Introducing HyperLog

Selmer Bringsjord

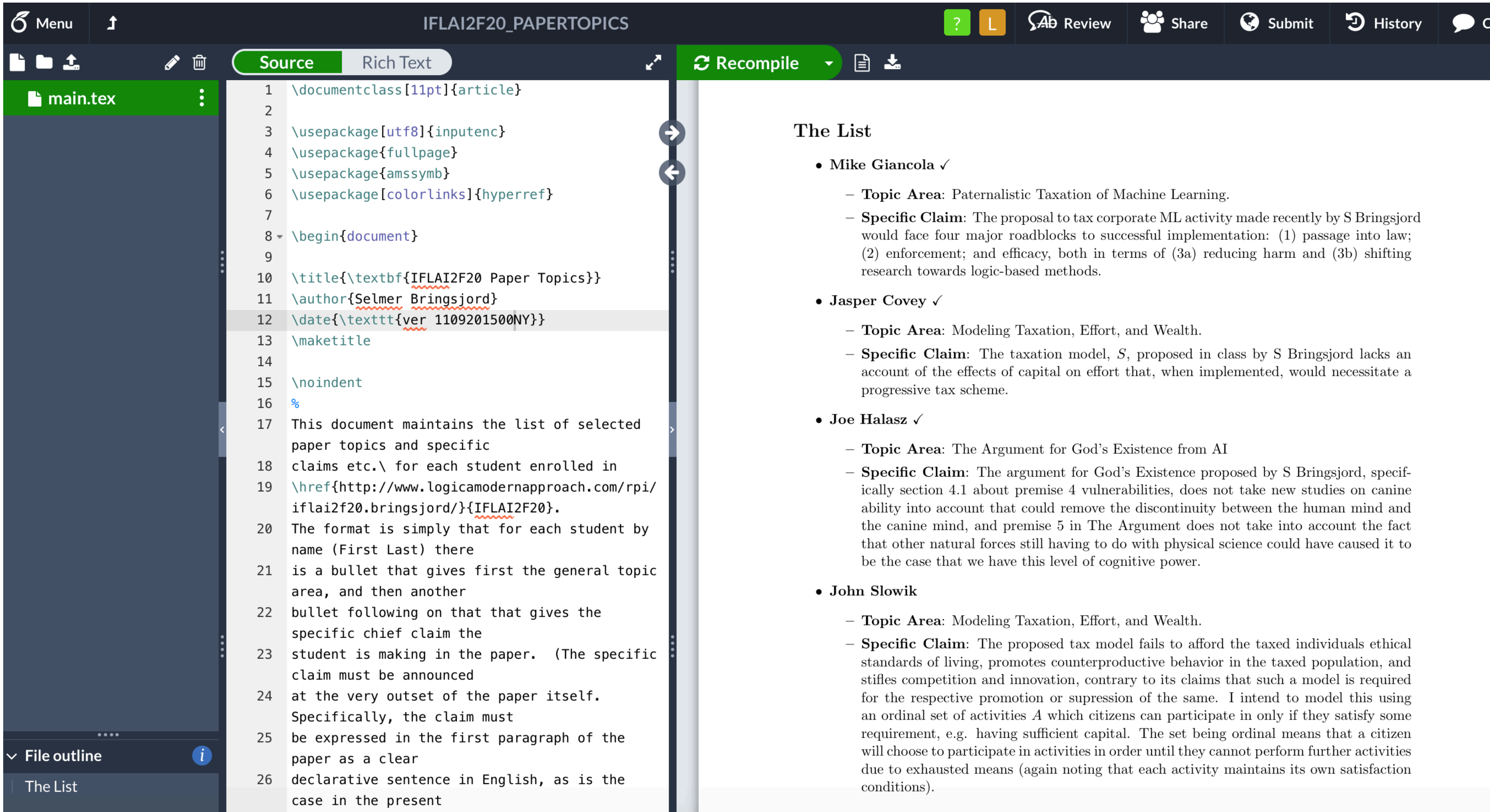
Rensselaer AI & Reasoning (RAIR) Lab
Department of Cognitive Science
Department of Computer Science
Lally School of Management & Technology
Rensselaer Polytechnic Institute (RPI)
Troy, New York 12180 USA

IFLAI2
11/19/2020
ver 1119201645NY

Rensselaer AI and Reasoning Lab

Misc topics …

Still some non-checkmarks. Btw, I hereby announce first draft is now due *after* Tgiving break: 3pm NY time on Dec 3.



Schedule

Schedule

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond.* We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.
- **Nov 9**: *Hypergraphical Proof and Programming in HyperSlate®.* We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.
- **Nov 12**: *Quantified Modal Logic.* We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.
- **Nov 16**: *Killer Robots,* ***D****, and Beyond in HyperSlate® to* $\mathcal{DCEC}$. We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates.

Schedule

√

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond*. We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.
- **Nov 9**: *Hypergraphical Proof and Programming in HyperSlate®*. We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.
- **Nov 12**: *Quantified Modal Logic*. We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.
- **Nov 16**: *Killer Robots, **D**, and Beyond in HyperSlate® to $\mathcal{DCEC}$*. We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates.

Schedule

✓

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond*. We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.
- **Nov 9**: *Hypergraphical Proof and Programming in HyperSlate®*. We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.

✓

- **Nov 12**: *Quantified Modal Logic*. We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.
- **Nov 16**: *Killer Robots, **D**, and Beyond in HyperSlate® to $\mathcal{DCEC}$*. We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates.

Schedule

√

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond.* We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.
- **Nov 9**: *Hypergraphical Proof and Programming in HyperSlate®.* We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.

√

- **Nov 12**: *Quantified Modal Logic.* We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.

√

- **Nov 16**: *Killer Robots, **D**, and Beyond in HyperSlate® to $\mathcal{DCEC}$.* We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates.

Schedule

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond.* We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.
- **Nov 9**: *Hypergraphical Proof and Programming in HyperSlate®.* We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.
- **Nov 12**: *Quantified Modal Logic.* We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.
- **Nov 16**: *Killer Robots, **D**, and Beyond in HyperSlate® to $\mathcal{DCEC}$.* We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates. We review that modal logic **D** is painfully inadequate, but now move to some exploration of a version of $\mathcal{DCEC}$ in HyperSlate®.
- **Nov 19**: *The Logicist AI-ification of the Doctrines of N Effect to Solve the PAID Problem.*
- **Nov 23**: ***ZFC***. We review and expand our understanding of axiomatic set theory, and of the relative size of infinite sets. **ZFC** in HyperSlate® is visited and explored. **Note**: This is the last day of any in-person instruction.
- **Nov 26**: **No Class** (Thanksgiving).

