

Pure General Logic Programming (PGLP)

Selmer Bringsjord
(with Naveen Sundar G.)

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

IFLAI2
Nov 5 2020
ver. 1106200830NY



Some Logistics

Some Logistics

- Why are human beings touching ballots (or high-stakes exams) instead of just AIs?

Some Logistics

- Why are human beings touching ballots (or high-stakes exams) instead of just AIs?
- Office Hrs scheduling understood?

Some Logistics

- Why are human beings touching ballots (or high-stakes exams) instead of just AIs?
- Office Hrs scheduling understood?
- Questions about **3SAT**? I did it quickly.

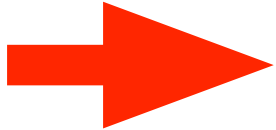
Some Logistics

- Why are human beings touching ballots (or high-stakes exams) instead of just AIs?
- Office Hrs scheduling understood?
- Questions about **3SAT**? I did it quickly.
- Let's visit Overleaf, & our paper-topic file ...

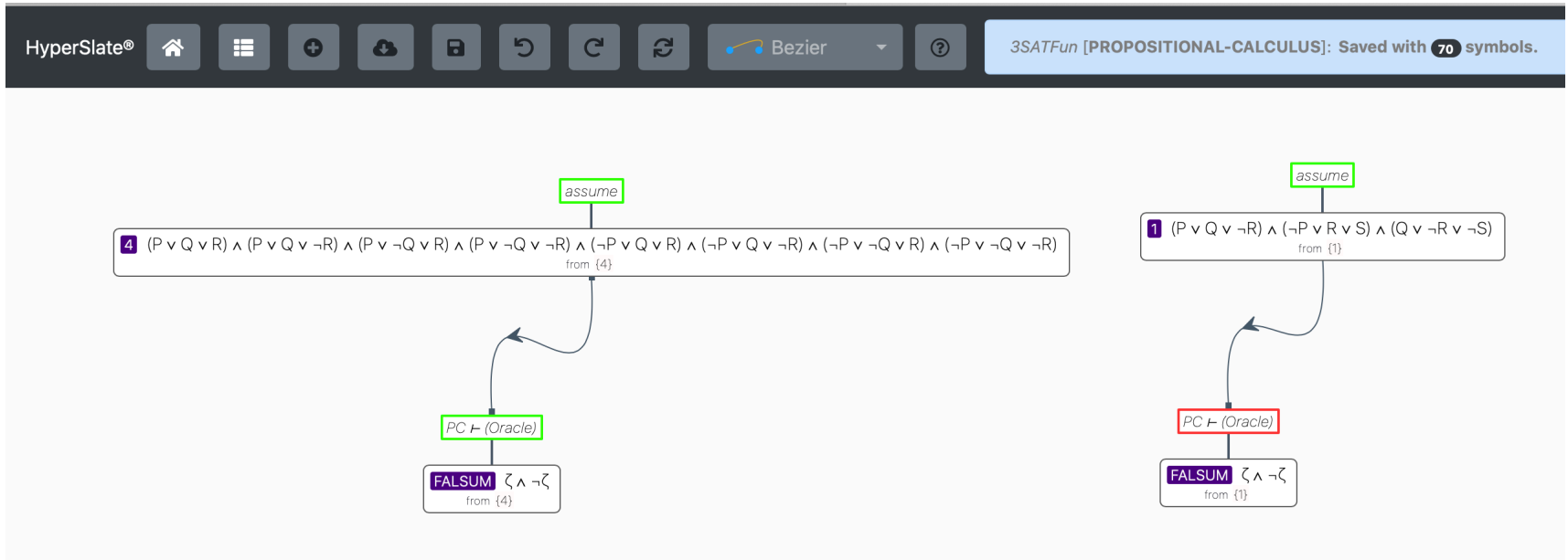
Some Logistics

- Why are human beings touching ballots (or high-stakes exams) instead of just AIs?
- Office Hrs scheduling understood?
- Questions about **3SAT**? I did it quickly.
- Let's visit Overleaf, & our paper-topic file ...
- Let's visit our web site ...

Some Logistics

- Why are human beings touching ballots (or high-stakes exams) instead of just AIs?
- Office Hrs scheduling understood?
- Questions about **3SAT**? I did it quickly. 
- Let's visit Overleaf, & our paper-topic file ...
- Let's visit our web site ...

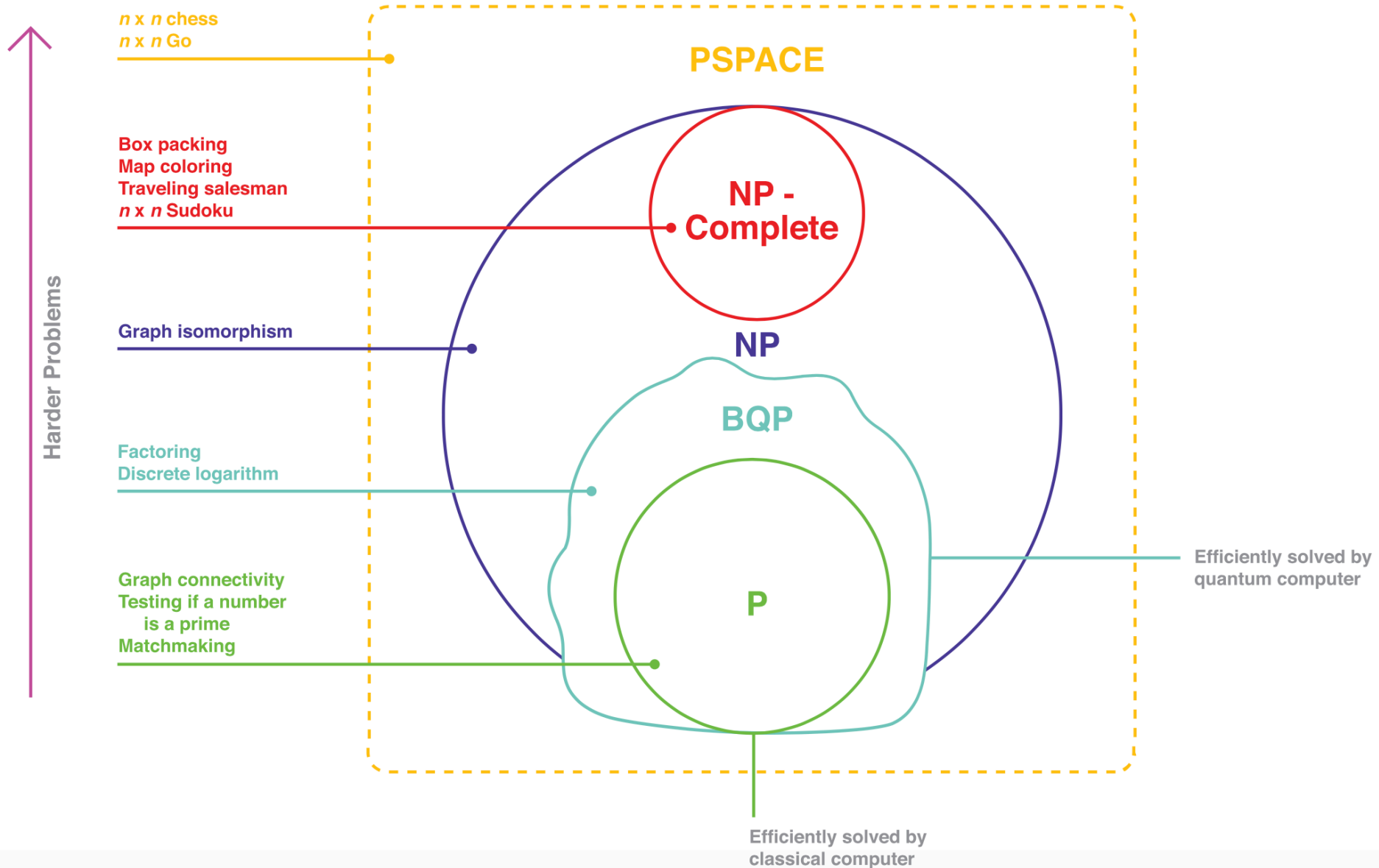
3SATFun



Next, delivering on part
of promissory note ...

What about (oft vaunted) quantum computers?

What about (oft vaunted) quantum computers?



What about (oft vaunted) quantum computers?



GCI

Harder Problems

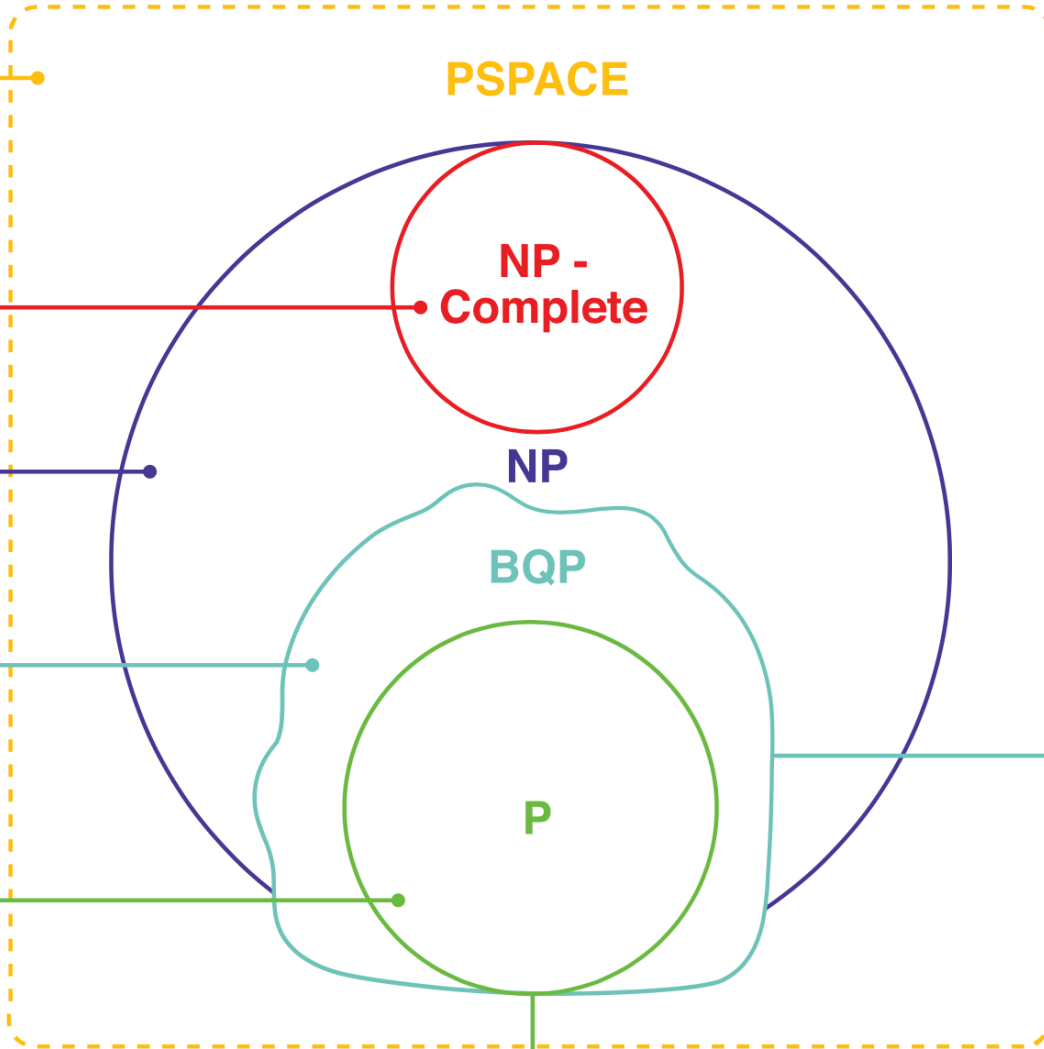
$n \times n$ chess
 $n \times n$ Go

Box packing
Map coloring
Traveling salesman
 $n \times n$ Sudoku

Graph isomorphism

Factoring
Discrete logarithm

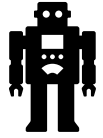
Graph connectivity
Testing if a number
is a prime
Matchmaking



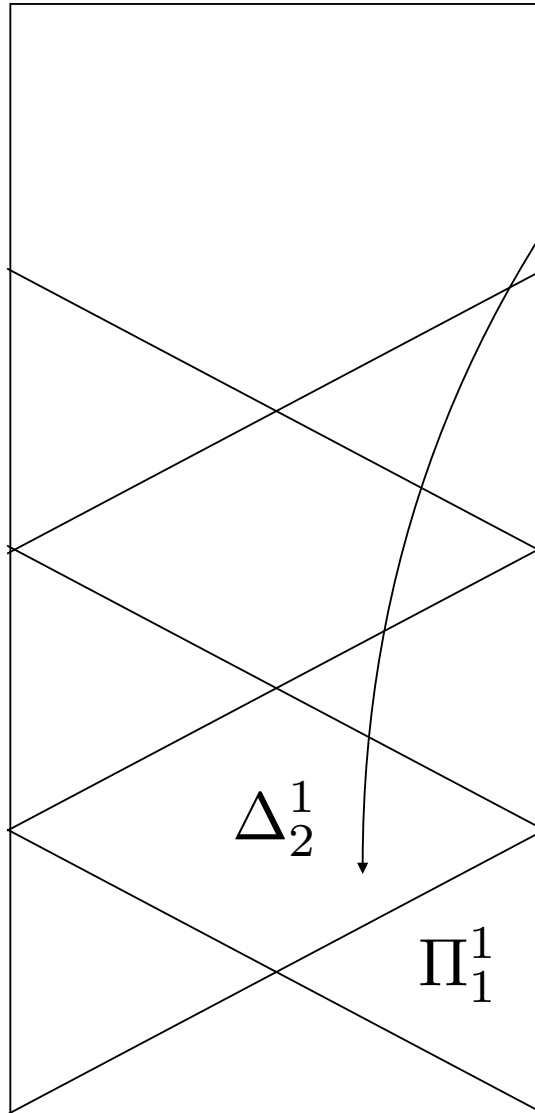
Efficiently solved by quantum computer

Efficiently solved by classical computer

CogSci and AI need to say more about where AI falls/can fall in the landscape.

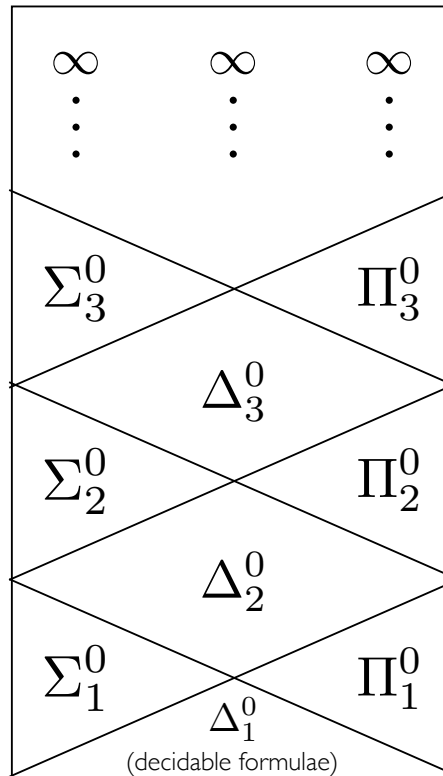


$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

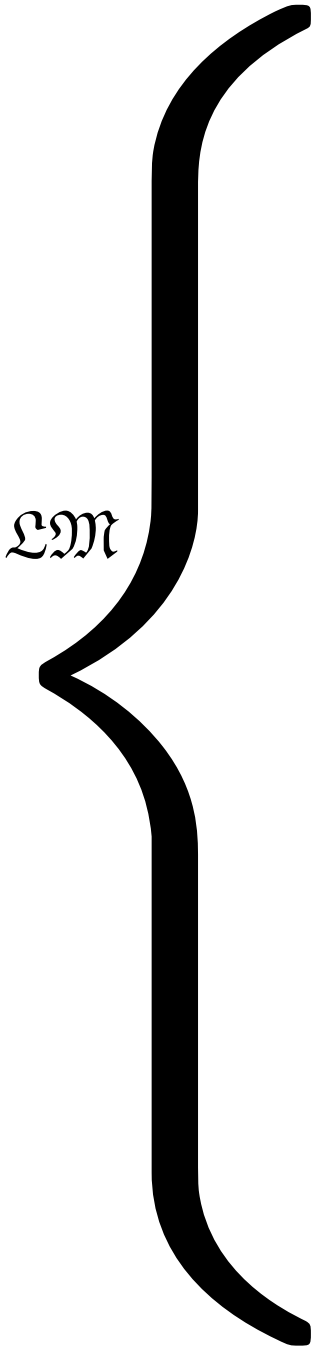
Human Brains (according to Granger)



\mathcal{CH} (Chomsky Hierarchy)

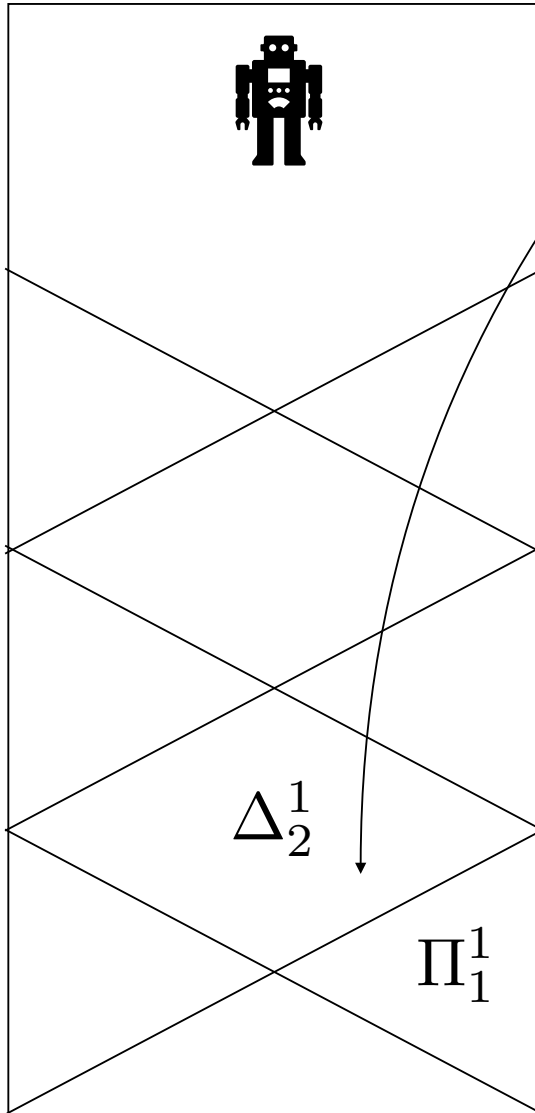
Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)

\mathcal{EM}



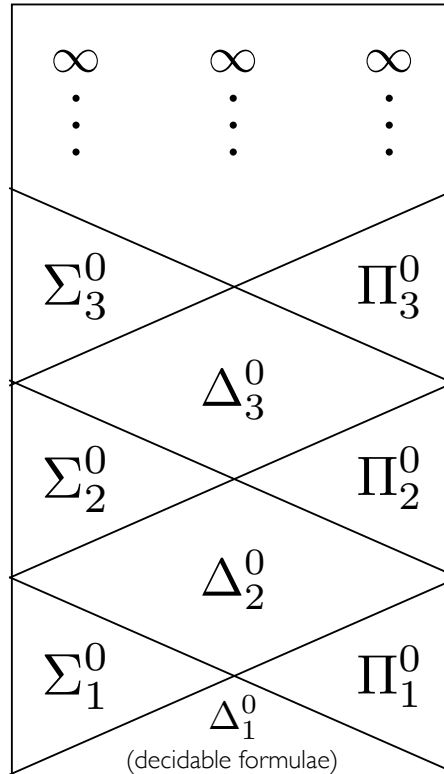
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

Human Brains (according to Granger)



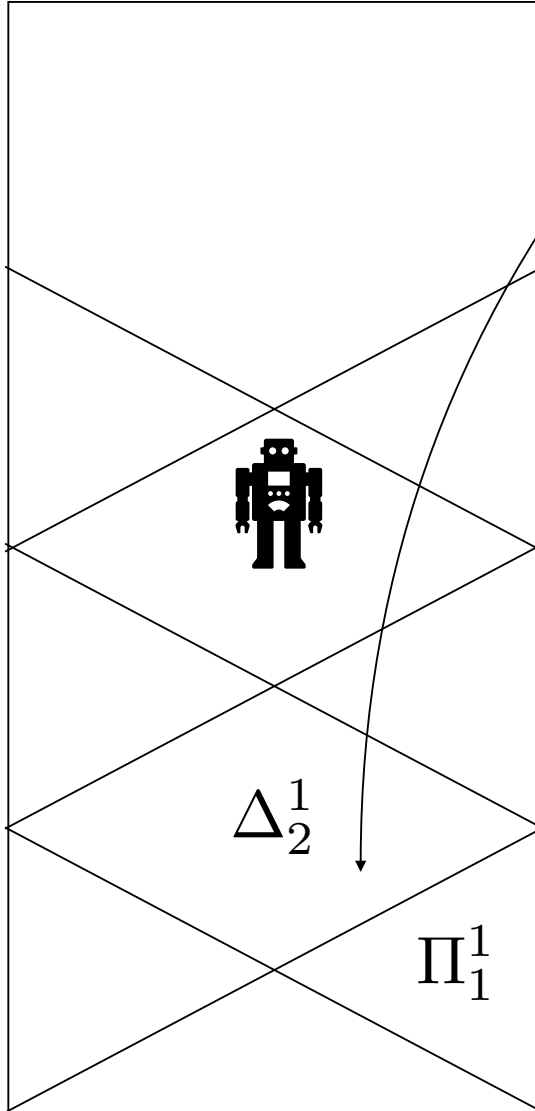
\mathcal{CH} (Chomsky Hierarchy)

Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)

\mathcal{EM}

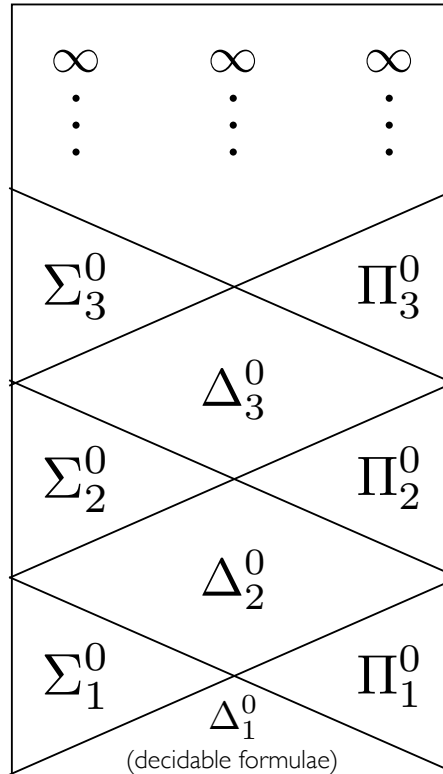
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

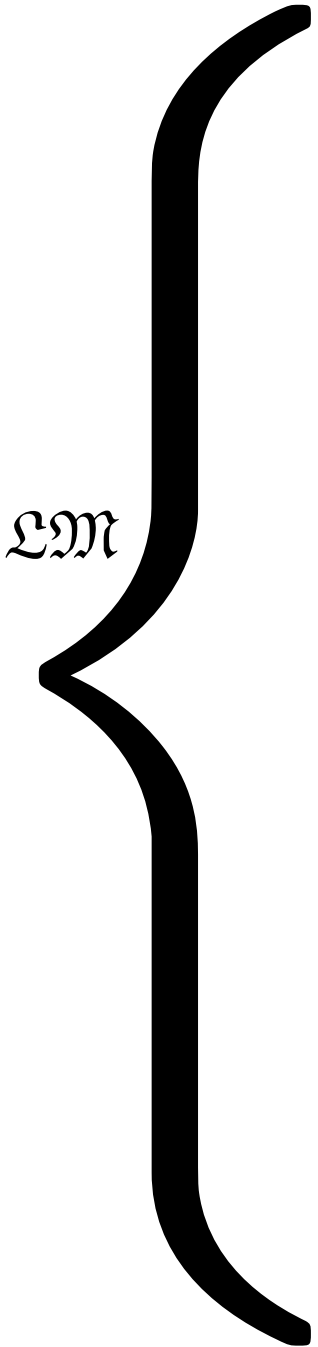
Human Brains (according to Granger)



