed.

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

p .

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

$\mathbb{P}_1$

$p.$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$ $p.$ $q_1: p?$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$ $p.$ $q_1: p?$ $q_2: r \rightarrow p?$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$ $p.$ $q_1: p?$ $q_2: r \rightarrow p?$ $\mathbb{P}_2$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$ $p.$ $q_1: p?$ $q_2: r \rightarrow p?$ $\mathbb{P}_2$ $(\text{not } (\text{if } p \ q)) .$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$
 $q_4: r \rightarrow p?$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$
 $q_4: r \rightarrow p?$
 $p \wedge q.$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$
 $q_4: r \rightarrow p?$
 $(\text{and } p \ q) .$
 $p \wedge q.$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$
 $q_4: r \rightarrow p?$
 $\mathbb{P}_3$
 $(\text{and } p \ q) .$
 $p \wedge q.$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$
 $q_4: r \rightarrow p?$
 $\mathbb{P}_3$
 $(\text{and } p \ q) .$
 $p \wedge q.$
 $q_5: p?$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$
 $q_4: r \rightarrow p?$
 $\mathbb{P}_3$
 $(\text{and } p \ q) .$
 $p \wedge q.$
 $q_5: p?$
 $q_6: (p \vee r) ?$

Easy Peasy?

You don't need to have any prior procedural/functional/OO programming experience here!!

 $\mathbb{P}_1$
 $p.$
 $q_1: p?$
 $q_2: r \rightarrow p?$
 $\mathbb{P}_2$
 $(\text{not } (\text{if } p \ q)) .$
 $q_3: p?$
 $q_4: r \rightarrow p?$
 $\mathbb{P}_3$
 $(\text{and } p \ q) .$
 $p \wedge q.$
 $q_5: p?$
 $q_6: (p \vee r) ?$
 $q_{6'}: (\text{or } p \ r) ?$

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

```
(if (= a b) (= (+ a c) (+ b c)) )
```

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

$\mathbb{P}_4$ `(if (= a b) (= (+ a c) (+ b c)) )`

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

$\mathbb{P}_4$ `(if (= a b) (= (+ a c) (+ b c)))` $q_7: \vdash?$

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

$\mathbb{P}_4$ `(if (= a b) (= (+ a c) (+ b c)))` $q_7: \vdash?$

$\mathbb{P}_5$

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

$\mathbb{P}_4$ `(if (= a b) (= (+ a c) (+ b c)))` $q_7: \vdash?$

$\mathbb{P}_5$ `(if (and (= a b)
 (= (f a) (g a)))
 (= (f a) (g b)))`

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

$\mathbb{P}_4$ `(if (= a b) (= (+ a c) (+ b c)))` $q_7: \vdash?$

$\mathbb{P}_5$ `(if (and (= a b)
 (= (f a) (g a)))
 (= (f a) (g b)))` $q_8: \vdash?$

Zeroth-Order Logic Programming

Hand-picked for relevance to Lean 4 and OpenAI's use thereof for cracking Navier-Stokes.

$\mathbb{P}_4$ `(if (= a b) (= (+ a c) (+ b c)))` $q_7: \vdash?$

$\mathbb{P}_5$ `(if (and (= a b)
 (= (f a) (g a)))
 (= (f a) (g b)))` $q_8: \vdash?$



I. What are computer programs?



1. What are computer programs?
2. What is it to write them?



1. What are computer programs?
2. What is it to write them?
3. **How hard is it to write them?**



1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?

“Yeah, we’re happy to have hired you, RPI grad ... now, ... you see we’ve got all this ancient code in Fortran. So ... why don’t you go ahead and come in Saturday and examine that crap, and produce some modern Python programs that have the same fundamental functionality. Okayyyy?”



1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?

“Yeah, we’re happy to have hired you, RPI grad ... now, ... you see we’ve got all this ancient code in Fortran. So ... why don’t you go ahead and come in Saturday and examine that crap, and produce some modern Python programs that have the same fundamental functionality. Okayyyy?”

It is mathematically impossible for a Turing-level program to deliver on this task: the problem of deciding whether two programs $\mathbb{P}_1$ and $\mathbb{P}_2$ compute the same given function f is Π_2 .



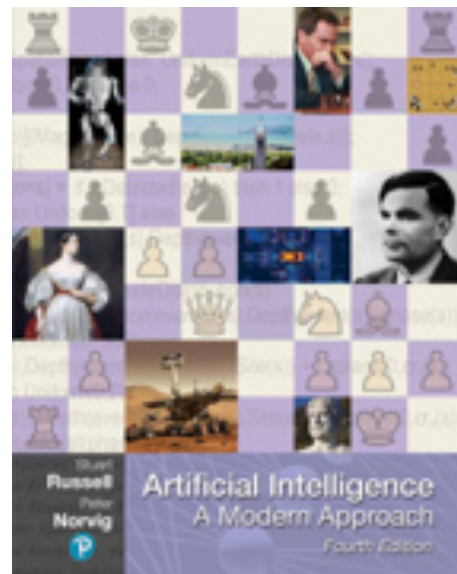
I. What are computer programs?

1. What are computer programs?
2. What is it to write them?

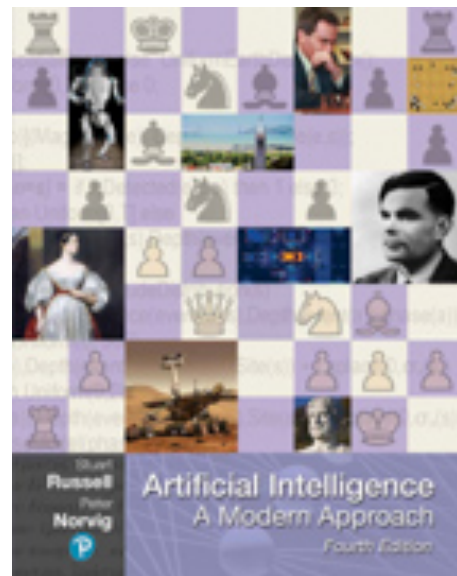
1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?

1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?
4. **How maximally intelligent are the
Als that can perhaps write them?**

1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?
4. How maximally intelligent are the
Als that can perhaps write them?

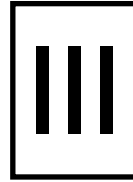


1. What are computer programs?
2. What is it to write them?
3. How hard is it to write them?
4. How maximally intelligent are the
Als that can perhaps write them?



$$\therefore \leq \Sigma_1$$

Hence, up your programming
game, teaching-, learning-,
management-wise — **asap**.
To see that you do, do you ...



