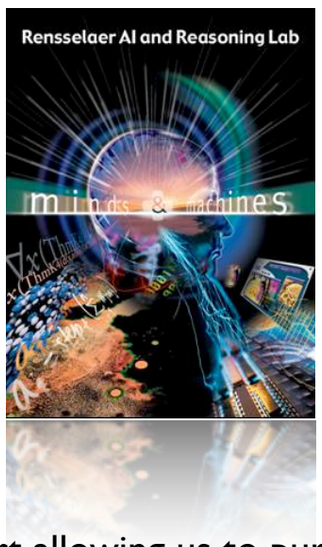
# In Further Defense of the ... Unprovability of the Church-Turing Thesis*

Selmer Bringsjord & Naveen Sundar G.

Department of Computer Science
Department of Cognitive Science
Lally School of Management & Technology
Rensselaer Polytechnic Institute (RPI)
Troy NY 12180 USA
selmer@rpi.edu • govinn@rpi.edu
June 4 2011

Trends in Logic IX
*Studia Logica* International Conference:
*Church Thesis: Logic, Mind and Nature*
Kraków Poland




*We are indebted to the John Templeton Foundation for support allowing us to pursue Palette$^{\infty}$ Machines.

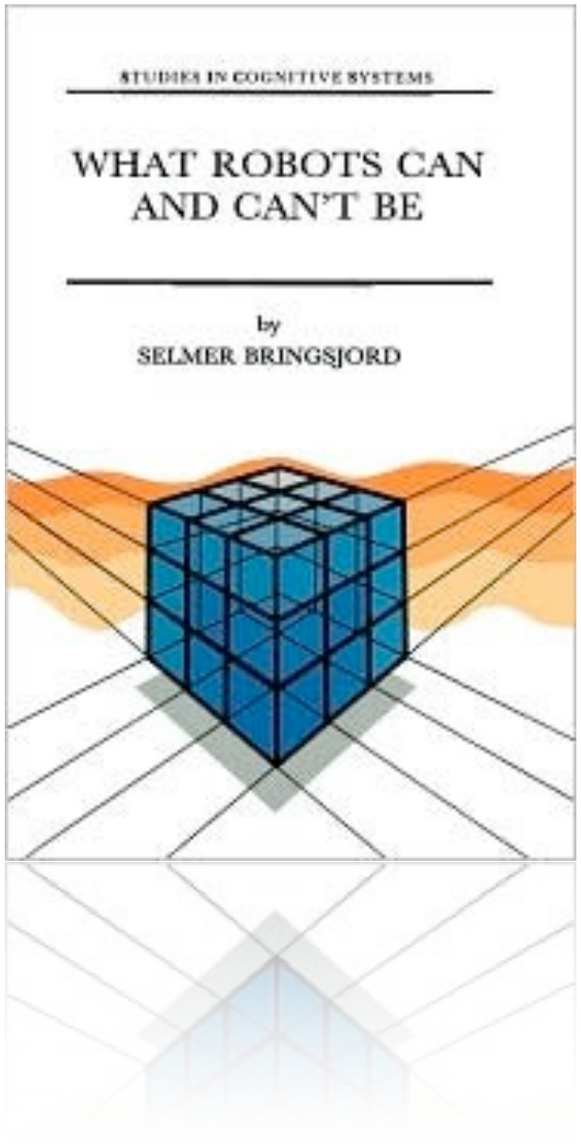
# Some Context ...

Turing machines, Abstract State Machines, Pointer Machines, etc.; all model just a tiny, *dim* part of human information processing ....

Human Cognition Includes — using terminology courtesy of Copeland — "Hyperlooping" (because human persons have free will)

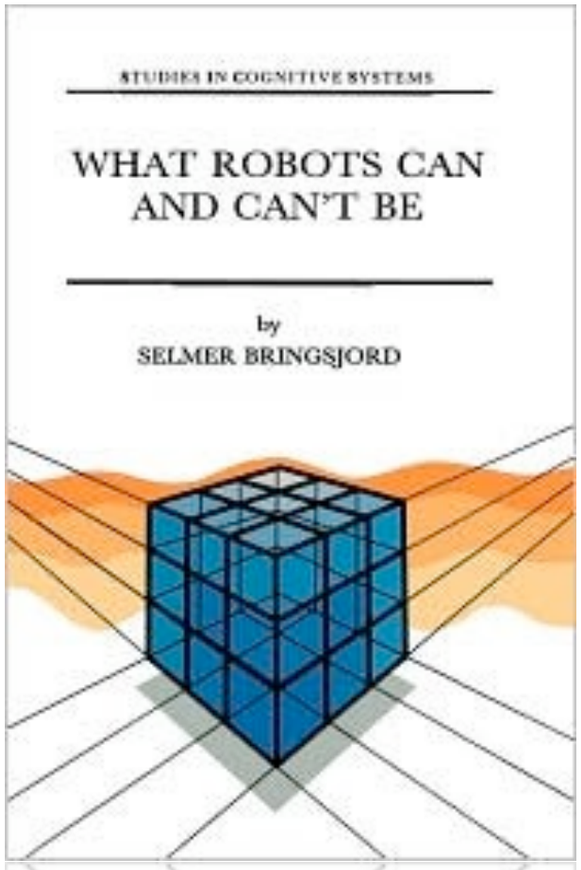
# Human Cognition Includes — using terminology courtesy of Copeland — "Hyperlooping" (because human persons have free will)

1992



Human Cognition Includes — using terminology courtesy of Copeland — "Hyperlooping" (because human persons have free will)

1992



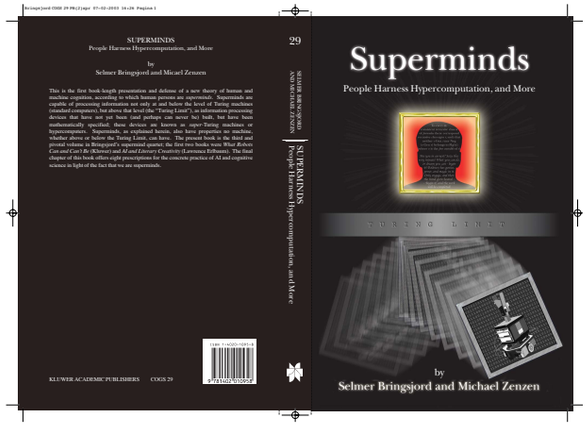

1987

*The Failure of Computationalism*

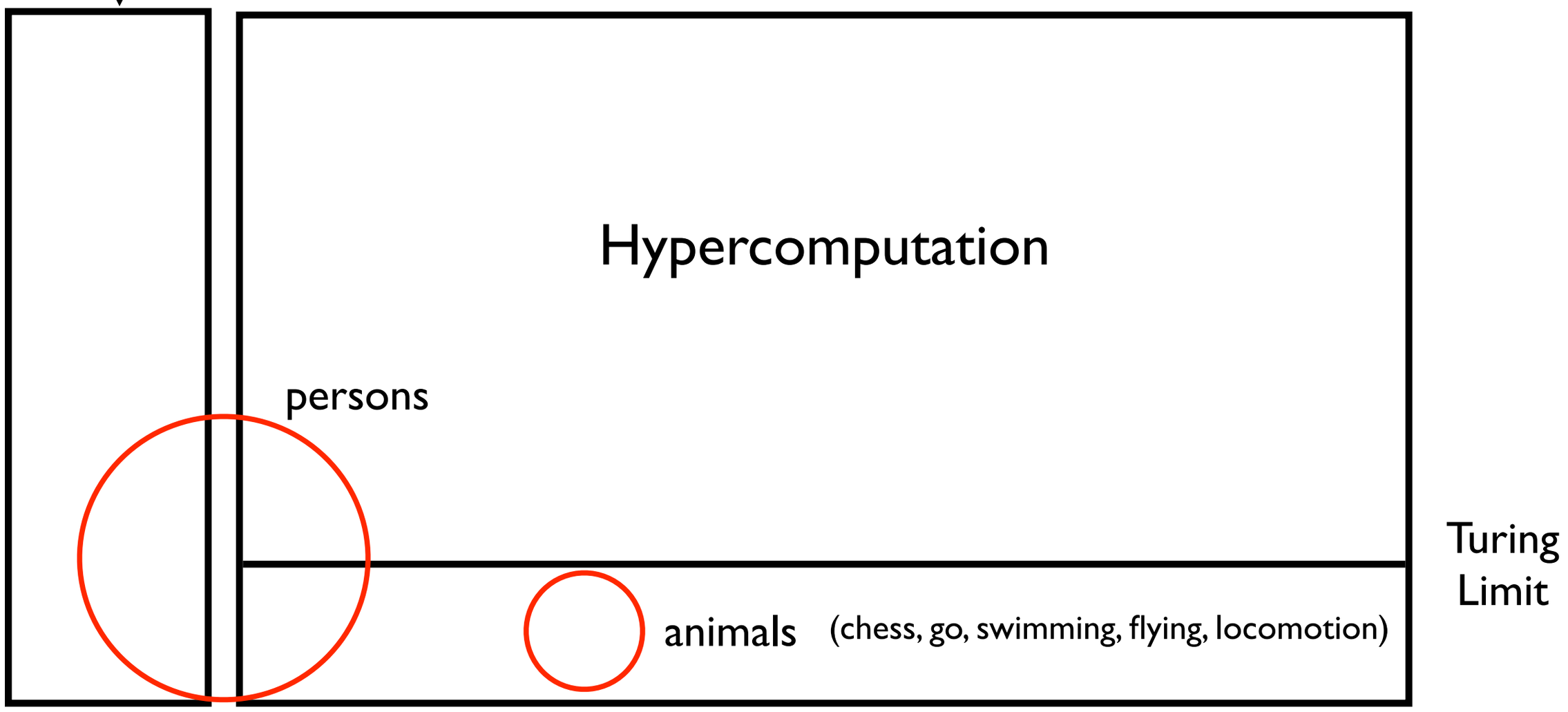
# *Superminds*

*People Harness Hypercomputation, and More*

(2003)



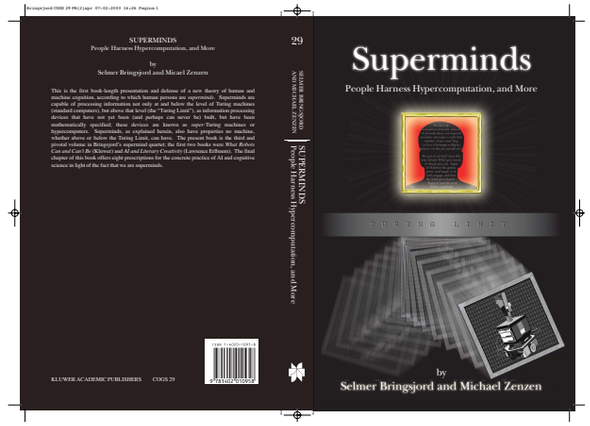

Subjective consciousness, qualia, etc. — phenomena in the incorporeal realm that can't be expressed in any third-person scheme



# *Superminds*

*People Harness Hypercomputation, and More*

(2003)




Subjective consciousness, qualia, etc. — phenomena in the incorporeal realm that can't be expressed in any third-person scheme

## Information Processing

## Hypercomputation

persons

$$\forall u \forall v [\exists k H(n, k, u, v) \leftrightarrow \exists k' H(m, k', u, v)]$$

(programming)

animals (chess, go, swimming, flying, locomotion)

Turing Limit

# Disciplinary Perspectives ...

There are different permutations for “flight paths” that will allow one to take off and consider CT/TT/CTT ...

Logic

AI/CogSci

Applied Computer Science

Theoretical Computer Science

Physics

Philosophy

Mathematics

Logic

AI/CogSci

Applied Computer Science

Theoretical Computer Science

Physics

Philosophy

Mathematics

*Cognitive* Take on CT/TT/CTT, which are after all intrinsically cognitive theses.

Logic

AI/CogSci

Applied Computer Science

Theoretical Computer Science

Physics

Philosophy

Mathematics

Logic

AI/CogSci

Applied Computer Science

Theoretical Computer Science

Physics

Philosophy

Mathematics

Logic

AI/CogSci

Applied Computer Science

Theoretical Computer Science

Physics

Mathematics

Philosophy

Since the only rational route for trying to establish CT/CTT is based on a rigorous analysis of *person* (perhaps a "partial" person) and *algorithm*, philosophy is indispensable for anyone aiming to establish CT/CTT.

# Some Relevant Prior Work on CT/TT/CTT ...

# Refutation of Mendelson and Argument Against CTT

Mendelson, E. (1986) "Second Thoughts About Church's Thesis and Mathematical Proofs" *Journal of Philosophy* **87.5**: 225–233.

Bringsjord, S. & Arkoudas, K. (2006) "On the Provability, Veracity, and AI-Relevance of the Church-Turing Thesis" in A. Olszewski, J. Wolenski & R. Janusz, eds., *Church's Thesis After 70 Years* (Frankfurt, Germany: Ontos Verlag), pp. 66–118. This book is in the series *Mathematical Logic*, edited by W. Pohlers, T. Scanlon, E. Schimmerling, R. Schindler, and H. Schwichtenberg.

**URL**: http://kryten.mm.rpi.edu/ct_bringsjord_arkoudas_final.pdf

Selmer Bringsjord, Konstantine Arkoudas*

# On the Provability, Veracity, and AI-Relevance of the Church-Turing Thesis

After providing some brief background on the Church-Turing thesis, we discuss MENDELSON's much-publicized change of heart regarding the provability of that thesis, and defend the standard conception of the thesis as a mathematically unprovable proposition. Next, the first author (BRINGSJORD) offers an argument that aims to establish the outright falsity of the thesis. The argument is controverted by the second author (ARKOUDAS), who puts forth several objections against it. BRINGSJORD replies to these objections, and to a number of other potential objections, and proceeds to consider previous attacks on the Church-Turing thesis and compare them with his own. The final section of the paper is devoted to an examination of arguments for the computational conception of mind from the Church-Turing thesis. We distinguish between strong and weak forms of computationalism, and we analyze an argument for weak computationalism on the basis of the Church-Turing thesis, which we show, *contra* COPELAND, to be formally valid. Accordingly, *if* the thesis is true, then, by virtue of this argument, weak computationalism is validated.