\mathcal{CH} (Chomsky Hierarchy)

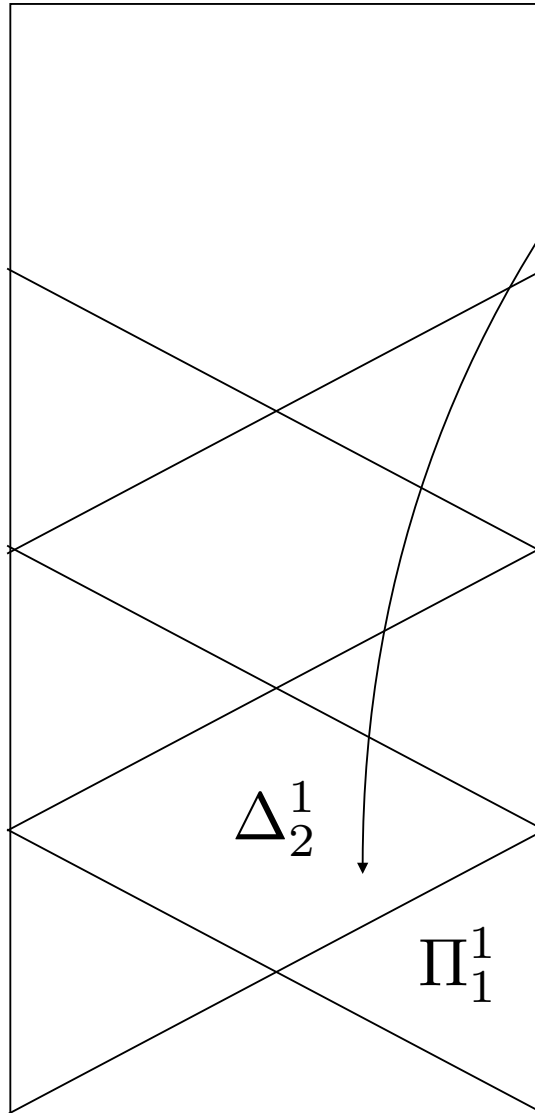
Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)

EM



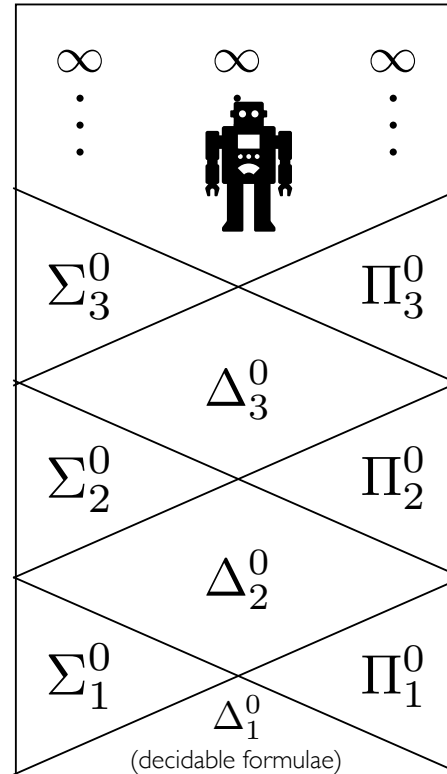
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

Human Brains (according to Granger)



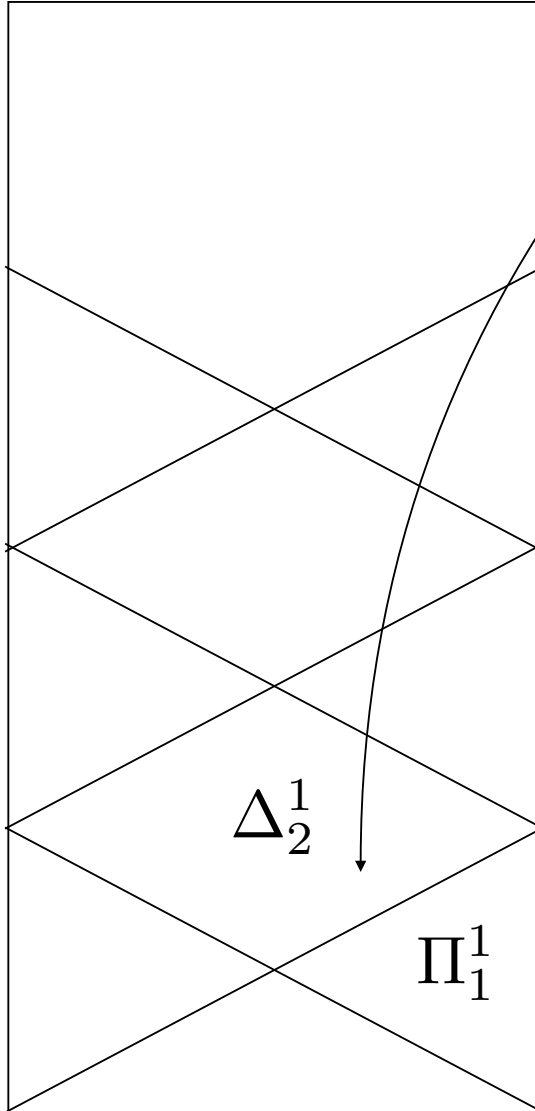
\mathcal{CH} (Chomsky Hierarchy)

Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)



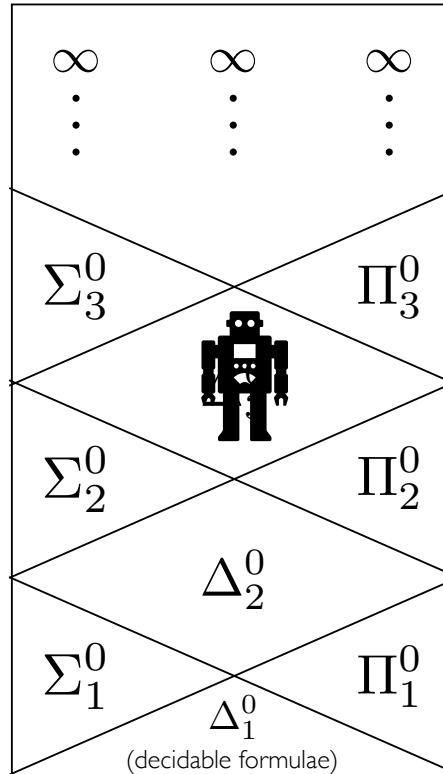
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons
(according to Bringsjord)

Human Brains
(according to Granger)



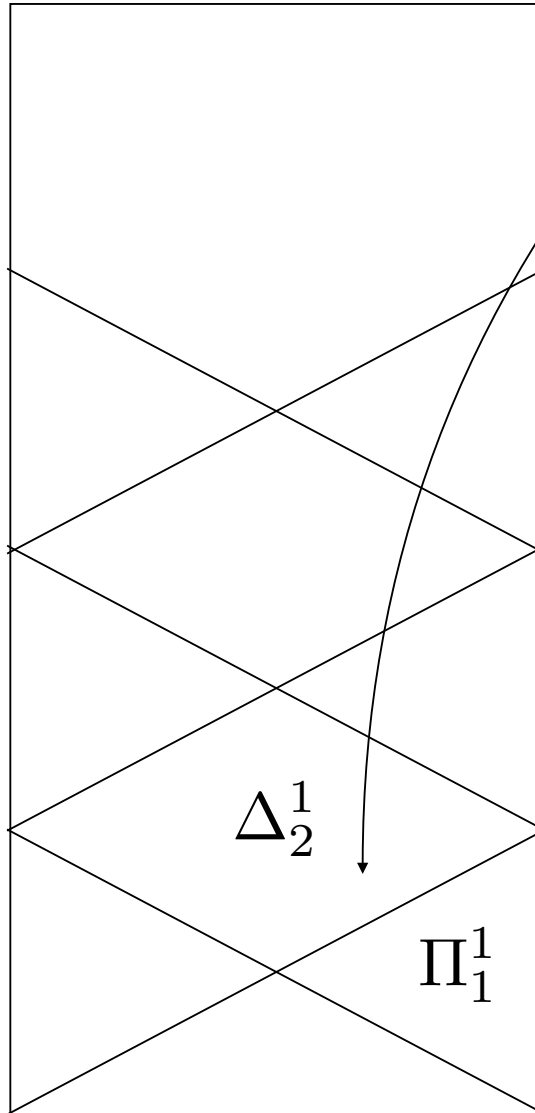
\mathcal{CH} (Chomsky Hierarchy)

Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)



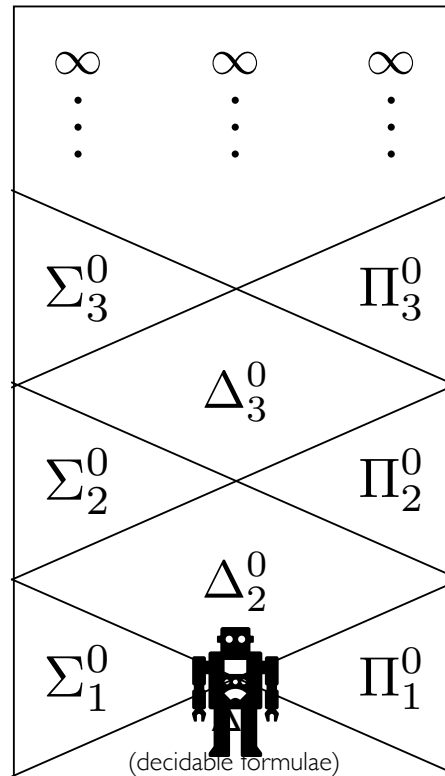
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

Human Brains (according to Granger)



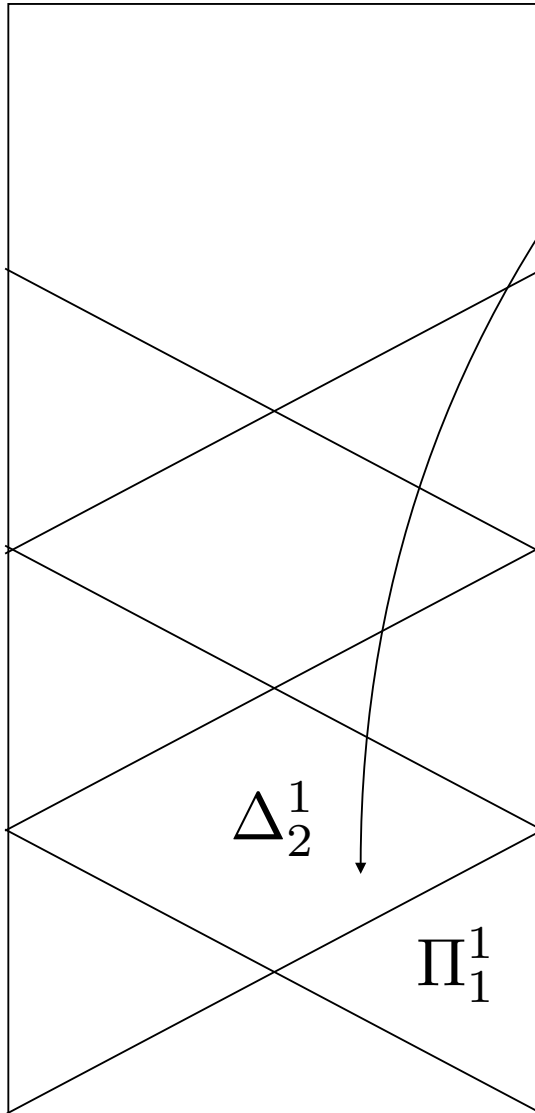
\mathcal{CH} (Chomsky Hierarchy)

Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)



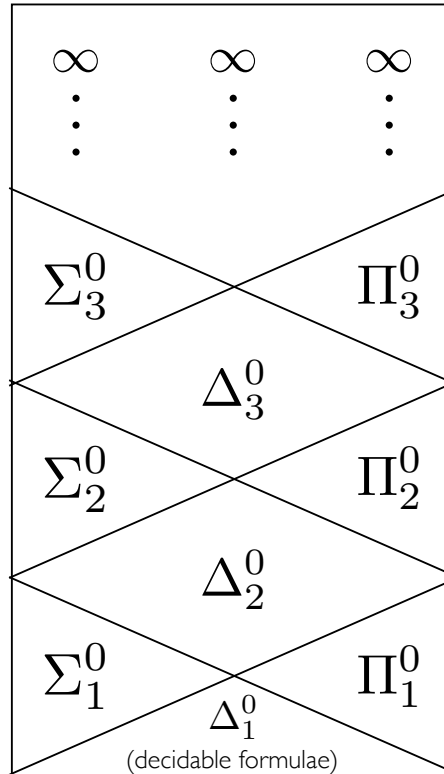
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)

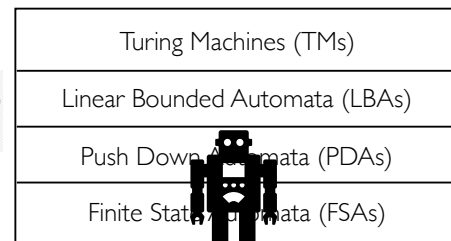


Human Persons (according to Bringsjord)

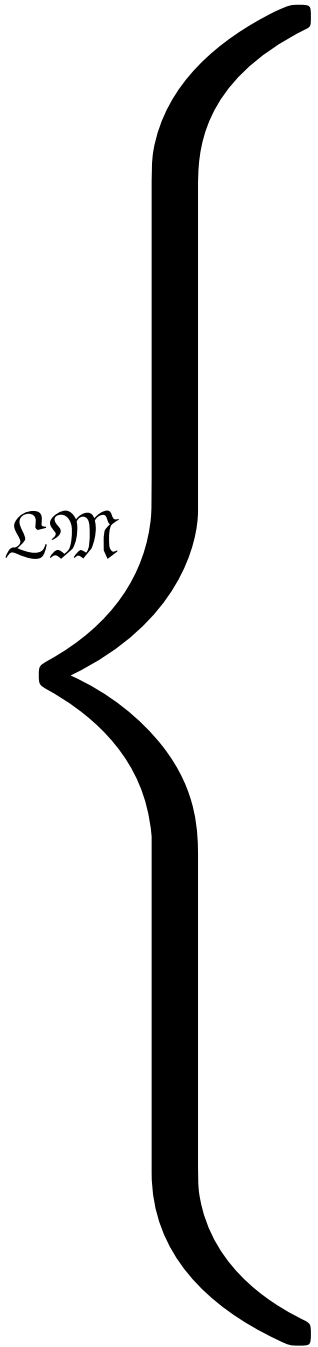
Human Brains (according to Granger)



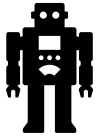
\mathcal{CH} (Chomsky Hierarchy)



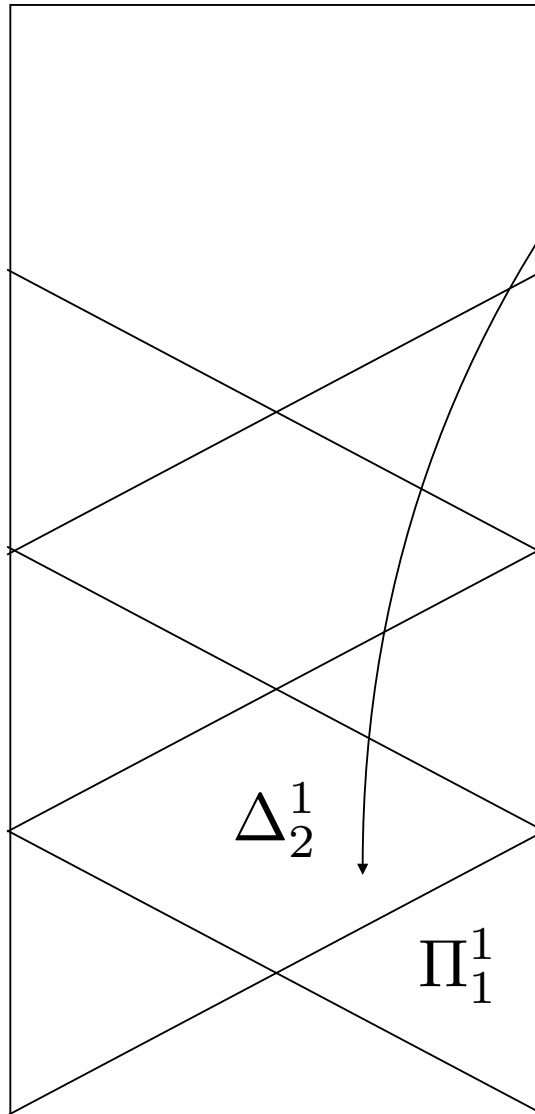
EM



CogSci and AI need to say more about where AI falls/can fall in the landscape.

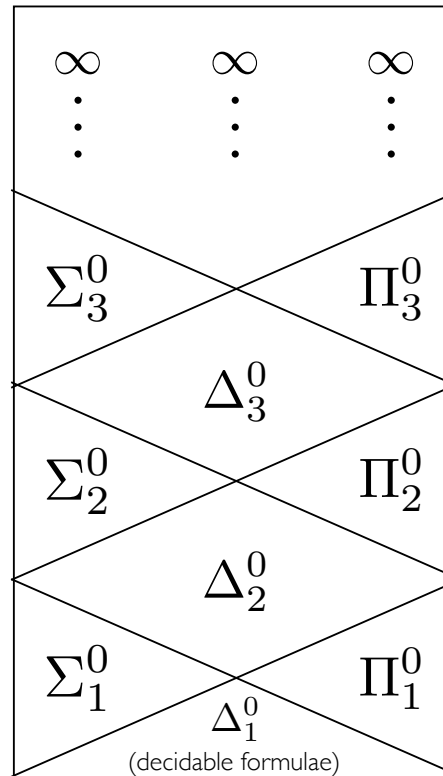


$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons
(according to Bringsjord)

Human Brains
(according to Granger)



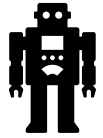
\mathcal{CH} (Chomsky Hierarchy)

Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)

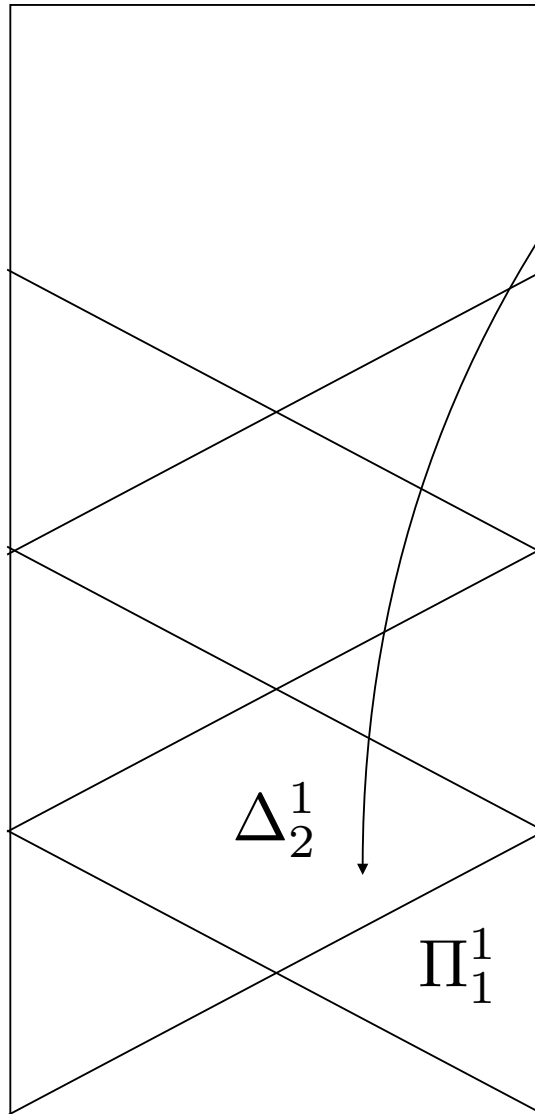


EM

CogSci and AI need to say more about where AI falls/can fall in the landscape.

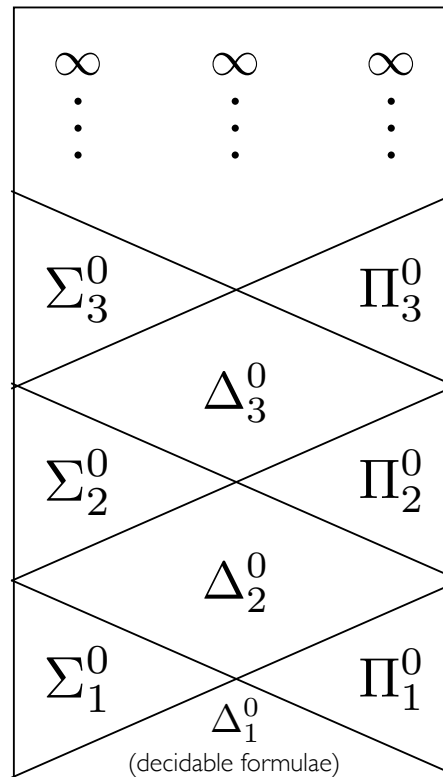


$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

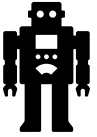
Human Brains (according to Granger)



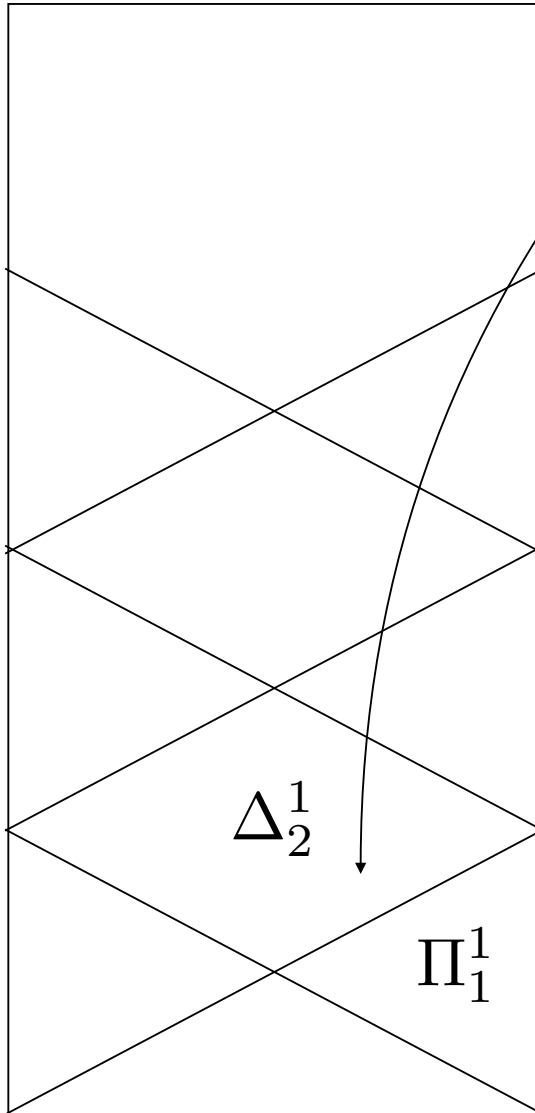
\mathcal{CH} (Chomsky Hierarchy)

Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)

CogSci and AI need to say more about where AI falls/can fall in the landscape.

