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 4 2021
ver. 1104211145NY

R A I R
Rensselaer AI and Reasoning Lab

Some Review & Logistics

Some Review & Logistics

- Re exploring Polynomial Hierarchy in HyperSlate®; e.g. **3SAT**? …

Some Review & Logistics

- Re exploring Polynomial Hierarchy in HyperSlate®; e.g. **3SAT**? …
- Let's visit Overleaf, & our paper-topic file …

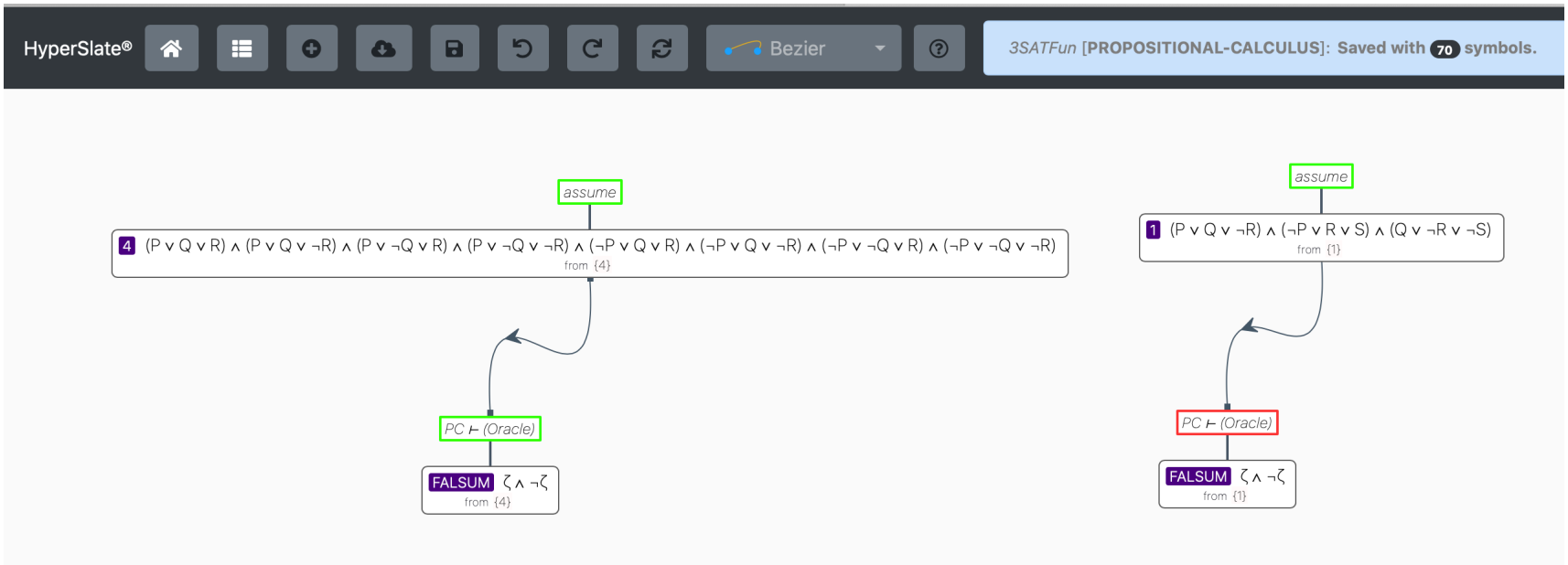
Some Review & Logistics

- Re exploring Polynomial Hierarchy in HyperSlate®; e.g. **3SAT**? …
- Let's visit Overleaf, & our paper-topic file …
- Let's visit our web site …

Some Review & Logistics

- Re exploring Polynomial Hierarchy in HyperSlate®; e.g. **3SAT**? …
- Let's visit Overleaf, & our paper-topic file …
- Let's visit our web site …

3SATFun


HyperSlate®
Bezier
3SATFun [PROPOSITIONAL-CALCULUS]: Saved with 70 symbols.
assume
4 (P ∨ Q ∨ R) ∧ (P ∨ Q ∨ ¬R) ∧ (P ∨ ¬Q ∨ R) ∧ (P ∨ ¬Q ∨ ¬R) ∧ (¬P ∨ Q ∨ R) ∧ (¬P ∨ Q ∨ ¬R) ∧ (¬P ∨ ¬Q ∨ R) ∧ (¬P ∨ ¬Q ∨ ¬R)
from {4}
PC ⊢ (Oracle)
FALSUM ζ ∧ ¬ζ
from {4}
assume
1 (P ∨ Q ∨ ¬R) ∧ (¬P ∨ R ∨ S) ∧ (Q ∨ ¬R ∨ ¬S)
from {1}
PC ⊢ (Oracle)
FALSUM ζ ∧ ¬ζ
from {1}

We can prove $\mathbf{3SAT} \in \Sigma_1^P = \mathbf{NP}$ because we have a polytime relation R s.t. $\phi \in \mathbf{3SAT}$ iff

$$\exists \bar{y} R(\phi \in \mathscr{L}_{pc}, \bar{y} = \langle \text{assignments to the 3 Boolean vars} \rangle)$$

,

where these assignments produce truth.