*S. BRINGSJORD, K. ARKOUDAS, Department of Cognitive Science, Department of Computer Science, Rensselaer AI & Reasoning (RAIR) Lab, Rensselaer Polytechnic Institute, Troy NY 12180 USA, selmer@rpi.edu, arkouk@rpi.edu, <http://www.rpi.edu/~brings>.



e objections to BRINGSJORD's argument, , and Section 4.3. BRINGSJORD replies to ceeds to consider some additional possible RINGSJORD discusses previous attacks on

on that CT is false first arose in connection *ctive* sets, which have two properties:

ndecidable (=no program, Turing machine, sets).

le function $f$ from the set of all standard h set, a function which, when given a canfor deciding the set in question), yields an r which $P$ will fail.

productive iff it's not only classically uny program proposed to decide $A$ can be me element of $A$. Clearly, if a set $A'$ has $\notin \Sigma_0$ and $A' \notin \Sigma_1$. If $A'$ falls somewhere decidable, then CTT falls. But what could NGSJORD has become convinced that the ries provides a perfect fit.

s you a bit off guard. Interesting stories? ou that the view that there are productive om unprecedented. Douglas HOFSTADTER s that the set $\mathcal{A}$ of all As is a productive 1, $\mathcal{A}$ must forever resist attempts to write is set; in order to satisfy P2, there must at to "stump" a program intended to decide hese conditions isn't all that implausible—up to the unpredictable variability seen in ke a look at Figure 1, taken from *Graphic Manual* [1981].

success in the present section doesn't render the answer is simply that, together, we have both endeavored to take MENDELSON seriously: To grapple directly with the provability issue, independent of other arguments.



e 1: Various Letter As

n to decide $\mathcal{A}$, it must capitalize on some sence" of the letter in question. But what be? Does the bar in the middle need to tly not (see 2 A). Does there have to be connecting the sides? Apparently not (see es for other proposed rules.[4]

onceded that no *argument* for the produced by HOFSTADTER. For all we know, rrow announce a letter recognition system The situation is a bit different in the case er KUGEL [1986], who makes clever use of unmistakably *arguing* that the set of all d above $\Sigma_1$ in AH:

> recognize, as beautiful, pieces of music nly could not have composed. There is partially computable sets that says that cedure for turning a procedure for recuch sets into a procedure for generating edure is uniform—you can use the same

TER's LETTER SPIRIT program, which generates s in the font in question. For an argument that t really creative, see [BRINGSJORD, FERRUCCI and

How are things to be improved? How is one to go about building an agent capable of creating interesting stories? It was the sustained attempt to answer this question, in conjunction with the concept of productivity discussed above, that persuaded BRINGSJORD that CT is indeed false. Let us explain.

First, to ease exposition, let $\mathcal{S}^I$ denote the set of all interesting stories. Now, recall that productive sets must have two properties, P1 and P2; let's take them in turn, in connection with $\mathcal{S}^I$. First, $\mathcal{S}^I$ must be classically undecidable; i.e., there is no program (or TM, etc.) which answers the question, for an arbitrary story in $\mathcal{S}^I$, whether or not it's interesting. Second, there must be some computable function $f$ from the set of all programs to $\mathcal{S}^I$ which, when given as input a program $P$ that purportedly decides $\mathcal{S}^I$, yields an element of $\mathcal{S}^I$ for which $P$ fails. It seems to us that $\mathcal{S}^I$ does have both of these properties—because, in a nutshell, Bringsjord and colleagues seemed to invariably and continuously turn up these two properties "in action." Every time someone suggested an algorithm-sketch for deciding $\mathcal{S}^I$, it was easily shot down by a counter-example consisting of a certain story which is clearly interesting despite the absence in it of those conditions regarded by the proposal to be necessary for interestingness. (It has been suggested that interesting stories must have inter-character conflict, but monodramas can involve only one character. It has been suggested that interesting stories must embody age-old plot structures, but some interesting stories are interesting precisely because they violate such structures, and so on.)

The situation we have arrived at can be crystallized in deductive form as follows.[8]

**Arg$_3$**

| | | | |
|---|---|---|---|
| (9) | If $\mathcal{S}^I \in \Sigma_1$ (or $\mathcal{S}^I \in \Sigma_0$), then there exists a procedure $P$ which adapts programs for deciding members of $\mathcal{S}^I$ so as to yield programs for enumerating members of $\mathcal{S}^I$. | | |
| (10) | There's no procedure $P$ which adapts programs for deciding members of $\mathcal{S}^I$ so as to yield programs for enumerating members of $\mathcal{S}^I$. | | |
| ∴ (11) | $\mathcal{S}^I \notin \Sigma_1$ (or $\mathcal{S}^I \notin \Sigma_0$). | 10, 11 | |
| (12) | $\mathcal{S}^I \in$ AH. | | |
| ∴ (13) | $\mathcal{S}^I \in \Pi_1$ (or above in the AH). | disj syll | |
| (14) | $\mathcal{S}^I$ is effectively decidable. | | |
| ∴ (15) | CT is false. | *reductio* | |

[8]Please note that the labeling in this argument is intentional. This argument is one Bringsjord is defending anew in the present chapter, and desires to preserve it precisely as it has been previously articulated [see BRINGSJORD and ZENZEN 2003]. The argument is now followed by new objections from ARKOUDAS, given in section 4.



Clearly, Arg$_3$ is formally valid. Premise (9) is not only true, but necessarily true, since it's part of the canon of elementary computability theory. What about premise (10)? Well, this is the core idea, the one expressed above by KUGEL, but transferred now to a different domain: People who can *decide* $\mathcal{S}^I$, that is, people who can decide whether something is an interesting story, can't necessarily *generate* interesting stories. Student researchers in BRINGSJORD's laboratory have been a case in point: with little knowledge of, and skill for, creating interesting stories, they have nonetheless recognized such narrative. That is, students who are, by their own admission, egregious creative writers, are nonetheless discriminating critics. They can decide which stories are interesting (which is why they know that the story generators AI has produced so far are nothing to write home about), but *producing* the set of all such stories (including, as it does, such works as not only *King Lear*, but *War and Peace*) is quite another matter. These would be, necessarily, the *same* matter if the set of all interesting stories, $\mathcal{S}^I$, was in either $\Sigma_0$ or $\Sigma_1$, the algorithmic portion of AH.

But what's the rationale behind (14), the claim that $\mathcal{S}^I$ is effectively decidable? The rationale is simply the brute fact that a normal, well-adjusted human computist can effectively decide $\mathcal{S}^I$. Try it yourself: First, start with the sort of story commonly discussed in AI; for example:

"Shopping"

> Jack was shopping at the supermarket. He picked up some milk from the shelf. He paid for it and left.[9]

[9]From page 502 of [CHARNIAK and MCDERMOTT 1985]. The story is studied in the context of attempts to resolve pronouns: How do we know who the first

# Refutation of Smith's "Proof" of CT

Smith, P. (2007) "Proving the Thesis?" in his *An Introduction to Gödel's Theorems* (Cambridge, UK: Cambridge University Press).

Bringsjord, S. & Govindarajulu, N.S. (2011) "In Defense of the Unprovability of the Church-Turing Thesis" *International Journal of Unconventional Computing* **6**: 353–374.

**URL**: http://kryten.mm.rpi.edu/SB_NSG_CTTnotprovable_091510.pdf

represented as two different cells joined by a single arrow. Each atom appearing in the argument is represented by a unidirectional chain of circles. The number of circles in the chain for an atom is equal to the number of literals the atom appears in. Each literal then connects to exactly one circle in the corresponding atom's chain of circles (in Figure 2 only the links for literals $A$ and $\neg A$ are shown). The circles with bisecting lines and grids in them serve as convenient "end-markers." The asterisk appears next to the current focal node. The important point to note is that there exists a KU machine which solves this problem. The exact formulation of such a machine is left as an exercise for the reader.*

### 5.2 The Argument Itself

Smith's argument directly matches the structure of a squeezing argument as we presented it in section 4. With obvious symbolization, this means that the squeeze is accomplished when the following chain† is established:

$$Tcomputable(f) \rightarrow Effcomputable(f) \rightarrow KUcomputable(f) \rightarrow Tcomputable(f).$$

And as you will be able to infer, the chain is in turn established when the relevant trio of conditionals are established, to wit:

1*. If $Tcomputable(f)$ then $Effcomputable(f)$

2*. If $Effcomputable(f)$ then $KUcomputable(f)$

3*. If $KUcomputable(f)$ then $Tcomputable(f)$

Obviously, $\{1^*., 2^*., 3^*.\} \vdash$ CTT.‡ Conditional 3*. is provable. Conditional 1*. seems to us to be provable as well. (Remember that 1*. is a counterpart to the "easy" half of CTT, and Bringsjord has long ago conceded to Mendelson that

* See http://kryten.mm.rpi.edu/CTPROV/KU-TruthtreeAlgorithm.pdf for one such formulation.

† Smith prefers to start the chain with reference to $\mu$-recursiveness rather than Turing-computability, but this needlessly complicates things. Besides, Smith immediately appeals to Turing machines in order to justify the first conditional: see section 35.5, p. 332 in [28].

‡ The second of these conditionals, in keeping with our analysis of the counterpart conditionals in the above-visited squeezing argument from Kreisel, might be said to be derived from its contrapositive, but Smith, as we shall soon see when looking at what he says about 2*., makes no such move.



the easy half is indeed provable.) It thus follows that if Smith's argument fails, the source of the trouble must be 2*. In light of this situation, does Smith himself defend 2*.? Yes, he does. His defense is based, first, on the observation that if 2*. is to be overthrown, that must happen because its antecedent holds, while its consequent doesn't. The second and final part of the defense is to argue that such a counter-example would need to be a case wherein an agent processes an algorithm effectively, but where there is some violation of the KU-machine format — and to then show that no such case is possible.

In support of this exegesis, we read:

> The KU specification involves a conjunction of requirements (finite alphabet, logically navigable workspace, etc.). So for a proposed algorithmic procedure to fail to be covered, it must falsify one of the conjuncts. But how? By having and using an infinite number of primitive symbols? Then it isn't usable by a limited computing agent like us (and we are trying to characterize the idea of an algorithmic procedure of the general type that agents like us could at least in principle deploy). By making use of a different sort of dataspace? But the KU specification only requires that the space has *some* structure which enables the data to be locally navigable by a limited agent. By not keeping the size of the active patch of dataspace bounded? But algorithms are supposed to proceed by the repetition of "small" operations which are readily surveyable by limited agents. By not keeping the jumps from one active patch of dataspace to the next active patch limited? But again, a limited agent couldn't then always jump to the next patch 'in one go' and still know where he was going. By the program that governs the updating of the dataspace having a different form? But KU algorithms are entirely freeform; there is no more generality to be had. [28, p. 337]

These two points, then, should be entirely uncontroversial: If in fact there are cases where agents "like us" process algorithms in ways that violate the "KU conjunction," and these cases are found, Smith's case for CTT is refuted. And if *for all we know* such cases are theoretical possibilities, Smith's case for CTT is inconclusive. We proceed to give three objections, each of which, alone, implies the latter; and which, combined, seem to us to undeniably reveal Smith's case to be a very weak one.*

## 6 THREE OBJECTIONS: SMITH'S ARGUMENT IS INCONCLUSIVE

represented as two different cells joined by a single arrow. Each atom appearing in the argument is represented by a unidirectional chain of circles. The number of circles in the chain for an atom is equal to the number of literals the atom appears in. Each literal then connects to exactly one circle in the corresponding atom's chain of circles (in Figure 2 only the links for literals $A$ and $\neg A$ are shown). The circles with bisecting lines and grids in them serve as convenient "end-markers." The asterisk appears next to the current focal node. The important point to note is that there exists a KU machine which solves this problem. The exact formulation of such a machine is left as an exercise for the reader.*

### 5.2 The Argument Itself

Smith's argument directly matches the structure of a squeezing argument as we presented it in section 4. With obvious symbolization, this means that the squeeze is accomplished when the following chain† is established:

$$Tcomputable(f) \rightarrow Effcomputable(f) \rightarrow KUcomputable(f) \rightarrow Tcomputable(f).$$

And as you will be able to infer, the chain is in turn established when the relevant trio of conditionals are established, to wit:

1*. If $Tcomputable(f)$ then $Effcomputable(f)$

2*. If $Effcomputable(f)$ then $KUcomputable(f)$

3*. If $KUcomputable(f)$ then $Tcomputable(f)$

Obviously, $\{1^*., 2^*., 3^*.\} \vdash$ CTT.‡ Conditional 3*. is provable. Conditional 1*. seems to us to be provable as well. (Remember that 1*. is a counterpart to the "easy" half of CTT, and Bringsjord has long ago conceded to Mendelson that

* See http://kryten.mm.rpi.edu/CTPROV/KU-TruthtreeAlgorithm.pdf for one such formulation.

† Smith prefers to start the chain with reference to $\mu$-recursiveness rather than Turing-computability, but this needlessly complicates things. Besides, Smith immediately appeals to Turing machines in order to justify the first conditional: see section 35.5, p. 332 in [28].

‡ The second of these conditionals, in keeping with our analysis of the counterpart conditionals in the above-visited squeezing argument from Kreisel, might be said to be derived from its contrapositive, but Smith, as we shall soon see when looking at what he says about 2*., makes no such move.

represented as two different cells joined by a single arrow. Each atom appearing in the argument is represented by a unidirectional chain of circles. The number of circles in the chain for an atom is equal to the number of literals the atom appears in. Each literal then connects to exactly one circle in the corresponding atom's chain of circles (in Figure 2 only the links for literals $A$ and $\neg A$ are shown). The circles with bisecting lines and grids in them serve as convenient "end-markers." The asterisk appears next to the current focal node. The important point to note is that there exists a KU machine which solves this problem. The exact formulation of such a machine is left as an exercise for the reader.*

### 5.2 The Argument Itself

Smith's argument directly matches the structure of a squeezing argument as we presented it in section 4. With obvious symbolization, this means that the squeeze is accomplished when the following chain† is established:

$$Tcomputable(f) \rightarrow Effcomputable(f) \rightarrow KUcomputable(f) \rightarrow Tcomputable(f).$$

And as you will be able to infer, the chain is in turn established when the relevant trio of conditionals are established, to wit:

1*. If $Tcomputable(f)$ then $Effcomputable(f)$

2*. If $Effcomputable(f)$ then $KUcomputable(f)$

3*. If $KUcomputable(f)$ then $Tcomputable(f)$

Obviously, $\{1^*., 2^*., 3^*.\} \vdash$ CTT.‡ Conditional 3*. is provable. Conditional 1*. seems to us to be provable as well. (Remember that 1*. is a counterpart to the "easy" half of CTT, and Bringsjord has long ago conceded to Mendelson that

---

* See http://kryten.mm.rpi.edu/CTTRSWKU-TruthtreeAlgorithm.pdf for one such formulation.

† Smith prefers to start the chain with reference to $\mu$-recursiveness rather than Turing-computability, but this needlessly complicates things. Besides, Smith immediately appeals to Turing machines in order to justify the first conditional: see section 35.5, p. 332 in [28].

‡ The second of these conditionals, in keeping with our analysis of the counterpart conditionals in the above-visited squeezing argument from Kreisel, might be said to be derived from its contrapositive, but Smith, as we shall soon see when looking at what he says about 2*., makes no such move.