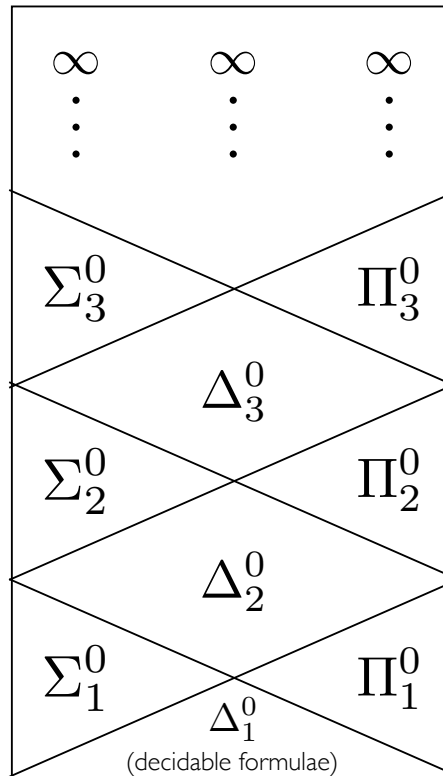


$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

Human Brains (according to Granger)

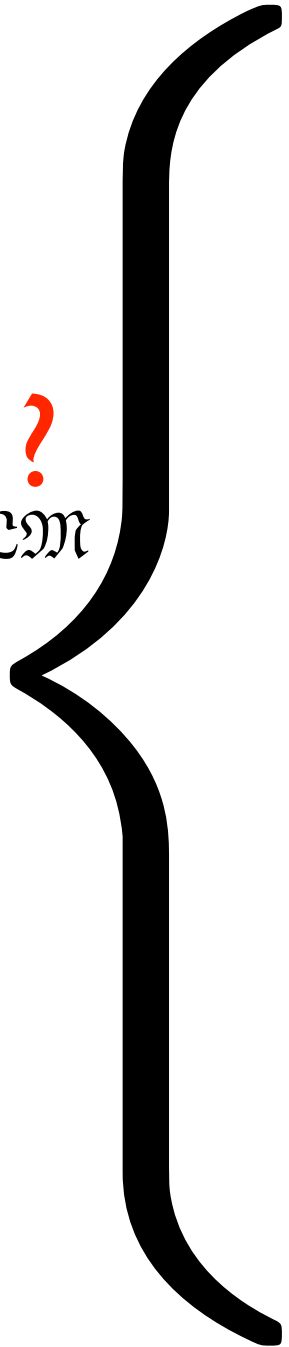


\mathcal{CH} (Chomsky Hierarchy)

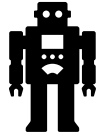
Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)



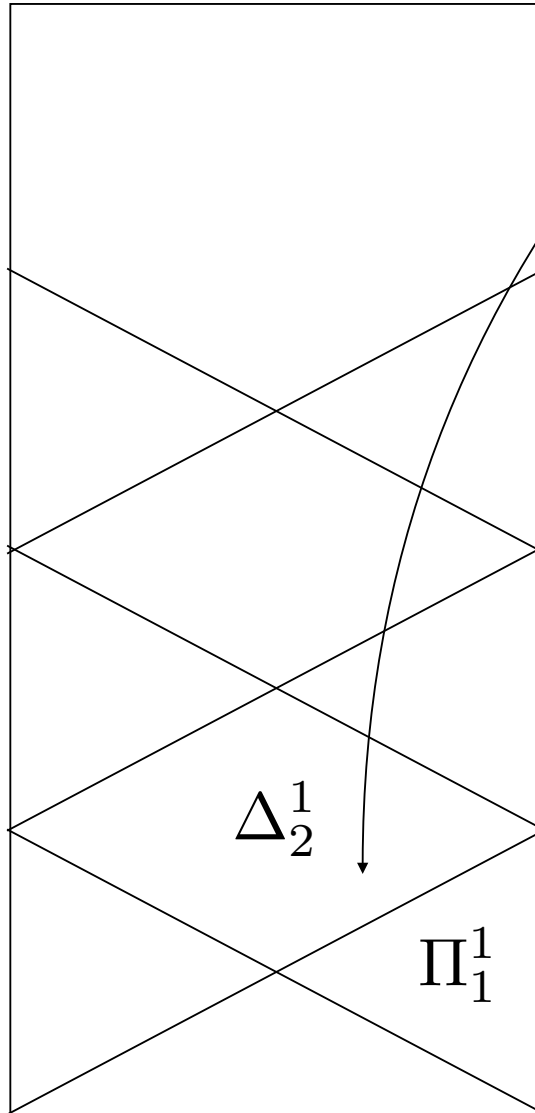
?
EM



CogSci and AI need to say more about where AI falls/can fall in the landscape.

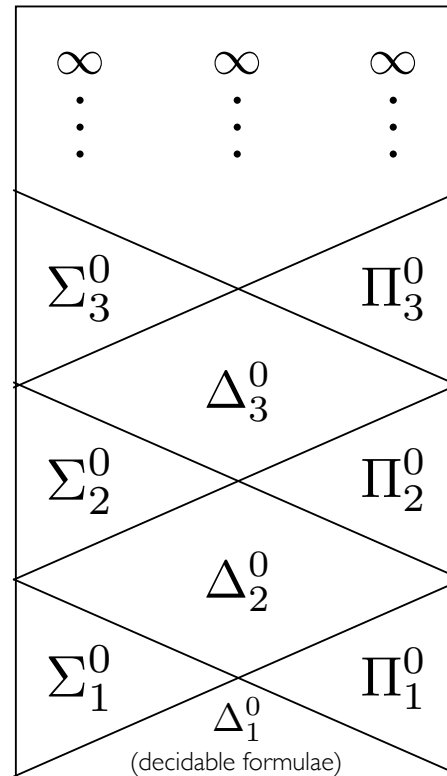


$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)



Human Persons (according to Bringsjord)

Human Brains (according to Granger)

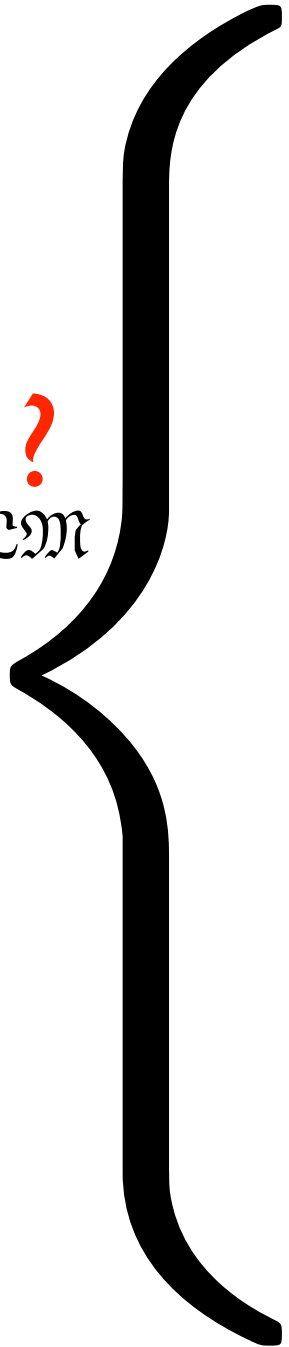


\mathcal{CH} (Chomsky Hierarchy)

- Turing Machines (TMs)
- Linear Bounded Automata (LBAs)
- Push Down Automata (PDAs)
- Finite State Automata (FSAs)

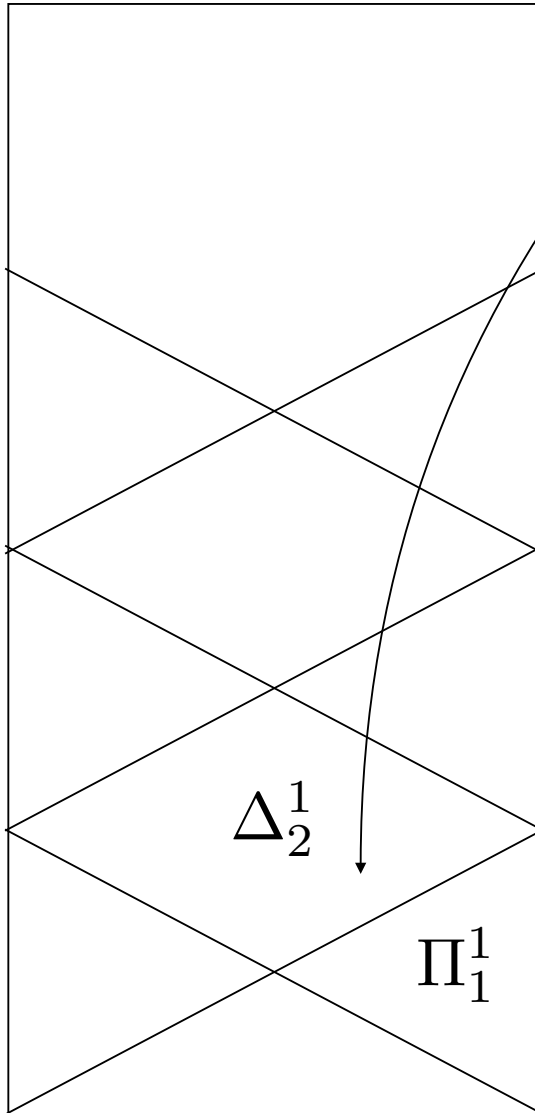


?
EM



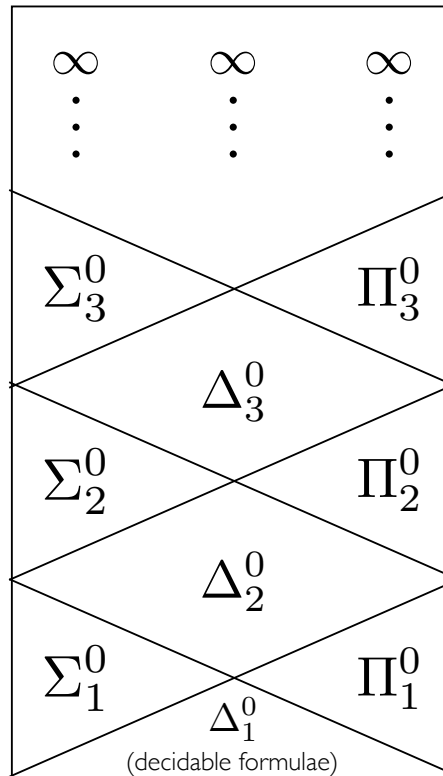
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)

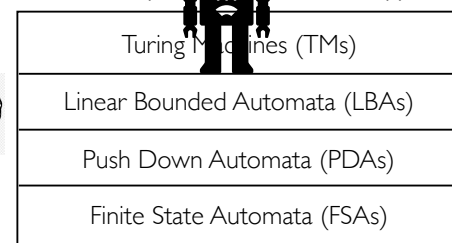


Human Persons (according to Bringsjord)

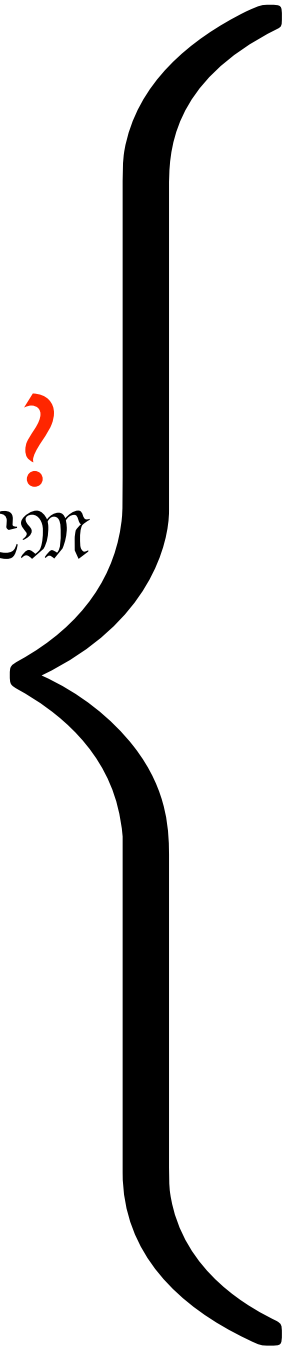
Human Brains (according to Granger)



\mathcal{CH} (Computational Hierarchy)

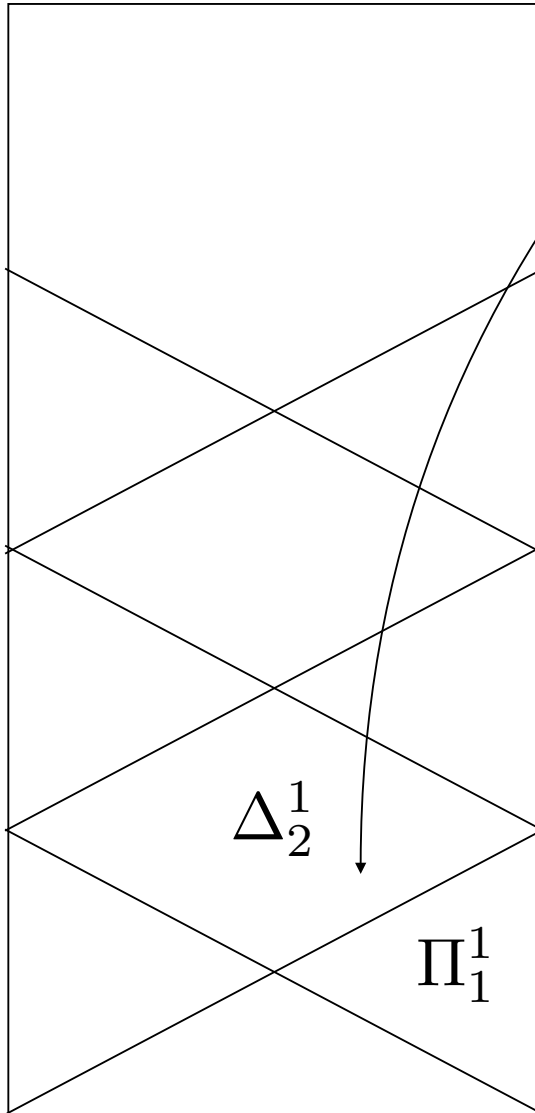


?
 \mathcal{EM}



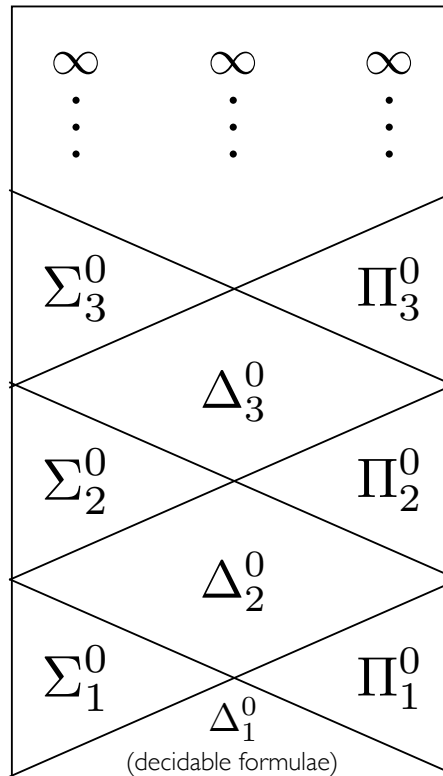
CogSci and AI need to say more about where AI falls/can fall in the landscape.

$A^n \mathcal{H}$ (Analytic Hierarchy)



Infinite Time Turing Machines (ITTMs)

$A^r \mathcal{H}$ (Arithmetic Hierarchy)

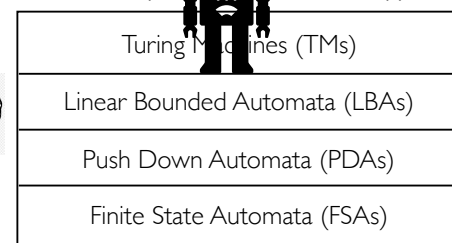


Human Persons (according to Bringsjord)

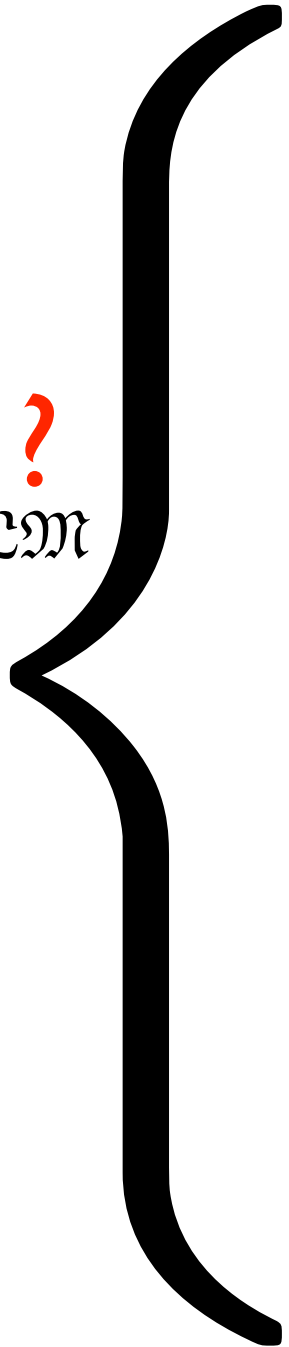
Human Brains (according to Granger)



\mathcal{CH} (Computational Hierarchy)

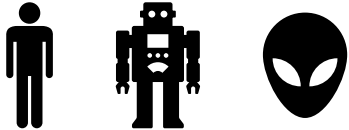


?
EM

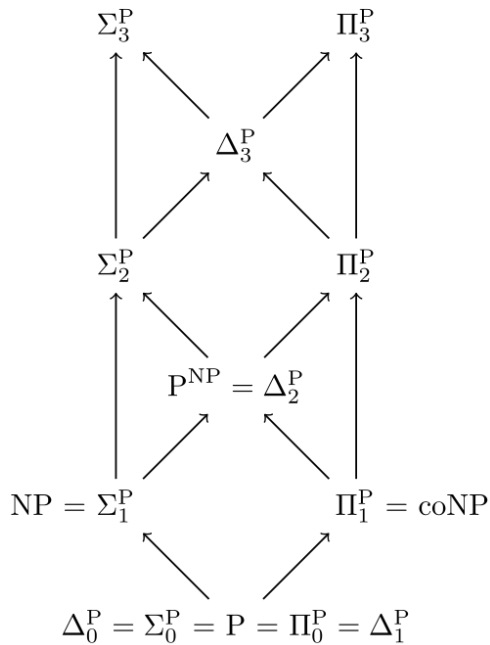


Polynomial Hierarchy, Part I

(via formal logic, directly; a start)



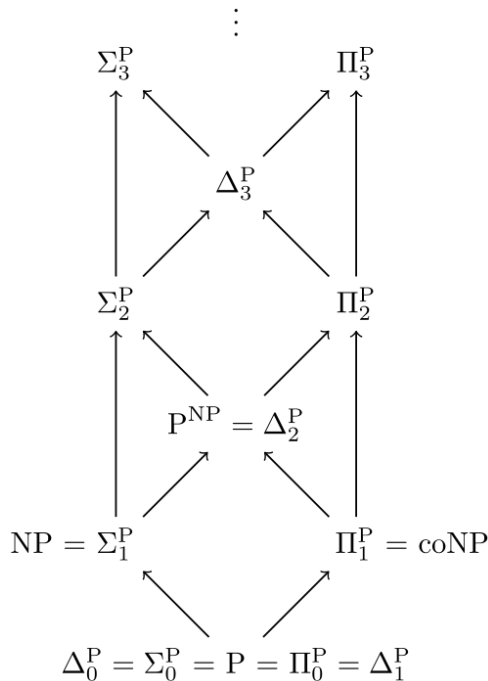
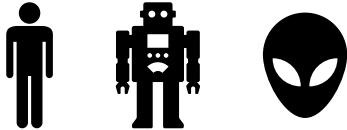
⋮



Polynomial Hierarchy, Part I

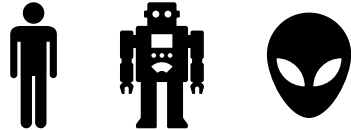
(via formal logic, directly; a start)

We say that a relation $R(u, y_1, \dots, y_n)$ is polytime iff there is a deterministic Turing Machine \mathbf{m} and a polynomial p s.t. \mathbf{m} decides this relation in $p(|u|)$.



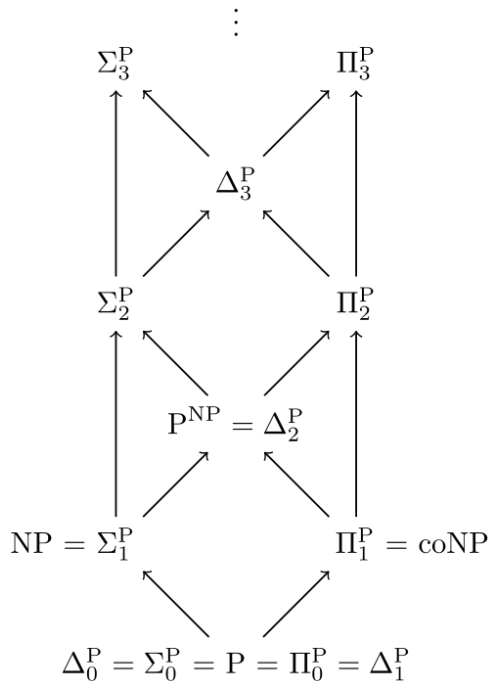
Polynomial Hierarchy, Part I

(via formal logic, directly; a start)



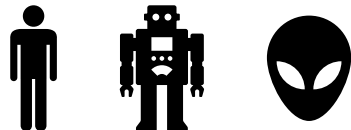
We say that a relation $R(u, y_1, \dots, y_n)$ is polytime iff there is a deterministic Turing Machine \mathbf{m} and a polynomial p s.t. \mathbf{m} decides this relation in $p(|u|)$.

$L \in \mathbf{NP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\exists y R(u, y)$.



Polynomial Hierarchy, Part I

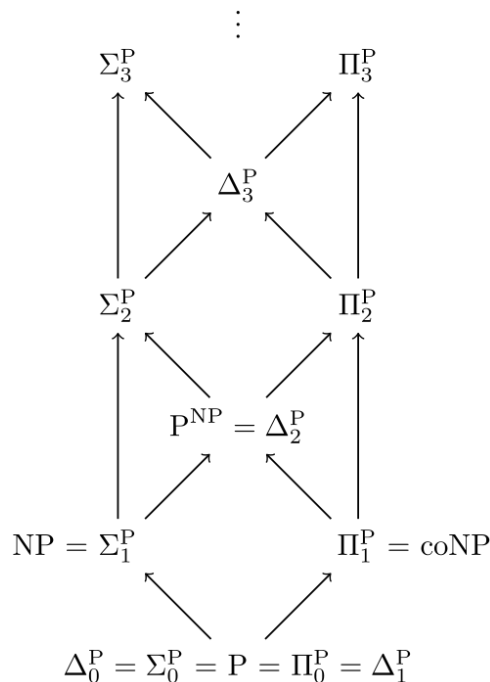
(via formal logic, directly; a start)



We say that a relation $R(u, y_1, \dots, y_n)$ is polytime iff there is a deterministic Turing Machine \mathbf{m} and a polynomial p s.t. \mathbf{m} decides this relation in $p(|u|)$.

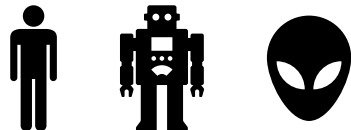
$L \in \mathbf{NP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\exists y R(u, y)$.