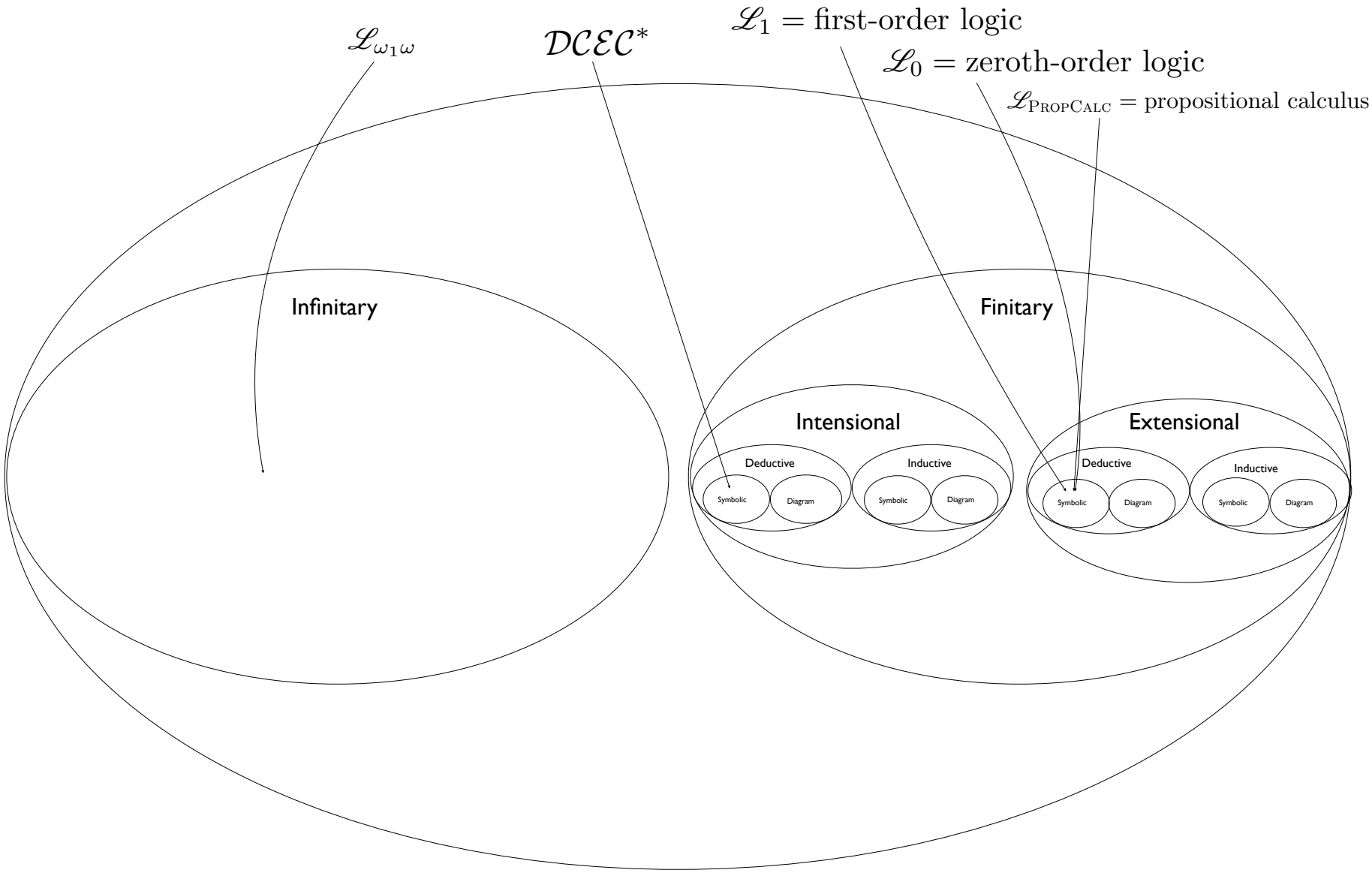
Some Logistics

Some Logistics

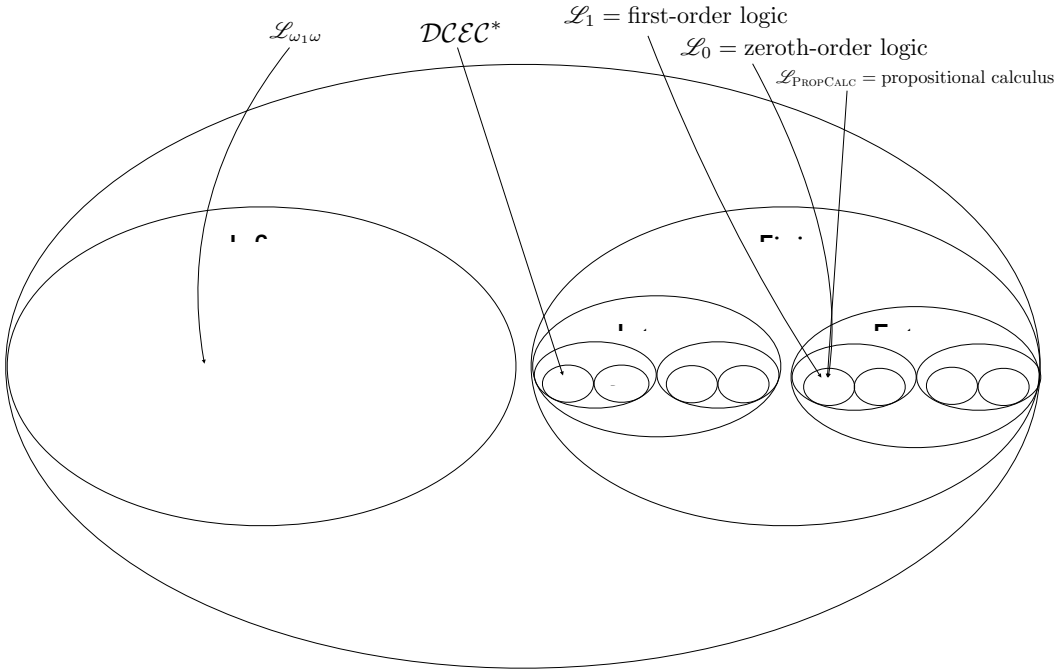
- Re exploring Polynomial Hierarchy in HyperSlate®; e.g. **3SAT**? …
- Let's visit Overleaf, & our paper-topic file …
- Let's visit our web site …