# Current CT Project: Evaluating* ...

# Current CT Project: Evaluating* ...

Dershowitz, N. & Gurevich, Y. (2008) "A Natural Axiomatization of Computability and Proof of Church's Thesis" *The Bulletin of Symbolic Logic* **14.3**: 299–350.

# Current CT Project: Evaluating* ...

Dershowitz, N. & Gurevich, Y. (2008) "A Natural Axiomatization of Computability and Proof of Church's Thesis" *The Bulletin of Symbolic Logic* **14.3**: 299–350.

**Abstract**. Church's Thesis asserts that the only numeric functions that can be calculated by effective means are the recursive ones, which are the same, extensionally, as the Turing-computable numeric functions. The Abstract State Machine Theorem states that every classical algorithm is behaviorally equivalent to an abstract state machine. This theorem presupposes three natural postulates about algorithmic computation. Here, we show that augmenting those postulates with an additional requirement regarding basic operations gives a natural axiomatization of computability and proof of Church's Thesis, as Gödel and others suggested may be possible. In a similar way, but with a different set of basic operations, one can prove Turing's Thesis, characterizing the effective string functions, and—in particular—the effectively-computable functions on string representations of numbers.

# Current CT Project: Evaluating* ...

Dershowitz, N. & Gurevich, Y. (2008) "A Natural Axiomatization of Computability and Proof of Church's Thesis" *The Bulletin of Symbolic Logic* **14.3**: 299–350.

**Abstract**. Church's Thesis asserts that the only numeric functions that can be calculated by effective means are the recursive ones, which are the same, extensionally, as the Turing-computable numeric functions. The Abstract State Machine Theorem states that every classical algorithm is behaviorally equivalent to an abstract state machine. This theorem presupposes three natural postulates about algorithmic computation. Here, we show that augmenting those postulates with an additional requirement regarding basic operations gives a natural axiomatization of computability and proof of Church's Thesis, as Gödel and others suggested may be possible. In a similar way, but with a different set of basic operations, one can prove Turing's Thesis, characterizing the effective string functions, and—in particular—the effectively-computable functions on string representations of numbers.

*We are indebted to D&G for a most stimulating and scholarly paper.

# What is CT in the Paper?

# What is CT in the Paper?

Page 303:

# What is CT in the Paper?

**[Church's] Thesis I**. Every effectively calculable function (effectively decidable predicate) is general recursive.

Page 303:

# What is CT in the Paper?

Page 303:

**[Church's] Thesis I**. Every effectively calculable function (effectively decidable predicate) is general recursive.

**[Church's] Thesis I**†. Every partial function which is effectively calculable (in the sense that there is an algorithm by which its value can be calculated for every n-tuple belonging to its range of definition) is ... partial recursive.

# What is CT in the Paper?

Page 303:

**[Church's] Thesis I**. Every effectively calculable function (effectively decidable predicate) is general recursive.

**[Church's] Thesis I†**. Every partial function which is effectively calculable (in the sense that there is an algorithm by which its value can be calculated for every n-tuple belonging to its range of definition) is ... partial recursive.

Label them, resp., as $CT_{p303}$ and $CT^{\dagger}_{p303}$.

# Will Ignore Scholarship Issues, and (for now) the Missing Distinction B/t "Easy" and "Hard" Conditionals

# Will Ignore Scholarship Issues, and (for now) the Missing Distinction B/t "Easy" and "Hard" Conditionals

Smith, P. (2007) "The Church-Turing Thesis" in his *An Introduction to Gödel's Theorems* (Cambridge, UK: Cambridge University Press); and Mendelson; and ... :

# Will Ignore Scholarship Issues, and (for now) the Missing Distinction B/t "Easy" and "Hard" Conditionals

Smith, P. (2007) "The Church-Turing Thesis" in his *An Introduction to Gödel's Theorems* (Cambridge, UK: Cambridge University Press); and Mendelson; and ... :

**TT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is computable by a Turing machine.

# Will Ignore Scholarship Issues, and (for now) the Missing Distinction B/t "Easy" and "Hard" Conditionals

Smith, P. (2007) "The Church-Turing Thesis" in his *An Introduction to Gödel's Theorems* (Cambridge, UK: Cambridge University Press); and Mendelson; and ... :

**TT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is computable by a Turing machine.

**CT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is $\mu$-recursive.

# Will Ignore Scholarship Issues, and (for now) the Missing Distinction B/t "Easy" and "Hard" Conditionals

Smith, P. (2007) "The Church-Turing Thesis" in his *An Introduction to Gödel's Theorems* (Cambridge, UK: Cambridge University Press); and Mendelson; and ... :

**TT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is computable by a Turing machine.

**CT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is $\mu$-recursive.

**CTT**: The effectively computable total numerical functions are the $\mu$-recursive/Turing computable functions.

# Will Ignore Scholarship Issues, and (for now) the Missing Distinction B/t "Easy" and "Hard" Conditionals

Smith, P. (2007) "The Church-Turing Thesis" in his *An Introduction to Gödel's Theorems* (Cambridge, UK: Cambridge University Press); and Mendelson; and ... :

**TT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is computable by a Turing machine.

**CT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is $\mu$-recursive.

**CTT**: The effectively computable total numerical functions are the $\mu$-recursive/Turing computable functions.

# Will Ignore Scholarship Issues, and (for now) the Missing Distinction B/t "Easy" and "Hard" Conditionals

Smith, P. (2007) "The Church-Turing Thesis" in his *An Introduction to Gödel's Theorems* (Cambridge, UK: Cambridge University Press); and Mendelson; and ... :

**TT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is computable by a Turing machine.

**CT**: A numerical (total) function is effectively computable by some algorithmic routine if and only if (= iff) it is $\mu$-recursive.

**CTT**: The effectively computable total numerical functions are the $\mu$-recursive/Turing computable functions.

And, for a function *f* to be effectively computable, is for a human agent/ computor/calculator/... to follow an algorithm in order to compute ... *f*.

# When is success achieved?

# When is success achieved?

Page 326:

# When is success achieved?

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

# When is success achieved?

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327:

# When is success achieved?

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #1

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #1

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #1

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

But:

"Church's Thesis" here
≠
$CT_{p303}$

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #1

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

But:

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

"Church's Thesis" here ≠ $CT_{p303}$

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

And:

The above corollary, rephrased, is precisely what we have set out to establish, namely:

"Church's Thesis" here ≠ $CT^{\dagger}_{p303}$

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #2

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #2

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #2

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

?

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #2

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

Do D&G mean to say 'function'? Perhaps. After all, that's the category invoked in the antecedent of Corollary 4.6.

?

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #2

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

Do D&G mean to say 'function'? Perhaps. After all, that's the category invoked in the antecedent of Corollary 4.6.

?

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

But what does Theorem 3.4 *say*?

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# We read:

# We read:

Page 322:

"THEOREM 3.4 (ASM Theorem [42]). For every process satisfying the Sequential Postulates, there is an abstract state machine in the same vocabulary (and with the same sets of states and initial states) that emulates it."

# We read:

Page 322:

"THEOREM 3.4 (ASM Theorem [42]). For every process satisfying the Sequential Postulates, there is an abstract state machine in the same vocabulary (and with the same sets of states and initial states) that emulates it."

# We read:

Page 322:

"THEOREM 3.4 (ASM Theorem [42]). For every process ? satisfying the Sequential Postulates, there is an abstract state machine in the same vocabulary (and with the same sets of states and initial states) that emulates it."

# We read:

Page 322:

"THEOREM 3.4 (ASM Theorem [42]). For every process ? satisfying the Sequential Postulates, there is an abstract state machine in the same vocabulary (and with the same sets of states and initial states) that emulates it."

Hence, as it stands, the reasoning is simply invalid.

# Problem #3

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #3

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# Problem #3

Page 326: "THEOREM 4.5. A numeric function is partial recursive if and only if it is computable by an arithmetical ASM."

Page 327: "It follows from Theorem 4.5 that:

COROLLARY 4.6. Every numeric function computed by a state-transition system satisfying the Sequential Postulates, and provided initially with only basic arithmetic, is partial recursive.

PROOF. By the ASM Theorem (Theorem 3.4), every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By Theorem 4.5, such an ASM computes a partial recursive function.

?! DEFINITION 4.7 (Arithmetical algorithm) A state-transition system satisfying the Sequential and Arithmetical Postulates is called an *arithmetical algorithm*.

The above corollary, rephrased, is precisely what we have set out to establish, namely:

THEOREM 4.8 (Church's Thesis) Every numeric (partial) function computed by an arithmetical algorithm is (partial) recursive."

# But:

# But:

The objective isn't to simply *stipulate* that CT (or CTT, etc.) is true!

# But:

The objective isn't to simply *stipulate* that CT (or CTT, etc.) is true!

> The challenge is to *show*, at a minimum by sound argument, and at a maximum by outright proof, that—to go with Sieg's felicitous term—a compu*tor* following an algorithm to compute *f* implies (given the D&G context) that a **PI–PIV** state-transition system can compute *f*.

# Problem #4

# Problem #4

If one replies that only this *type* of algorithm (viz., arithmetical) is being stipulatively defined, it makes no difference, because the problem infects the four postulates D&G present:

# Problem #4

If one replies that only this *type* of algorithm (viz., arithmetical) is being stipulatively defined, it makes no difference, because the problem infects the four postulates D&G present:

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*

- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*

- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed "glossary" of "critical" terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*

- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

# Problem #4

If one replies that only this *type* of algorithm (viz., arithmetical) is being stipulatively defined, it makes no difference, because the problem infects the four postulates D&G present:

“

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*

- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*

- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed “glossary” of “critical” terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*

- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

”

# Problem #4

If one replies that only this *type* of algorithm (viz., arithmetical) is being stipulatively defined, it makes no difference, because the problem infects the four postulates D&G present:

“

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*

- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*

- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed “glossary” of “critical” terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*

- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

”

# Problem #4

If one replies that only this *type* of algorithm (viz., arithmetical) is being stipulatively defined, it makes no difference, because the problem infects the four postulates D&G present:

This needs to be proved, not merely asserted.*

“

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*

- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*

- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed “glossary” of “critical” terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*

- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

”

# Problem #4

If one replies that only this *type* of algorithm (viz., arithmetical) is being stipulatively defined, it makes no difference, because the problem infects the four postulates D&G present:

This needs to be proved, not merely asserted.*

“

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*

- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*

- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed “glossary” of “critical” terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*

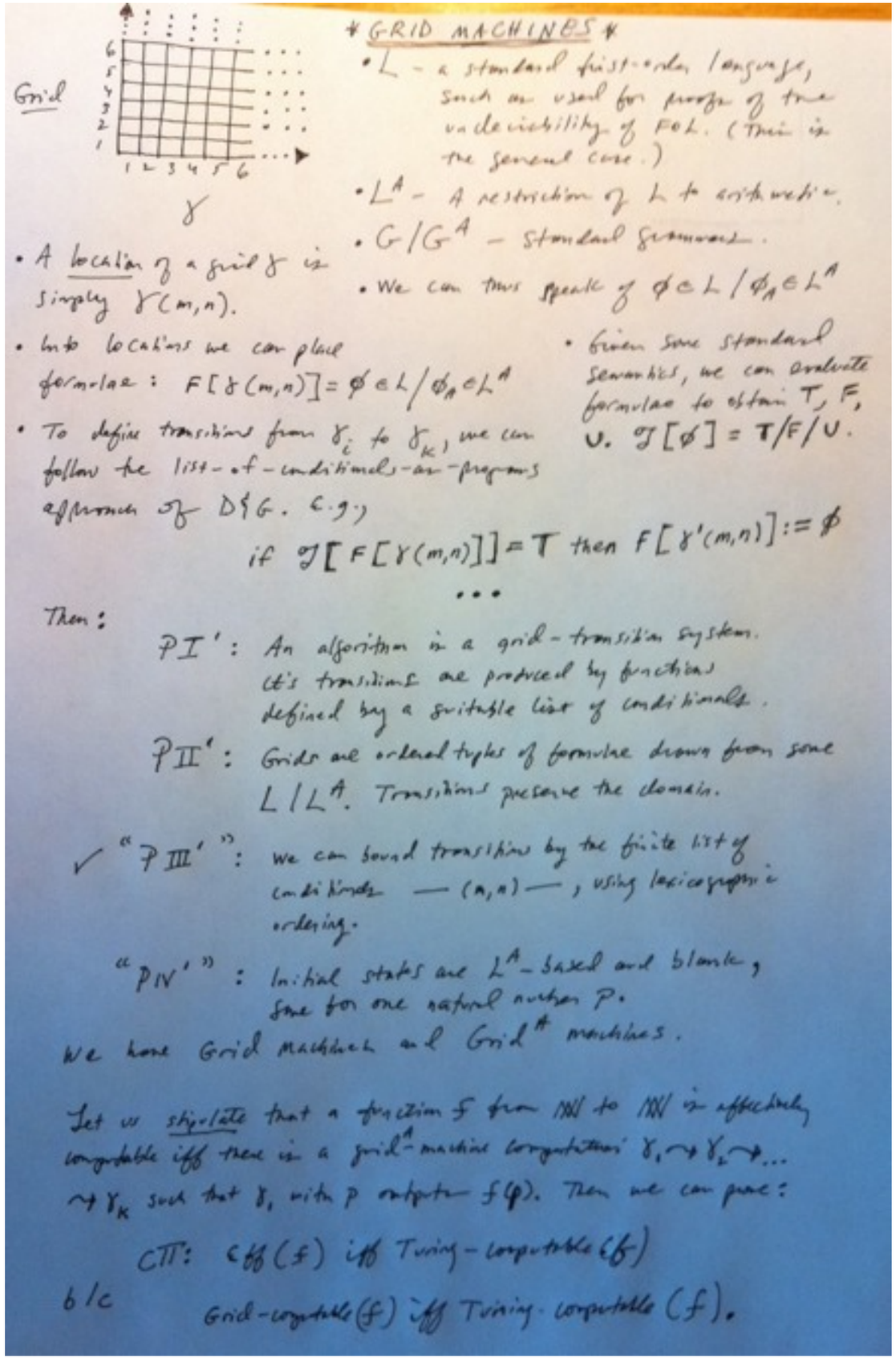
- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

”

*Demanding a sound argument or proof for *P* shouldn’t be conflated with the position that *P* is false. That said, since an algorithm *A* can be fully tokenized by a natural-language description *D*, and the assertion here (when cast in FOL) would imply the massively counter-intuitive position that such a *D* is identical with a state-transition system, we believe the assertion to be provably false. For example, imagine that the merge sort algorithm is described in such a *D*. How is is that this text is *identical* to a state-transition system?