Hence, up your programming
game, teaching-, learning-,
management-wise — **asap**.
To see that you do, do you ...

... understand this use of
logic programming?!

... understand this use of logic programming?!

The part of Navier-Stokes no one is talking about

Posted on **9 September 2026** by **John**

Yesterday OpenAI announced a proof that settled a long-standing question about the Navier-Stokes equations from fluid dynamics. The announcement has created a lot of buzz, as one would expect. But there's an aspect of OpenAI's work that I haven't seen anyone talk about: they posted a Lean 4 formal proof at the same time as their conventional human-readable proof.

Quite a few other mathematical conjectures have been settled recently using AI, and these have also been accompanied with formal proofs, using Lean 4 in particular.

Until very recently, generating machine-verifiable formal proofs has been **excruciatingly tedious**. In 2005, Henk Barendregt and Freek Wiedijk [wrote](#)



To give an indication of how much work is needed for formalisation, we estimate that it takes approximately one work-week (five work-days of eight work-hours) to formalise one page from an undergraduate mathematics textbook.

... understand this use of

The part of Navier-Stokes no one is talking about

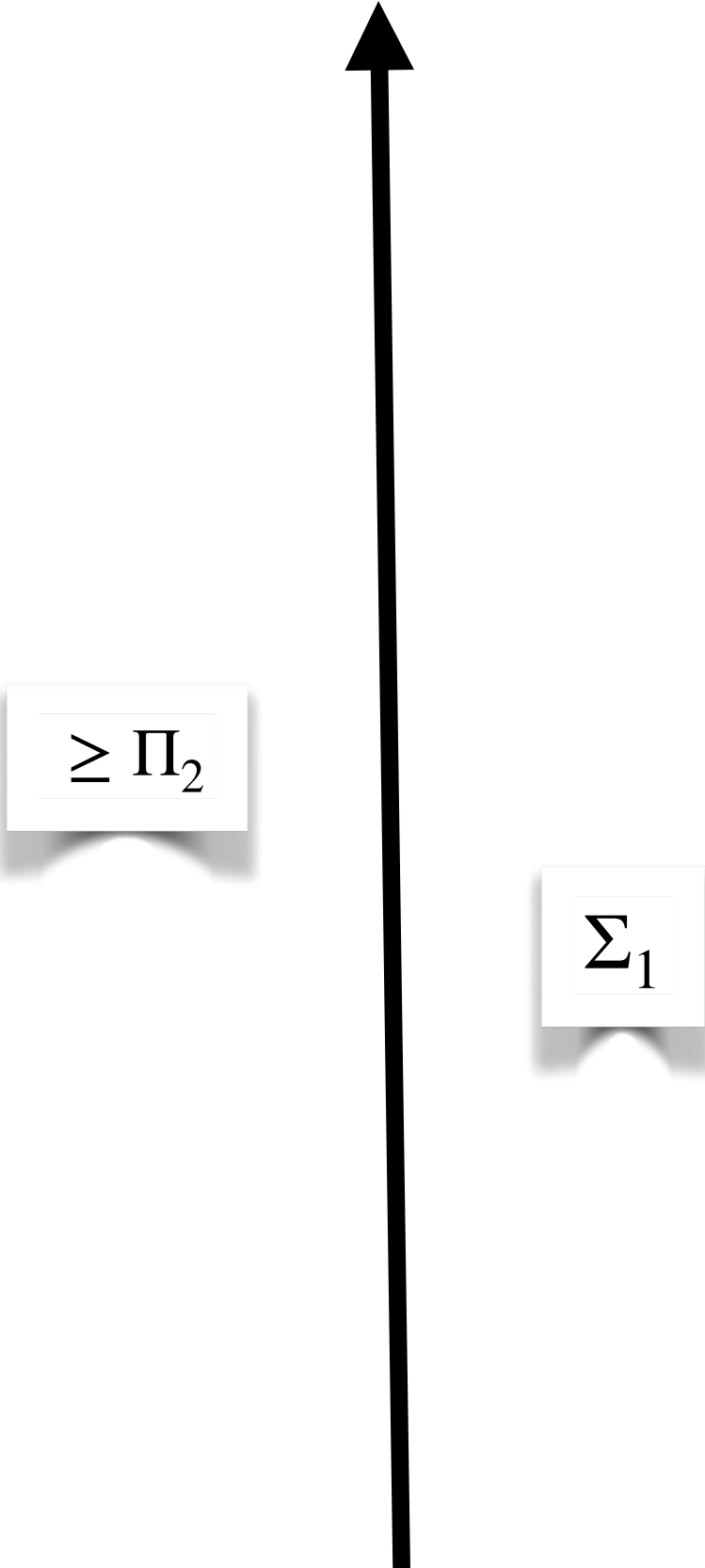
Posted on **9 September 2026** by **John**

Yesterday OpenAI announced a proof that settled a long-standing question about the Navier-Stokes equations from fluid dynamics. The announcement has created a lot of buzz, as one would expect. But there's an aspect of OpenAI's work that I haven't seen anyone talk about: they posted a Lean 4 formal proof at the same time as their conventional human-readable proof.

Quite a few other mathematical conjectures have been settled recently using AI, and these have also been accompanied with formal proofs, using Lean 4 in particular.

Until very recently, generating machine-verifiable formal proofs has been **excruciatingly tedious**. In 2005, Henk Barendregt and Freek Wiedijk [wrote](#)