Reminders …

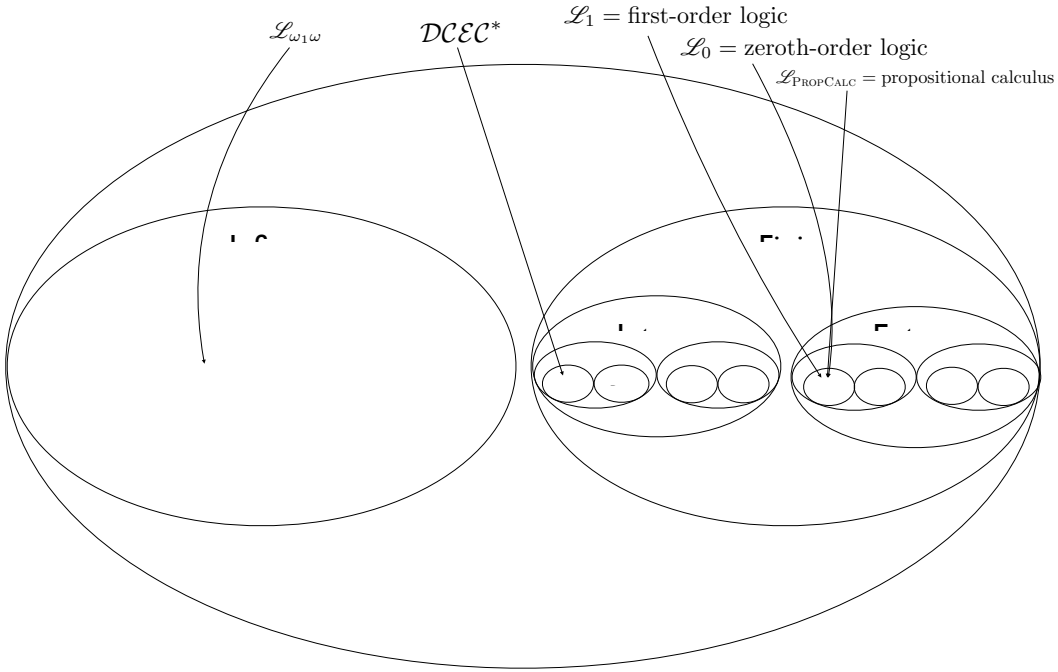
The Universe of Logics



The Universe of Logics

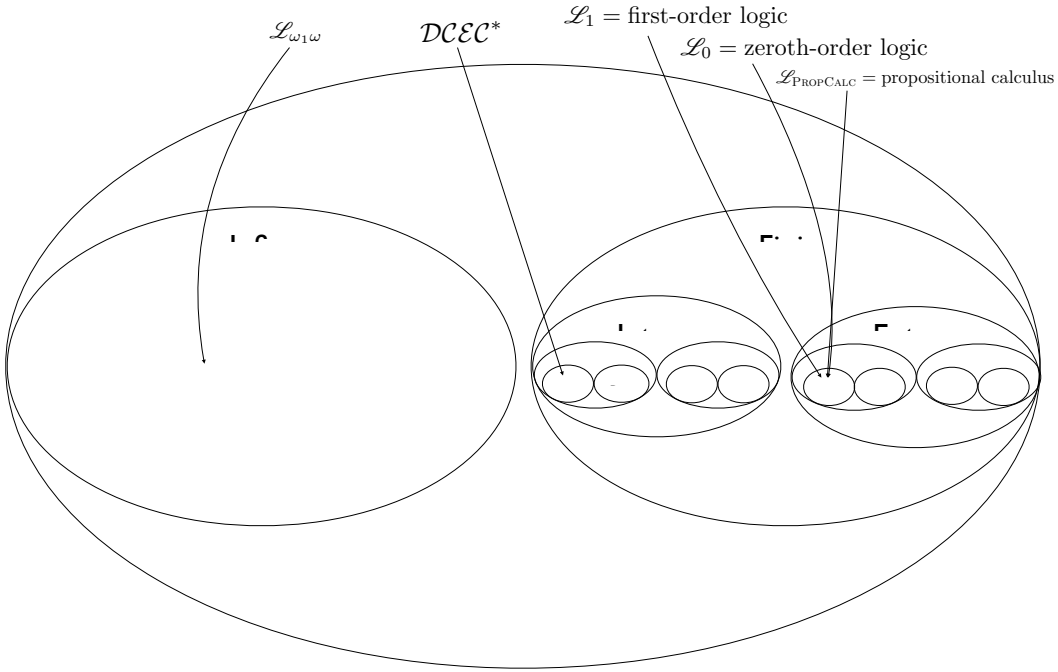


The Universe of Logics



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

The Universe of Logics



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

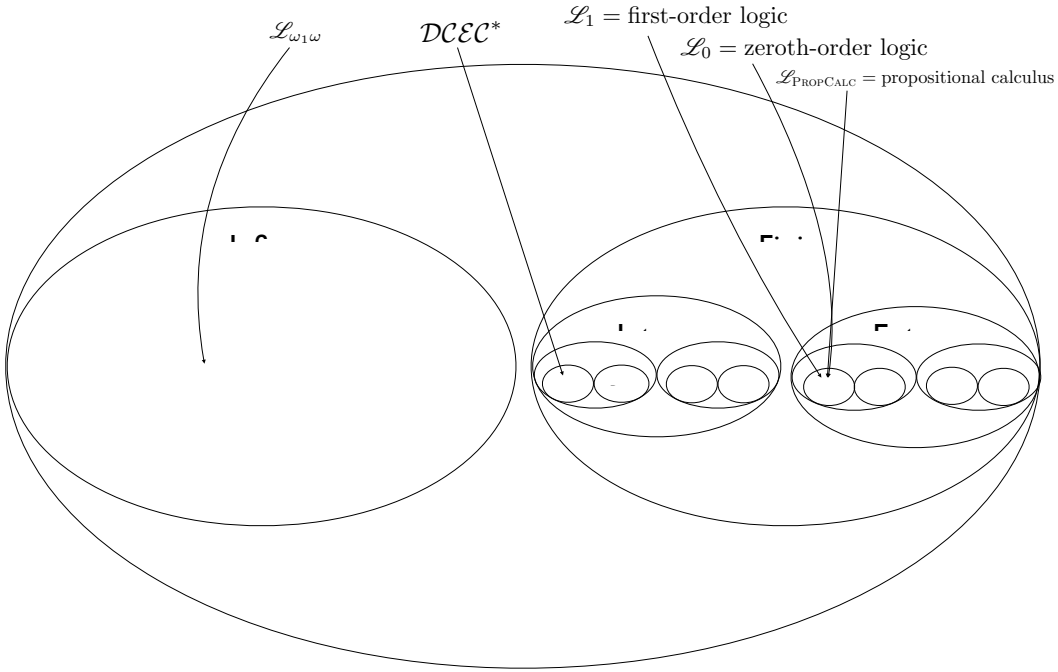
the logic

formal language

inference schemata

semantics

The Universe of Logics