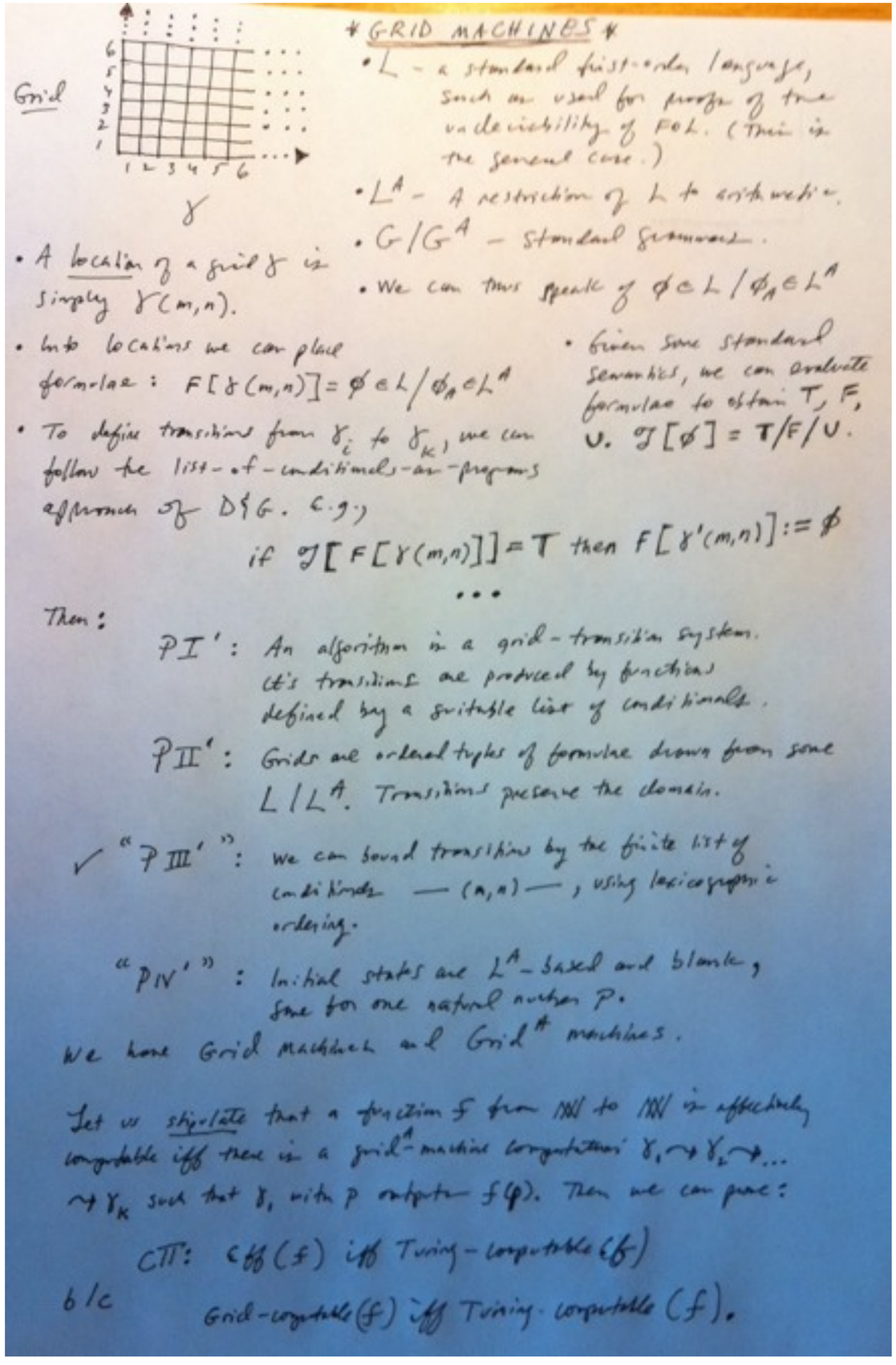
After all, anyone can stipulate
their way to “success” ...

After all, anyone can stipulate their way to "success" ...



# After all, anyone can stipulate their way to "success" ...

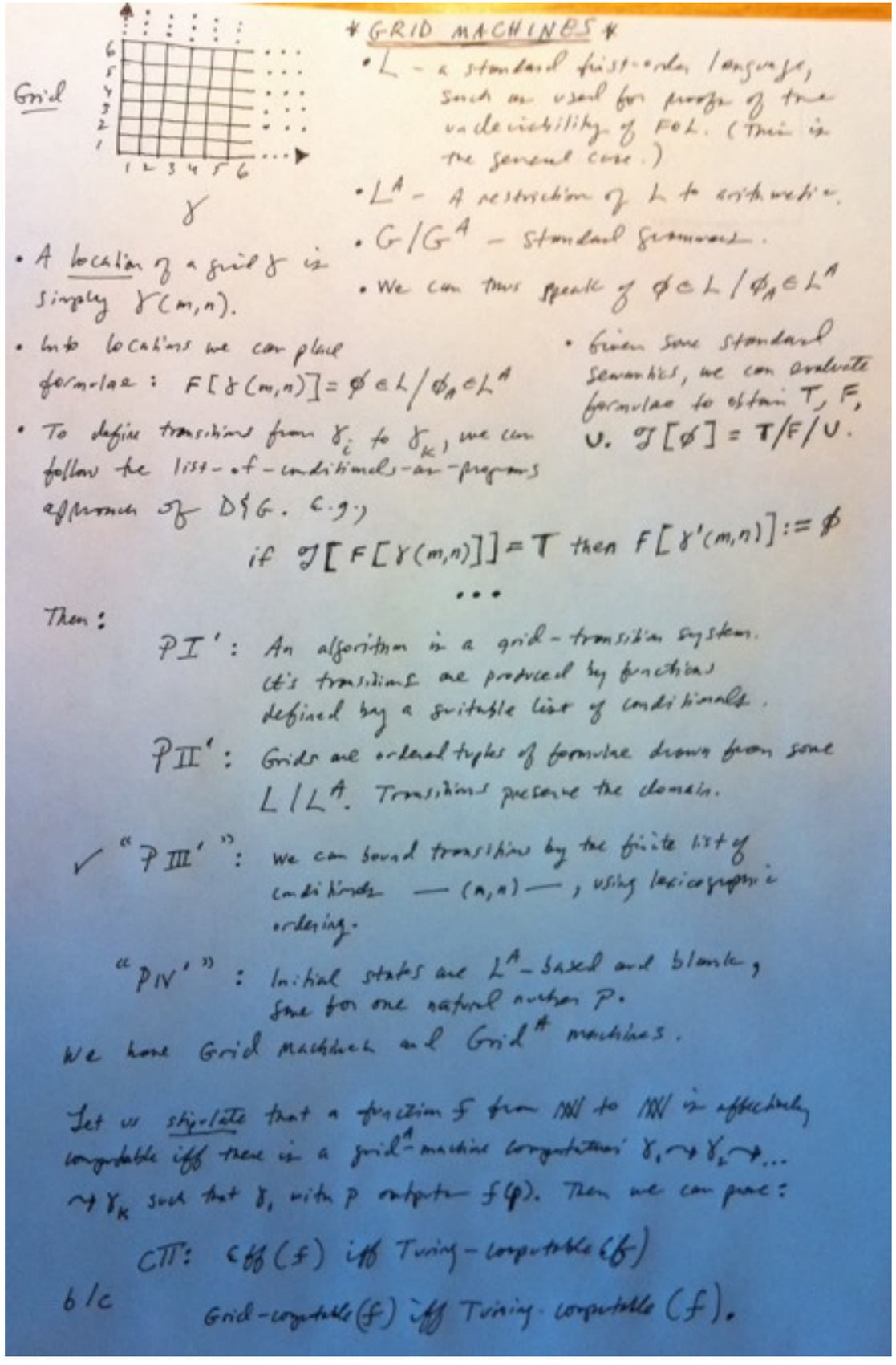
I define grid machines, whose cells carry predicate-calculus formulae interpretable as either **T**, **F**, or **U**.



# After all, anyone can stipulate their way to "success" ...

I define grid machines, whose cells carry predicate-calculus formulae interpretable as either **T**, **F**, or **U**.

Transitions are regimented by obeying a list of conditionals covering the next value of each cell.

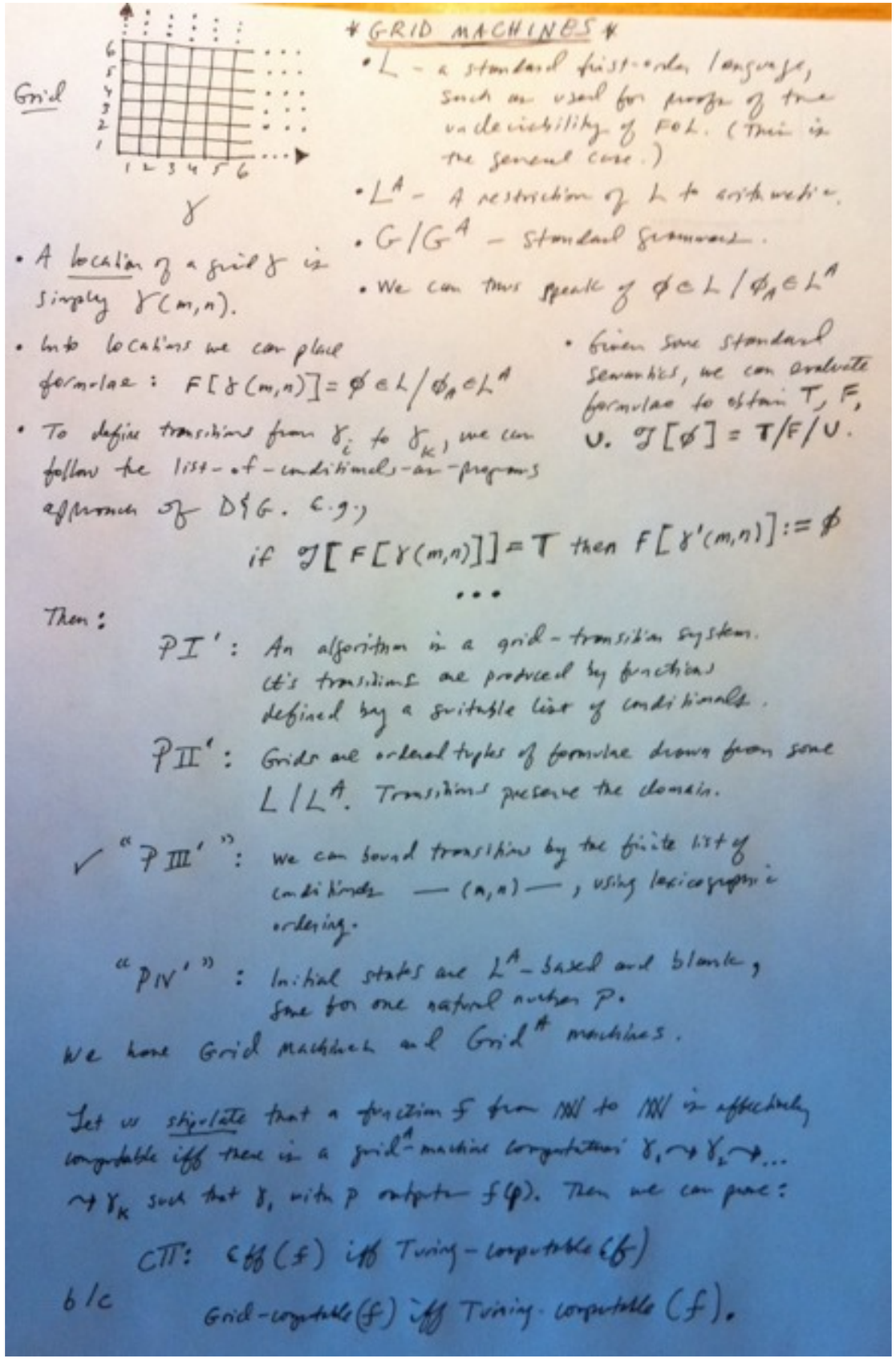


# After all, anyone can stipulate their way to "success" ...

I define grid machines, whose cells carry predicate-calculus formulae interpretable as either **T**, **F**, or **U**.

Transitions are regimented by obeying a list of conditionals covering the next value of each cell.

Four "postulates" are asserted, allowing me to prove that what is grid-computable is Turing-computable.



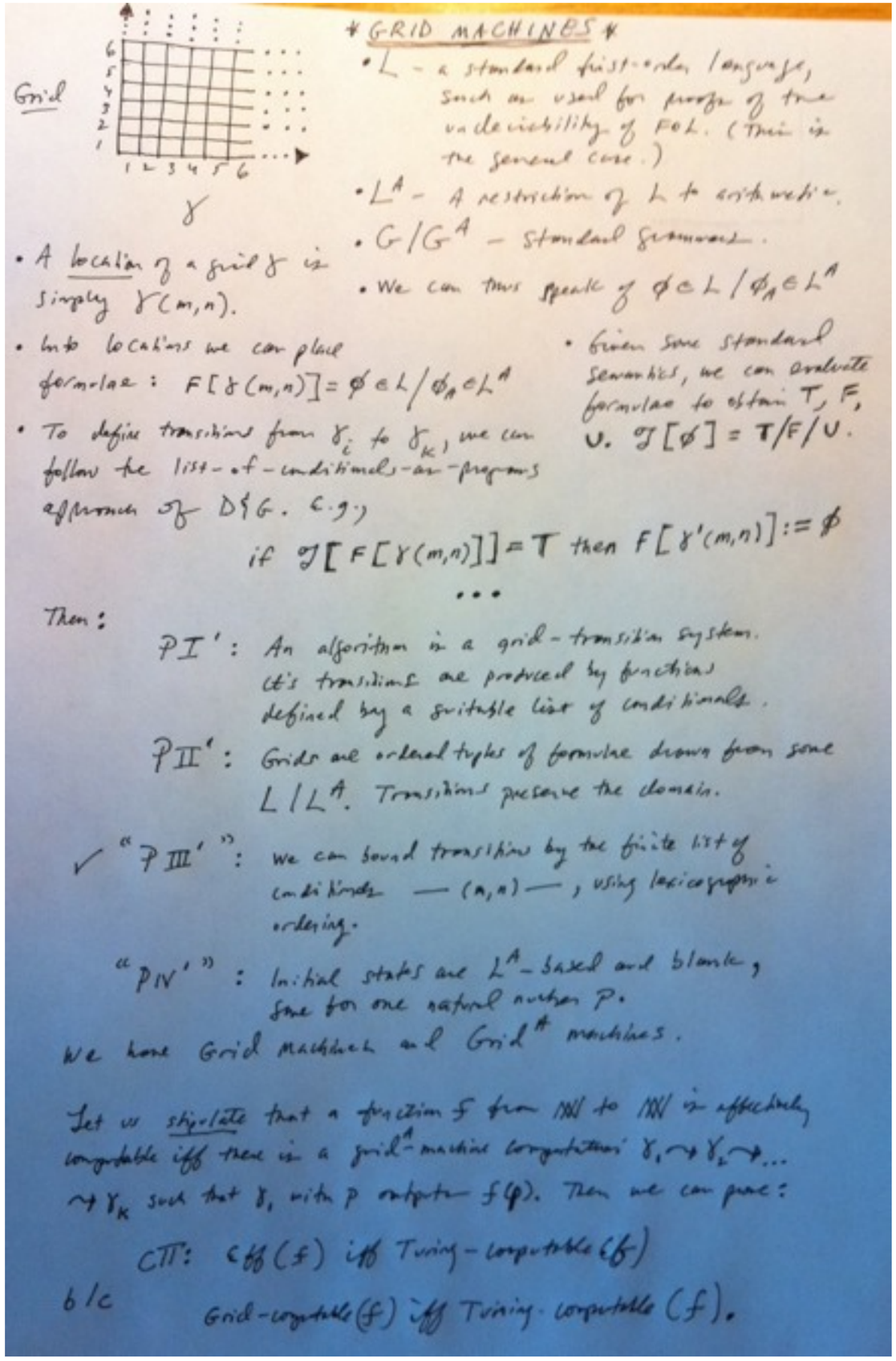
# After all, anyone can stipulate their way to "success" ...