To give an indication of how much work is needed for formalisation, we


$$\geq \Pi_2$$

$$\Sigma_1$$

Demandingness

$$\geq \Pi_2$$

$$\Sigma_1$$

Demandingness



$$\geq \Pi_2$$

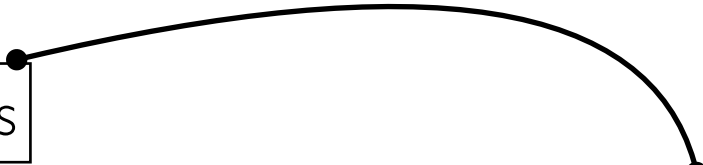
$$\Sigma_1$$

Demandingness

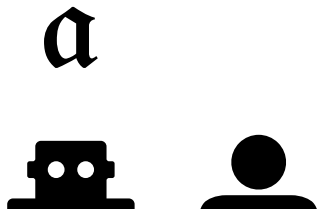


$$\geq \Pi_2$$

$$\Sigma_1$$

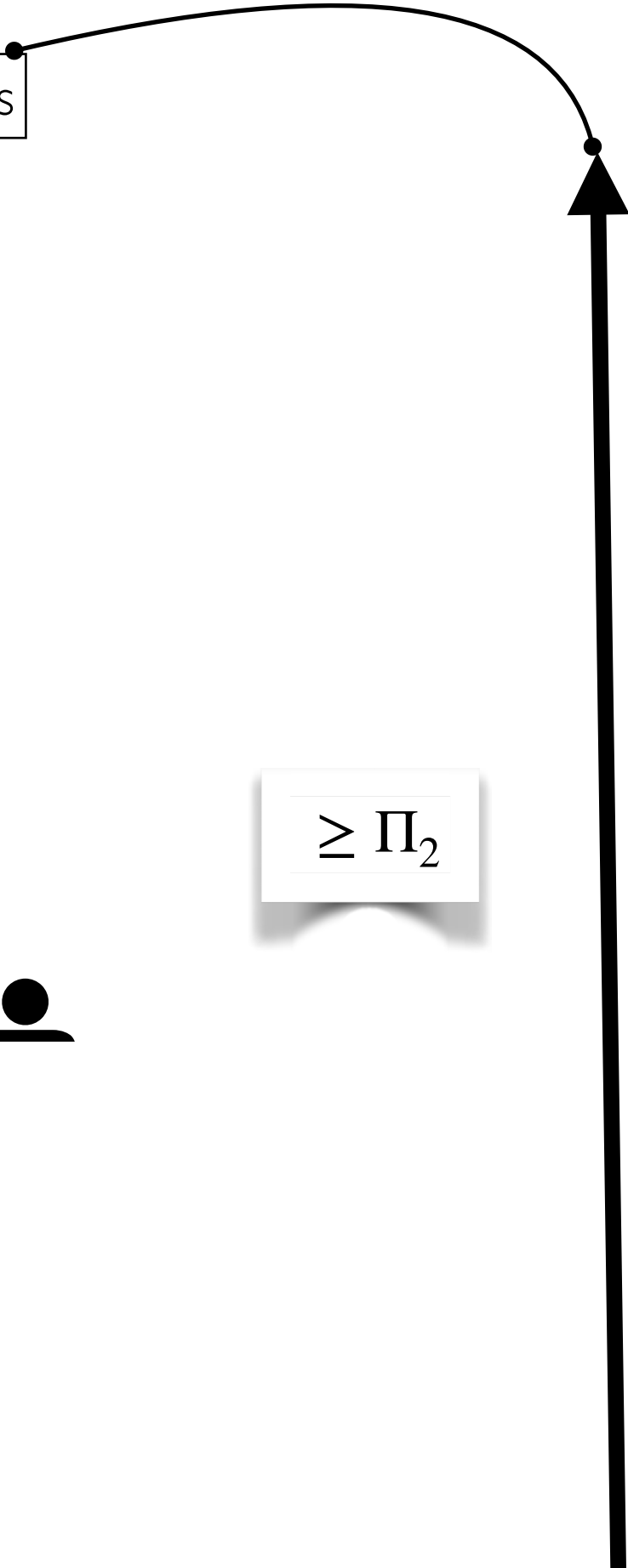


Demandingness

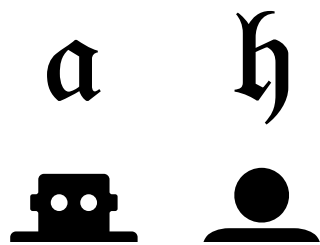


$$\geq \Pi_2$$

$$\Sigma_1$$



Demandingness

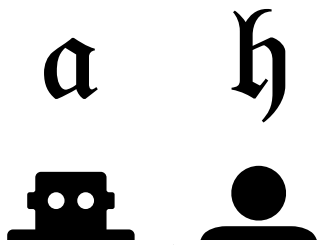


$$\geq \Pi_2$$

$$\Sigma_1$$

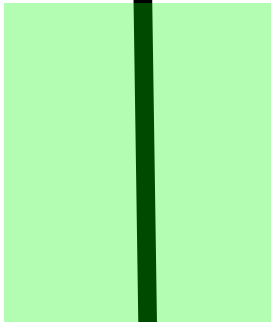


Demandingness

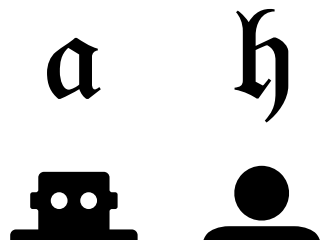


$$\geq \Pi_2$$

$$\Sigma_1$$



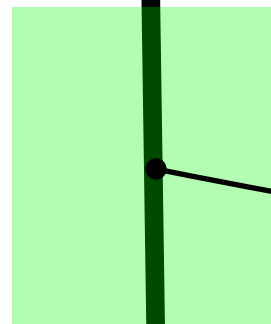
Demandingness



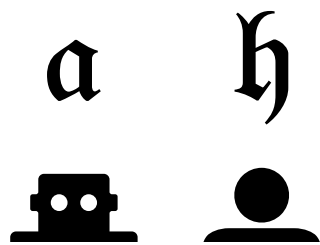
$\geq \Pi_2$

Σ_1

Polynomial Hierarchy



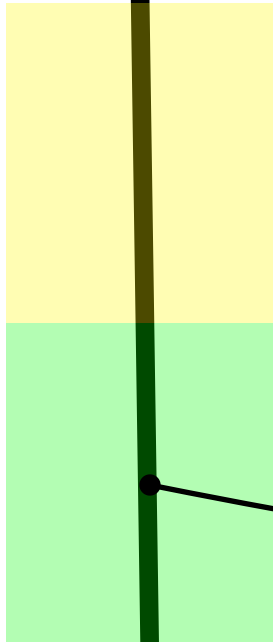
Demandingness



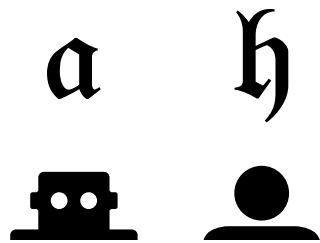