$$\mathscr{L} := \langle L, I, \quad \rangle$$

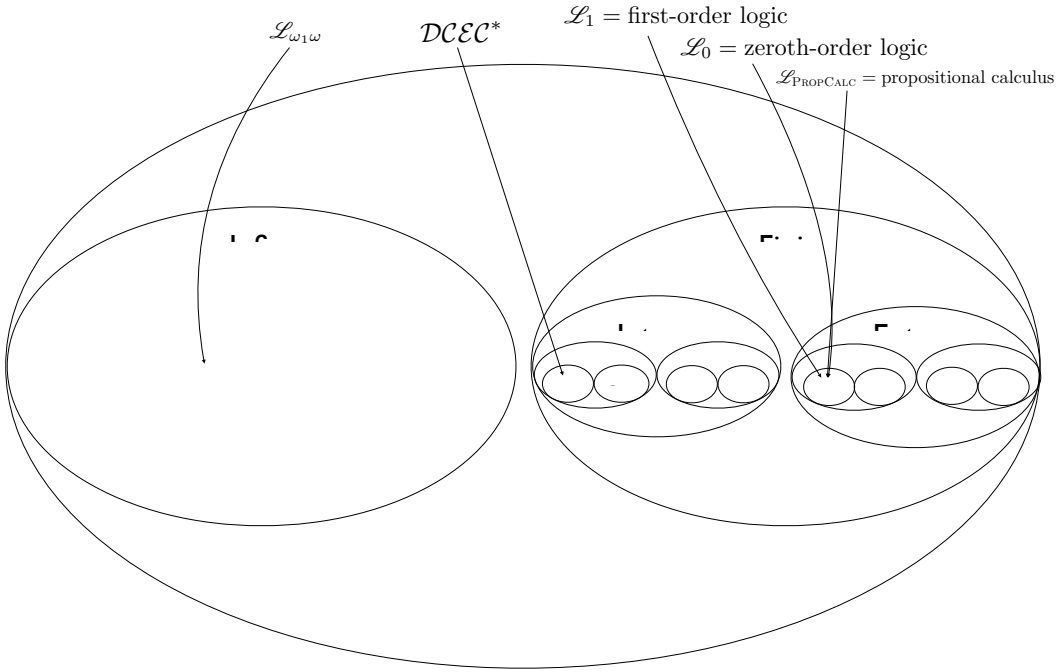
the logic

formal language

inference schemata

semantics

The Universe of Logics



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

the logic

formal language

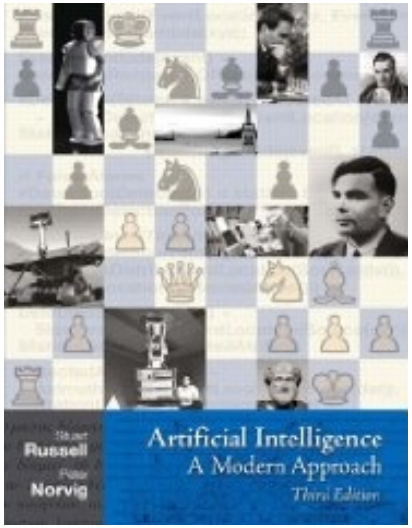
inference schemata

semantics

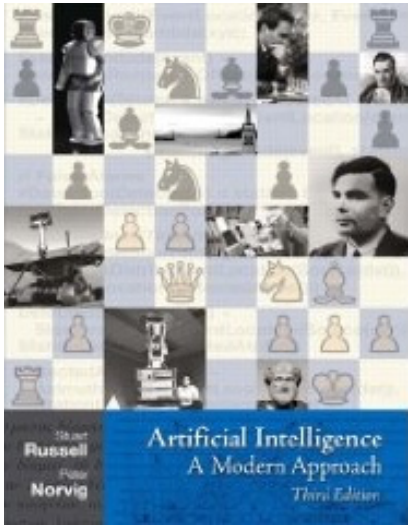
AI …

AI:

AI:


Russell
Norvig
Artificial Intelligence