E.g.: We can prove $\mathbf{SAT} \in \mathbf{NP}$ because we have a polytime relation R s.t. $\phi \in \mathbf{SAT}$ iff $\exists y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where these assignments produce truth.



Polynomial Hierarchy, Part I

(via formal logic, directly; a start)

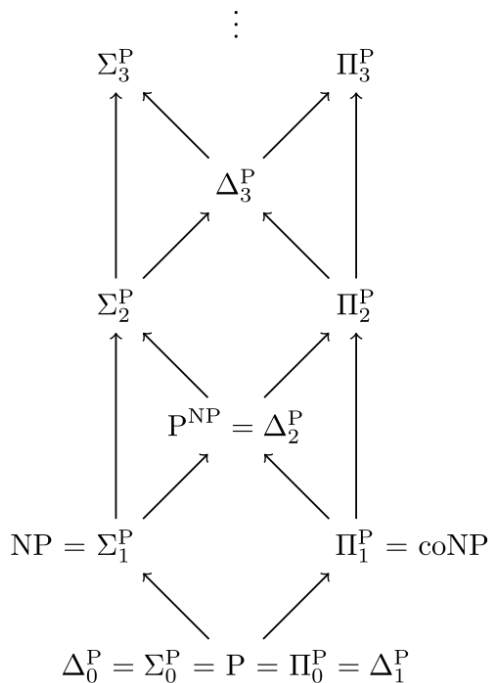


We say that a relation $R(u, y_1, \dots, y_n)$ is polytime iff there is a deterministic Turing Machine \mathbf{m} and a polynomial p s.t. \mathbf{m} decides this relation in $p(|u|)$.

$L \in \mathbf{NP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\exists y R(u, y)$.

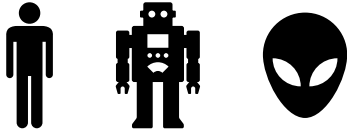
E.g.: We can prove $\mathbf{SAT} \in \mathbf{NP}$ because we have a polytime relation R s.t. $\phi \in \mathbf{SAT}$ iff $\exists y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where these assignments produce truth.

$L \in \mathbf{coNP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\forall y R(u, y)$.



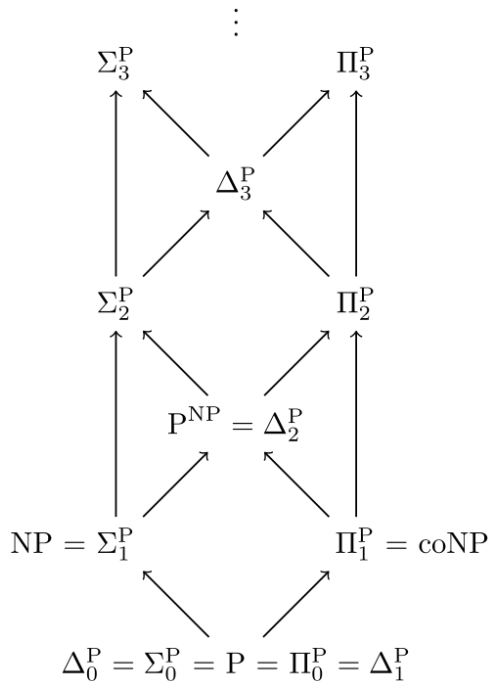
Polynomial Hierarchy, Part I

(via formal logic, directly; a start)



We say that a relation $R(u, y_1, \dots, y_n)$ is polytime iff there is a deterministic Turing Machine \mathbf{m} and a polynomial p s.t. \mathbf{m} decides this relation in $p(|u|)$.

$L \in \mathbf{NP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\exists y R(u, y)$.



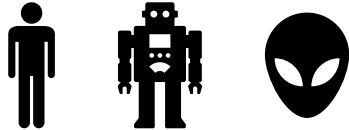
E.g.: We can prove $\mathbf{SAT} \in \mathbf{NP}$ because we have a polytime relation R s.t. $\phi \in \mathbf{SAT}$ iff $\exists y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where these assignments produce truth.

$L \in \mathbf{coNP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\forall y R(u, y)$.

To prove $\mathbf{coSAT} \in \mathbf{coNP}$, we note that we have a polytime relation R s.t. $\phi \in \mathbf{coSAT}$ iff $\forall y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where the assignments produce *falsity*.

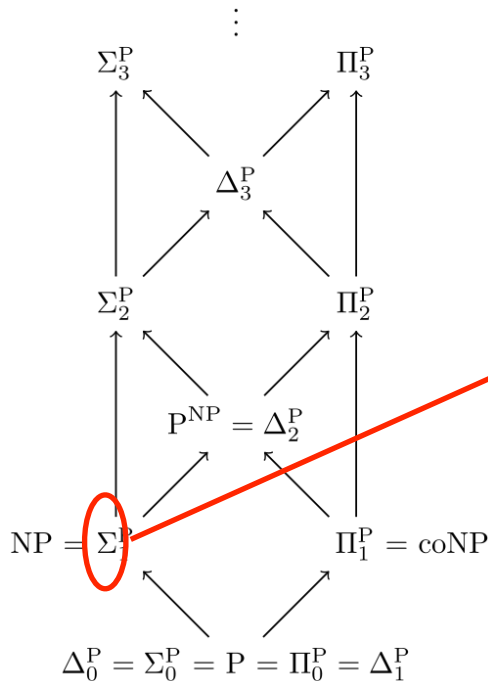
Polynomial Hierarchy, Part I

(via formal logic, directly; a start)



We say that a relation $R(u, y_1, \dots, y_n)$ is polytime iff there is a deterministic Turing Machine \mathbf{m} and a polynomial p s.t. \mathbf{m} decides this relation in $p(|u|)$.

$L \in \mathbf{NP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\exists y R(u, y)$.



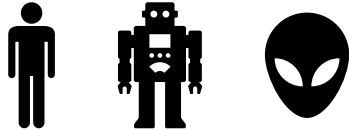
E.g.: We can prove $\mathbf{SAT} \in \mathbf{NP}$ because we have a polytime relation R s.t. $\phi \in \mathbf{SAT}$ iff $\exists y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where these assignments produce truth.

$L \in \mathbf{coNP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\forall y R(u, y)$.

To prove $\mathbf{coSAT} \in \mathbf{coNP}$, we note that we have a polytime relation R s.t. $\phi \in \mathbf{coSAT}$ iff $\forall y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where the assignments produce *falsity*.

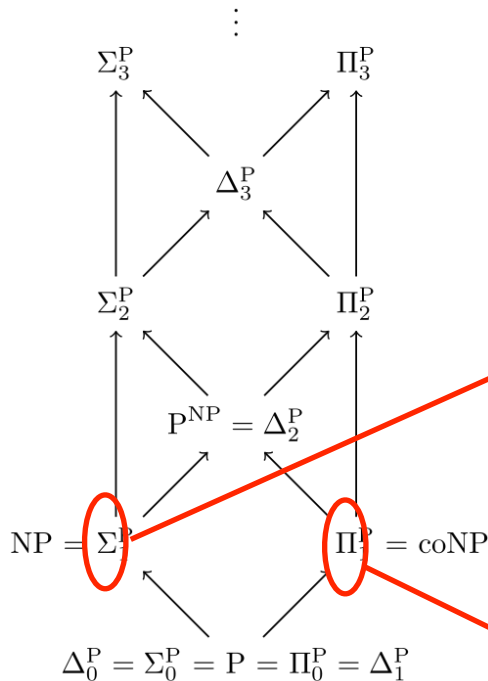
Polynomial Hierarchy, Part I

(via formal logic, directly; a start)



We say that a relation $R(u, y_1, \dots, y_n)$ is polytime iff there is a deterministic Turing Machine \mathbf{m} and a polynomial p s.t. \mathbf{m} decides this relation in $p(|u|)$.

$L \in \mathbf{NP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\exists y R(u, y)$.



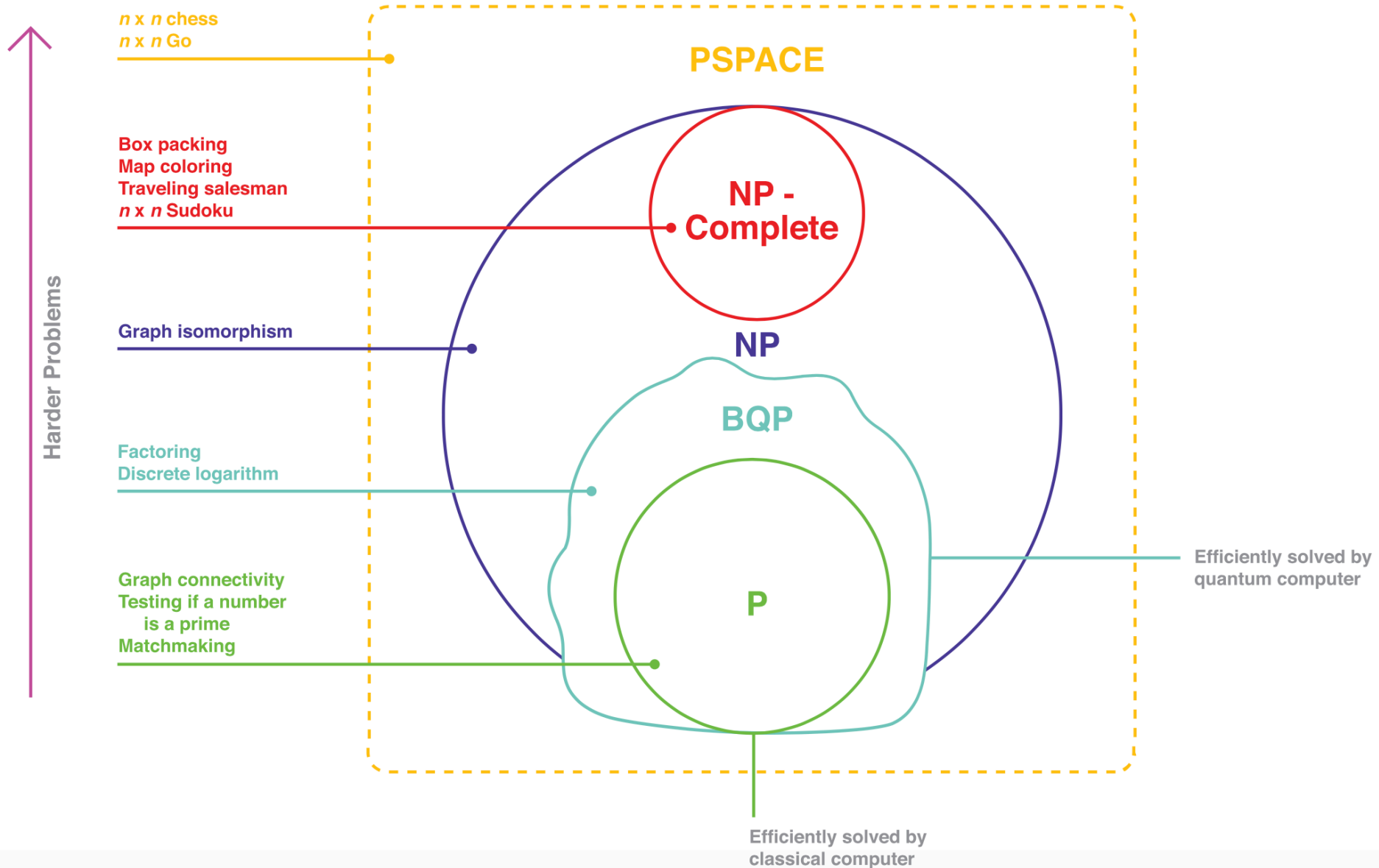
E.g.: We can prove $\mathbf{SAT} \in \mathbf{NP}$ because we have a polytime relation R s.t. $\phi \in \mathbf{SAT}$ iff $\exists y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where these assignments produce truth.

$L \in \mathbf{coNP}$ iff: there's a polytime relation R s.t. $u \in L$ iff $\forall y R(u, y)$.

To prove $\mathbf{coSAT} \in \mathbf{coNP}$, we note that we have a polytime relation R s.t. $\phi \in \mathbf{coSAT}$ iff $\forall y R(\phi \in \mathcal{L}_{pc}, \langle \text{assignments to Boolean vars} \rangle)$, where the assignments produce *falsity*.

What about (oft vaunted) quantum computers?

What about (oft vaunted) quantum computers?



What about (oft vaunted) quantum computers?



GCI

Harder Problems

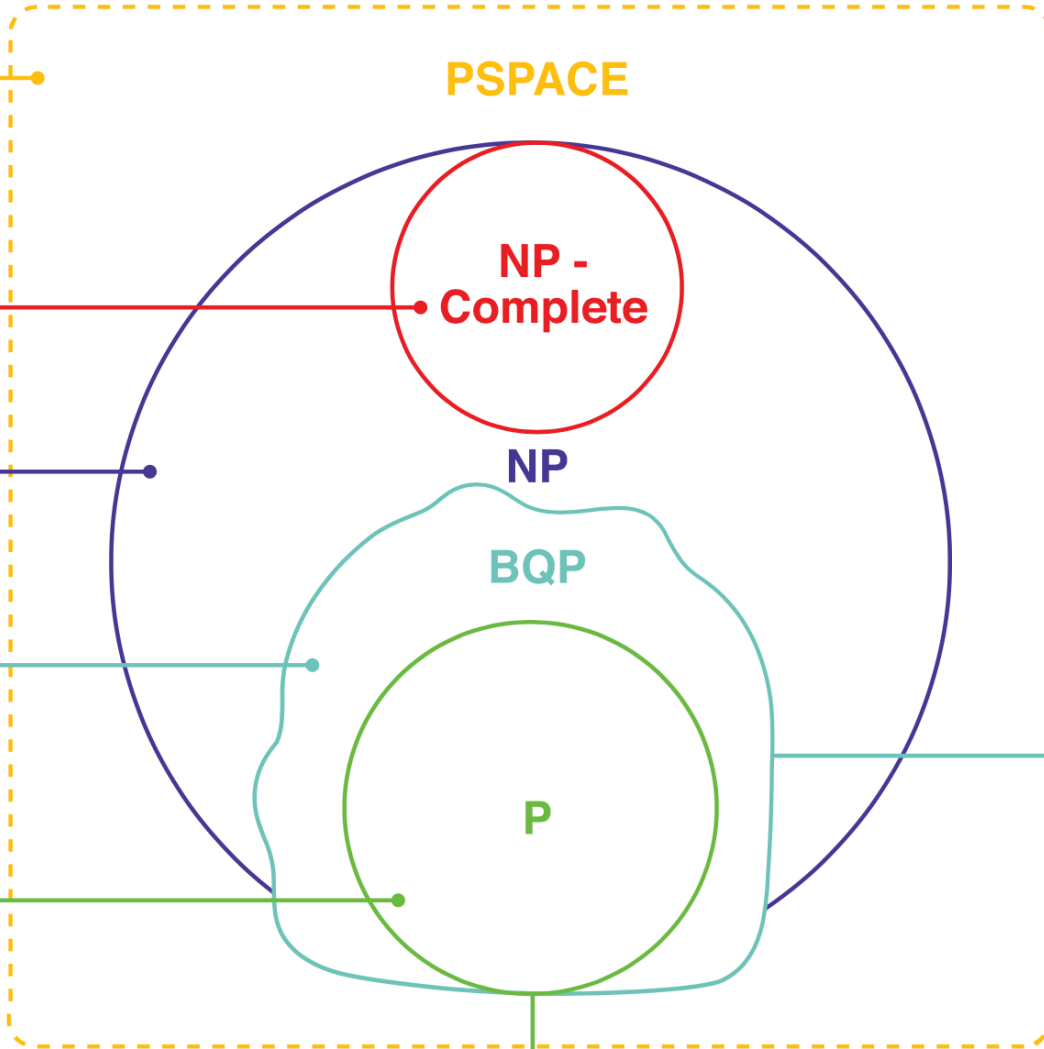
$n \times n$ chess
 $n \times n$ Go

Box packing
Map coloring
Traveling salesman
 $n \times n$ Sudoku

Graph isomorphism

Factoring
Discrete logarithm

Graph connectivity
Testing if a number
is a prime
Matchmaking



PSPACE

NP - Complete

NP

BQP

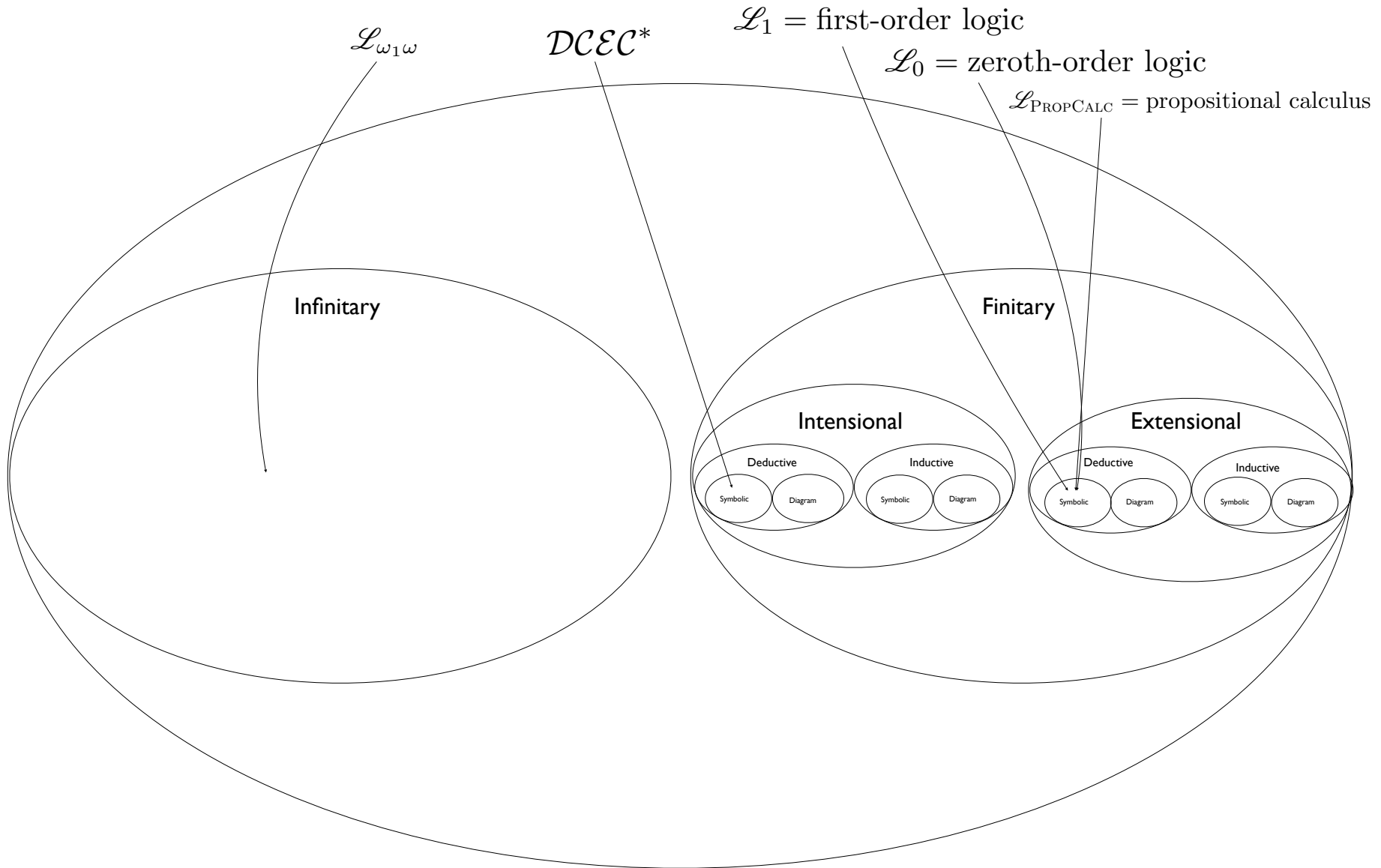
P

Efficiently solved by quantum computer

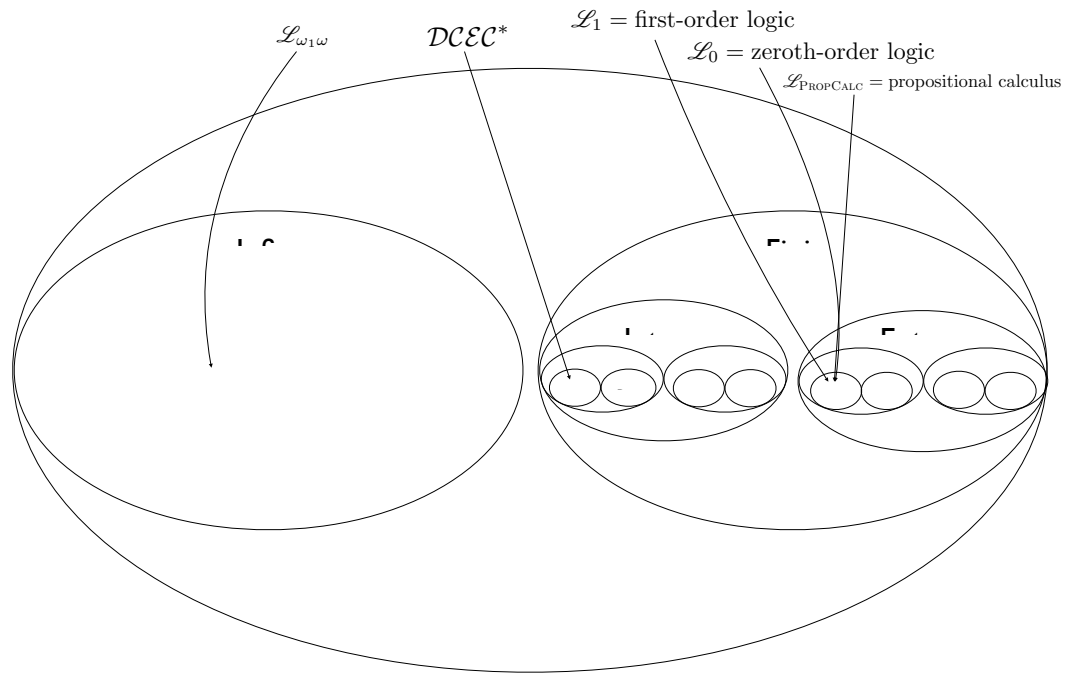
Efficiently solved by classical computer

Reminders ...