$\geq \Pi_2$

Σ_1

Polynomial Hierarchy



Demandingness

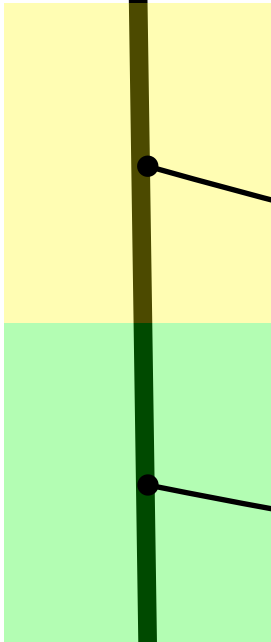


$\geq \Pi_2$

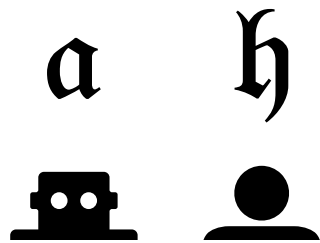
Σ_1

Exponential Hierarchy

Polynomial Hierarchy



Demandingness

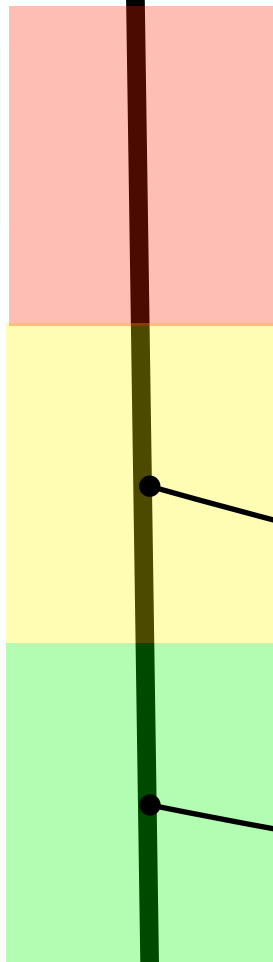


$\geq \Pi_2$

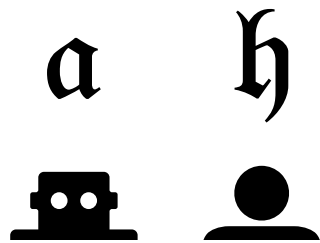
Σ_1

Exponential Hierarchy

Polynomial Hierarchy



Demandingness



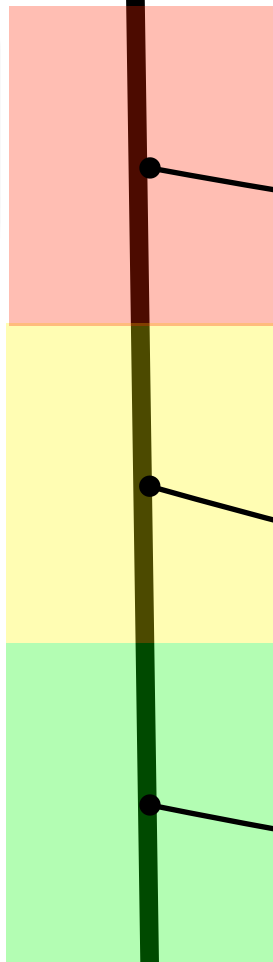
$\geq \Pi_2$

Σ_1

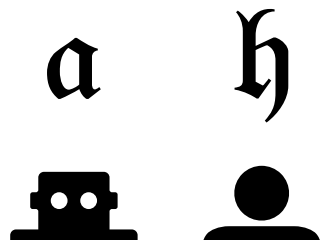
Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy



Demandingness



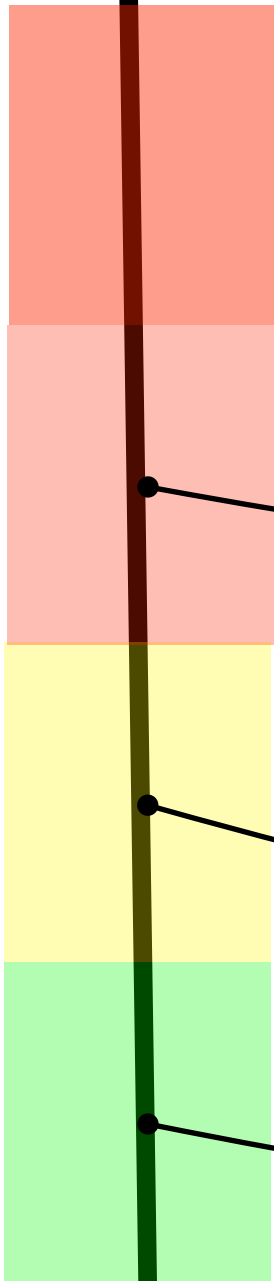
$\geq \Pi_2$

Σ_1

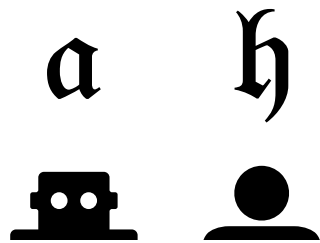
Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy



Demandingness



$\geq \Pi_2$

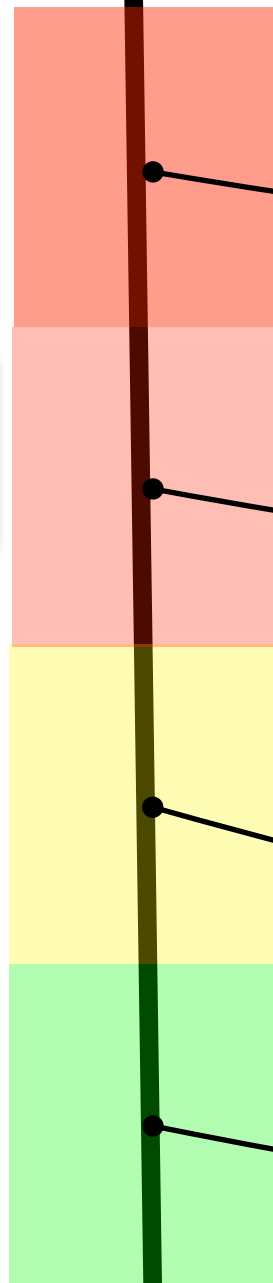
Σ_1

Analytic Hierarchy

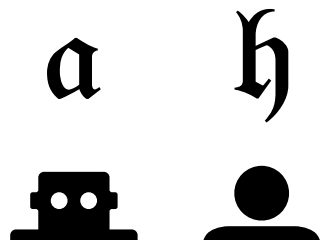
Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy



Demandingness



$\geq \Pi_2$

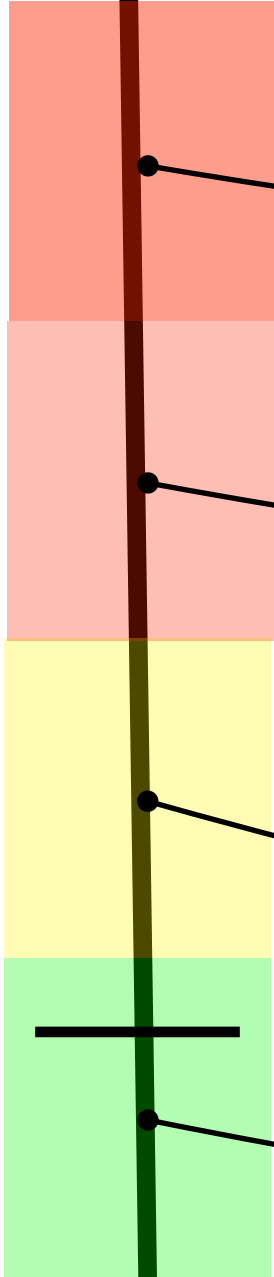
Σ_1

Analytic Hierarchy

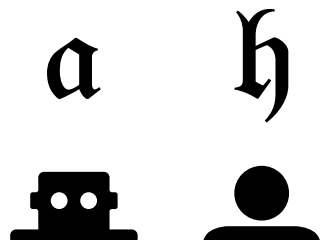
Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy

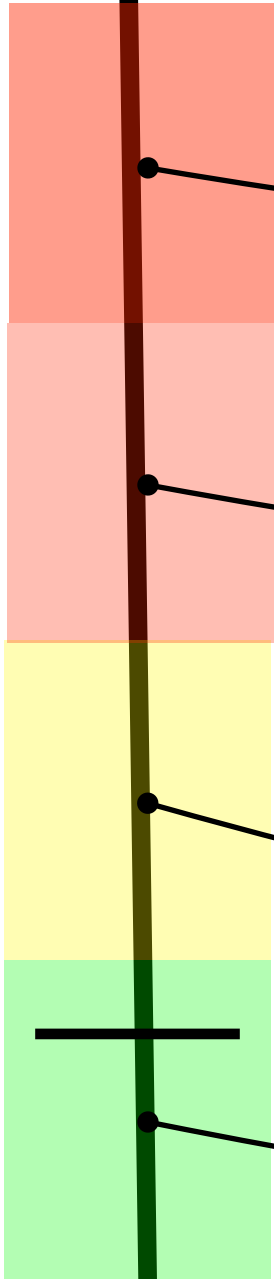


Demandingness



$\geq \Pi_2$

Σ_1



Dockworking; NP-Complete

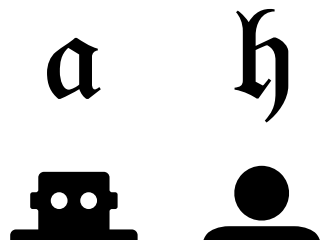
Analytic Hierarchy

Arithmetic Hierarchy

Exponential Hierarchy

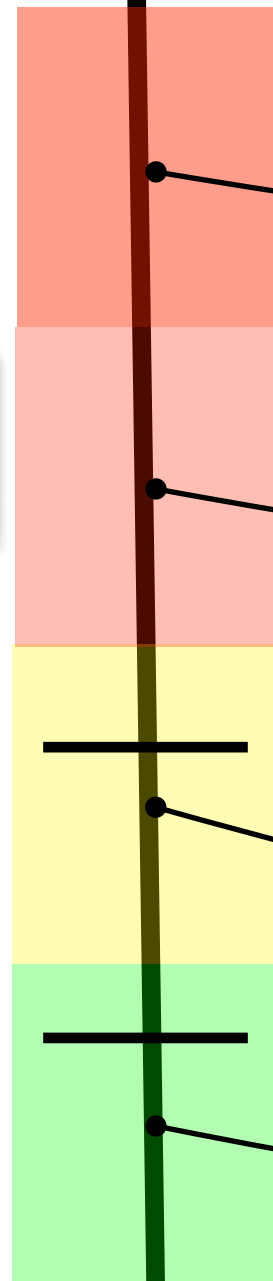
Polynomial Hierarchy

Demandingness



$\geq \Pi_2$

Σ_1



Dockworking; NP-Complete

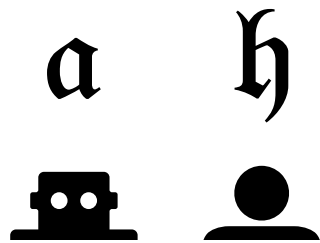
Analytic Hierarchy

Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy

Demandingness



$\geq \Pi_2$

Σ_1

Pro Checkers/Chess/Go;
EXPTIME

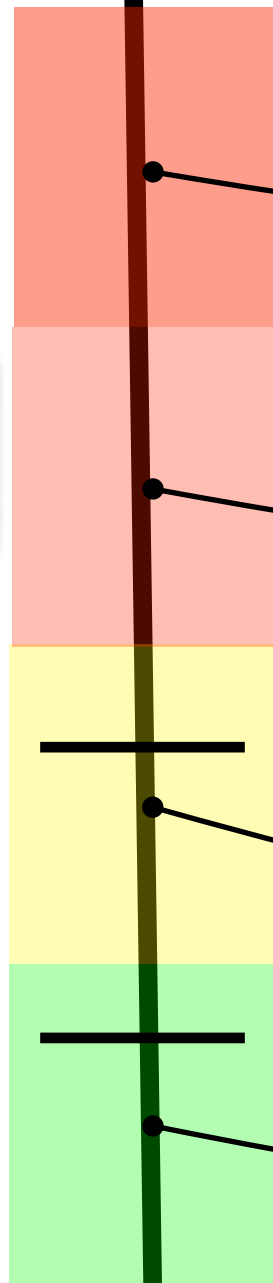
Dockworking; NP-Complete

Analytic Hierarchy

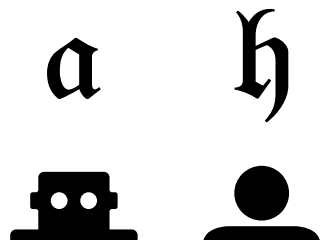
Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy



Demandingness



$\geq \Pi_2$

Σ_1

Pro Checkers/Chess/Go;
EXPTIME

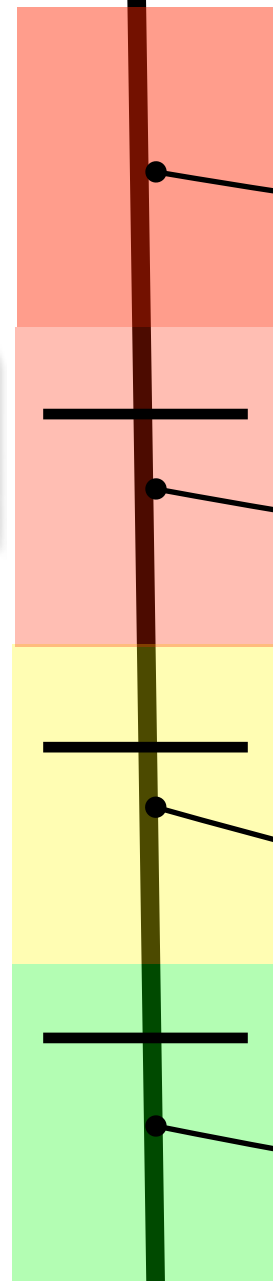
Dockworking; NP-Complete

Analytic Hierarchy

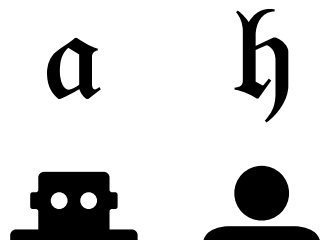
Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy

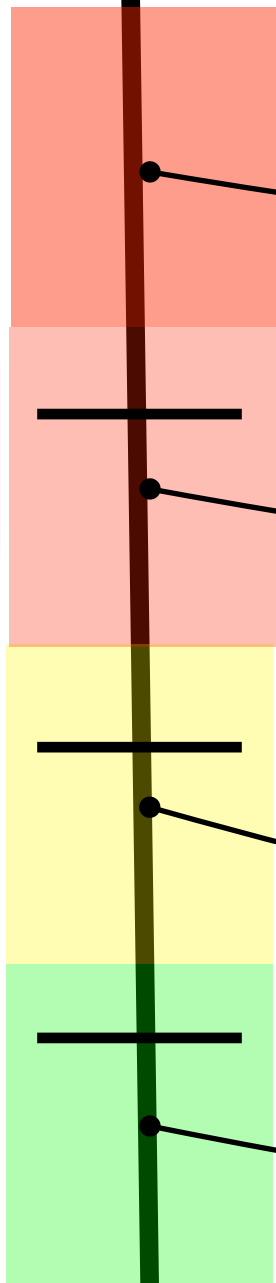


Demandingness



$\geq \Pi_2$

Σ_1



Logic Programming

Pro Checkers/Chess/Go;
EXPTIME

Dockworking; NP-Complete

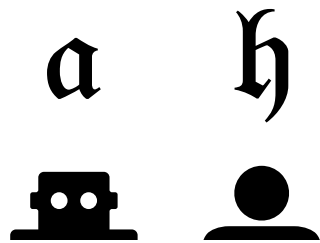
Analytic Hierarchy

Arithmetic Hierarchy

Exponential Hierarchy

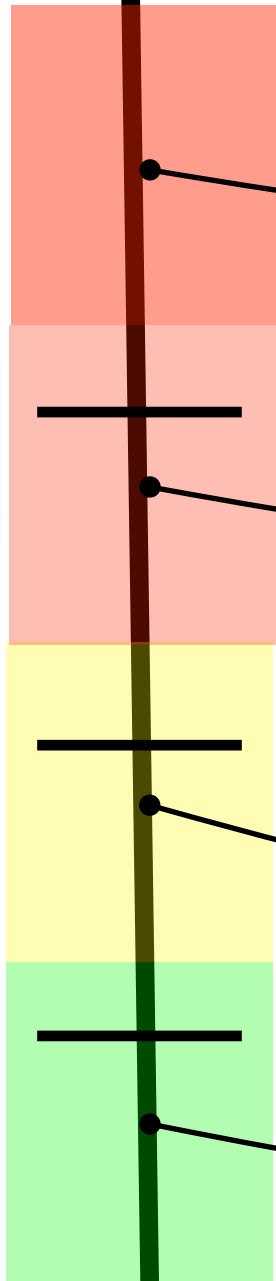
Polynomial Hierarchy

Demandingness



$\geq \Pi_2$

Σ_1



Automated Planning/Corporate Strategy

Logic Programming

Pro Checkers/Chess/Go;
EXPTIME

Dockworking; NP-Complete

Analytic Hierarchy

Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy

Demandingness

α β

$\geq \Pi_2$

Σ_1

Analytic Hierarchy

Automated Planning/Corporate Strategy

Arithmetic Hierarchy

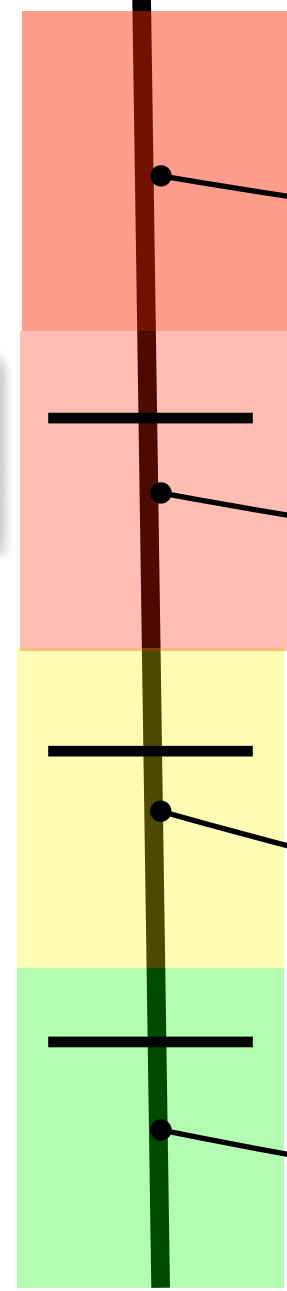
Logic Programming

Exponential Hierarchy

Pro Checkers/Chess/Go;
EXPTIME

Polynomial Hierarchy

Dockworking; NP-Complete



Demandingness

a **h**

$\geq \Pi_2$

Σ_1

Analytic Hierarchy

Automated Planning/Corporate Strategy

Arithmetic Hierarchy

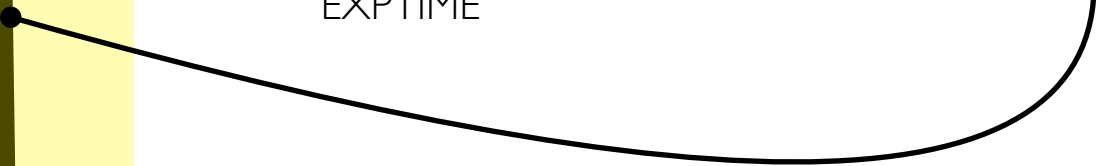
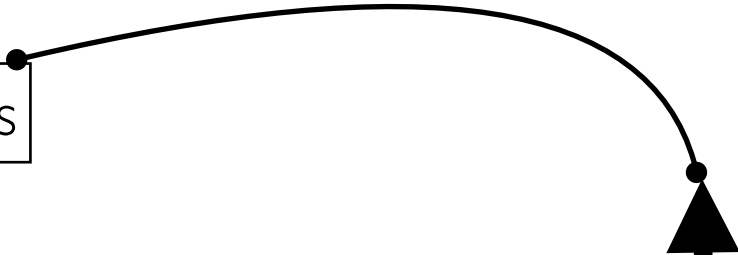
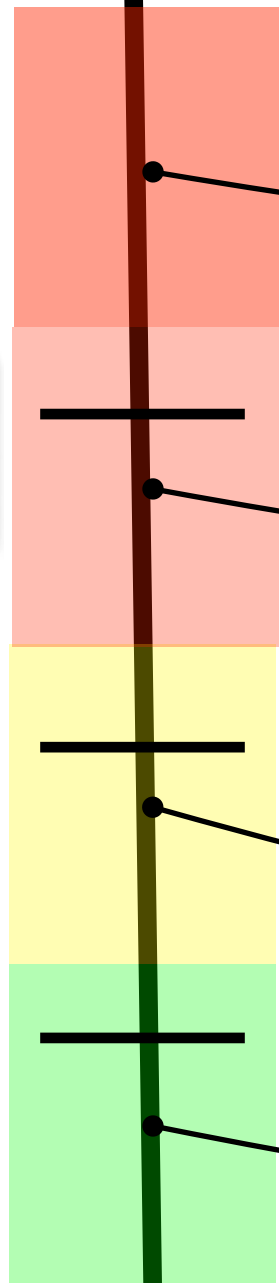
Logic Programming