A Modern Approach
Third Edition

AI:



Stanford Encyclopedia of Philosophy

Browse About Support SEP

Search SEP

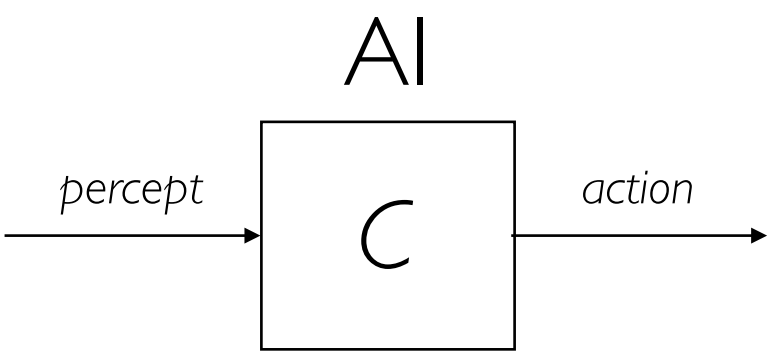
Entry Contents
Bibliography
Academic Tools
Friends PDF Preview
Author and Citation Info
Back to Top

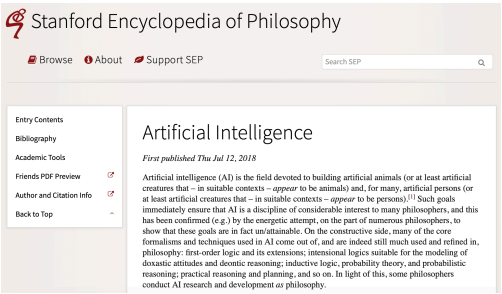
Artificial Intelligence

First published Thu Jul 12, 2018

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

AI:





Stanford Encyclopedia of Philosophy

Browse About Support SEP

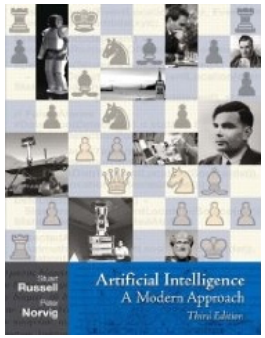
Search SEP

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

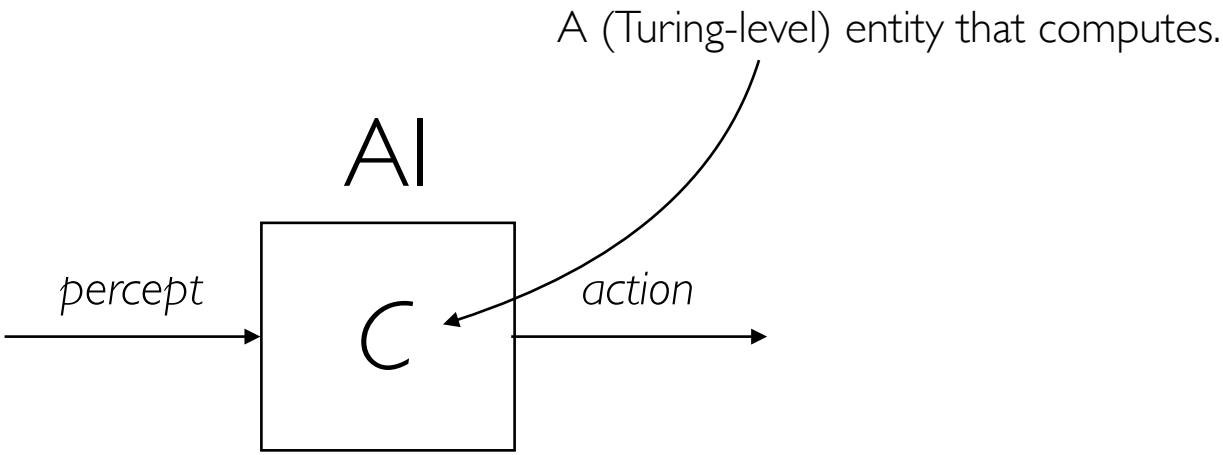
Artificial Intelligence

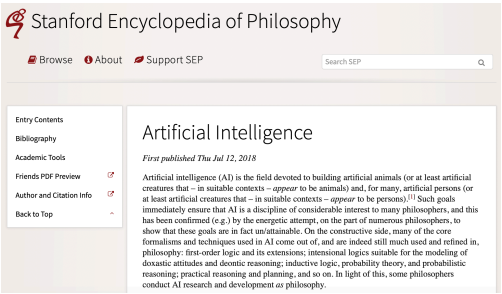
First published Thu Jul 12, 2018

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



AI:





Stanford Encyclopedia of Philosophy

Browse About Support SEP

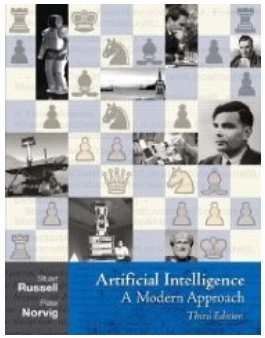
Search SEP

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.



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

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

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

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

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

Computational thinking thus has the following characteristics:

Computational Thinking

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

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

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

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

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

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

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

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

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 incentivize about this vision is that computing and computers facilitate the spread of computational thinking.

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

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

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

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

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

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

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

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

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

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

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 incentivize about this vision is that computing and computers facilitate the spread of computational thinking.

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

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

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

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

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

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

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 incentivize about this vision is that computing and computers facilitate the spread of computational thinking.

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

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

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

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

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

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

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

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

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

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

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

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

efficiently,
practically,
physically,
and ethically.

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 incentivize about this vision is that computing and computers facilitate the spread of computational thinking.

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

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

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

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

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

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

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

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

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

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

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

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

efficiently,
practically,
physically,
and ethically.

– Rapaport, "phics" book

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

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

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

program

query

degree of "confidence"

proof(s)

argument(s)

reasoner

checker

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.

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

program

query

degree of "confidence"

proof(s)

argument(s)

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

reasoner

checker

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.

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

program

query

degree of "confidence"

proof(s)

argument(s)

$$\begin{array}{ll} \mathbb{P} & L \\ \mathfrak{q} & L \\ \hline \end{array}$$

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

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

reasoner

checker

What about infinitary logics?

Allows e.g. $\phi_1 \vee \phi_2 \vee \ldots$

1	$\phi_1 \vee \phi_2 \vee \ldots$
2	$\neg\phi_{634}$

$\mathscr{L}_{\omega_1\omega}$

$\phi_1 \vee \phi_2 \vee \phi_{633} \vee \phi_{635} \vee \ldots$

$$\mathbb{P} \qquad L$$

$$\mathfrak{q} \qquad L$$

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

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

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

What about infinitary logics?