I define grid machines, whose cells carry predicate-calculus formulae interpretable as either **T**, **F**, or **U**.

Transitions are regimented by obeying a list of conditionals covering the next value of each cell.

Four "postulates" are asserted, allowing me to prove that what is grid-computable is Turing-computable.

I stipulate that a function is efffectively computable iff there's a grid machine that computes it.



# After all, anyone can stipulate their way to "success" ...

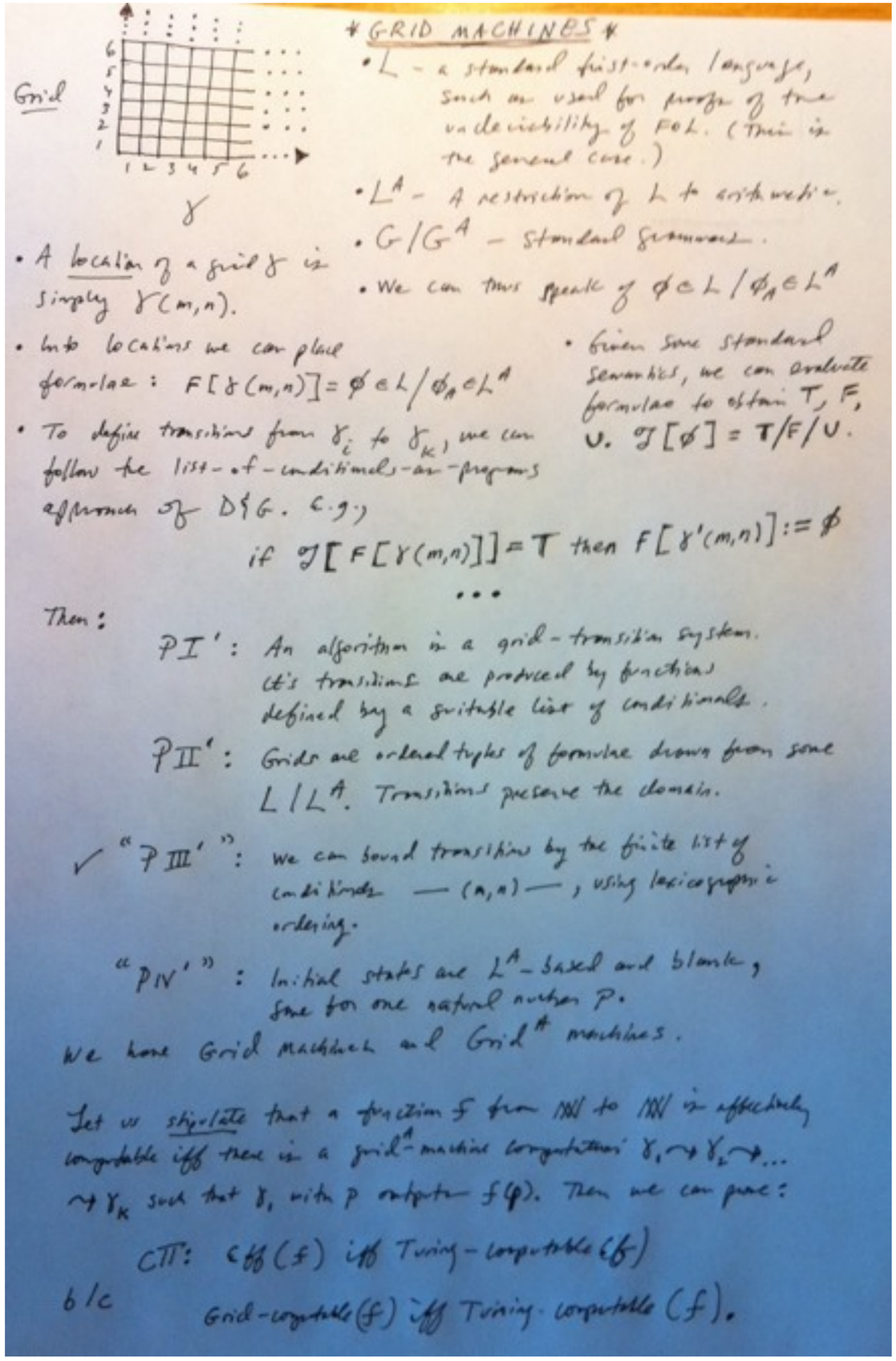
I define grid machines, whose cells carry predicate-calculus formulae interpretable as either **T**, **F**, or **U**.

Transitions are regimented by obeying a list of conditionals covering the next value of each cell.

Four "postulates" are asserted, allowing me to prove that what is grid-computable is Turing-computable.

I stipulate that a function is efffectively computable iff there's a grid machine that computes it.

CTT is "proved."



# After all, anyone can stipulate their way to "success" ...

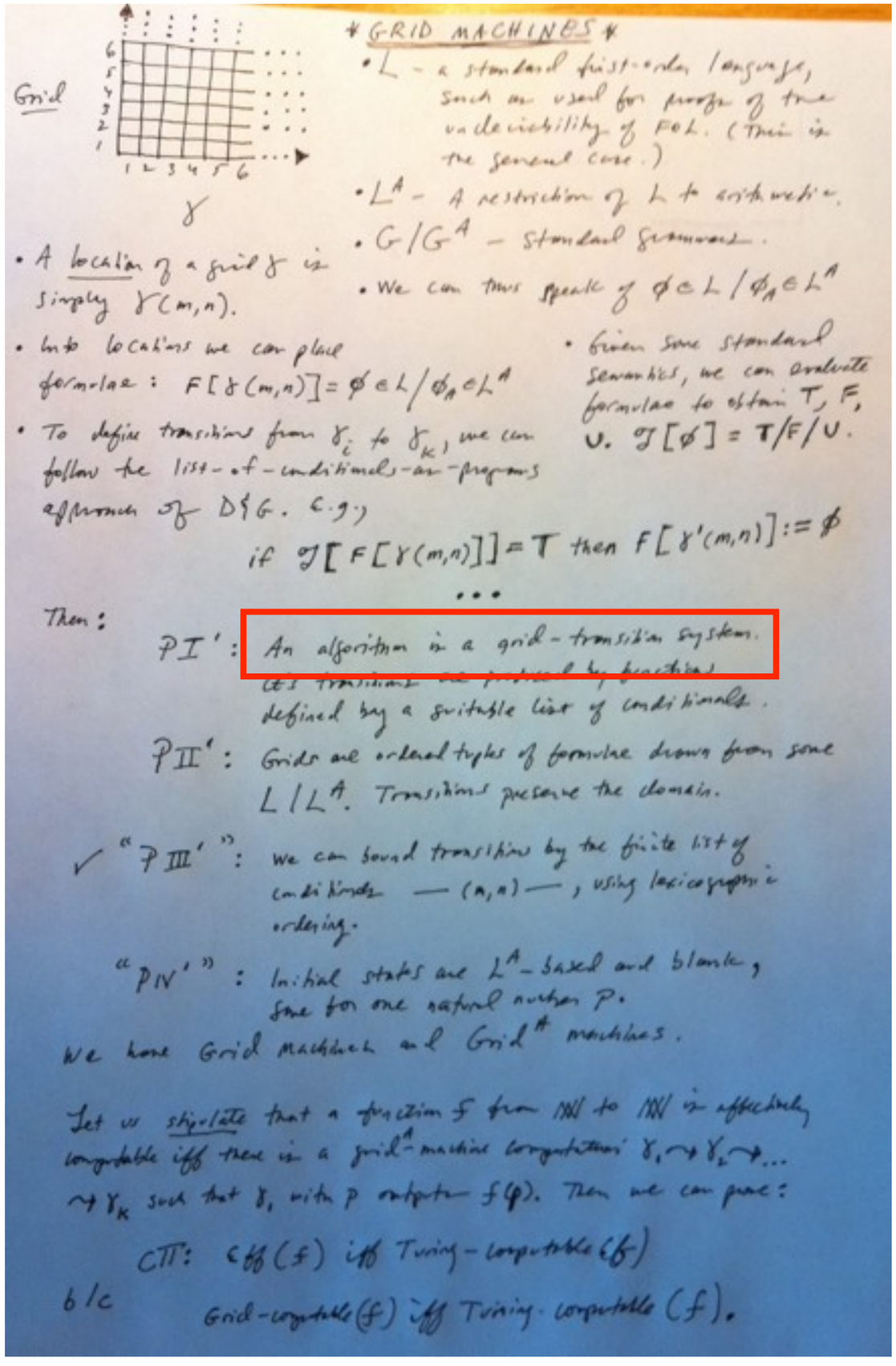
I define grid machines, whose cells carry predicate-calculus formulae interpretable as either **T**, **F**, or **U**.

Transitions are regimented by obeying a list of conditionals covering the next value of each cell.

Four "postulates" are asserted, allowing me to prove that what is grid-computable is Turing-computable.

I stipulate that a function is efffectively computable iff there's a grid machine that computes it.

CTT is "proved."



# Problem #5

Nowhere does the core concept of a computing agent/computor/calculating agent... appear in the postulates, so it's logically impossible that the standard form of CT/CTT can be proved.

"

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*

- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*

- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed "glossary" of "critical" terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*

- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

"

# Problem #6

D&G tell us that "structures" at most require a level of expressivity available via standard first-order languages.

# Problem #6

## D&G tell us that "structures" at most require a level of expressivity available via standard first-order languages.

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*

- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*

- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed "glossary" of "critical" terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*

- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

# Problem #6

D&G tell us that "structures" at most require a level of expressivity available via standard first-order languages.

"

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*
- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*
- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed "glossary" of "critical" terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*
- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

"

# Problem #6

D&G tell us that "structures" at most require a level of expressivity available via standard first-order languages.

"

- POSTULATE I (Sequential time). *An algorithm is a state-transition system. Its transitions are partial functions.*
- POSTULATE II (Abstract state). *States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.*
- POSTULATE III (Bounded exploration). *Transitions are determined by a fixed "glossary" of "critical" terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that the states that agree on the values of these glossary terms, also agree on all next-step state changes*
- POSTULATE IV (Arithmetical States). *Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values*

"

But in a world where (to use Post's famous phrase) the mathematizing power of *homo sapiens* includes making use of propositions that cannot be expressed in (finitary) FOL, at the very least, to have a proof, we *need* a proof—that effective computability excludes such cognition. Yet D&G don't supply it.

# Why not Palette$^{\infty}$ Machines?

**6.2 Objection #2: The Cognitive Possibility of the "Effective Infinitary"**

For those open-minded about the possibility of hypercomputing minds, it is interesting to explicitly consider frameworks in which the customary bounds that enforce the Turing Limit are dissolved. In the context of KU machines, it would specifically seem worth considering problem-solving frameworks in which cells contain formulas that in and of themselves apparently break these bounds. Smith, of course, is completely closed-minded with respect to such breakage, as confirmed by this text, which we visited above: "So for a proposed algorithmic procedure to fail to be covered [by KU-machines], it must falsify one of the conjuncts. But how? By having and using an infinite number of primitive symbols? Then it isn't usable by a limited computing agent like us ..." Clearly, Smith is of the view that a problem-solving system allowing an infinite number of primitive symbols is not appropriate when the agents involved are human persons. But is this view justified? Is it consistent with what human persons do when they problem-solve? Might it not be that limited problem-solving agents make use of techniques that are at once effective and infinitary? We are inclined to answer the first two questions in the negative, and the third in the affirmative.

For example, how might KU machines be modified to reflect steadfast open-mindedness regarding more expressive underlying languages? Well, consider the simple infinitary logic $\mathcal{L}_{\omega_1\omega}$, in which countably infinite conjunctions are allowed. Hence, cells in this case would be permitted to contain countably infinite expressions such as

$$\bigwedge \phi$$

and the equivalent

$$\phi_1 \wedge \phi_2 \wedge \ldots.$$

As is well-known, $\mathcal{L}_{\omega_1\omega}$ occupies a somewhat special place among infinitary logics, because inference in this system can be considered quite mechanistic. So for example we might have a cell like

$$\boxed{\bigvee \psi}$$

and the cell

$$\boxed{\neg\psi_{23}}$$

in the active patch at some time during the computation, and at the next moment move to