Schedule

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond.* We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.
- **Nov 9**: *Hypergraphical Proof and Programming in HyperSlate®.* We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.
- **Nov 12**: *Quantified Modal Logic.* We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.
- **Nov 16**: *Killer Robots, **D**, and Beyond in HyperSlate® to $\mathcal{DCEC}$.* We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates. We review that modal logic **D** is painfully inadequate, but now move to some exploration of a version of $\mathcal{DCEC}$ in HyperSlate®.
- **Nov 19**: *The Logicist AI-ification of the Doctrines of N Effect to Solve the PAID Problem.*
- **Nov 23**: ***ZFC***. We review and expand our understanding of axiomatic set theory, and of the relative size of infinite sets. **ZFC** in HyperSlate® is visited and explored. **Note**: This is the last day of any in-person instruction.
- **Nov 26**: **No Class** (Thanksgiving).

Schedule

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond.* We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.
- **Nov 9**: *Hypergraphical Proof and Programming in HyperSlate®.* We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.
- **Nov 12**: *Quantified Modal Logic.* We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.
- **Nov 16**: *Killer Robots, **D**, and Beyond in HyperSlate® to $\mathcal{DCEC}$.* We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates. We review that modal logic **D** is painfully inadequate, but now move to some exploration of a version of $\mathcal{DCEC}$ in HyperSlate®.
- Nov 16 **19**: *The Logicist AI-ification of the Doctrines of N Effect to Solve the PAID Problem.*
- **Nov 23**: ***ZFC***. We review and expand our understanding of axiomatic set theory, and of the relative size of infinite sets. **ZFC** in HyperSlate® is visited and explored. **Note**: This is the last day of any in-person instruction.
- **Nov 26**: **No Class** (Thanksgiving).

Schedule

- **Nov 5**: *Pure General Logic Programming, Functional Programming, Turing-Completeness, and Beyond.* We review the basic paradigms of computer programming. For the imperative case, we use the simple imperative language of (Davis, Sigal & Weyuker 1994), and also discuss register machines, Turing machines (again), KU machines. We also discuss whether programming beyond the Turing Limit makes sense and can be pursued.

Today, Nov 19

pergraphical Proof and Programming in HyperSlate®. We here introduce the availability of writing Clojure functions in the context of proofs in HyperSlate®.

- **Nov 12**: *Quantified Modal Logic.* We here explore quantified **S5**, the infamous Barcan Formula. HyperSlate® is used.
- **Nov 16**: *Killer Robots, **D**, and Beyond in HyperSlate® to $\mathcal{DCEC}$.* We begin here by stating the "PAID Problem," and then the approach to it from Bringsjord et al. advocates. We review that modal logic **D** is painfully inadequate, but now move to some exploration of a version of $\mathcal{DCEC}$ in HyperSlate®.

Nov 16

19: *The Logicist AI-ification of the Doctrines of N Effect to Solve the PAID Problem.*

- **Nov 23**: ***ZFC***. We review and expand our understanding of axiomatic set theory, and of the relative size of infinite sets. **ZFC** in HyperSlate® is visited and explored. **Note**: This is the last day of any in-person instruction.
- **Nov 26**: **No Class** (Thanksgiving).

DCEC in HyperSlate® …

Inference Schemata

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

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

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

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

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

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

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

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

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

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

DCEC (supported fragment)

First-order (Propositional) Schema

- Assume
- Not Elim, Not Intro
- And Elim, And Intro
- Or Elim, Or Intro
- If Elim, If Intro
- Iff Elim, Iff Intro
- Forall Elim, Forall Intro
- Exists Elim, Exists Intro
- Higher Order Forall Elim, Higher Order Forall Intro
- Higher Order Exists Elim, Higher Order Exists Intro
- Eq Elim, Eq Intro
- Pc Oracle, Fol Oracle

Modal Schema

- R_1, R_2, R_3, R_4,
- R_k, R_b,
- R_{14}

Inference Schemata

Modal

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

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

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

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

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

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

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

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

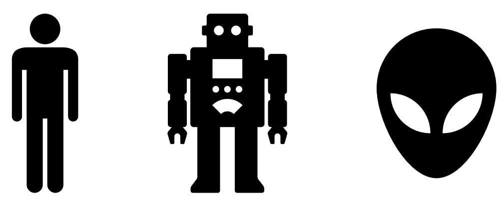
$$\frac{\mathbf{S}(s,h,t,\phi)}{\mathbf{B}(h,t,\mathbf{B}(s,t,\phi))}\ [R_{12}] \qquad \frac{\mathbf{I}(a,t,happens(action(a^*,\alpha),t'))}{\mathbf{P}(a,t,happens(action(a^*,\alpha),t))}\ [R_{13}]$$

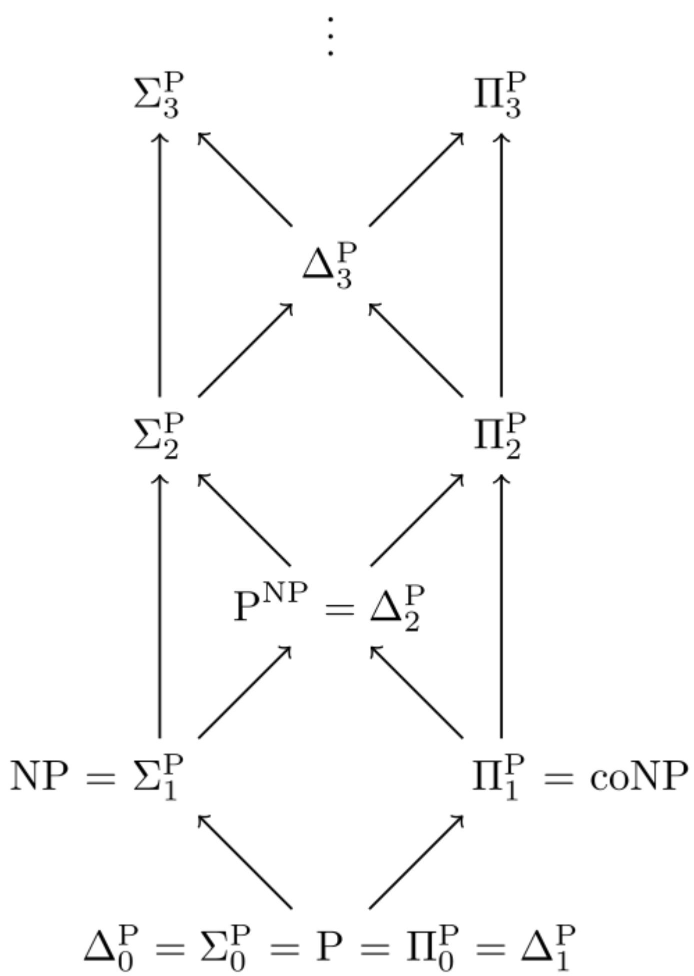
$$\frac{\mathbf{B}(a,t,\phi)\quad \mathbf{B}(a,t,\mathbf{O}(a,t,\phi,\chi))\quad \mathbf{O}(a,t,\phi,\chi)}{\mathbf{K}(a,t,\mathbf{I}(a,t,\chi))}\ [R_{14}]$$