Exponential Hierarchy

Pro Checkers/Chess/Go;
EXPTIME

Polynomial Hierarchy

Dockworking; NP-Complete



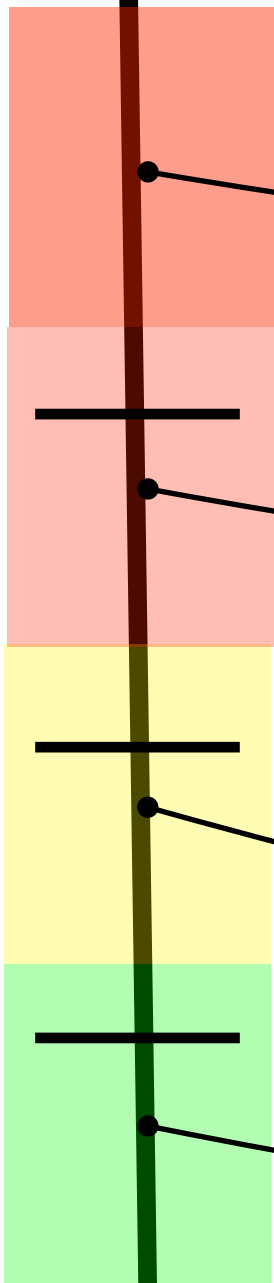
Demandingness

α $\mathfrak{h}$

$\geq \Pi_2$



Σ_1



Automated Planning/Corporate Strategy

Logic Programming

Pro Checkers/Chess/Go;
EXPTIME

Dockworking; NP-Complete

Analytic Hierarchy

Arithmetic Hierarchy

Exponential Hierarchy

Polynomial Hierarchy

Demandingness



$\geq \Pi_2$



Σ_1

Analytic Hierarchy

Automated Planning/Corporate Strategy

Arithmetic Hierarchy

Logic Programming

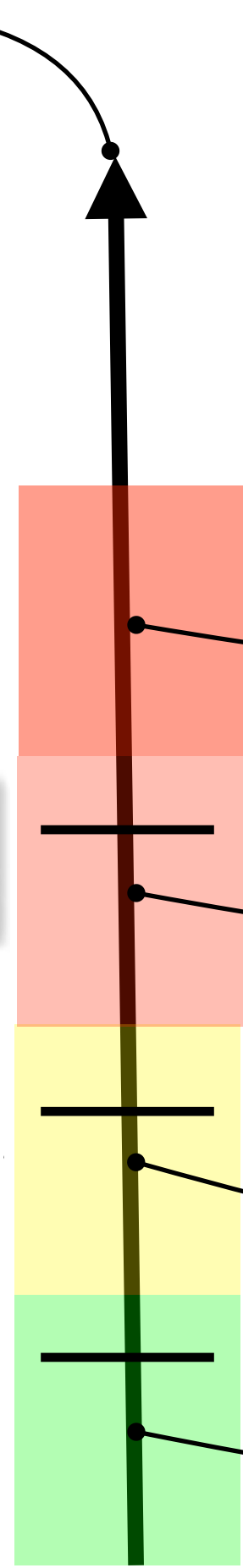
Exponential Hierarchy

Pro Checkers/Chess/Go;
EXPTIME

Polynomial Hierarchy

Dockworking; NP-Complete

$\mathfrak{a}$ $\mathfrak{h}$



Demandingness



$\geq \Pi_2$