$$\boxed{\psi_1 \vee \psi_2 \vee \ldots \psi_{22} \vee \psi_{24} \vee \ldots}$$

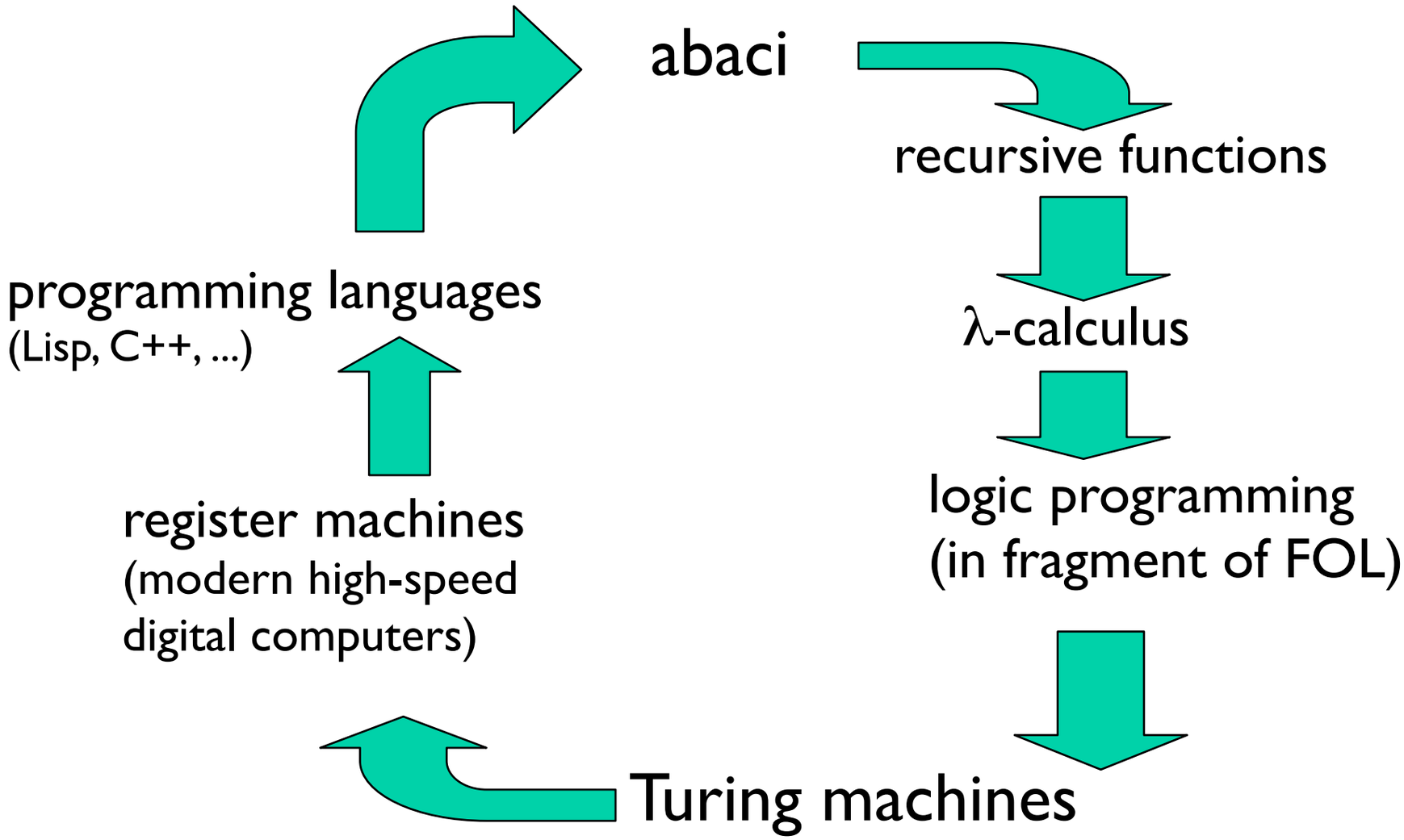
This would seem to be perfectly standard reasoning for limited agents like us, and hence appears to be a suitable target for what a KU-like framework should accommodate — and yet such reasoning is over infinitely long expressions.*

Comments made in the final paragraph of section 6.1 apply *mutatis mutandis* to what we have said regarding infinitary logic and human problem-solving. In short, to repeat with but a touch of new emphasis: KU-like machines "boosted" in direction of $\mathcal{L}_{\omega_1\omega}$ *appear* to offer a framework suggested by aspects of human problem-solving. We do not claim that *in fact* such boosting is at present known to be necessary. Once again, the burden is on Smith to show that such boosting is inappropriate.

What, then, have D&G accomplished, if anything?

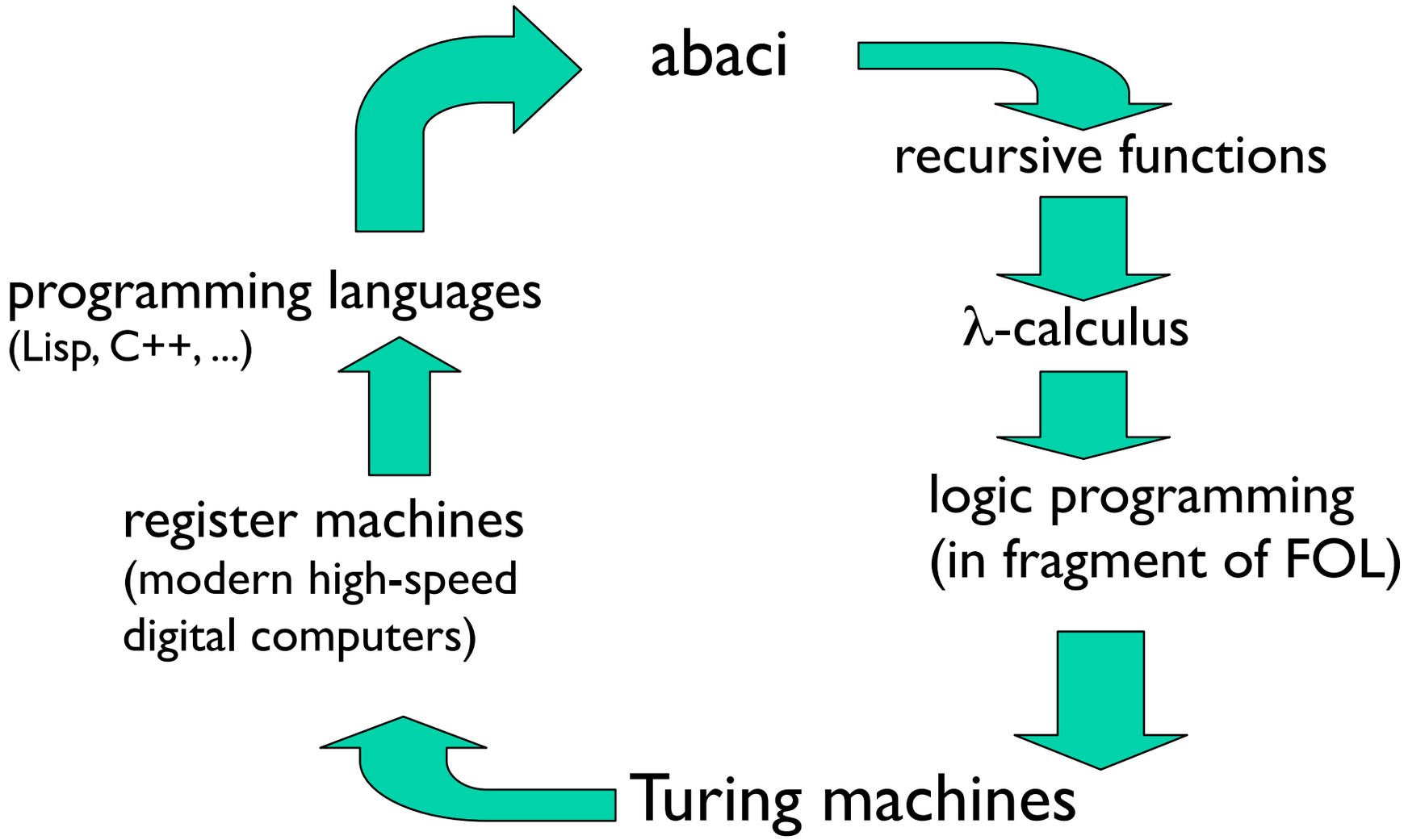
They have simply shown that yet another type of Turing-level device can be added to what Bringsjord has in another venue called the Humble "Circle" of Computation.

# "Humble" Circle of Computation



# "Humble" Circle of Computation

Insert ASMs anywhere you wish!



*finis*

# Kolmogorov-Uspenskii (KU) Machines ...

# KU Machines

# KU Machines

- KU machines operate on a set of states ***S***.

# KU Machines

- KU machines operate on a set of states ***S***.
- The states are directed graphs with color-coded edges.

# KU Machines

- KU machines operate on a set of states ***S***.
- The states are directed graphs with color-coded edges.
- The nodes act as data cells and have symbols from a finite alphabet written in them.

# KU Machines

- KU machines operate on a set of states ***S***.
- The states are directed graphs with color-coded edges.
- The nodes act as data cells and have symbols from a finite alphabet written in them.
- The two conditions below limit the number of neighbors for any node.

# KU Machines

- KU machines operate on a set of states ***S***.
- The states are directed graphs with color-coded edges.
- The nodes act as data cells and have symbols from a finite alphabet written in them.
- The two conditions below limit the number of neighbors for any node.

    1. The number of edge colors is fixed.

# KU Machines

- KU machines operate on a set of states ***S***.
- The states are directed graphs with color-coded edges.
- The nodes act as data cells and have symbols from a finite alphabet written in them.
- The two conditions below limit the number of neighbors for any node.

  1. The number of edge colors is fixed.
  2. No two edges adjacent to a node have the same color.

# KU Machines

- KU machines operate on a set of states ***S***.
- The states are directed graphs with color-coded edges.
- The nodes act as data cells and have symbols from a finite alphabet written in them.
- The two conditions below limit the number of neighbors for any node.

    1. The number of edge colors is fixed.
    2. No two edges adjacent to a node have the same color.

- At every stage in the computation, there is a unique node called the *focal node*: *f*.

# KU Machines

# KU Machines

- Around the focal node, all the nodes within a fixed distance, n (called the *attention span*), form the *active patch*.

# KU Machines

- Around the focal node, all the nodes within a fixed distance, n (called the *attention span*), form the *active patch*.
- A KU machine program is a finite domain function *m* such that if the active patch is isomorphic to *P* it gets replaced with *m(P)*

# KU Machines

- Around the focal node, all the nodes within a fixed distance, n (called the *attention span*), form the *active patch*.
- A KU machine program is a finite domain function *m* such that if the active patch is isomorphic to *P* it gets replaced with *m(P)*

$$\{(P_1, m(P_1)), (P_2, m(P_2)), \ldots, (P_k, m(P_k))\}$$

# KU Machines

- Around the focal node, all the nodes within a fixed distance, n (called the *attention span*), form the *active patch*.
- A KU machine program is a finite domain function *m* such that if the active patch is isomorphic to *P* it gets replaced with *m(P)*

$$\{(P_1, m(P_1)), (P_2, m(P_2)), \ldots, (P_k, m(P_k))\}$$

- Associated with each pair *(P, m(P))* in the machine program is a mapping *s* between nodes in the boundary of the active patch *P* to certain nodes in *m(P)*. This aligns the new active patch with the rest of the state

# KU Machines

# KU Machines

- Initial states, *I*, and final states, *F*, are characterized by certain special symbols in the focal node *f*.

# KU Machines

- Initial states, *I*, and final states, *F*, are characterized by certain special symbols in the focal node *f*.
- The initial states contain an encoding of the input.

# KU Machines

- Initial states, *I*, and final states, *F*, are characterized by certain special symbols in the focal node *f*.
- The initial states contain an encoding of the input.
- The final states contain an encoding of the output.

# Example: Validity of an Argument in Propositional Calculus

To check if

$$\{\phi_1, \phi_2, \ldots, \phi_n\} \vdash \psi$$

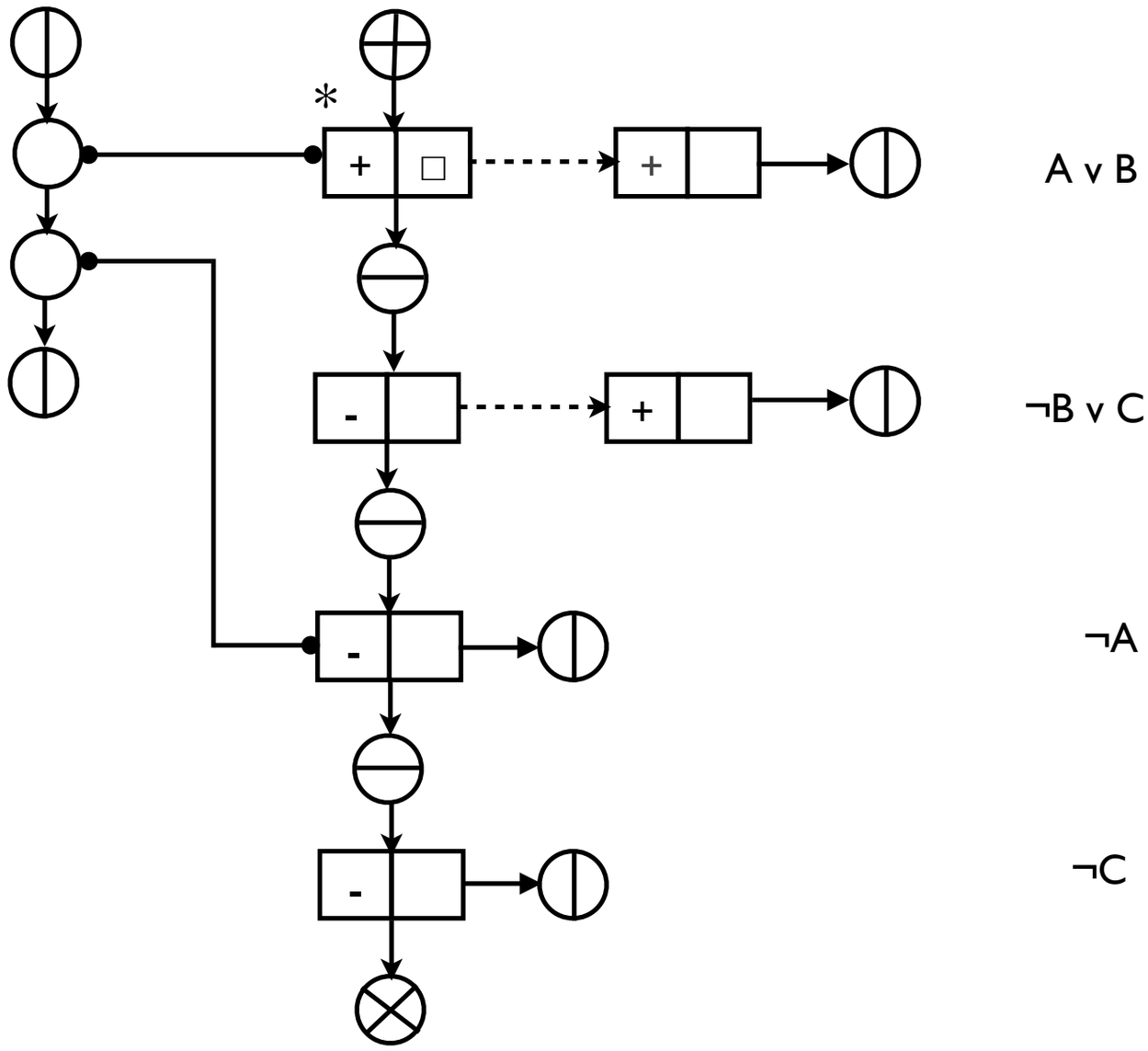
# The *truth-tree* algorithm

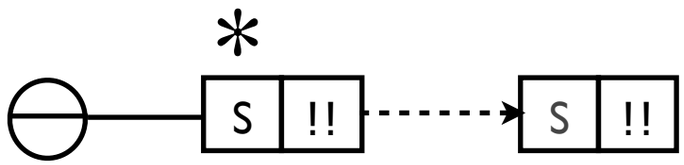
If we find that the set of formulas, $\Gamma = \{\phi_i | i = 1 \ldots n\} \cup \{\neg\psi\}$, comprising the premises and the negated conclusion, is not satisfiable, we can infer that the argument in question is valid, i.e., $\alpha = \{\phi_1, \phi_2, \ldots, \phi_n\} \vdash \psi$. The algorithm $\mathcal{T}$ works by constructing a *truth-tree* $\tau$ for the set of formulas $\Gamma$. The truth tree $\tau$ is a *rooted tree* and is constructed using the formulas in $\Gamma$ as follows:

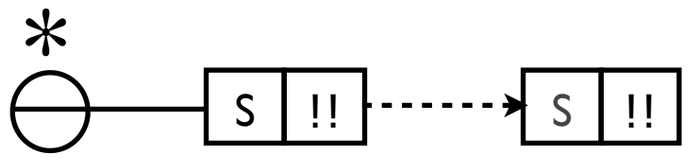
1. Place the root $r$.
2. The list of formulas in $\Gamma$ is converted to a list of formulas $\Gamma'$ such that $\Gamma$ is not satisfiable iff $\Gamma'$ is not satisfiable, and $\Gamma'$ consists of formulas which are either disjuncts or conjuncts of literals. There is a straightforward algorithm to accomplish this.
3. For each formula in $\Gamma'$:
   (a) If the formula is a disjunct of literals, for each leaf $l$ in the unfinished tree place the literals as new leaves in a *parallel* fashion and connect each literal to the old leaf $l$.
   (b) If the formula is a conjunct of literals, for each leaf $l$ in the unfinished tree place the literals as new leaves in a *serial* fashion and connect the topmost literal to the old leaf $l$.