Delivered on promissory note re building hierarchies via formal logic … questions?

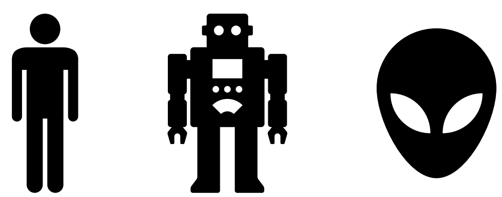
Polynomial Hierarchy, Part II

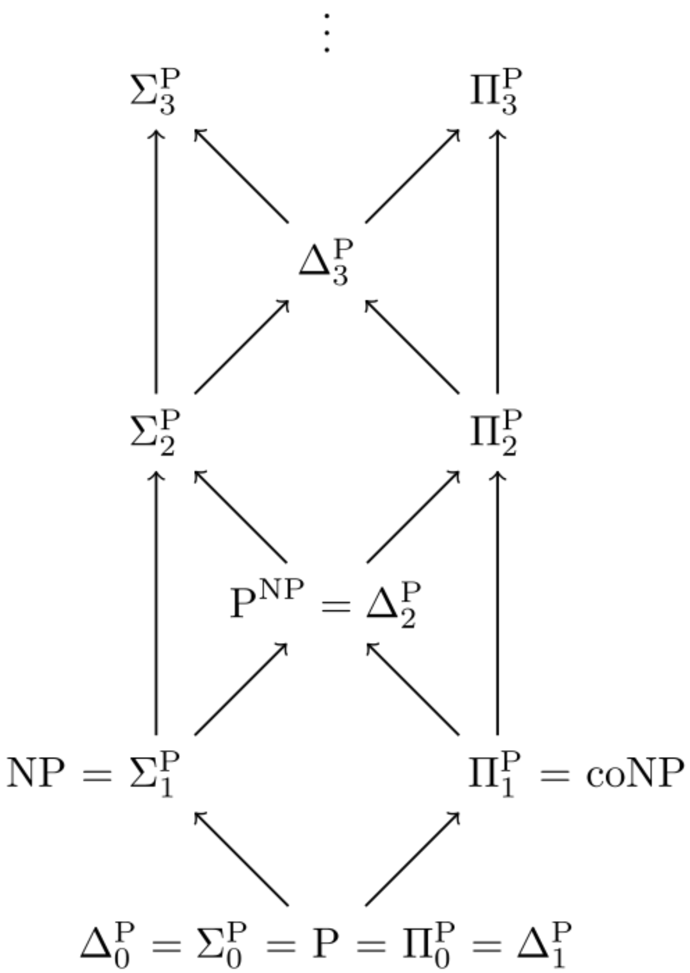
(via formal logic, directly)





Polynomial Hierarchy, Part II
(via formal logic, directly)

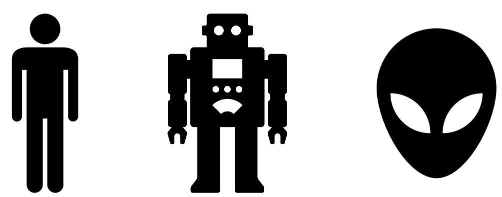


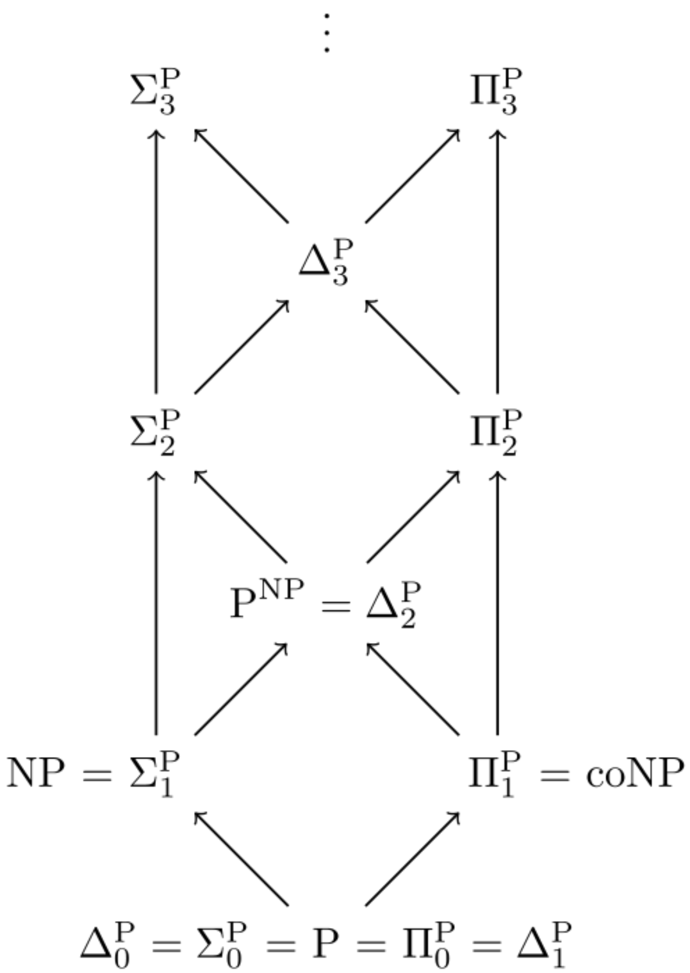


Eg:

Polynomial Hierarchy, Part II

(via formal logic, directly)



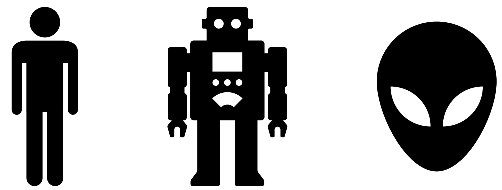


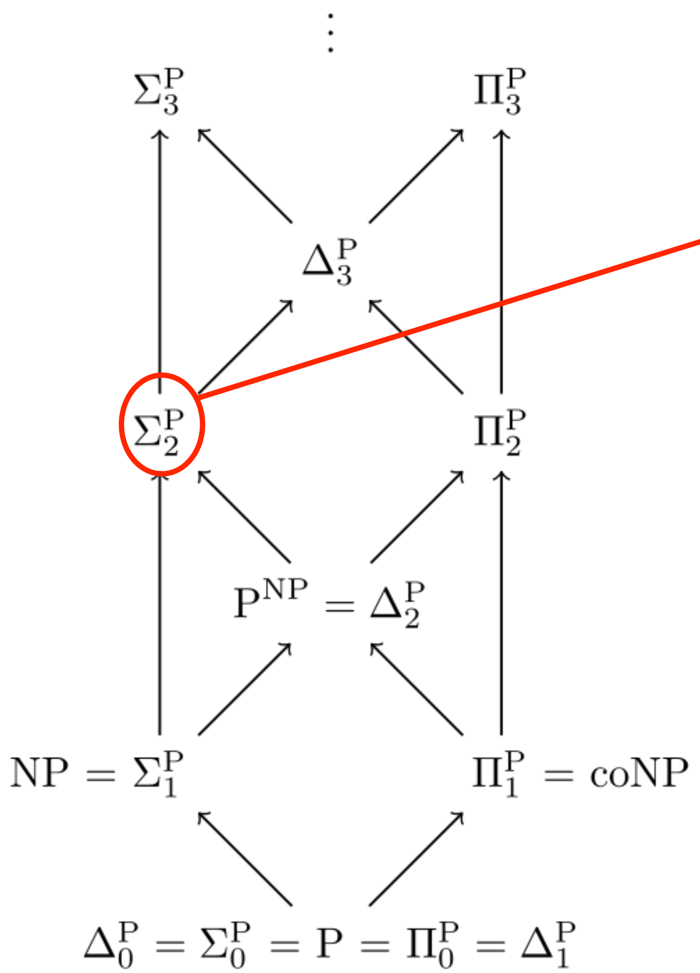
Eg:

$$\langle \phi_1, k \rangle \in L \text{ iff } \exists \phi_2 \forall \alpha KLogEquiv(\phi_1, \phi_2, |\phi_2| \leq k, \alpha(\phi_1) = \alpha(\phi_2))$$

Polynomial Hierarchy, Part II

(via formal logic, directly)



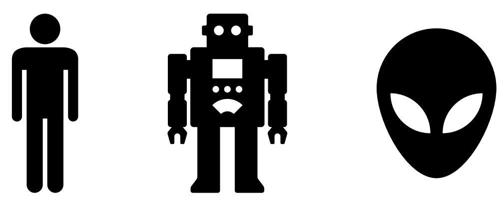


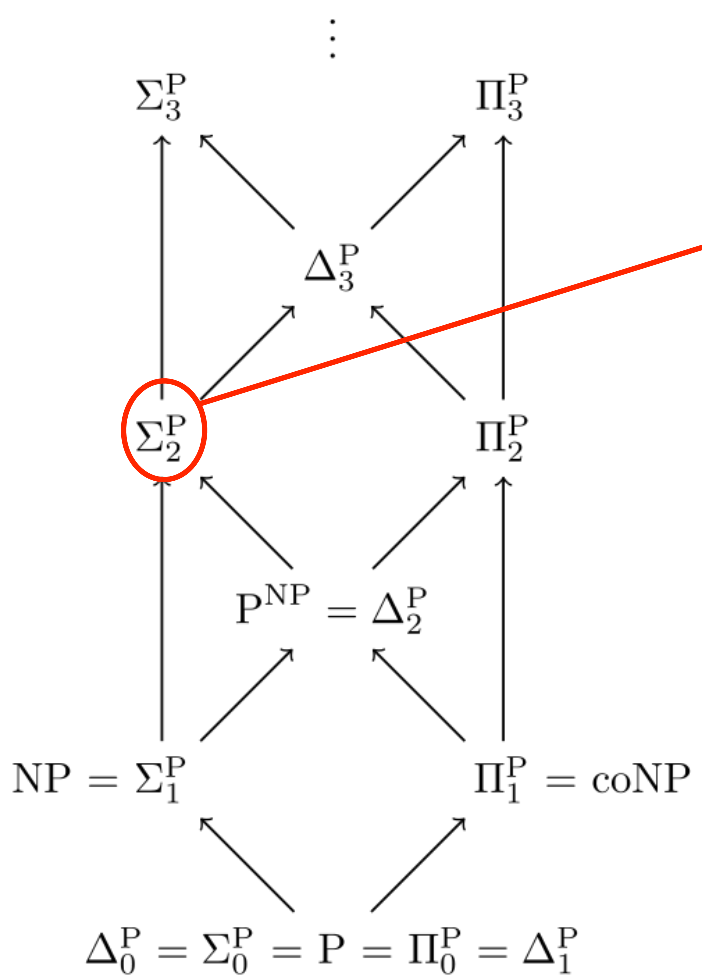
Eg:

$$\langle \phi_1, k \rangle \in L \text{ iff } \exists \phi_2 \forall \alpha KLogEquiv(\phi_1, \phi_2, |\phi_2| \leq k, \alpha(\phi_1) = \alpha(\phi_2))$$

Polynomial Hierarchy, Part II

(via formal logic, directly)