Σ_1

Analytic Hierarchy

Automated Planning/Corporate Strategy

Arithmetic Hierarchy

Logic Programming

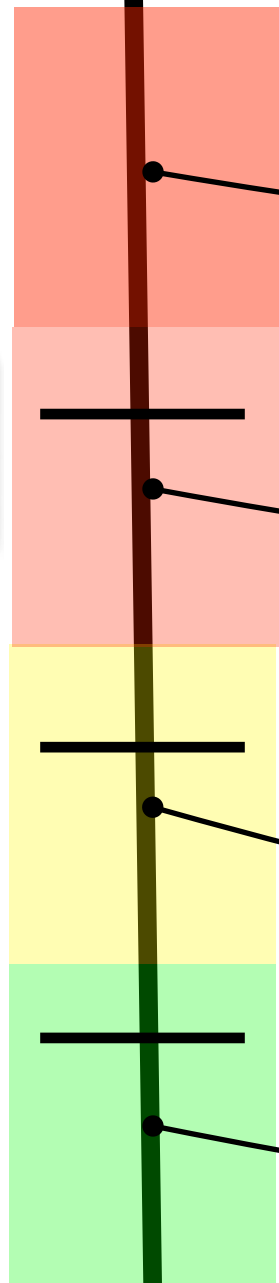
Exponential Hierarchy

Pro Checkers/Chess/Go;
EXPTIME

Polynomial Hierarchy

Dockworking; NP-Complete

$\mathfrak{a}$ $\mathfrak{h}$



Silly Alternative Scale

Leopold Aschenbrenner

SITUATIONAL AWARENESS

The Decade Ahead

JUNE 2024

Silly Alternative Scale

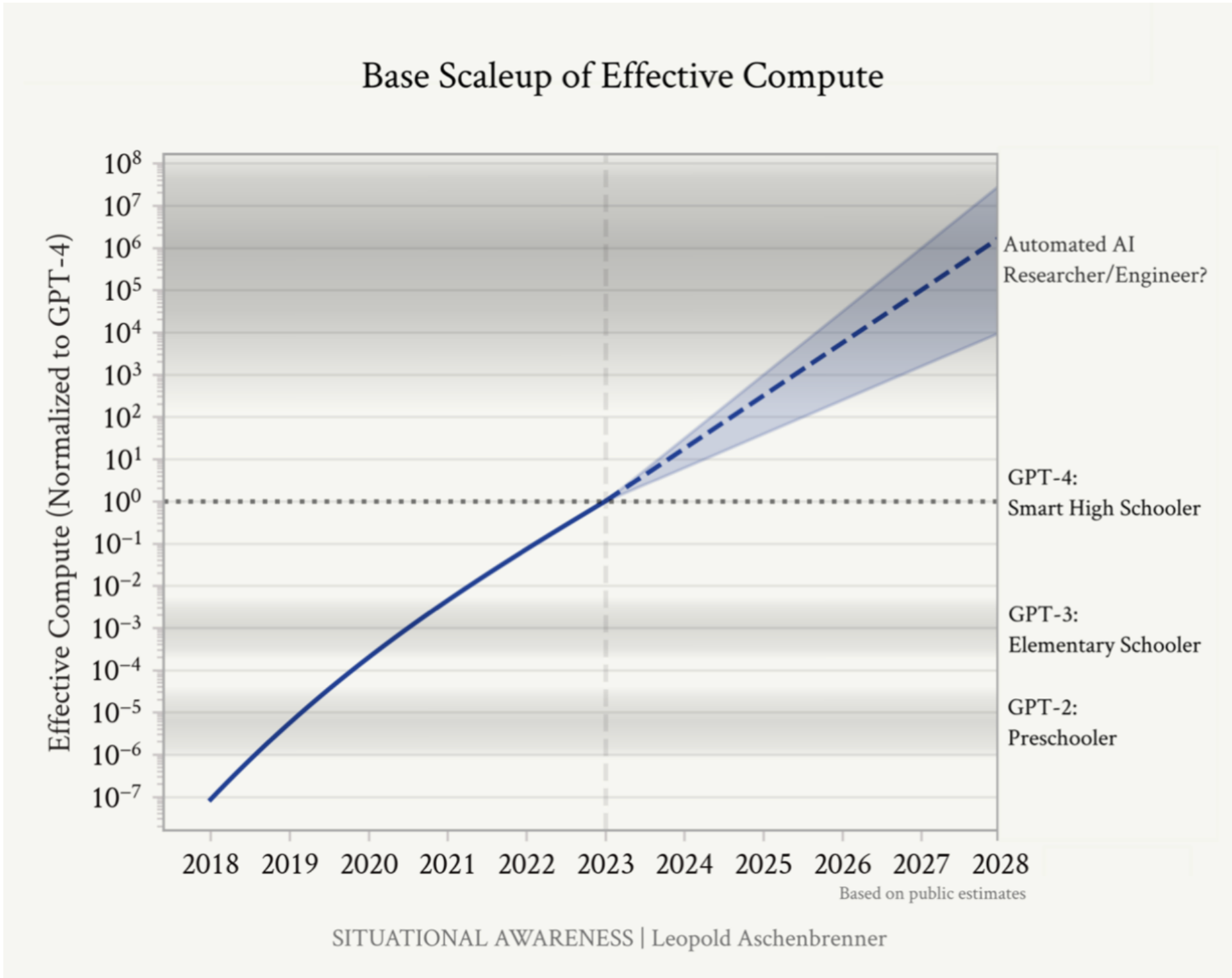
Leopold Aschenbrenner

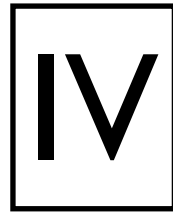
SITUATIONAL AWARENESS

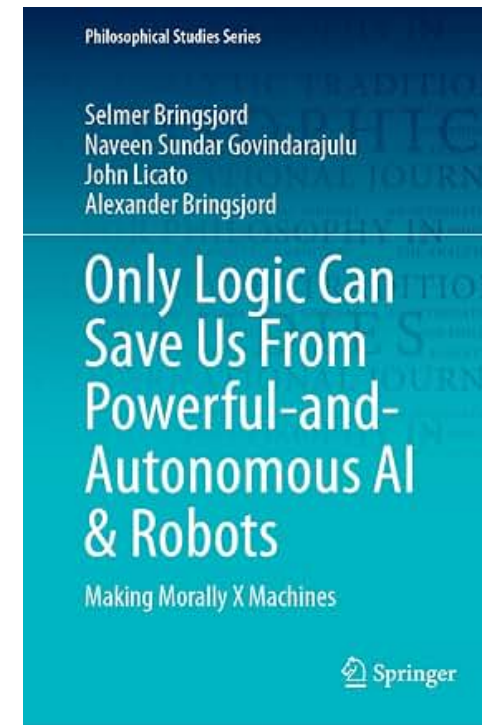
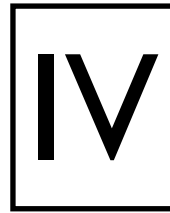
The Decade Ahead

JUNE 2024

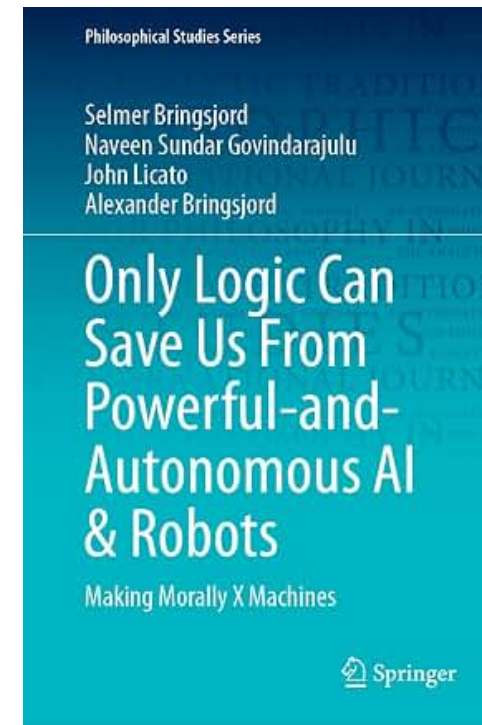
Silly Alternative Scale







IV



Logikk kan også gi dyp glede!