Allows e.g. $\phi_1 \vee \phi_2 \vee \ldots$

1	$\phi_1 \vee \phi_2 \vee \ldots$
2	$\neg\phi_{634}$

$\mathscr{L}_{\omega_1\omega}$

$\phi_1 \vee \phi_2 \vee \phi_{633} \vee \phi_{635} \vee \ldots$

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

$$\begin{array}{cc} \mathbb{P} & L \\ \mathfrak{q} & L \end{array}$$

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

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

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

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

Semantics

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

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

Semantics

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

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

Semantics

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

$\mathscr{L}$

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

Semantics

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

$\mathscr{L}$

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

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

Semantics

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

$\mathscr{L}$

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

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

Semantics

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

$\mathscr{L}$

+ ShadowReasoner

On the Anatomy of a PGLP Program

Linguistics

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

Inference

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

Semantics

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

$\mathscr{L}$

Logic-Machines Hierarchy

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Ali has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Ali's, namely that no Turing machine can be a universal computer, the Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Ali's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Ali, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Ali's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathcal{A}_{Ali}$. Because I shall use '$\mathcal{R}_{TM}$' to denote the statement that no Turing machine can be a universal computer, the subscript in '$\mathcal{A}_{Ali}$' is just a convenient reminder that $\mathcal{R}_{TM}$ is the conclusion of this argument.

* I'm indebted to Selim Ali for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and supernatural patience.

1

Logic-Machines Hierarchy

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that no Turing machine can be a universal computer, the Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathcal{A}_{\mathrm{Alì}}$. Because I shall use '$\mathrm{Eq}_{\mathcal{T}}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathcal{A}_{\mathrm{Alì}}$ is just a convenient reminder that $\mathrm{Eq}_{\mathcal{T}}$ is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and supernatural patience.

1

Logic-Machines Hierarchy

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that universal computation is equivalent to Turing's and Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathcal{A}_{\mathrm{Alì}}$. Because I shall use '$\mathrm{Eq_T}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathcal{A}_{\mathrm{Alì}}$ is just a convenient reminder that $\mathrm{Eq_T}$ is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Arkoudas for guidance and representational patience.

1

Logic-Machines Hierarchy

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_0 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that universal computation is not equivalent to Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathcal{A}_{\mathrm{Alì}}$. Because I shall use '$\mathrm{Eq_T}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathcal{A}_{\mathrm{Alì}}$ is just a convenient reminder that $\mathrm{Eq_T}$ is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Arkoudas for guidance and representational patience.

1

Logic-Machines Hierarchy

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_1 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_0 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that universal computation is equivalent to Church-Turing Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, in a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathcal{A}_{\mathrm{Alì}}$. Because I shall use '$\mathrm{Krg}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathcal{A}_{\mathrm{Alì}}$ is just a convenient reminder that Krg is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Arkoudariy for guidance and representational patience.

1

Logic-Machines Hierarchy

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_2 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_1 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_0 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that universal computation is not equivalent to Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as '$\mathcal{A}_{\mathrm{Myt}}$'. Because I shall use '$\mathrm{Myt}$' to denote the statement that no Turing machine can be a universal computer, the subscript in '$\mathcal{A}_{\mathrm{Myt}}$' is just a convenient reminder that Myt is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and representational patience.

1

Logic-Machines Hierarchy

$$\vdots$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_2 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_1 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_0 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that no machine can be a universal computer, given the Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, in a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathscr{A}_{\mathrm{Alì}}$. Because I shall use '$\mathscr{A}_{\mathrm{Alì}}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathscr{A}_{\mathrm{Alì}}$ is just a convenient reminder that $\mathscr{A}_{\mathrm{Alì}}$ is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and representational patience.

1

Logic-Machines Hierarchy

$$\vdots$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_2 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_1 \quad \mathbb{R} \quad \mathbb{C}$$

PH $\quad \mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_0 \quad \mathbb{R} \quad \mathbb{C}$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that no physical machine can be a universal machine in the Kolmogorov-Uspensky sense. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathcal{A}_{\mathrm{Alì}}$. Because I shall use '$\mathrm{Myth}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathcal{A}_{\mathrm{Alì}}$ is just a convenient reminder that Myth is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and supernatural patience.

1

Logic-Machines Hierarchy

$$\vdots$$

	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_2$	$\mathbb{R}$	$\mathbb{C}$
Turing Machines	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_1$	$\mathbb{R}$	$\mathbb{C}$
PH	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_0$	$\mathbb{R}$	$\mathbb{C}$
	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_{\mathrm{PC}}$	$\mathbb{R}$	$\mathbb{C}$

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that no physical machine can be a universal machine like a Kolmogorov-Uspensky machine. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, in a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as '$\mathcal{A}_{\mathrm{Myth}}$'. Because I shall use '$\mathcal{A}_{\mathrm{Myth}}$' to denote the statement that no Turing machine can be a universal computer, the subscript in '$\mathcal{A}_{\mathrm{Myth}}$' is just a convenient reminder that $\mathcal{A}_{\mathrm{Myth}}$ is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and representational patience.

1

Logic-Machines Hierarchy

$$\vdots$$

	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_2$	$\mathbb{R}$	$\mathbb{C}$
Turing Machines	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_1$	$\mathbb{R}$	$\mathbb{C}$
PH	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_0$	$\mathbb{R}$	$\mathbb{C}$
	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_{\mathrm{PC}}$	$\mathbb{R}$	$\mathbb{C}$

GCI

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that universal computation is equivalent to Turing's and Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathscr{A}_{\mathrm{Myth}}$. Because I shall use '$\mathrm{Myth}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathscr{A}_{\mathrm{Myth}}$ is just a convenient reminder that Myth is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and representational patience.