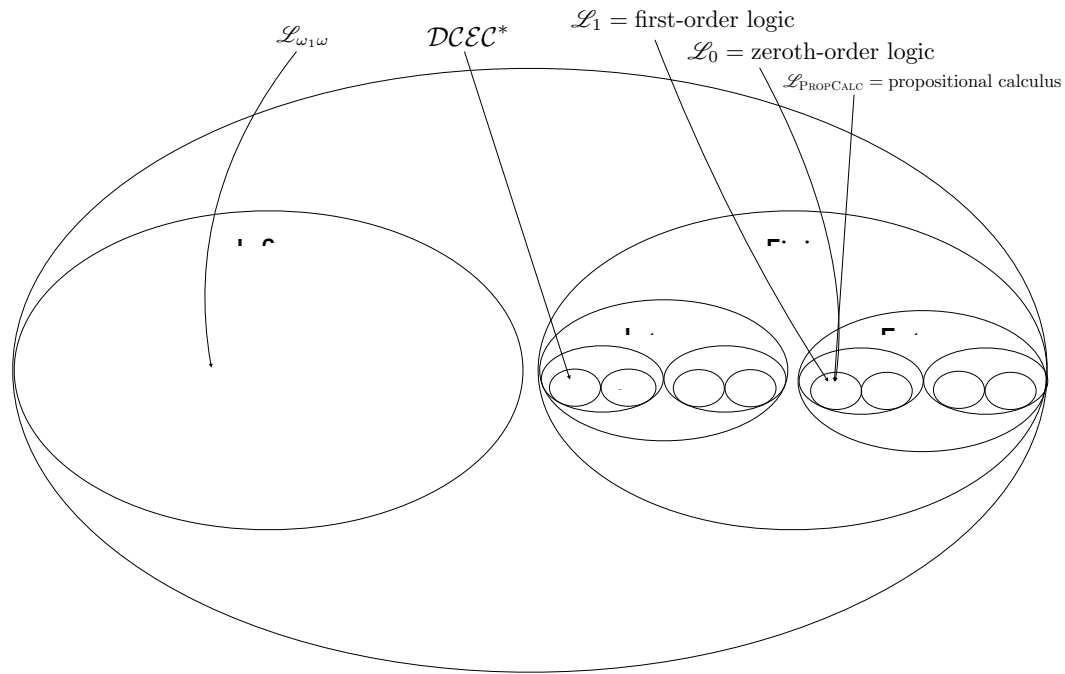
The Universe of Logics



The Universe of Logics

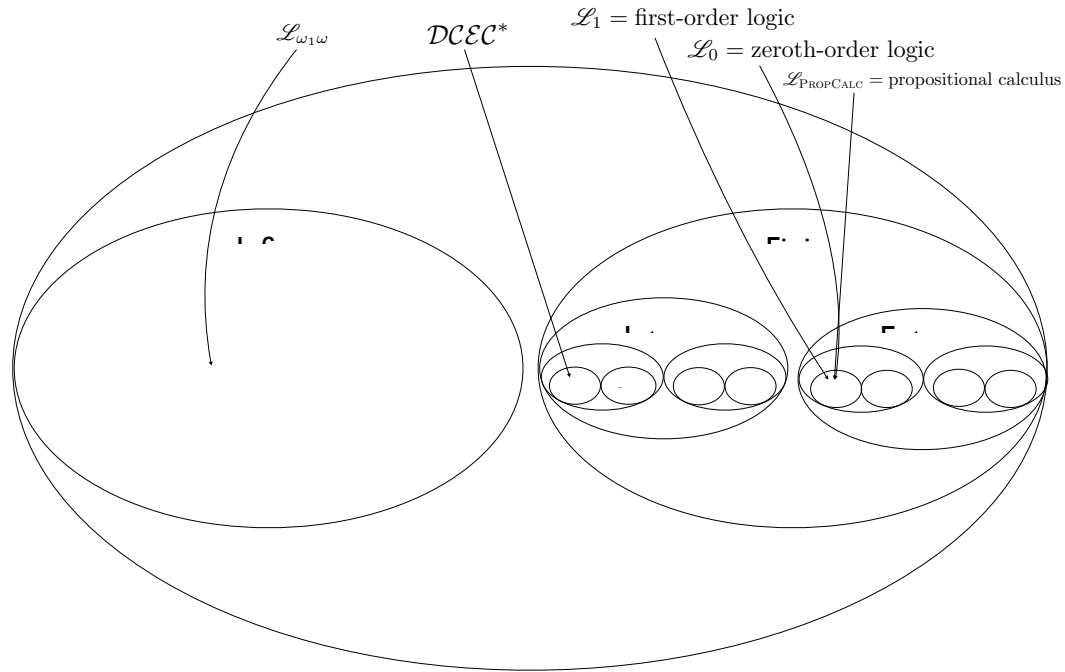


The Universe of Logics



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

The Universe of Logics



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

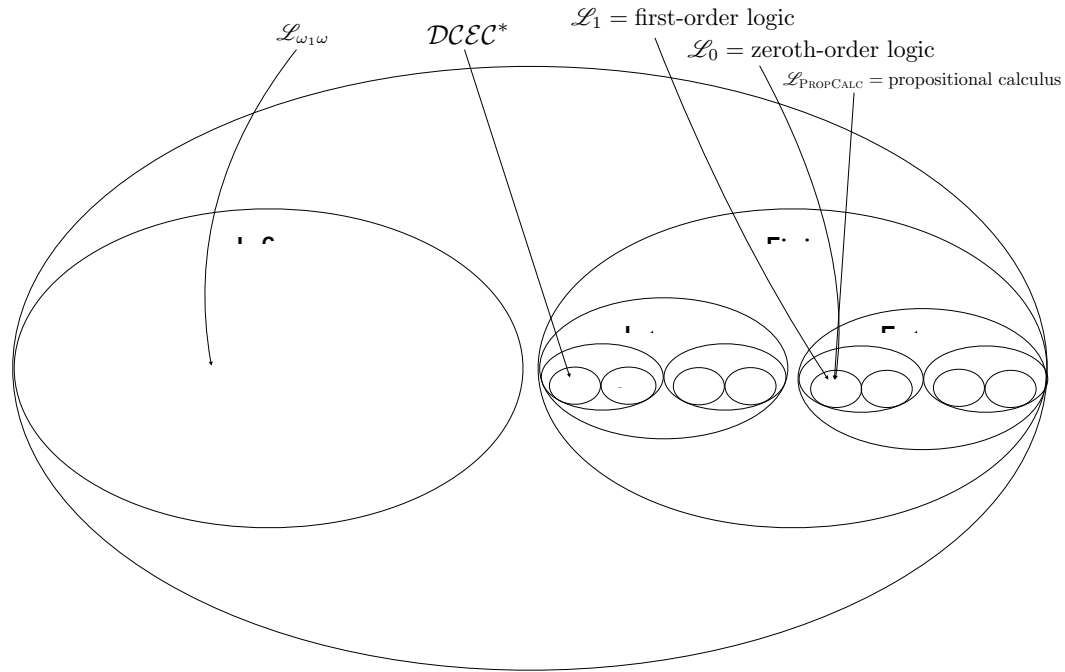
the logic

formal language

inference schemata

semantics

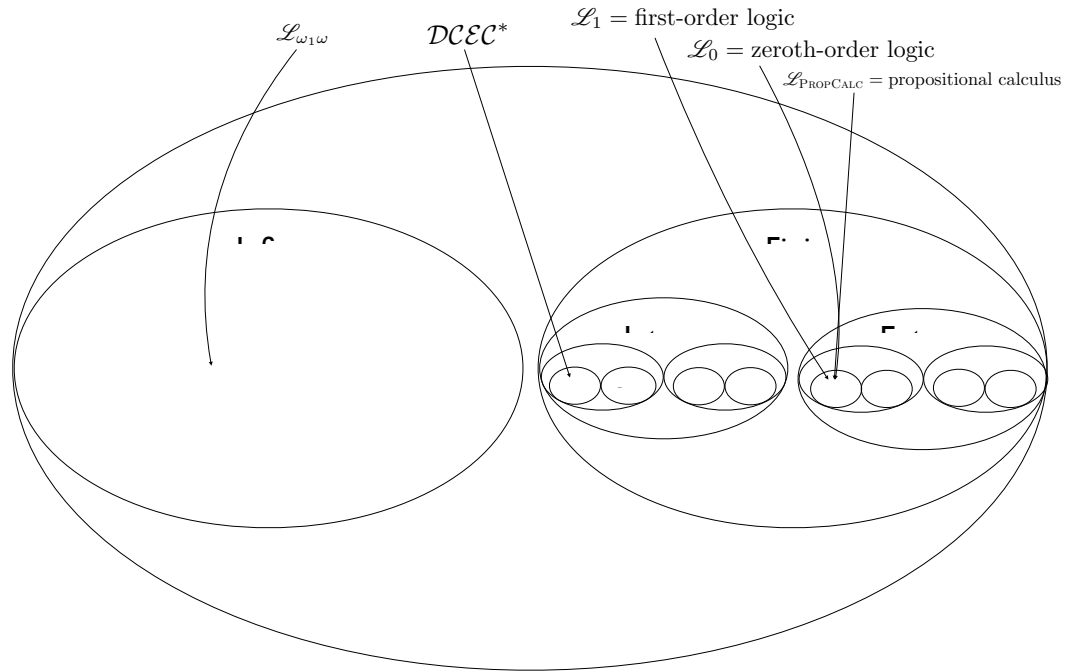
The Universe of Logics



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

the logic \nearrow formal language \nearrow inference schemata \nearrow semantics

The Universe of Logics



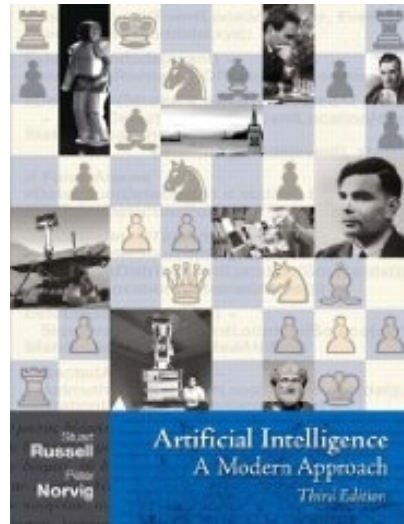
$$\mathcal{L} := \langle L, I, \text{ } \rangle$$

the logic \nearrow \mathcal{L}
 formal language \nearrow L
 inference schemata \nearrow I
 semantics \nearrow $\text{ } \rangle$

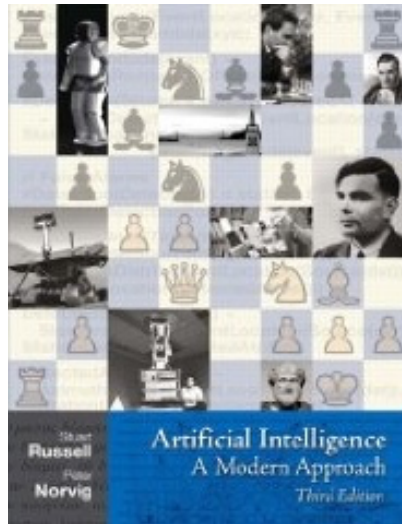
AI ...

Al:




AI:



AI:



Stanford Encyclopedia of Philosophy

 Browse  About  Support SEP

Search SEP

[Entry Contents](#)

[Bibliography](#)

[Academic Tools](#)

[Friends PDF Preview](#) 

[Author and Citation Info](#) 

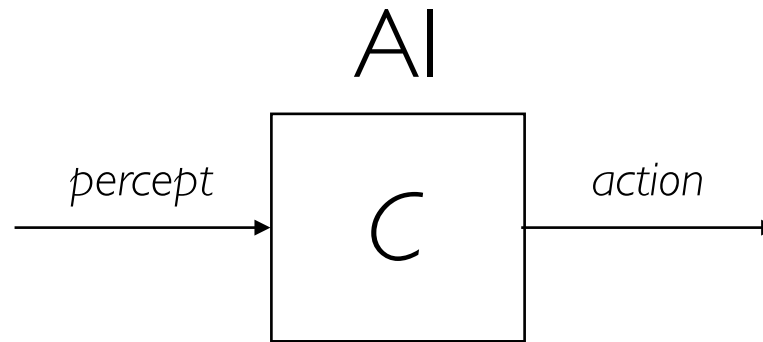
[Back to Top](#) 

Artificial Intelligence

First published Thu Jul 12, 2018

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

AI:



Stanford Encyclopedia of Philosophy

Browse About Support SEP

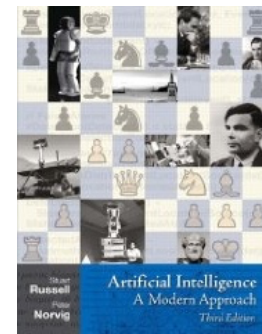
Search SEP

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

Artificial Intelligence

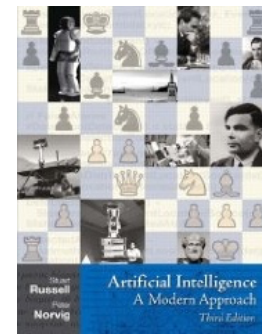
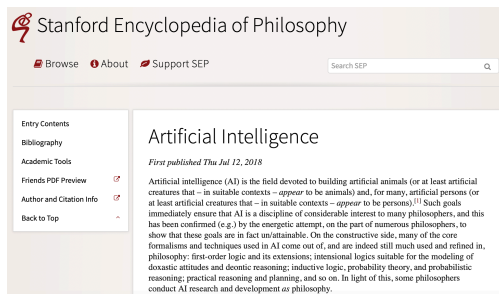
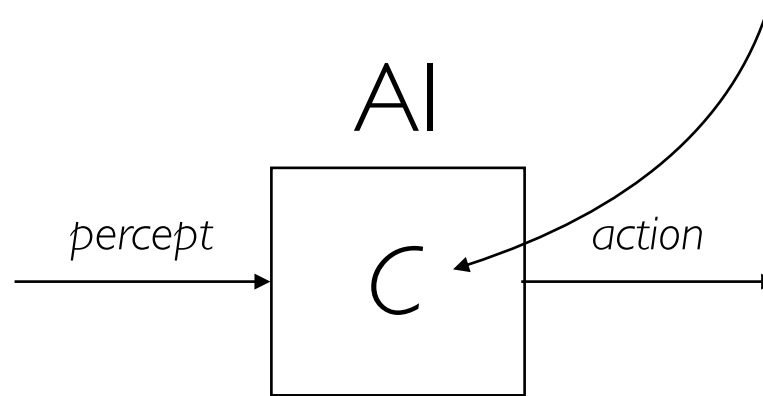
First published Thu Jul 12, 2018

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



AI:

A (Turing-level) entity that computes.



(Pure General) Logic Programming ...

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 pre-

cisely. 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

collection
scientists;

nputa-
xample,
s. Statisti-
a scale, in
imagination
ments in
uter scienc
racing
tments.
biology is
enefit

to
action.

science's
bility to
data look-
atures
actions
of pro-
Compu-
gists
ry is
comput-
a comput-

the skill set
e else.
putational
puting was
ity; com-

ation—
ute it.

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

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

Computational thinking thus has the following characteristics:

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 prob-

lems 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 pre-

cisely. 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

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 benefits 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 facing alone. Computational thinking conditions the habits 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. In 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 ideas of what is appropriately measured about the values of that computing and computers facilitate the spread of computational thinking.

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

Facing 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 tries to add the intellectual 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 constraints as 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 understandings 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 syntax and the design of classes 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 design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when analyzing large complex tasks. It is 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 invariant 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

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 benefits 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 solving alone. Computational thinking conditions the habits 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. In 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 ideas of what is appropriate, we must ensure that what is appropriate to computers and computers facilitate the spread of computational thinking.

Computational thinking involves asking problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking is a broad 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 tries to add the intellectual underpinnings to answer such questions precisely.

Stating the difficulty of a problem accords for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's constraints as 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 heuristics 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 naming 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 design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when modeling a large complex task. It is 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 iteration to discover a system's behavior iteratively 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



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 benefits 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 facing alone. Computational thinking conditions the habits 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. In 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 measured about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves asking problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking is a broadly applicable 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 tries to add 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 constraints as its resource constraints, and its operating environment.

By asking a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use understanding 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 naming 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 design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when modeling a large complex task. It is 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 inventiveness 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)

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 benefits 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 facing alone. Computational thinking conditions the habits 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. In 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 ideas of what is appropriate, we should encourage the ideas of what computing and computers facilitate the spread of computational thinking.

Computational thinking involves asking problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking is a broad 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 tries to add the intellectual underpinnings to answer such questions precisely.

Stating the difficulty of a problem accurately for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's constraints as its resource constraints, and its operating environment.

By asking a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use understanding 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 naming 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 design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when modeling a large complex task. It is designing a large complex system. It is separating it into components. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using inventiveness 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)



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 benefits 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 conditions the habits 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. In 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 Bible, what is appropriately measured about the vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves making problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking is 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 tries to add the intellectual 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 constraints as its resources, and its operating environment, whether we can use mathematics to our advantage, and whether false promises or false progress are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by induction, 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 intent and the danger of sharing 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 design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when modeling large complex tasks. It is designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the correct approach to a problem to make it tractable. It is using iteration 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.

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 benefits 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 conditions the habits 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. In 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 Bible, what is appropriately measured about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves making problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking is a family 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 runs 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 constraints as its resources, and its operating environment, to ask whether an approximate solution is good enough, whether we can use understanding to our advantage, and whether false promises or false progress are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by induction, 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 intent and the danger of sharing 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 design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when analyzing large complex tasks. It is designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the correct approach to a problem to make it tractable. It is using iteration 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,

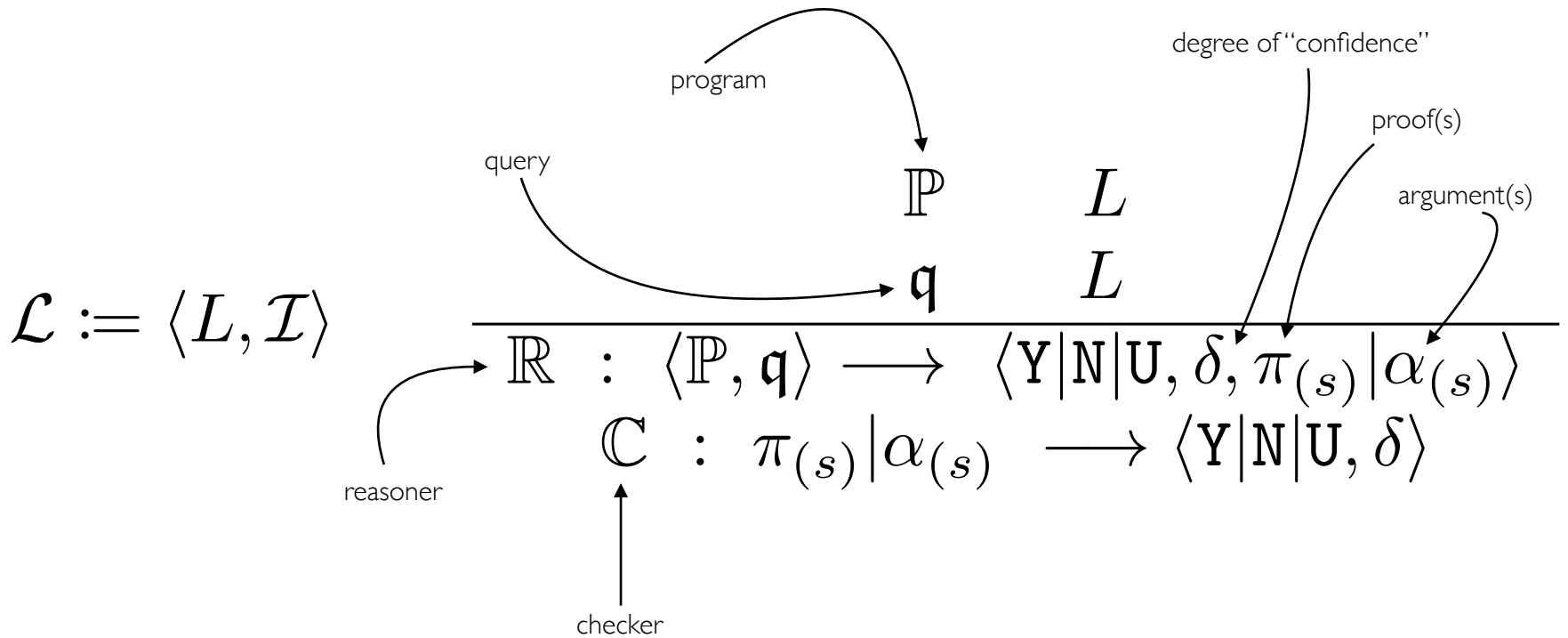
and then to provide algorithms to show how this can be done:

efficiently,
practically,
physically,
and ethically.

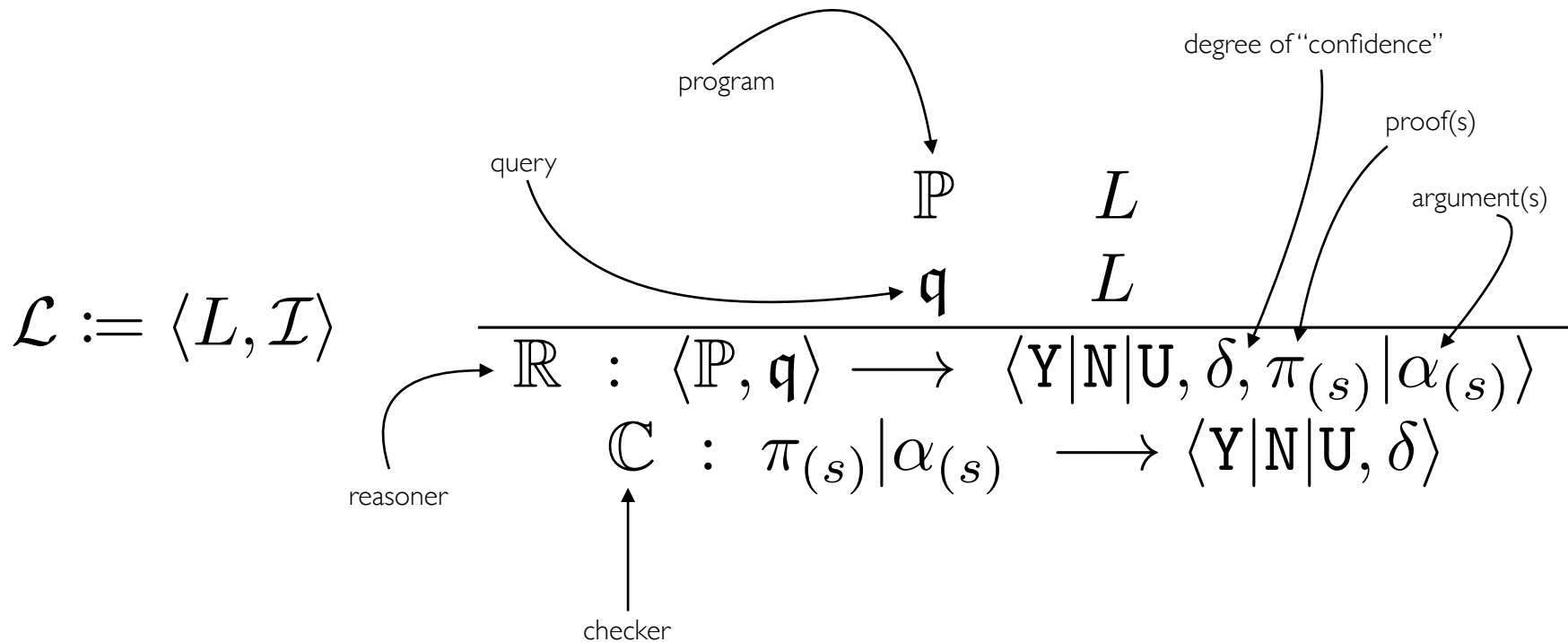
– Rapaport, “phics” book

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

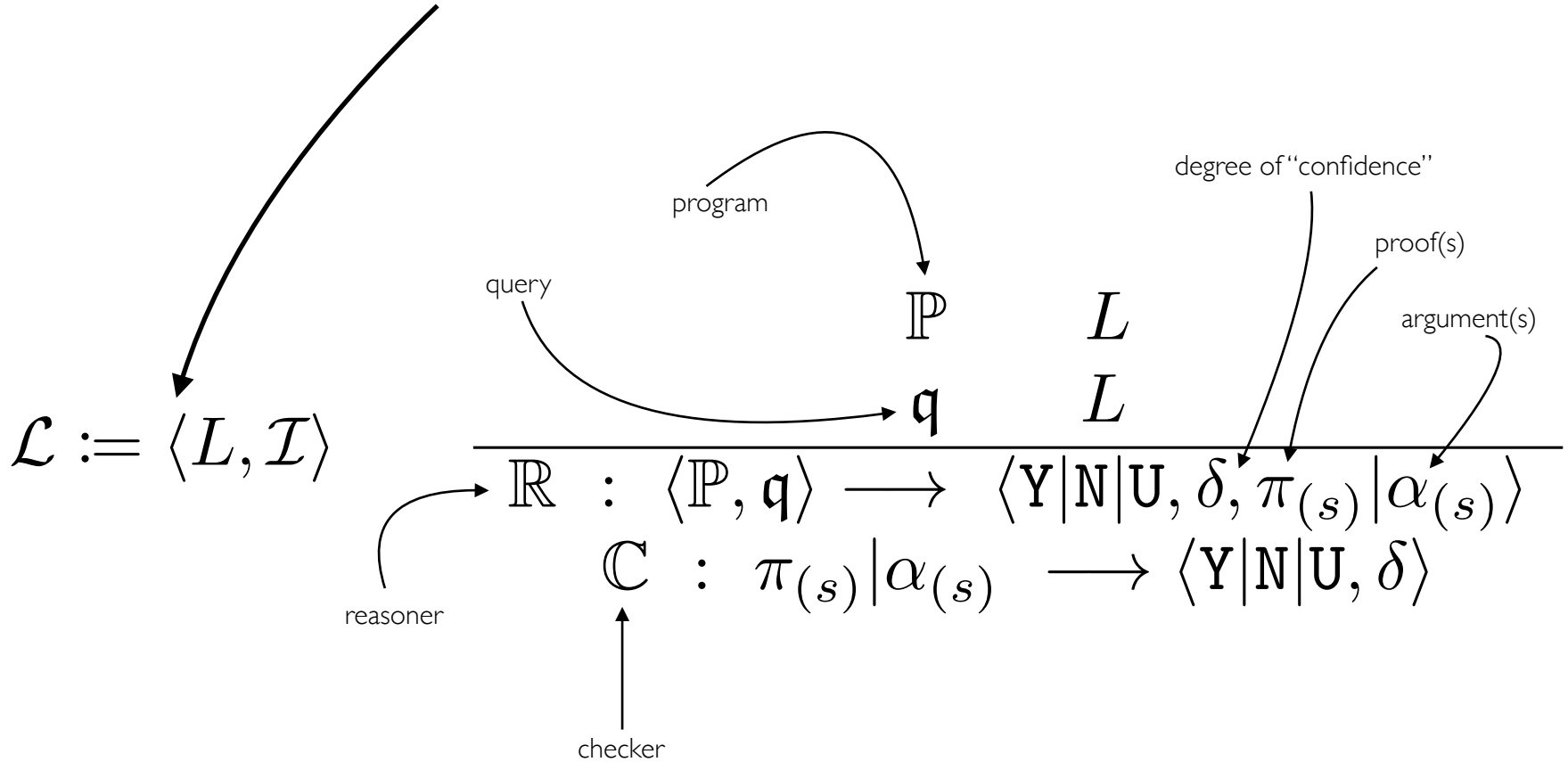
$$\begin{array}{ccc}
 & \mathbb{P} & L \\
 & \mathfrak{q} & L \\
 \hline
 \mathbb{R} & : \langle \mathbb{P}, \mathfrak{q} \rangle \longrightarrow & \langle \mathbf{Y} | \mathbf{N} | \mathbf{U}, \delta, \pi_{(s)} | \alpha_{(s)} \rangle \\
 \mathbb{C} & : \pi_{(s)} | \alpha_{(s)} \longrightarrow & \langle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



For just “logic programming,” and a vintage approach that goes back to circa 1970, restrict this to a FOL or a fragment thereof, and use resolution as the only inference schema.