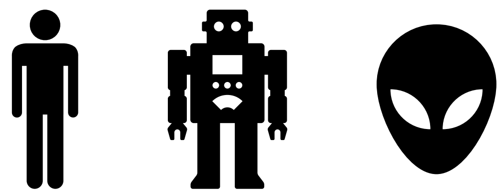
free variables

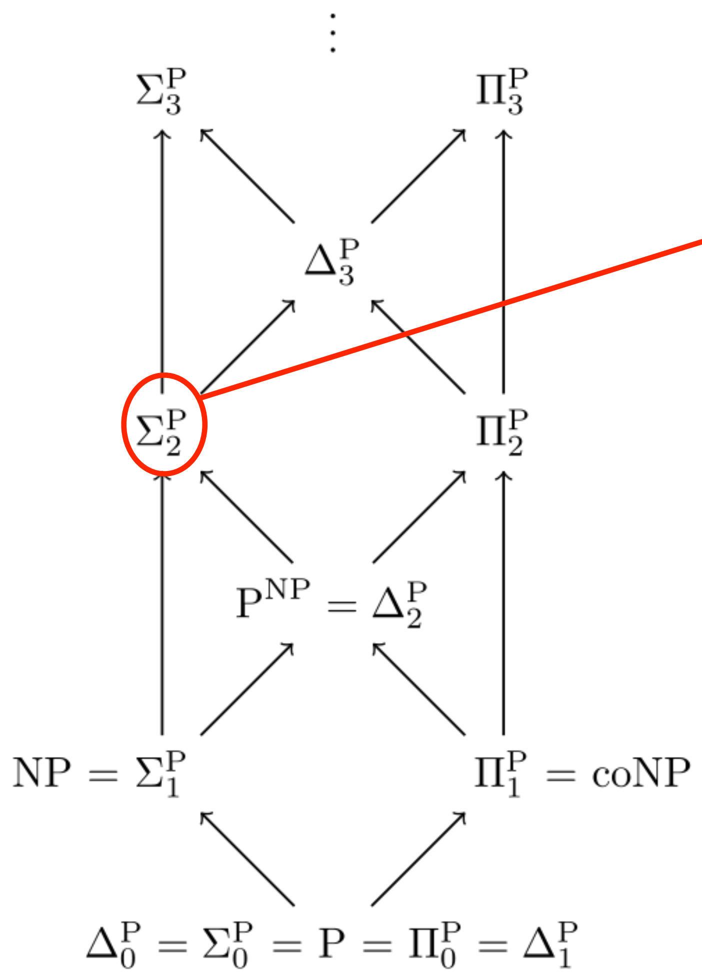
Eg:

$$\langle \phi_1, k \rangle \in L \text{ iff } \exists \phi_2 \forall \alpha KLogEquiv(\phi_1, \phi_2, |\phi_2| \leq k, \alpha(\phi_1) = \alpha(\phi_2))$$

Polynomial Hierarchy, Part II

(via formal logic, directly)





free variables

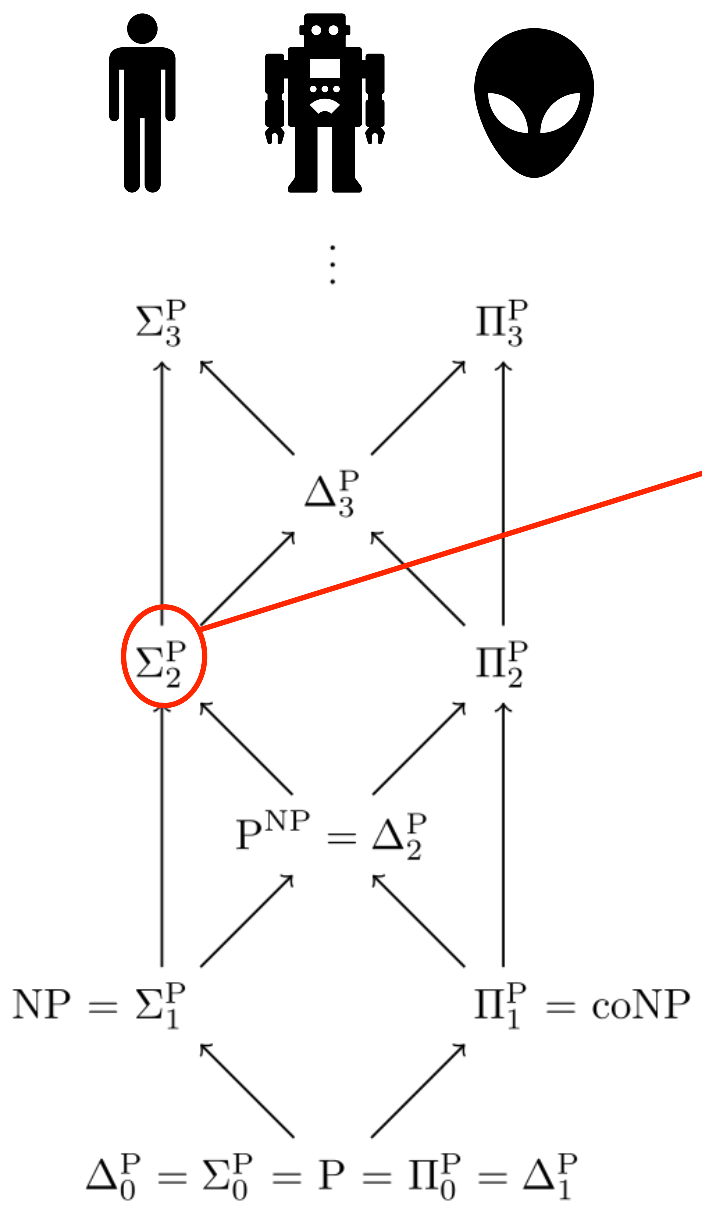
Eg:

$$\langle \phi_1, k \rangle \in L \text{ iff } \exists \phi_2 \forall \alpha KLogEquiv(\phi_1, \phi_2, |\phi_2| \leq k, \alpha(\phi_1) = \alpha(\phi_2))$$

Now we generalize:

Polynomial Hierarchy, Part II

(via formal logic, directly)



free variables

Eg:

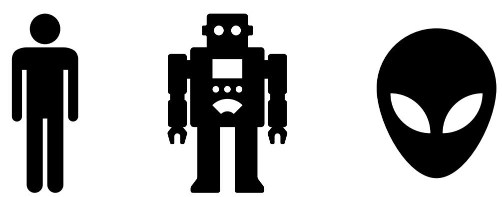
$$\langle \phi_1, k \rangle \in L \text{ iff } \exists \phi_2 \forall \alpha KLogEquiv(\phi_1, \phi_2, |\phi_2| \leq k, \alpha(\phi_1) = \alpha(\phi_2))$$