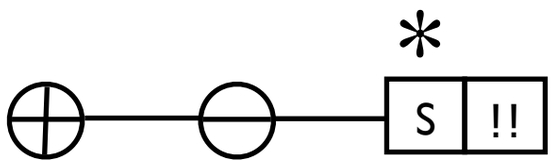
# Example KU Machine State of an Implementation of the Truth-tree Algorithm

Symbols={!, □ ,!!,! □ , ◫ ,+,-,+w,-w,+D,-D}

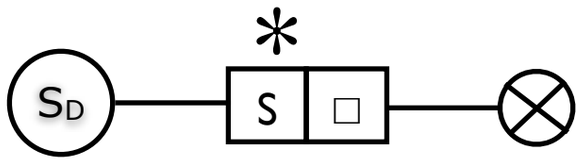




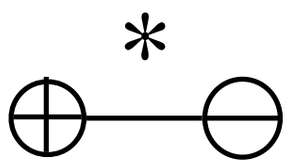



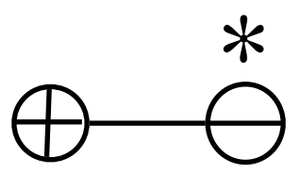



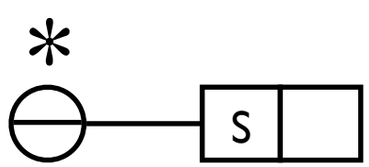

No

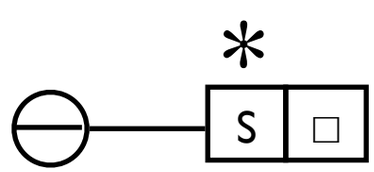


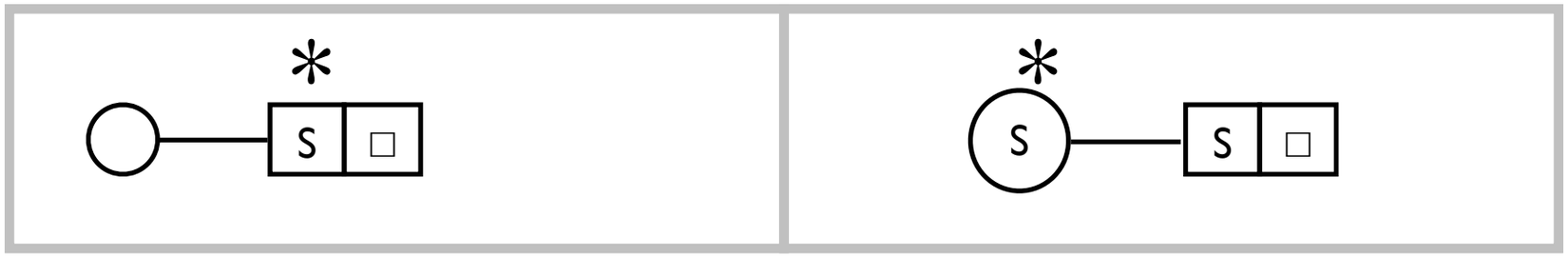

Yes

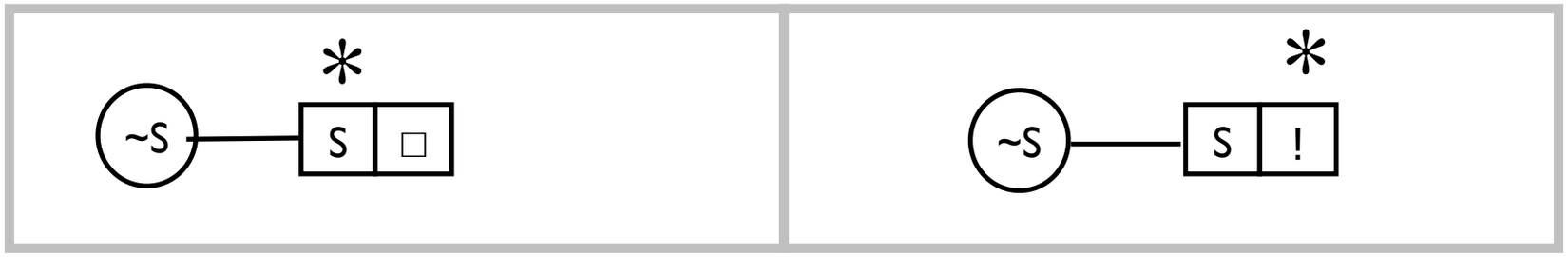


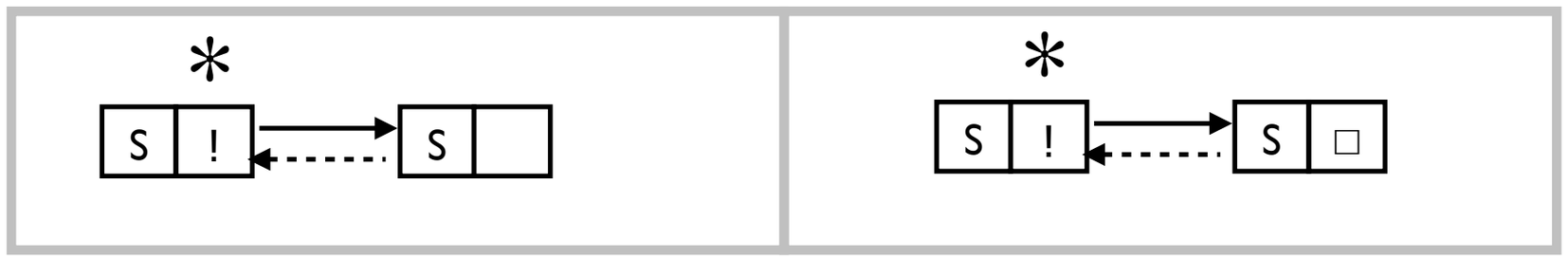



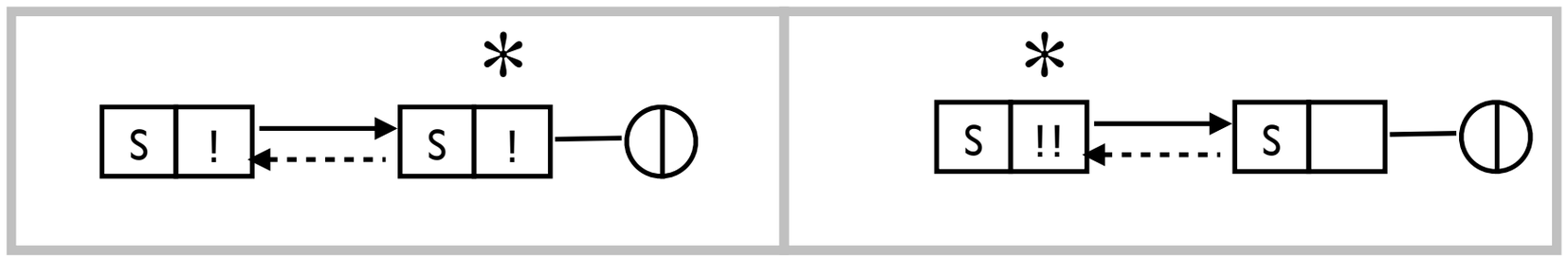



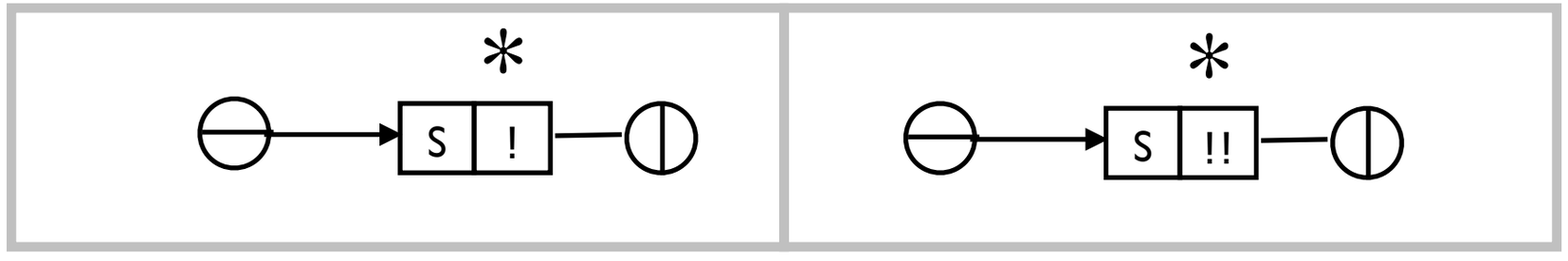



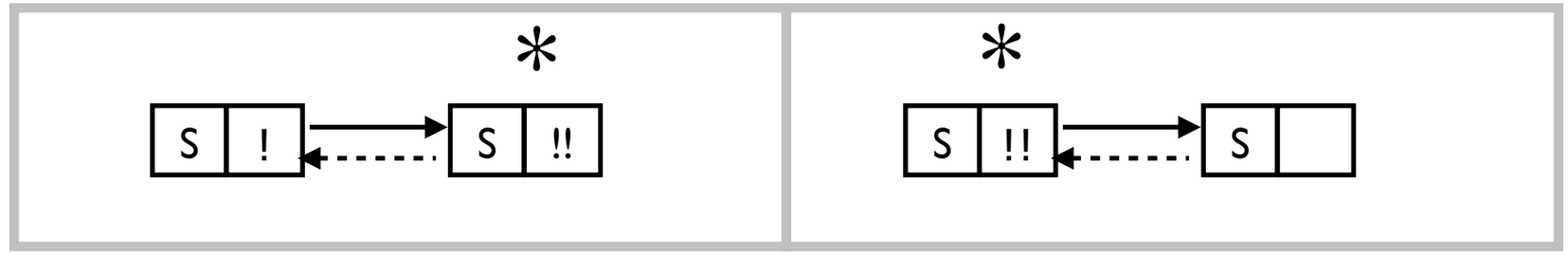

*
S
□
*
S
S
□

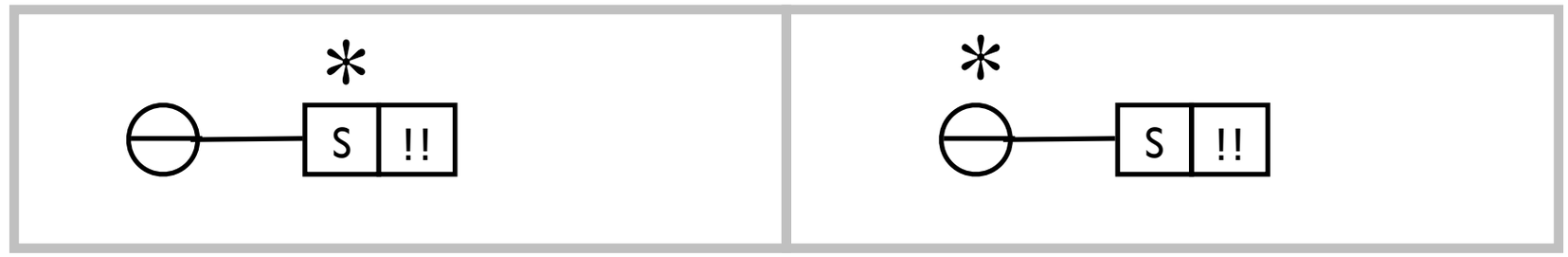

*
~S
S
□
*
~S
S
!