For just “logic programming,” and a vintage approach that goes back to circa 1970, restrict this to a FOL or a fragment thereof, and use resolution as the only inference schema.



On the Anatomy of a PGLP Program

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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

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

Semantics

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

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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

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

Semantics

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

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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

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

Semantics

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

\mathcal{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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

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

Semantics

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

\mathcal{L}

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

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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

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

Semantics

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

\mathcal{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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

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

Semantics

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

\mathcal{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	$\exists x \text{rank}(\phi) = x \quad \{\phi\} \vdash \psi \quad \mathfrak{A} \models \phi$
L	object-level language	$\phi \quad \psi \quad \delta$

Inference

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

Semantics

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

\mathcal{L}



Logic-Machines Hierarchy

Chapter 1 Is Universal Computation a Myth?*

John Hopcroft

Abstract. All has claimed that universal computation is a myth, and has offered a variety of ingenious arguments in support of this claim, one of which involves the halting problem and the notion of oracle Turing machines. I provide a brief survey of the arguments, and conclude that the claim is unfounded. I then discuss the related but more subtle claim that Church-Turing Thesis, and its various extended versions of Kolmogorov-Chaitin complexity, which I conclude that it is equally unfounded. I then discuss the notion of oracles that universal computation is, or can be, and I conclude by providing several ways that I believe can vindicate the counter claim that universal computation is significant, and meaningful.

1.1 Introduction

While all is mind-boggling, every possible (reasonable) computation has an end, and every other (reasonable) computation is (effectively) computable. For the former claim, the usual argument is simply algorithmic: if a computation does not terminate, then it is not a computation. The latter claim is more subtle, and more controversial. The latter claim is more subtle, and more controversial. The latter claim is more subtle, and more controversial.

* In addition to being a book, this is a lecture, by an otherwise invisible author, and one of several related books, from which you can see what I have learned in practice and experimental papers.



Logic-Machines Hierarchy

\mathbb{P} \mathbb{q} L_{PC} \mathbb{R} \mathbb{C}

Chapter 1 Is Universal Computation a Myth?

John D. Bruggel

Abstract All has claimed that universal computation is a myth, and has offered a variety of arguments against its support. In this paper, we will show that the challenge of finding the location of an object, even when the object is hidden, is a problem that can be solved by a Turing machine. This result is a direct consequence of the fact that the Church-Turing Thesis, and its various generalizations, are true. It is a direct consequence of the fact that the Church-Turing Thesis, and its various generalizations, are true. It is a direct consequence of the fact that the Church-Turing Thesis, and its various generalizations, are true.

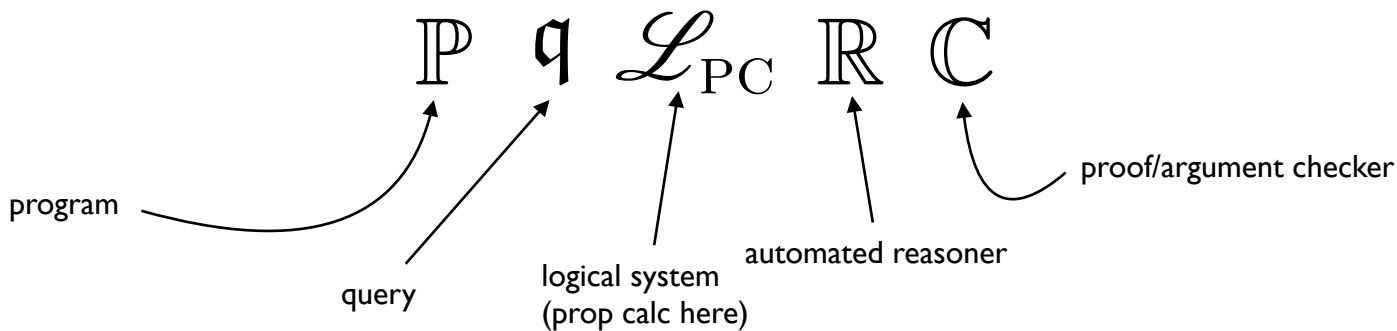
1.1 Introduction

When all is considered, every possible computation is a Turing machine. This is the central thesis of this paper. The central thesis of this paper is that every possible computation is a Turing machine. This is the central thesis of this paper. The central thesis of this paper is that every possible computation is a Turing machine. This is the central thesis of this paper.

¹ In addition to being a Turing machine, every computation is also a Turing machine. This is the central thesis of this paper. The central thesis of this paper is that every possible computation is a Turing machine. This is the central thesis of this paper.



Logic-Machines Hierarchy



Chapter 1 Is Universal Computation a Myth?*

John D. Bruggen

Abstract: All has claimed that universal computation is a myth, and has offered a variety of arguments against its support. In this paper, we will check some of the challenges of finding the location of our right, our wrong, our truth, or lies. I provide a brief survey of the arguments of the computer science community in support of a myth. This is the abstract and more general than the Church-Turing Thesis, and so we may understand a variety of subsequent technical results. What I consider that I have interpreted. I have from the history of our relations that universal computation is, or one the most, if possible by getting around a simple that I believe can challenge the counter-claim that universal computation is impossible, and unrealistic.

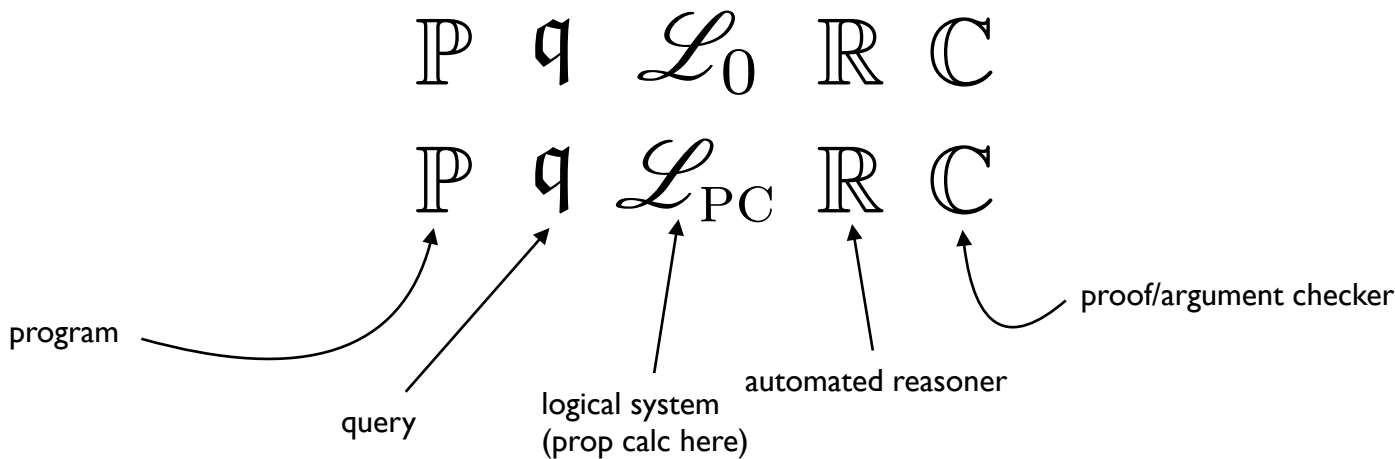
1.1 Introduction

John D. Bruggen's survey provides a comprehensive overview for me to write about the history, logic, formal and philosophical, of computation. For the present value, the author gives a single algorithmic, by induction and recursion. The title of *Universal Computation*, I will take the word "universal" as a synonym for "computable" or "decidable". The author's paper, in this paper, is a survey of the history of multiple related concepts in the field of *Universal Computation*. I will, however, I will not be able to discuss the abstract that I believe, in this paper, is a survey of the history of multiple related concepts in the field of *Universal Computation*. I will, however, I will not be able to discuss the abstract that I believe, in this paper, is a survey of the history of multiple related concepts in the field of *Universal Computation*.

* I am grateful to John D. Bruggen for an extensive review of this paper, and for his kind support and advice. I have also received many helpful comments from other researchers in the field of universal computation.



Logic-Machines Hierarchy



Chapter 1
Is Universal Computation a Myth?*

John Heintzel

Abstract: All has claimed that universal computation is a myth, and has offered a variety of operations arguments in support of this claim, one of which is the challenge of finding the location of our right-most Turing machine. I provide a brief survey of these arguments, and conclude that the claim is false. In the abstract, all operations arguments are based on a single idea: the idea that the location of our right-most Turing machine is not well-defined, and hence that the Turing machine is not well-defined.

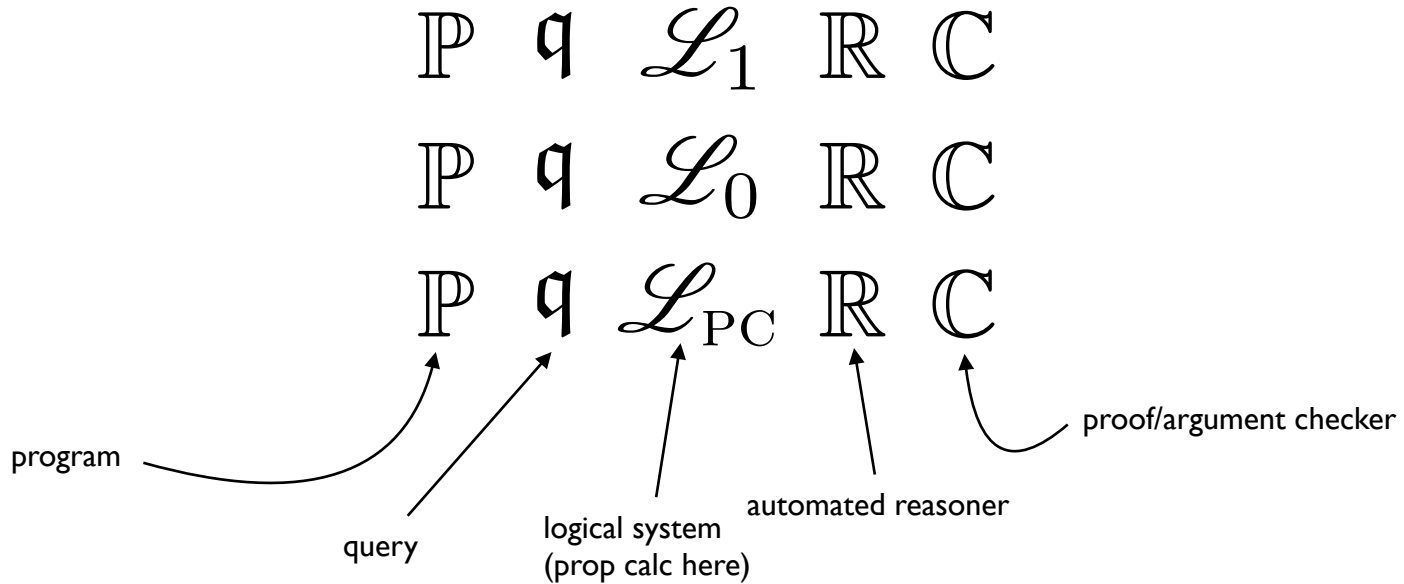
1.1 Introduction

When I'm reminded, every possible computation is finite, and hence that the Turing machine is well-defined, and hence that the Turing machine is well-defined. For the present, I will focus on the claim that the Turing machine is not well-defined, and hence that the Turing machine is not well-defined. The rest of the paper is devoted to the claim that the Turing machine is not well-defined, and hence that the Turing machine is not well-defined.

* I'm grateful to John Heintzel for an extremely thorough and thoughtful review, and to the anonymous referees for their helpful comments and suggestions.



Logic-Machines Hierarchy



Chapter 1
Is Universal Computation a Myth?*

John D. Bruggen

Abstract: All has claimed that universal computation is a myth, and has offered a variety of arguments against its support. In this paper, we will check to see if the challenge of finding the location of our right, our wrong, and our middle. I provide a brief survey of the arguments of the computer science community in favor of a myth. In the abstract and more practical than the Church-Turing Thesis, and in an even more practical sense of Kolmogorov's Universal Machine, which I consider that I have interpreted. I also find the source of our reliance that universal computation is, or can be, and I provide a partial proof to show that I believe our challenge to consider that universal computation is a myth, and is not.

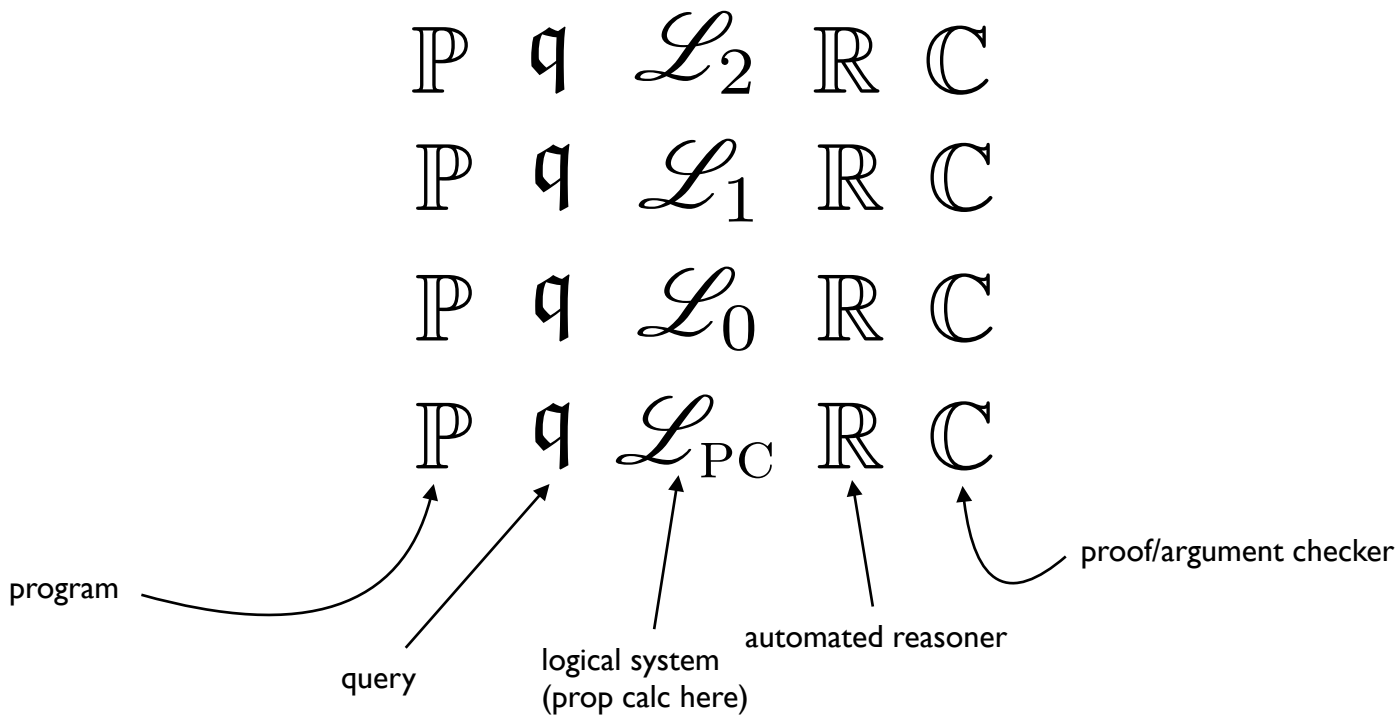
1.1 Introduction

While all is considered every possible computation is possible for us to carry about the boundaries, both formal and philosophical, of computation. For the present value, the world is a single algorithmic, by definition, and computation. The field of Universal Computation (UC) has, by its nature, the ability to carry out any computation that can be carried out by a Turing machine, and in this paper we examine the history of multiple other attempts to do so. I believe that the concept of UC is a very important concept, and I believe that the history of UC is a very important concept. The subject of UC is, in just a convenient manner that I use in the remainder of this paper.

* I am grateful to John D. Bruggen for an extensive review of this paper, and to all of the other people who have helped me in my work on this document. I am grateful to the people who have helped me in my work on this document.



Logic-Machines Hierarchy



Chapter 1 Is Universal Computation a Myth?

John D. Beatty

Abstract: All has claimed that universal computation is a myth, and has offered a variety of arguments in support of this claim, one of which is the challenge of finding the location of our rightmost Turing machine. I provide a brief survey of the arguments and counter-arguments to this claim, and then show that the claim that there is no rightmost Turing machine is false. I conclude that the claim that there is no rightmost Turing machine is false, and that the claim that there is no rightmost Turing machine is false.

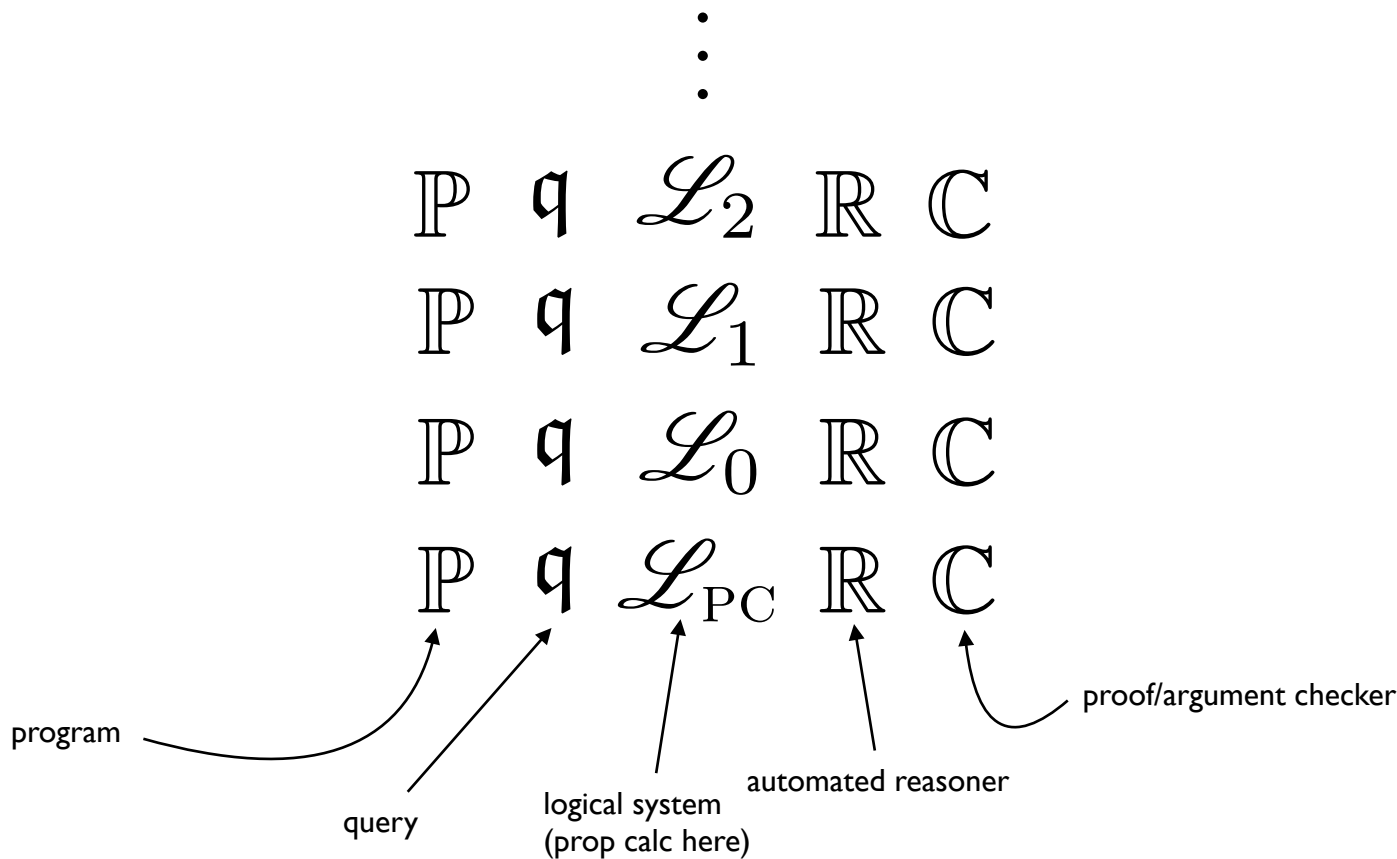
1.1 Introduction

John D. Beatty, "Is Universal Computation a Myth?", *Journal of Universal Computation*, 2014. This paper is published in the *Journal of Universal Computation*, a journal of the International Association of Universal Computation. The paper is published in the *Journal of Universal Computation*, a journal of the International Association of Universal Computation. The paper is published in the *Journal of Universal Computation*, a journal of the International Association of Universal Computation.