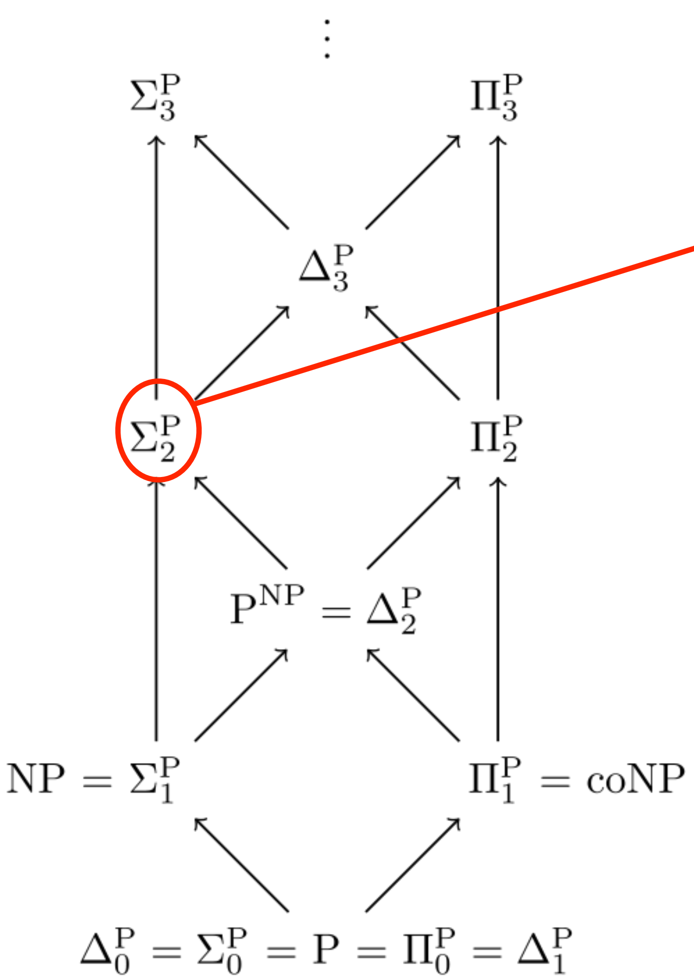
Now we generalize:

$$x \in \Sigma_i \text{ iff } \exists R \; \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

Polynomial Hierarchy, Part II

(via formal logic, directly)





free variables

Eg:

$$\langle \phi_1, k \rangle \in L \text{ iff } \exists \phi_2 \forall \alpha KLogEquiv(\phi_1, \phi_2, |\phi_2| \leq k, \alpha(\phi_1) = \alpha(\phi_2))$$

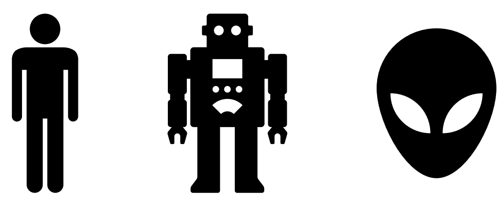
Now we generalize:

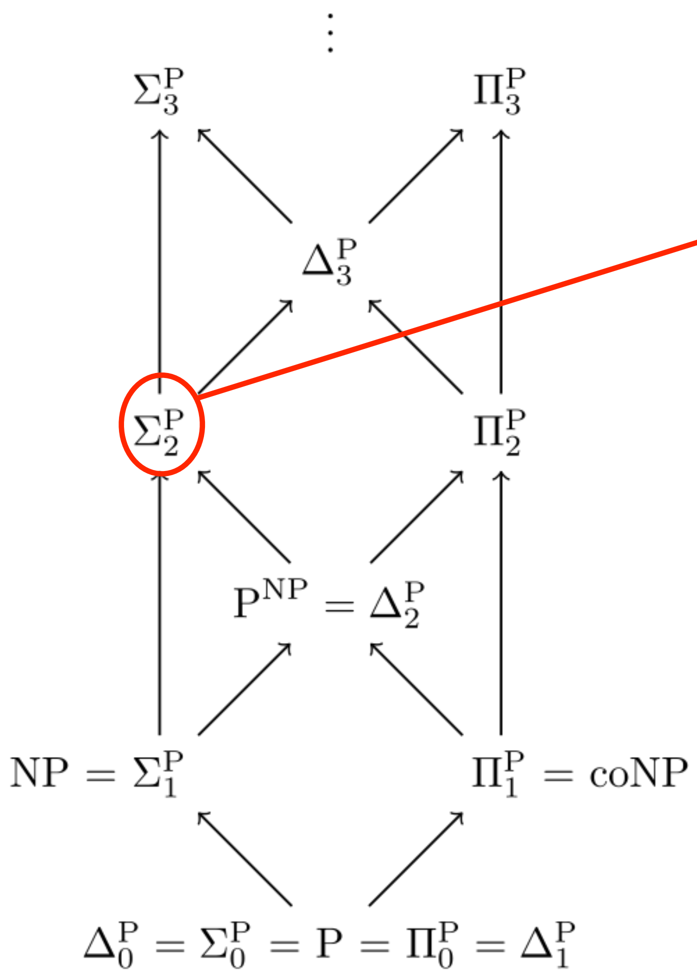
$$x \in \Sigma_i \text{ iff } \exists R \; \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R \; \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Polynomial Hierarchy, Part II

(via formal logic, directly)