1

Logic-Machines Hierarchy

$$\vdots$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_2 \quad \mathbb{R} \quad \mathbb{C}$$

Turing Machines

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_1 \quad \mathbb{R} \quad \mathbb{C}$$

PH

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_0 \quad \mathbb{R} \quad \mathbb{C}$$

$$\mathbb{P} \quad \mathfrak{q} \quad \mathscr{L}_{\mathrm{PC}} \quad \mathbb{R} \quad \mathbb{C}$$

GCI

program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker

Chapter 1
Is Universal Computation a Myth?*

Selmer Bringsjord

Abstract Alì has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than Alì's, namely that universal computation is not equivalent to Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

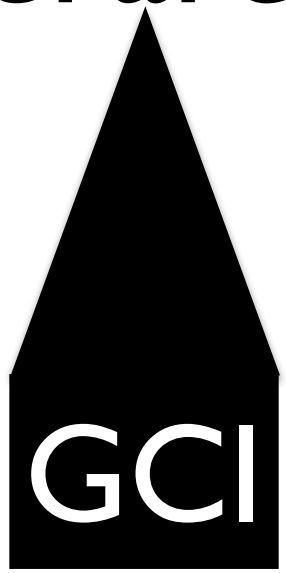
1.1 Introduction

Selim Alì's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" (Alì, 2005). My analysis, to a further narrowing, is specifically targeted at a key argument of Alì's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as $\mathcal{A}_{\mathrm{Myth}}$. Because I shall use '$\mathrm{Myth}$' to denote the statement that no Turing machine can be a universal computer, the subscript in $\mathcal{A}_{\mathrm{Myth}}$ is just a convenient reminder that Myth is the conclusion of this argument.

* I'm indebted to Selim Alì for bringing to my attention countless stimulating ideas, only one of which I explore herein. Many thanks are due as well to Andy Adamatzky for guidance and representational patience.

1

Logic-Machines Hierarchy



$$\vdots$$

	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_2$	$\mathbb{R}$	$\mathbb{C}$
Turing Machines	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_1$	$\mathbb{R}$	$\mathbb{C}$
PH	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_0$	$\mathbb{R}$	$\mathbb{C}$
	$\mathbb{P}$	$\mathfrak{q}$	$\mathscr{L}_{\mathrm{PC}}$	$\mathbb{R}$	$\mathbb{C}$

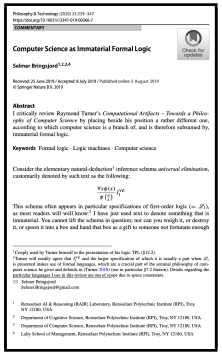
program

query

logical system
(prop calc here)

automated reasoner

proof/argument checker



Philosophy & Technology (2020) 33:339–347
https://doi.org/10.1007/s13347-019-00366-7

COMMENTARY

Check for updates

Computer Science as Immaterial Formal Logic

Selmer Bringsjord[1,2,3,4]

Received: 25 June 2019 / Accepted: 8 July 2019 / Published online: 5 August 2019
© Springer Nature B.V. 2019

Abstract
I critically review Raymond Turner's *Computational Artifacts – Towards a Philosophy of Computer Science* by placing beside 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 elementary natural-deduction[1] inference schema *universal elimination*, customarily denoted by such text as the following:

$$\frac{\forall x\, \phi(x)}{\phi(t)}\ \forall E$$

This schema often appears in particular specifications of first-order logic (= $\mathscr{L}_1$), as most readers will well know.[2] I have just used text to denote something that is immaterial. You cannot lift the schema in question; nor can you weigh it, or destroy it, or spoon it into a box and hand that box as a gift to someone not Fortunate enough

[1] Crisply used by Turner himself in the presentation of his logic TPL (§12.2).
[2] Turner will readily agree that $\mathscr{L}_1^{\mathbb{N}}$ and the larger specification of which it is usually a part when $\mathscr{L}_1$ is presented makes use of formal languages, which are a crucial part of the minimal philosophy of computer science he gives and defends in (Turner 2018) (see in particular §7.2 theorem). Details regarding the relevant formal languages in this review are out of scope due to space constraints.

✉ Selmer Bringsjord
Selmer.Bringsjord@gmail.com

1 Rensselaer AI & Reasoning (RAR) Laboratory, Rensselaer Polytechnic Institute (RPI), Troy, NY 12180, USA
2 Department of Cognitive Science, Rensselaer Polytechnic Institute (RPI), Troy, NY 12180, USA
3 Department of Computer Science, Rensselaer Polytechnic Institute (RPI), Troy, NY 12180, USA
4 Lally School of Management, Rensselaer Polytechnic Institute (RPI), Troy, NY 12180, USA

Med nok penger, kan logikk løse alle problemer.