¹ I am grateful to John D. Beatty for his invitation to participate in this special issue, and to the anonymous reviewers for their helpful and constructive comments.



Logic-Machines Hierarchy



Chapter 1
Is Universal Computation a Myth?*

John D. Bruggen

Abstract. All has claimed that universal computation is a myth, and has offered a variety of opinions regarding its support or its status, one of which has been the challenge of finding the location of our right, ever moving, reference line. I provide that line as a function of the computer, my computer system, its host, or a host, that is, the abstract and more practical than the Church-Turing Thesis, and an ever more generalized version of Kolmogorov's function. What I contend is that I have identified a line from the source of our reference that universal computation is, or can be, and I provide by getting inside a computer that I believe can challenge the counter-claim that universal computation is specific, and invariable.

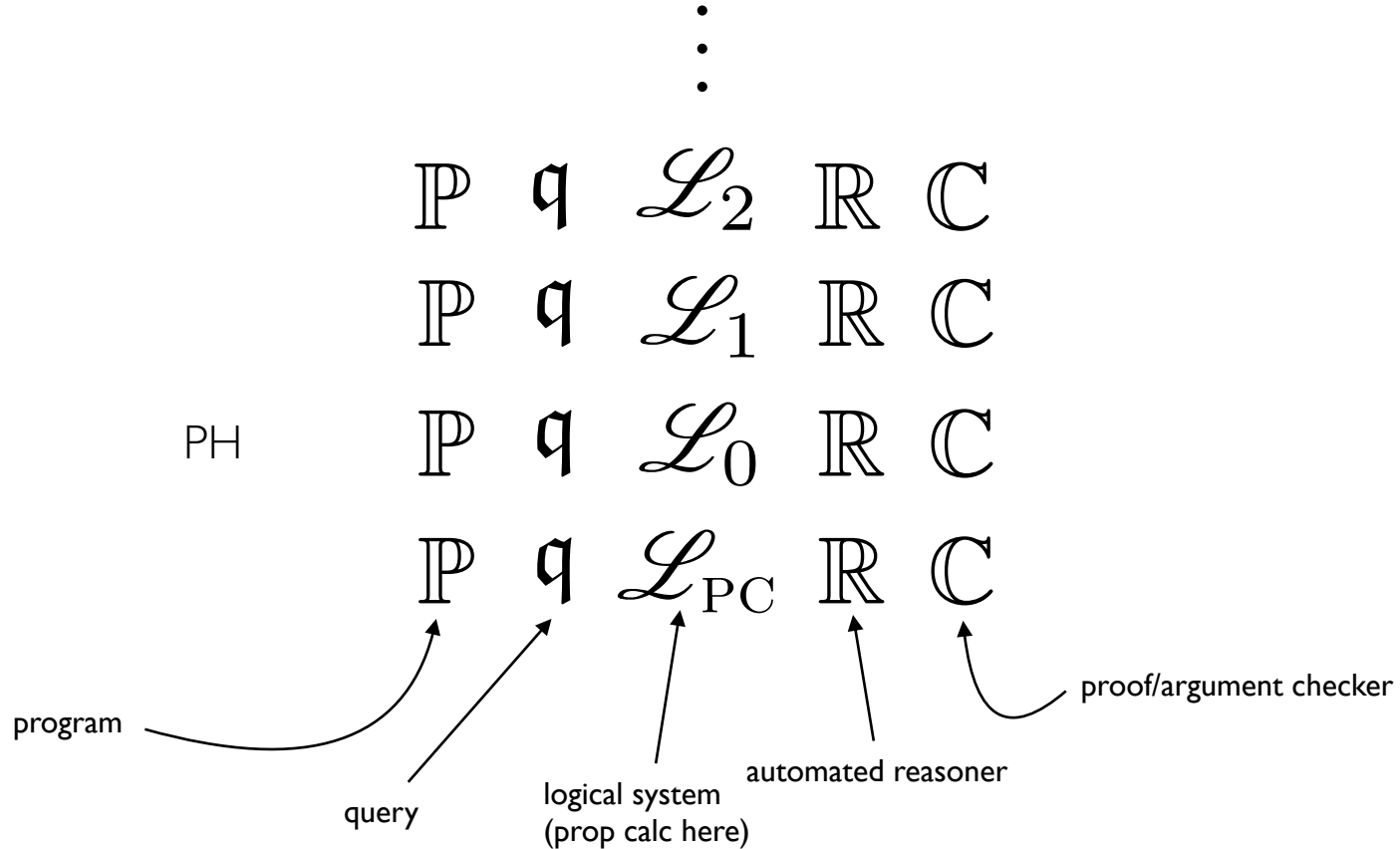
1.1 Introduction

When all is considered, every possible (non-recursive) operation has the same to offer about the location of our right and left-hand reference lines. For the present, the line is a single reference line, the location of our right and left-hand reference lines, and the location of our right and left-hand reference lines. The line is a function of the computer, my computer system, its host, or a host, that is, the abstract and more practical than the Church-Turing Thesis, and an ever more generalized version of Kolmogorov's function. What I contend is that I have identified a line from the source of our reference that universal computation is, or can be, and I provide by getting inside a computer that I believe can challenge the counter-claim that universal computation is specific, and invariable.

* In addition to being All the things to an abstract counter-claiming line, and one of many other lines, from the source of our right and left-hand reference lines and reference points.



Logic-Machines Hierarchy



Chapter 1
Is Universal Computation a Myth?*

John D. Borger

Abstract: All has claimed that universal computation is a myth, and has offered a variety of heuristic arguments in support of this claim, one of which has been the challenge of finding the location of our right-most Turing machine. I provide a brief survey of the literature on this question, and conclude that the claim is unfounded. I then discuss the relationship between the Turing machine and the lambda calculus, and the relationship between the Turing machine and the lambda calculus, and the relationship between the Turing machine and the lambda calculus. I then discuss the relationship between the Turing machine and the lambda calculus, and the relationship between the Turing machine and the lambda calculus. I then discuss the relationship between the Turing machine and the lambda calculus, and the relationship between the Turing machine and the lambda calculus.

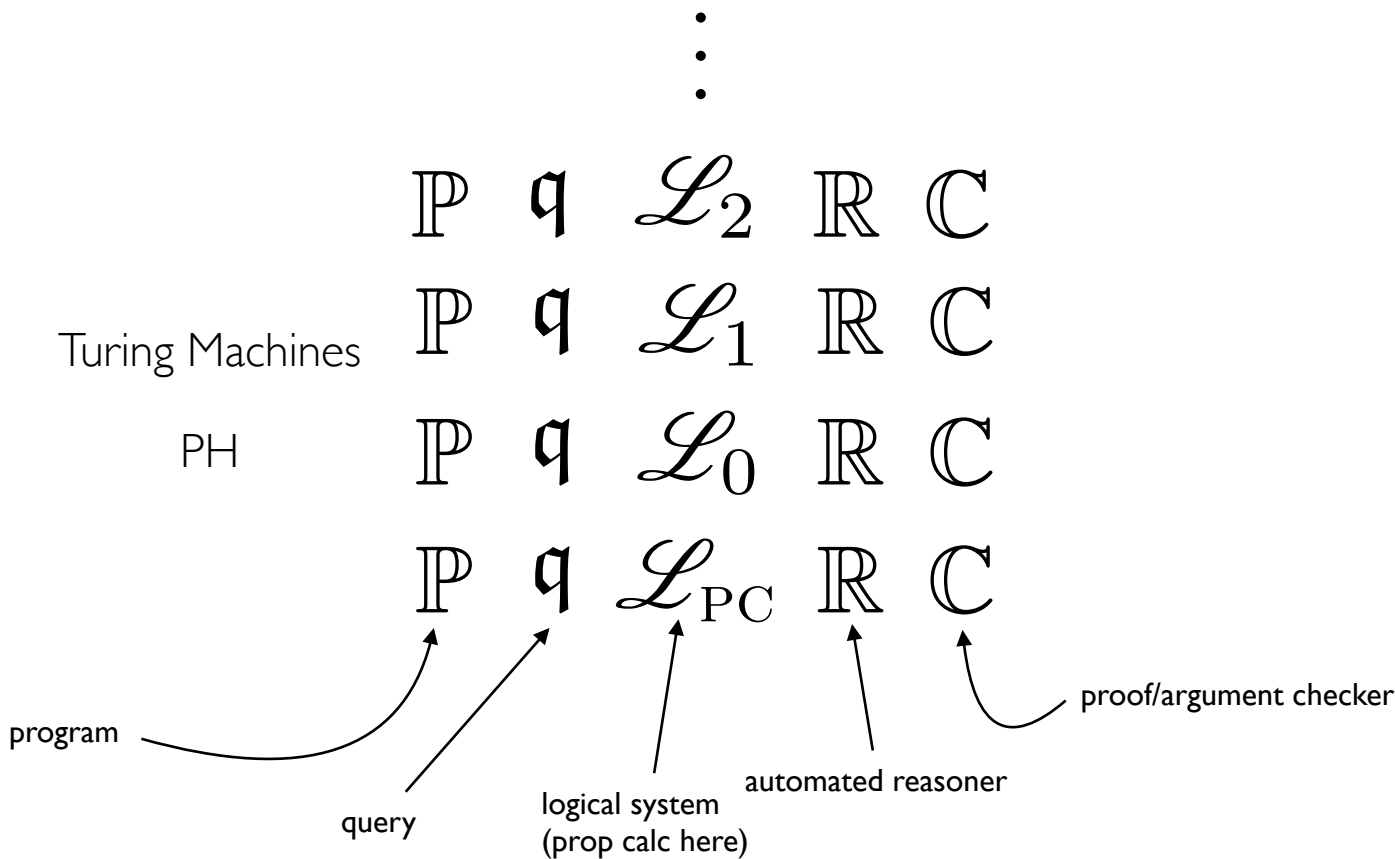
1.1 Introduction

When I first read the survey, I was struck by the author's apparent lack of interest in the history of the subject. The author's argument is based on a misunderstanding of the Turing machine, and is completely unconvincing. The Turing machine is a simple machine, and it is not clear why it should be considered so important. The Turing machine is a simple machine, and it is not clear why it should be considered so important. The Turing machine is a simple machine, and it is not clear why it should be considered so important.

* I am grateful to John D. Borger for his invitation to contribute to this volume, and to the other authors for their contributions to the volume.



Logic-Machines Hierarchy



Chapter 1
Is Universal Computation a Myth?¹

John D. Bragg

Abstract: All has claimed that universal computation is a myth, and has offered a variety of arguments against its support. In this paper, we will look at some of the challenges of building the machine of our dreams, one using cellular automata. It is possible that we may be able to do this, or at least, we may be able to do it in a way that is not as difficult as we think it is. We will look at the work of John D. Bragg, and we will see how universal computation is possible. We will also look at the work of John D. Bragg, and we will see how universal computation is possible. We will also look at the work of John D. Bragg, and we will see how universal computation is possible.

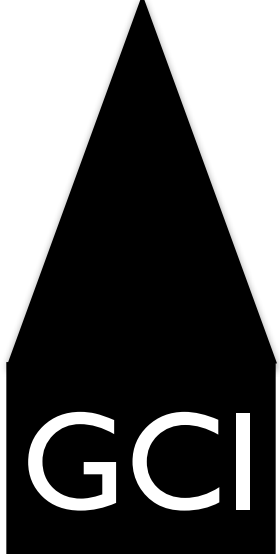
1.1 Introduction

John D. Bragg is a mathematician who has spent most of his life trying to prove that universal computation is a myth. He has written a book on this topic, and he has also written a paper on this topic. He has also written a paper on this topic, and he has also written a paper on this topic.

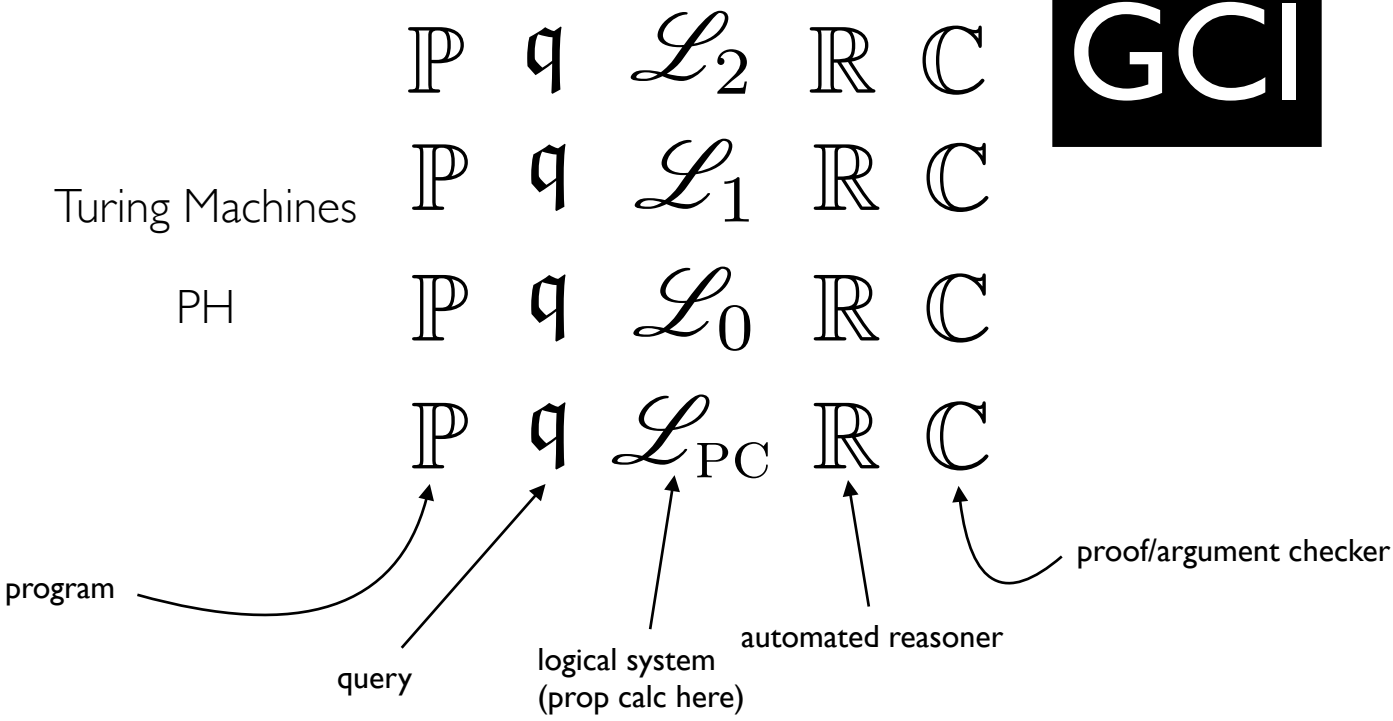
¹ It is possible to build a machine that is universal, and that can compute anything that can be computed by a Turing machine. This is the claim of the Church-Turing thesis, and it is a claim that has been widely accepted.



Logic-Machines Hierarchy



⋮



Chapter 1
Is Universal Computation a Myth?*

John Heintzel

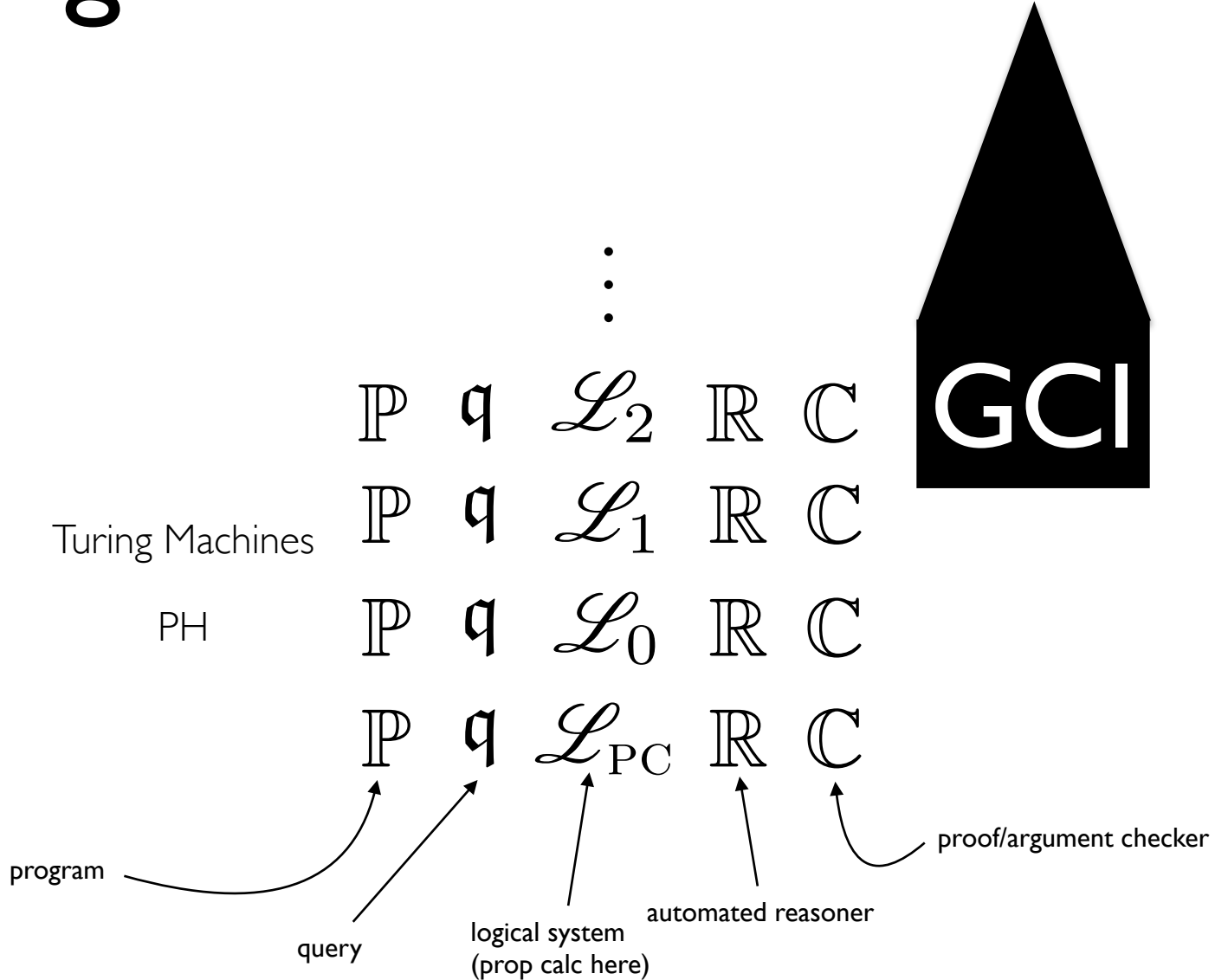
Abstract All has claimed that universal computation is a myth, and has offered a variety of arguments against in support of this claim, one of which has been the challenge of finding the location of our rightmost computing electron on Mars. I provide here an alternative to this argument, one which is more closely related to the claim that the halting problem is not provably undecidable. The rest of the paper is devoted to the construction of a checker for the Church-Turing Thesis, and an alternative version of Kolmogorov's complexity function. What I conclude that I have developed here is the notion of an *argument checker* and a *proof/argument checker*. The rest of the paper is devoted to the construction of a checker for the Church-Turing Thesis, and an alternative version of Kolmogorov's complexity function. What I conclude that I have developed here is the notion of an *argument checker* and a *proof/argument checker*. The rest of the paper is devoted to the construction of a checker for the Church-Turing Thesis, and an alternative version of Kolmogorov's complexity function. What I conclude that I have developed here is the notion of an *argument checker* and a *proof/argument checker*.

1.1 Introduction

All has claimed that universal computation is a myth, and has offered a variety of arguments against in support of this claim, one of which has been the challenge of finding the location of our rightmost computing electron on Mars. I provide here an alternative to this argument, one which is more closely related to the claim that the halting problem is not provably undecidable. The rest of the paper is devoted to the construction of a checker for the Church-Turing Thesis, and an alternative version of Kolmogorov's complexity function. What I conclude that I have developed here is the notion of an *argument checker* and a *proof/argument checker*.

* I am grateful to John Heintzel for his invitation to contribute to this volume, and to the other authors for their kind and helpful comments.

Logic-Machines Hierarchy



Chapter 1
Is Universal Computation a Myth?*

John D. Bruggen

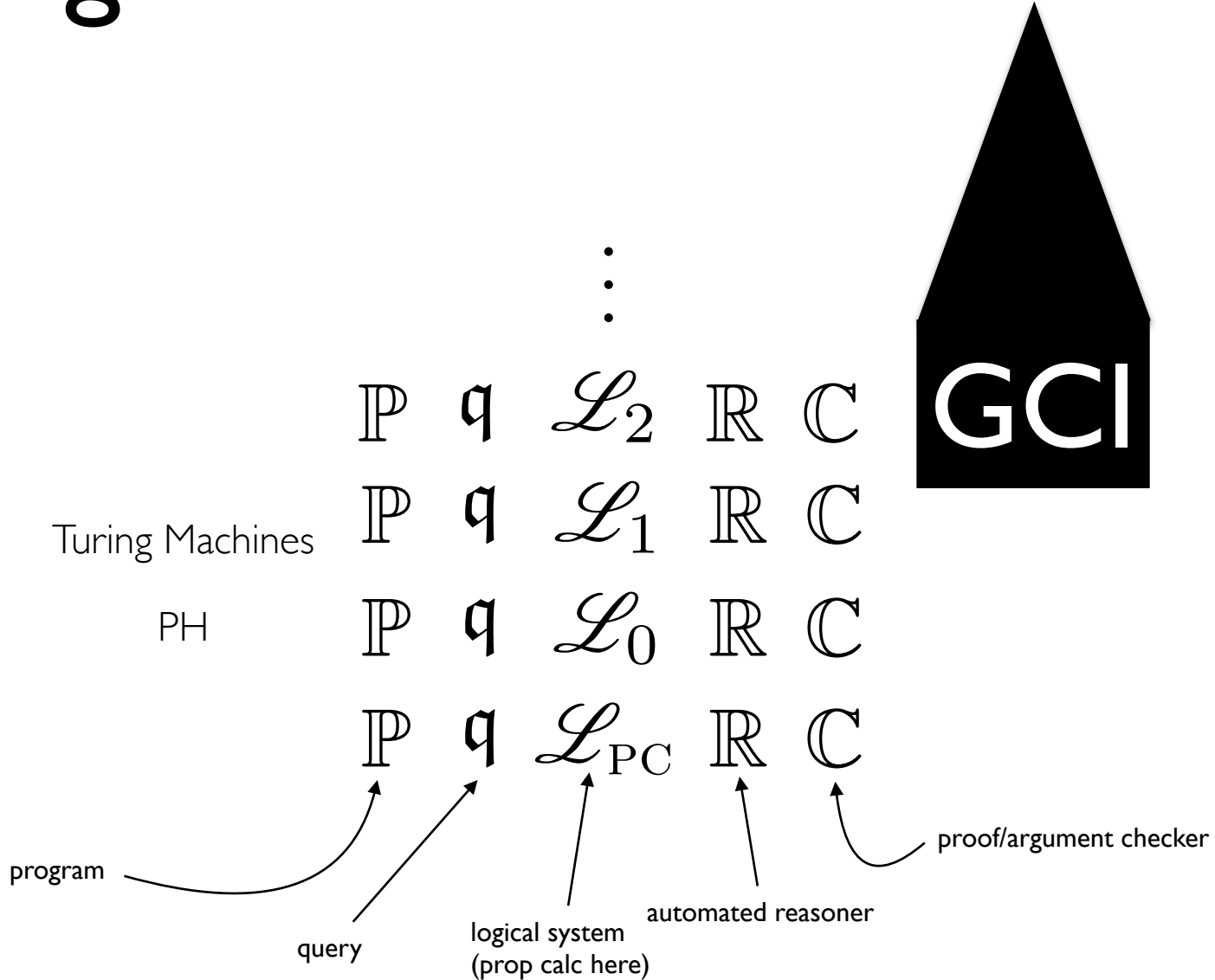
Abstract: All has claimed that universal computation is a myth, and has offered a variety of arguments against it. In this paper, we will show that such claims are unfounded. We will show that the Church-Turing Thesis, and its extensions, are not only true, but also that they are the only reasonable extensions of it. We will also show that the Church-Turing Thesis, and its extensions, are not only true, but also that they are the only reasonable extensions of it. We will also show that the Church-Turing Thesis, and its extensions, are not only true, but also that they are the only reasonable extensions of it.