free variables

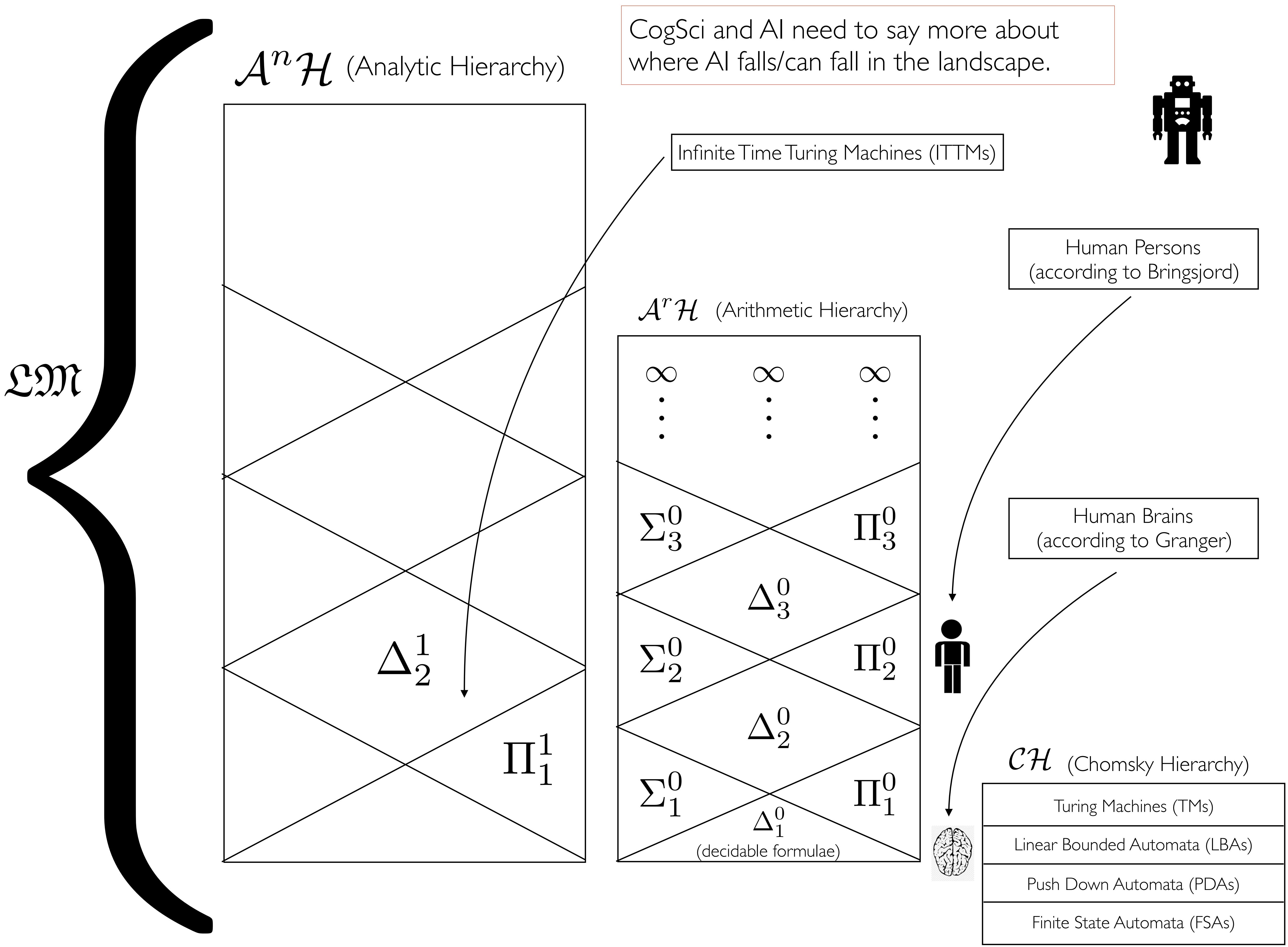
Eg:

$$\langle \phi_1, k \rangle \in L \text{ iff } \exists \phi_2 \forall \alpha \, KLogEquiv(\phi_1, \phi_2, |\phi_2| \leq k, \alpha(\phi_1) = \alpha(\phi_2))$$

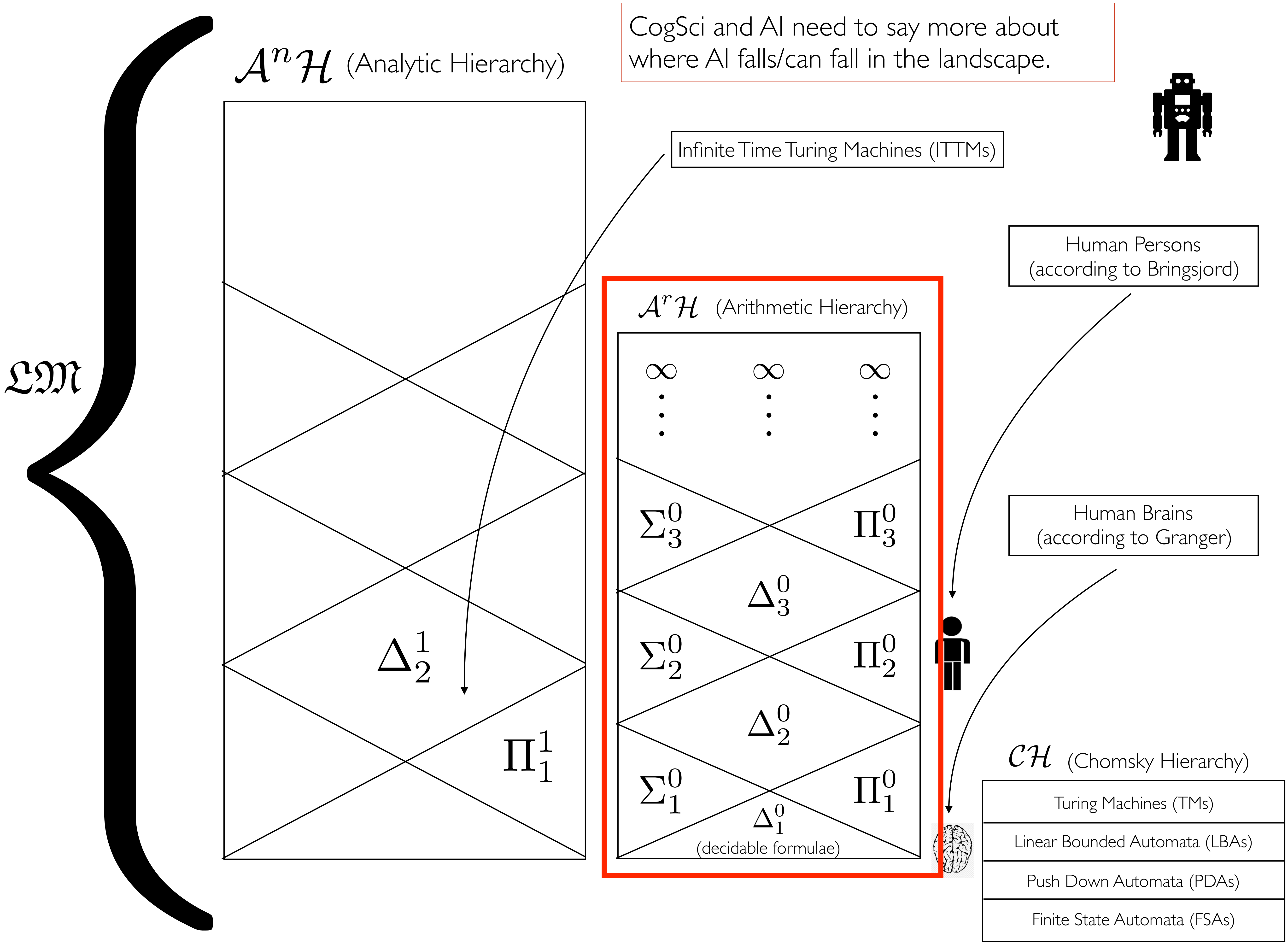
Now we generalize:

$$x \in \Sigma_i \text{ iff } \exists R \; \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R \; \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$