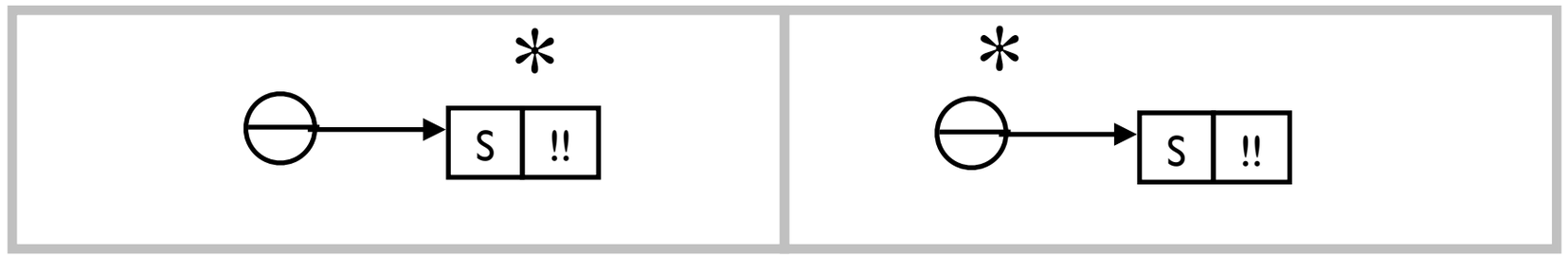

*
S
!
S
*
S
!
S
□

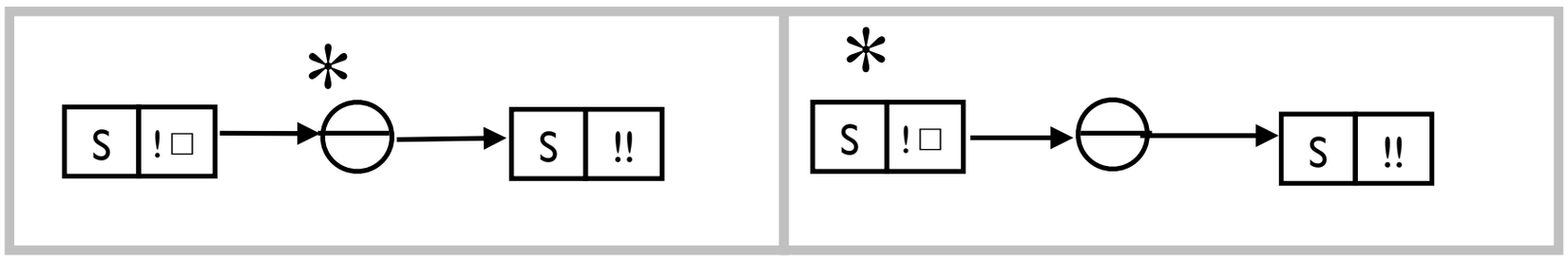

*
S
!
S
!
*
S
!!
S

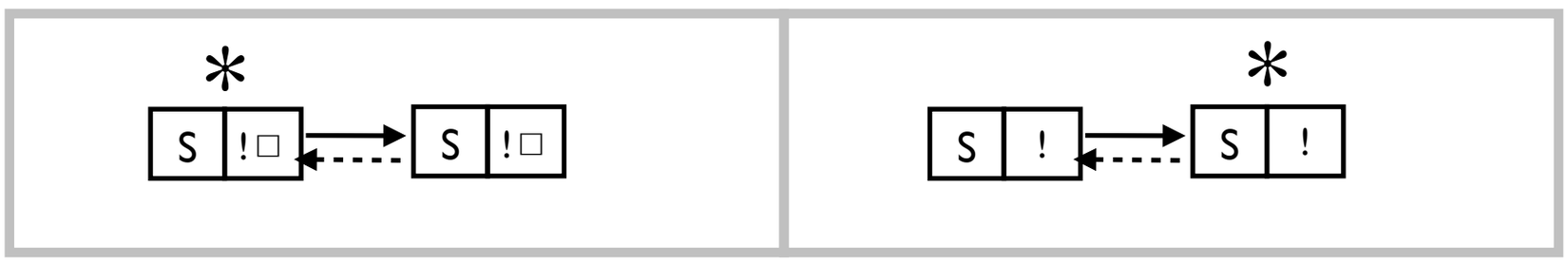

*
S
!
*
S
!!

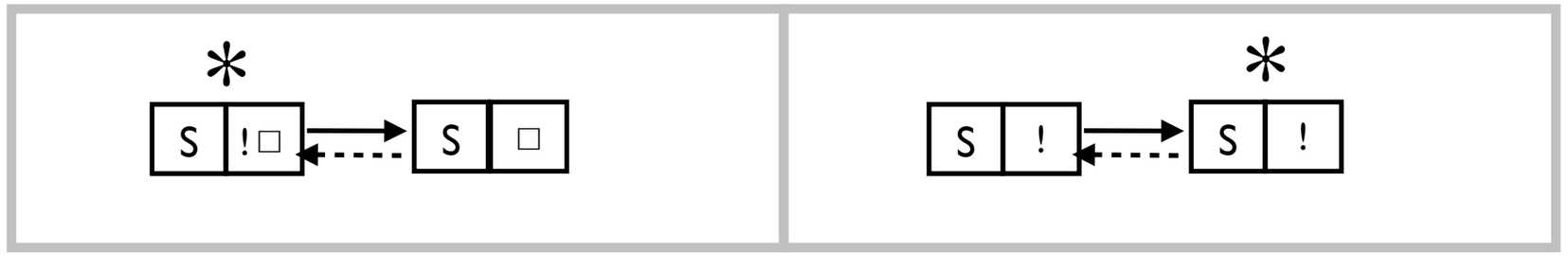

*
S
!
S
!!
*
S
!!
S


*
S
!!
*
S
!!

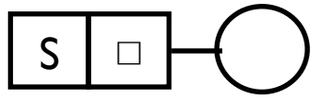

*
S
!!
*
S
!!

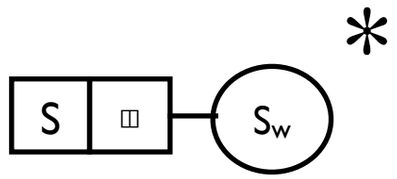

*
S
!□
S
!!
*
S
!□
S
!!

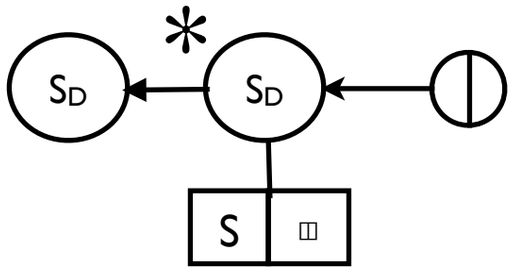

*
S
!□
S
!□
*
S
!
S
!

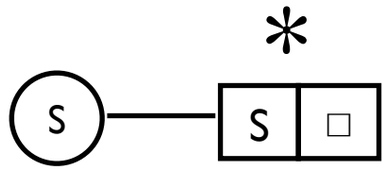

S
!□
S
□
S
!
S
!
S
!
S
□
S
!
S
!
S
!
S
□
S
S
!
S
!
S
S
S
!□
S
!□

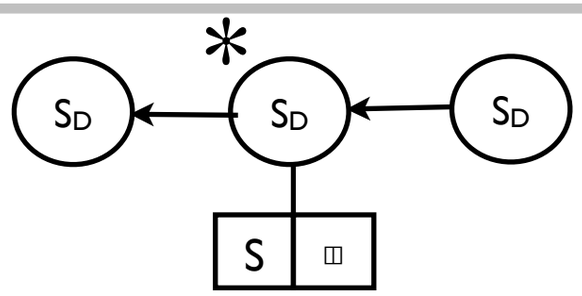
# variable setting routine

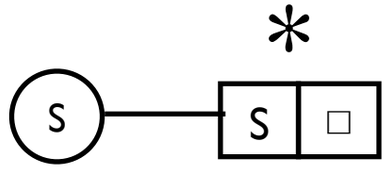


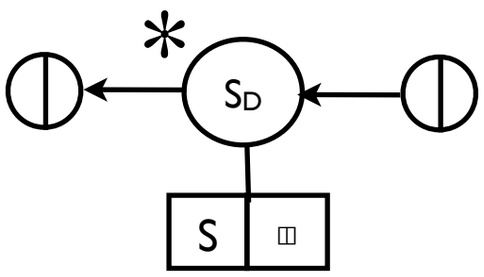



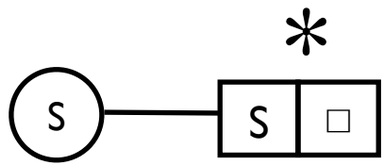



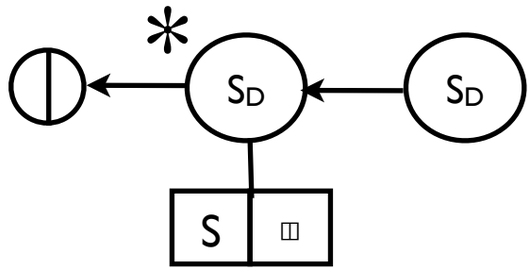



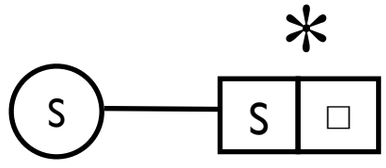



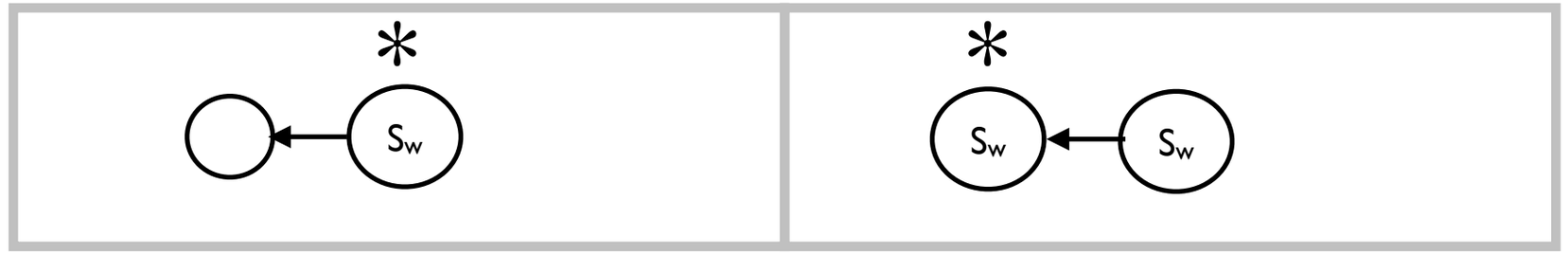



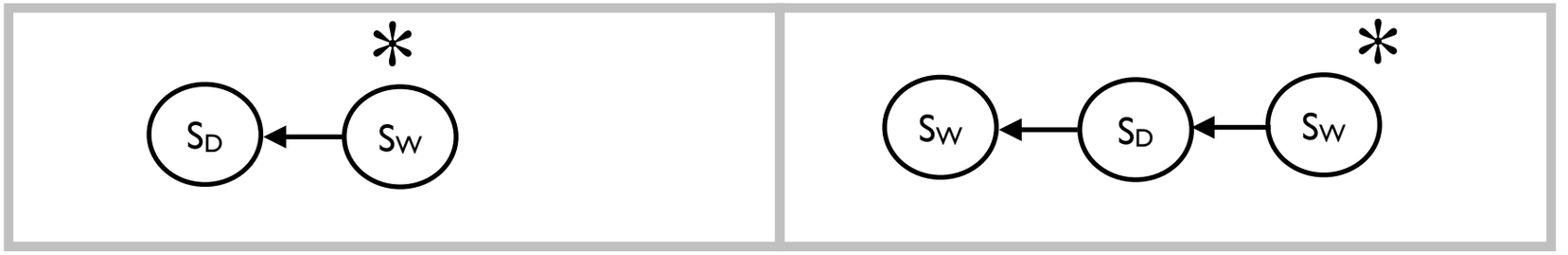



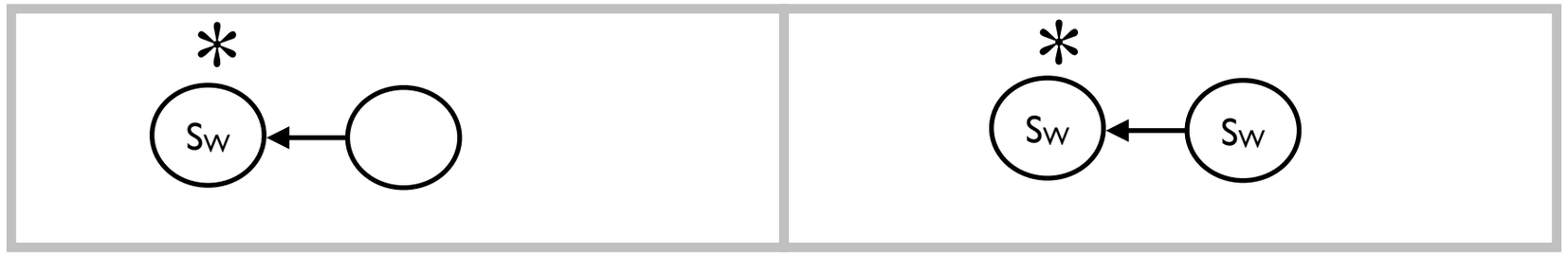



variable setting routine



variable setting routine



# Infinitary KU machines

# Infinitary KU machines

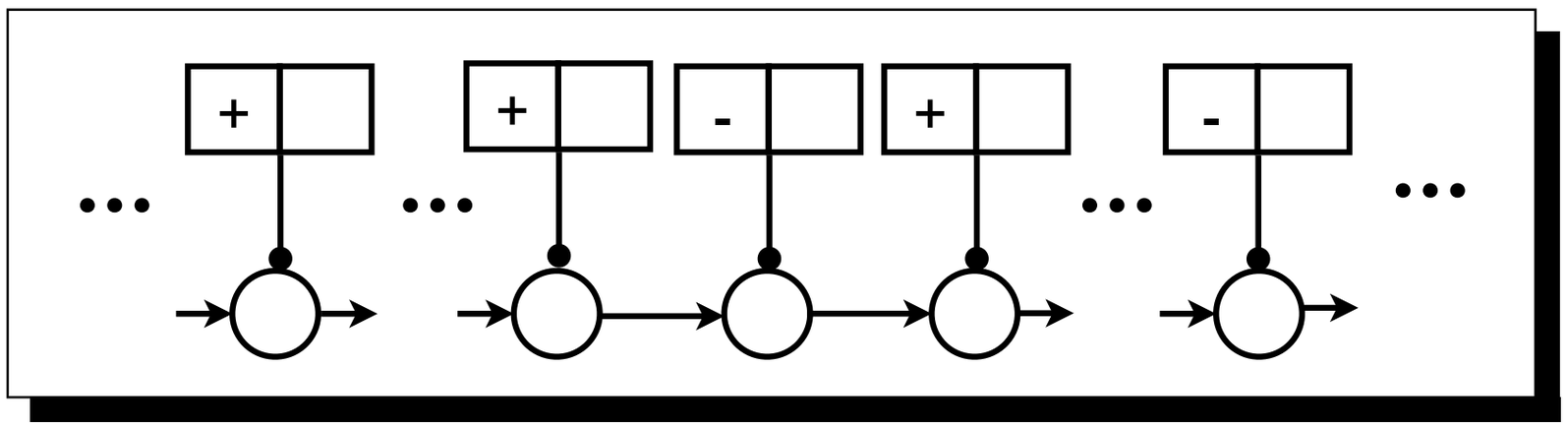
- The workspace can be *infinitary*.

# Infinitary KU machines

- The workspace can be *infinitary*.
- The representation of the workspace is effective and *finite* (e.g. using the dot notation).

# Infinitary KU machines

- The workspace can be *infinitary*.
- The representation of the workspace is effective and *finite* (e.g. using the dot notation).



# A bit more precisely

# A bit more precisely

- The states are infinite directed graphs.

# A bit more precisely

- The states are infinite directed graphs.
- Note that some infinite graphs can be represented finitely and effectively.

# A bit more precisely

- The states are infinite directed graphs.
- Note that some infinite graphs can be represented finitely and effectively.

- Set-theoretically,

# A bit more precisely

- The states are infinite directed graphs.
- Note that some infinite graphs can be represented finitely and effectively.

- Set-theoretically,
  - Standard-KU-machine states correspond to finite states.

# A bit more precisely

- The states are infinite directed graphs.
- Note that some infinite graphs can be represented finitely and effectively.

- Set-theoretically,
  - Standard-KU-machine states correspond to finite states.
  - Infinitary-KU-machine states correspond to finite and infinite sets.

# Our Case

- P∞Ms are essentially a special case of infinitary KU machines.