1.1 Introduction

Since All's introduction, many people have written responses to me to state their objections. I have found that the most common objection is that the Church-Turing Thesis is not a theorem, but a hypothesis. This is a common objection, but it is not a valid one. The Church-Turing Thesis is not a hypothesis, but a theorem. It is a theorem that has been proven to be true. It is a theorem that has been proven to be true. It is a theorem that has been proven to be true.

* In addition to being All for Turing, he is also a member of the Turing Society, and a member of the Turing Society. He is also a member of the Turing Society, and a member of the Turing Society.

Logic-Machines Hierarchy



Research Technology (2012) 11(1) 107
<https://doi.org/10.1080/17445019.2012.674087>
 ISSN: 1744-5019

Computer Science as Immaterial Formal Logic
 Seminar Bruggagel, S.A.¹
 Received 20 June 2012; Accepted 8 July 2012; Published online 1 August 2012
 © Springer Science & Media

Abstract
 I critically review Raymond Turner's Computational Arithmetic – Towards a Philosophy of Computer Science by checking inside his position a rather different one, according to which computer science is a branch of, and is therefore subsumed by, immaterial formal logic.

Keywords: Formal logic · Logic machines · Computer science

Consider the abstractness, instead of abstract? Inference schemes immaterial/abstractness, counterintuitively derived by such text as the following:

$$\frac{\text{forall } x \in D \text{ } P(x)}{P(a)}$$

This scheme often appears in particular specifications of first-order logic (see [2]), as most readers will well know. I have just used and used to derive something that is abstractness. The source of the scheme is, of course, not the one myself, as already it, or upon it, is a box and last but not least a gift to someone and fortunate enough to receive it.

¹Invited article by Thomas Bruggagel is an improvement of the original paper (2012).
 Thomas will readily agree that "I" and the larger identification of which is a usually a given value of, as presented here and of formal logic, which is a conceptual of the immaterial/abstractness of computer science as presented in Turner (2012) has a particular of formal logic (Turner regarding the formal logic machine).
 © Springer Science & Media
 ISSN: 1744-5019

1. Department of AI & Robotics (RAI) Laboratory, Research Institute for Systems (RIS), Troy, NY 12180, USA
 2. Department of Cognitive Science, Research Institute for Systems (RIS), Troy, NY 12180, USA
 3. Department of Computer Science, Research Institute for Systems (RIS), Troy, NY 12180, USA
 4. Lady Bird School of Management, Research Institute for Systems (RIS), Troy, NY 12180, USA

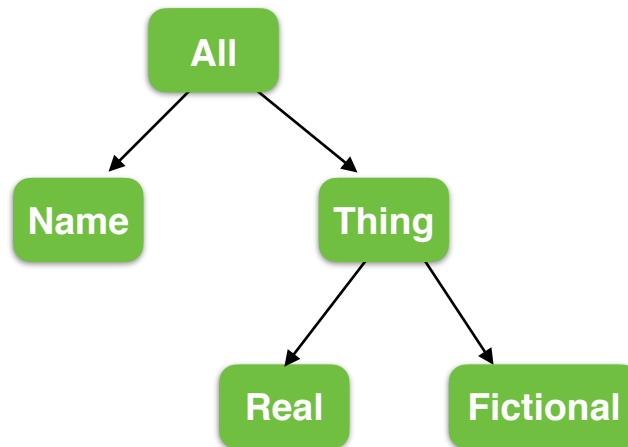
*Med nok penger, kan
logikk løse alle problemer.*

**Example #1:
Descartes' Cogito ...**

Cogito Ergo Sum

Sort system

```
(Believes! I (forall [x] (or (Name x) (Thing x))))  
(Believes! I (forall (x) (iff (Name x) (not (Thing x)))) )  
(Believes! I (forall (x) (if (Thing x) (or (Real x) (Fictional x)))) )  
(Believes! I (forall (x) (if (Thing x) (iff (Real x) (not (Fictional x)))) ) )
```



Cogito Ergo Sum

```
(Believes! I (forall (y) (if (Name y)
                             (iff (DeReExists y)
                                   (exists x (and (Real x) (= x (* y))))))))))
```

De re existence is real existence

```
(Believes! I (not (DeReExists I)))
```

I believe I don't really exist

Cogito Ergo Sum

Instance of axiom schema

```
(Believes!  
  I  
  (forall [?agent]  
    (if (Perceives! I (Believes! ?agent (not (DeReExists ?agent))))  
      (Real (* ?agent))))))
```

I believe that if I perceive an agent believe something then I believe that agent is real

Cogito Ergo Sum

```
{:name      "Cogito Ergo Sum"
 :description "A formalization of Descartes' Cogito Ergo Sum"
 :assumptions {

  S1 (Believes! I (forall [x] (or (Name x) (Thing x))))
  S2 (Believes! I (forall (x) (iff (Name x) (not (Thing x)))) )
  S3 (Believes! I (forall (x) (if (Thing x) (or (Real x) (Fictional x))))))
  S4 (Believes! I (forall (x) (if (Thing x) (iff (Real x) (not (Fictional x))))))
  ;;;
  A1 (Believes! I (forall (x) (if (Name x) (Thing (* x))))))
  A2 (Believes! I (forall (y) (if (Name y) (iff (DeReExists y) (exists x (and (Real x) (= x (* y))))))))

  ;;;
  ;
  Suppose (Believes! I (not (DeReExists I)))
  given (Believes! I (Name I))

  ;;;
  Perceive-the-belief (Believes! I (Perceives! I (Believes! I (not (DeReExists I))))))
  If_P_B (Believes!
    I
    (forall [?agent]
      (if (Perceives! I (Believes! ?agent (not (DeReExists ?agent))))
        (Real (* ?agent))))))
  }
:goal (and (Believes! I (not (Real (* I))))
           (Believes! I (Real (* I))))
}
```

1.7 seconds

Cogito Ergo Sum

```
{:name      "Cogito Ergo Sum"
 :description "A formalization of Descartes' Cogito Ergo Sum"
 :assumptions {

  S1 (Believes! I (forall [x] (or (Name x) (Thing x))))
  S2 (Believes! I (forall (x) (iff (Name x) (not (Thing x)))))
  S3 (Believes! I (forall (x) (if (Thing x) (or (Real x) (Fictional x)))))
  S4 (Believes! I (forall (x) (if (Thing x) (iff (Real x) (not (Fictional x)))))
  ;;;
  A1 (Believes! I (forall (x) (if (Name x) (Thing (* x)))))
  A2 (Believes! I (forall (y) (if (Name y) (iff (DeReExists y) (exists x (and (Real x) (= x (* y)))))))

  ;;;
  ;
  Suppose (Believes! I (not (DeReExists I)))
  given (Believes! I (Name I))

  ;;;
  Perceive-the-belief (Believes! I (Perceives! I (Believes! I (not (DeReExists I)))))
  If_P_B (Believes!
    I
    (forall [?agent]
      (if (Perceives! I (Believes! ?agent (not (DeReExists ?agent))))
        (Real (* ?agent)))))
  }

:goal (and (Believes! I (not (Real (* I))))
           (Believes! I (Real (* I))))
}
```

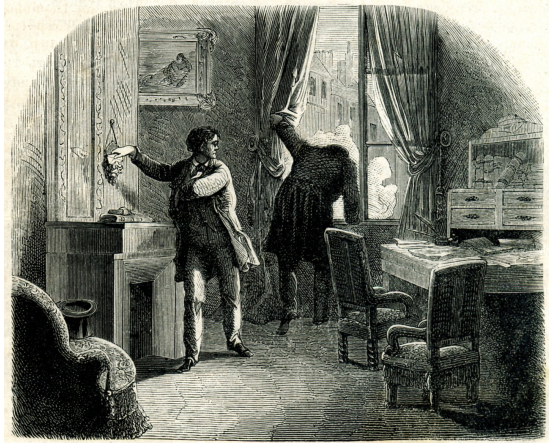
absurd belief

1.7 seconds

Example #3:
Poe's Detective Dupin ...



Shadow Prover



The Purloined Letter

```
{:name      "The Purloined Letter"  
 :description "Dupin's reasoning as he goes through the case"  
  
:assumptions {1 (Believes! g (hide m elaborate))  
 2 (Believes! d (or (hide m elaborate) (hide m plain)))  
 3 (Believes! m (Believes! g (hide m elaborate)))  
 4 (if (Believes! m (Believes! g (hide m elaborate))) (hide m plain))  
 5 (if (Believes! m (Believes! g (hide m plain))) (hide m elaborate))  
 6 (Believes! m (Believes! g (hide m elaborate)))  
 7 (Believes! d (if (Believes! m (Believes! g (hide m elaborate))) (hide m plain)))  
 8 (Believes! d (if (Believes! m (Believes! g (hide m plain))) (hide m elaborate)))  
 9 (Believes! d (Believes! m (Believes! g (hide m elaborate))))}  
  
:goal (Believes! d (hide m plain))}
```

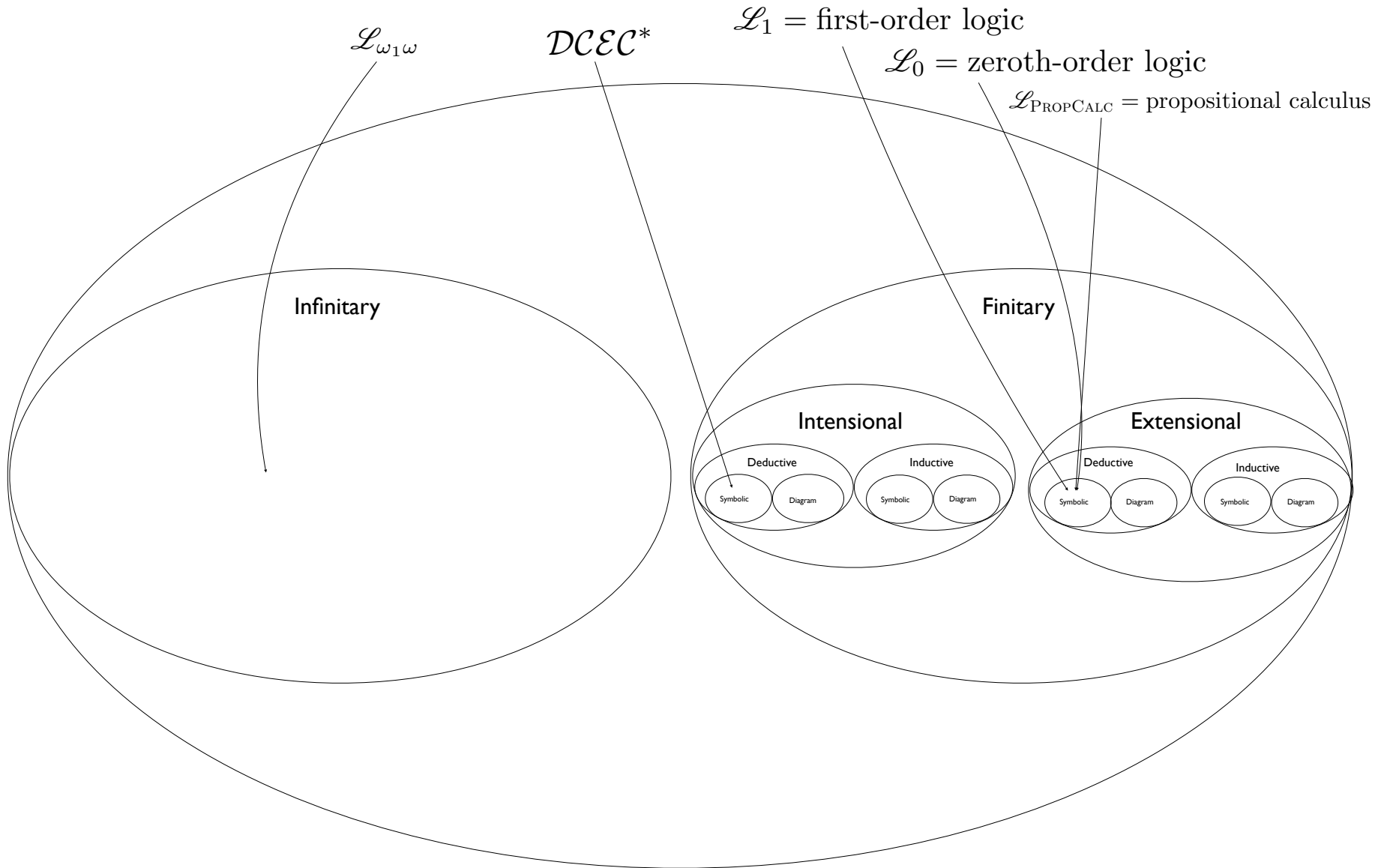
55 ms

Example #3:

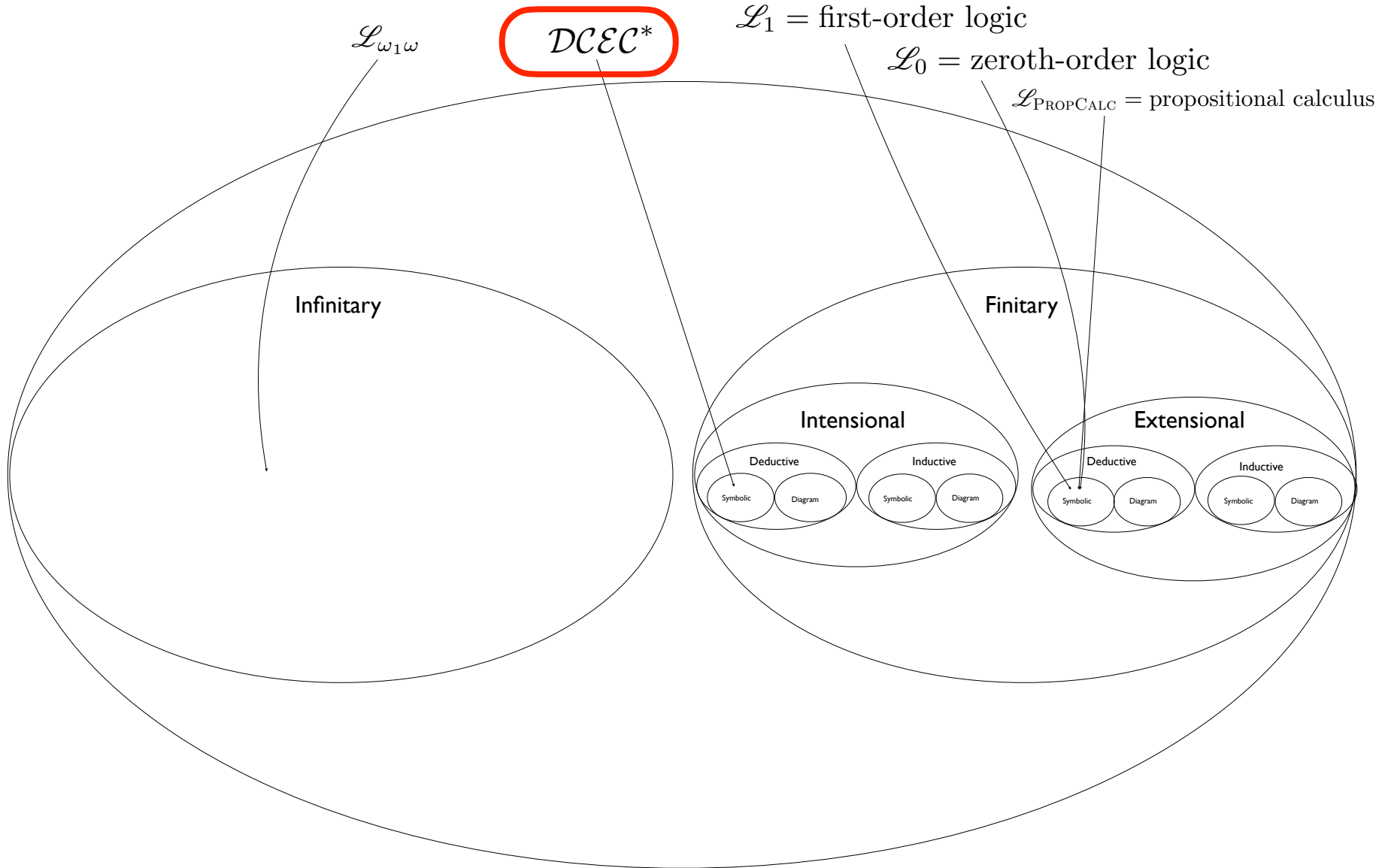
Ethical Control via a Program
Based on *DCEC** + ShadowProver

...

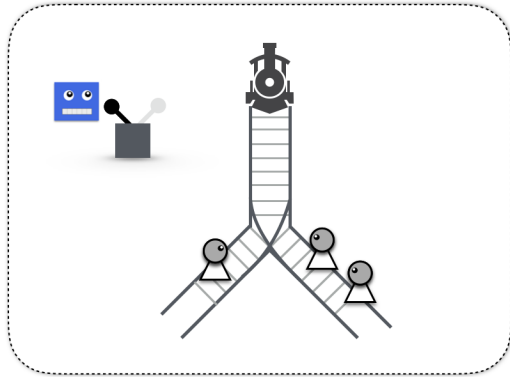
The Universe of Logics



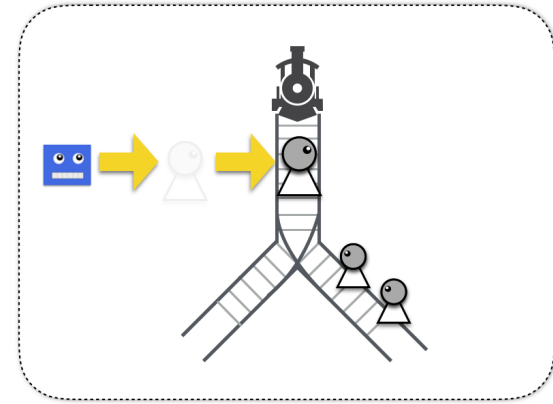
The Universe of Logics



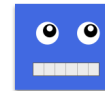
A Trolley Dilemma



This is allowed



This is not allowed!



Doctrine of Double Effect *DDÉ*

Doctrine of Double Effect *DDÉ*

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

Doctrine of Double Effect *DDÉ*

- 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 *DDÉ*

- 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 *DDÉ*



- 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

- C₁** 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);
- C₂** 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;
- C₄** the bad effects are not used as a means to obtain the good effects; and
- C₅** 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

- C₁** 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);
- C₂** 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;
- C₄** the bad effects are not used as a means to obtain the good effects; and
- C₅** 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



Univer
sal
Cognitiv

*DCEC**

1. The first part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.	2. The second part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.
3. The third part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.	4. The fourth part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.
5. The fifth part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.	6. The sixth part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.
7. The seventh part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.	8. The eighth part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.
9. The ninth part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.	10. The tenth part of the document is a list of the names of the members of the committee who have been appointed to study the subject of the DCEC.

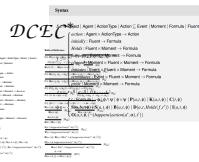


.5 centuries

“Univer
sal



Univer
sal
Cognitiv



.5 centuries

Syntax

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

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

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

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

“Univer
sal



Univers
al
Cognitiv

*DCEC**



.5 centuries

Syntax

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

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

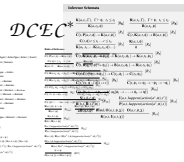
$t ::= x : S \mid c : S \mid f(t_1, \dots, t_n)$

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

“Univer
sal



Univers
al
Cognitiv



.5 centuries

Syntax

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

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

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

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

“Univer
sal



Univers
al
Cognitiv

*DCEC**

Agent	Time	Location	Event	Formula
...



R A I R
Reinstate AI and Reasoning Lab

5 centuries

Inference Schemata

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

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

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

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

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

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

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

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

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

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





Formal Conditions for \mathcal{DDE}

F₁ α carried out at t is not forbidden. That is:

$$\Gamma \not\vdash \neg \mathbf{O}(a, t, \sigma, \neg \text{happens}(\text{action}(a, \alpha), t))$$

F₂ 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. (**F₂** 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:

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

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:

$$\Gamma \not\vdash \mathbf{I}(a, t, \text{Holds}(f_b, y)) \text{ and}$$

$$\Gamma \not\vdash \mathbf{I}(a, t, \neg \text{Holds}(f_g, y))$$

F₄ The harmful effects don't cause the good effects. Four permutations, paralleling the definition of \triangleright 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:

$$\Gamma \vdash \neg \triangleright (\text{Holds}(f_b, t_1), \text{Holds}(f_g, t_2))$$



Formal Conditions for \mathcal{DDE}

F₁ α carried out at t is not forbidden. That is:

$$\Gamma \not\vdash \neg \mathbf{O}(a, t, \sigma, \neg \text{happens}(\text{action}(a, \alpha), t))$$

F₂ 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. (**F₂** 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:

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

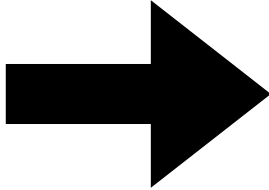
F_{3b} The agent a does not intend any bad effect. For all fluents f_b in $\alpha_I^{a,t}$ with $\mu(f_b, y) < 0$, or f_g in $\alpha_T^{a,t}$ with $\mu(f_g, y) > 0$, and for all y such that $t < y \leq H$ the following holds:

$$\Gamma \not\vdash \mathbf{I}(a, t, \text{Holds}(f_b, y)) \text{ and}$$

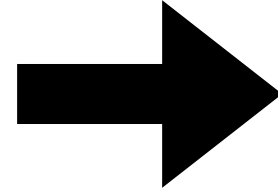
$$\Gamma \not\vdash \mathbf{I}(a, t, \neg \text{Holds}(f_g, y))$$

F₄ The harmful effects don't cause the good effects. Four permutations, paralleling the definition of \triangleright 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:

$$\Gamma \vdash \neg \triangleright (\text{Holds}(f_b, t_1), \text{Holds}(f_g, t_2))$$



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



Formal Conditions for \mathcal{DDE}

F₁ α carried out at t is not forbidden. That is:

$$\Gamma \not\vdash \neg \mathbf{O}(a, t, \sigma, \neg \text{happens}(\text{action}(a, \alpha), t))$$

F₂ 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. (**F₂** 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:

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

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:

$$\Gamma \not\vdash \mathbf{I}(a, t, \text{Holds}(f_b, y)) \text{ and}$$

$$\Gamma \not\vdash \mathbf{I}(a, t, \neg \text{Holds}(f_g, y))$$

F₄ The harmful effects don't cause the good effects. Four permutations, paralleling the definition of \triangleright 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:

$$\Gamma \vdash \neg \triangleright (\text{Holds}(f_b, t_1), \text{Holds}(f_g, t_2))$$



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