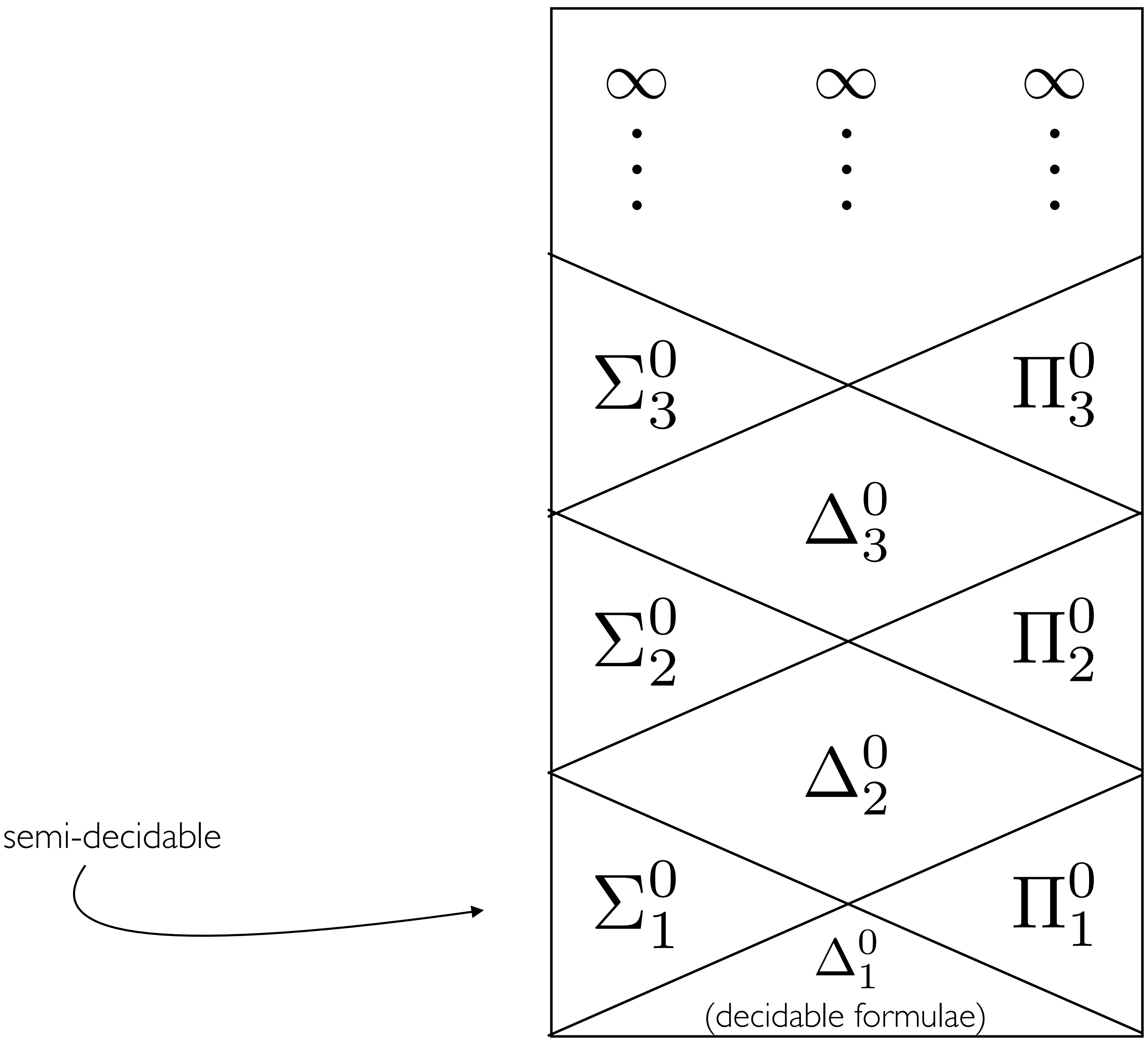
CogSci and AI need to say more about where AI falls/can fall in the landscape.
$\mathcal{LM}$
$\mathcal{A}^n\mathcal{H}$ (Analytic Hierarchy)
Infinite Time Turing Machines (ITTMs)
Δ^1_2
Π^1_1
Human Persons (according to Bringsjord)
$\mathcal{A}^r\mathcal{H}$ (Arithmetic Hierarchy)
∞
∞
∞
Σ^0_3
Π^0_3
Δ^0_3
Σ^0_2
Π^0_2
Δ^0_2
Σ^0_1
Π^0_1
Δ^0_1 (decidable formulae)
Human Brains (according to Granger)
$\mathcal{CH}$ (Chomsky Hierarchy)
Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)



CogSci and AI need to say more about where AI falls/can fall in the landscape.
$\mathfrak{LM}$
$\mathcal{A}^n\mathcal{H}$ (Analytic Hierarchy)
Δ^1_2
Π^1_1
Infinite Time Turing Machines (ITTMs)
$\mathcal{A}^r\mathcal{H}$ (Arithmetic Hierarchy)
∞ ∞ ∞
Σ^0_3 Π^0_3
Δ^0_3
Σ^0_2 Π^0_2
Δ^0_2
Σ^0_1 Π^0_1
Δ^0_1
(decidable formulae)
Human Persons (according to Bringsjord)
Human Brains (according to Granger)
$\mathcal{CH}$ (Chomsky Hierarchy)
Turing Machines (TMs)
Linear Bounded Automata (LBAs)
Push Down Automata (PDAs)
Finite State Automata (FSAs)

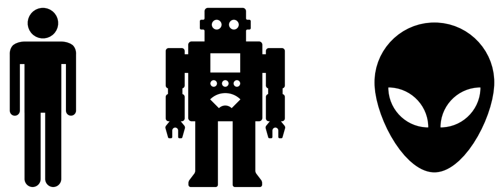
$$\mathbf{2SAMEFUNC} \coloneqq \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

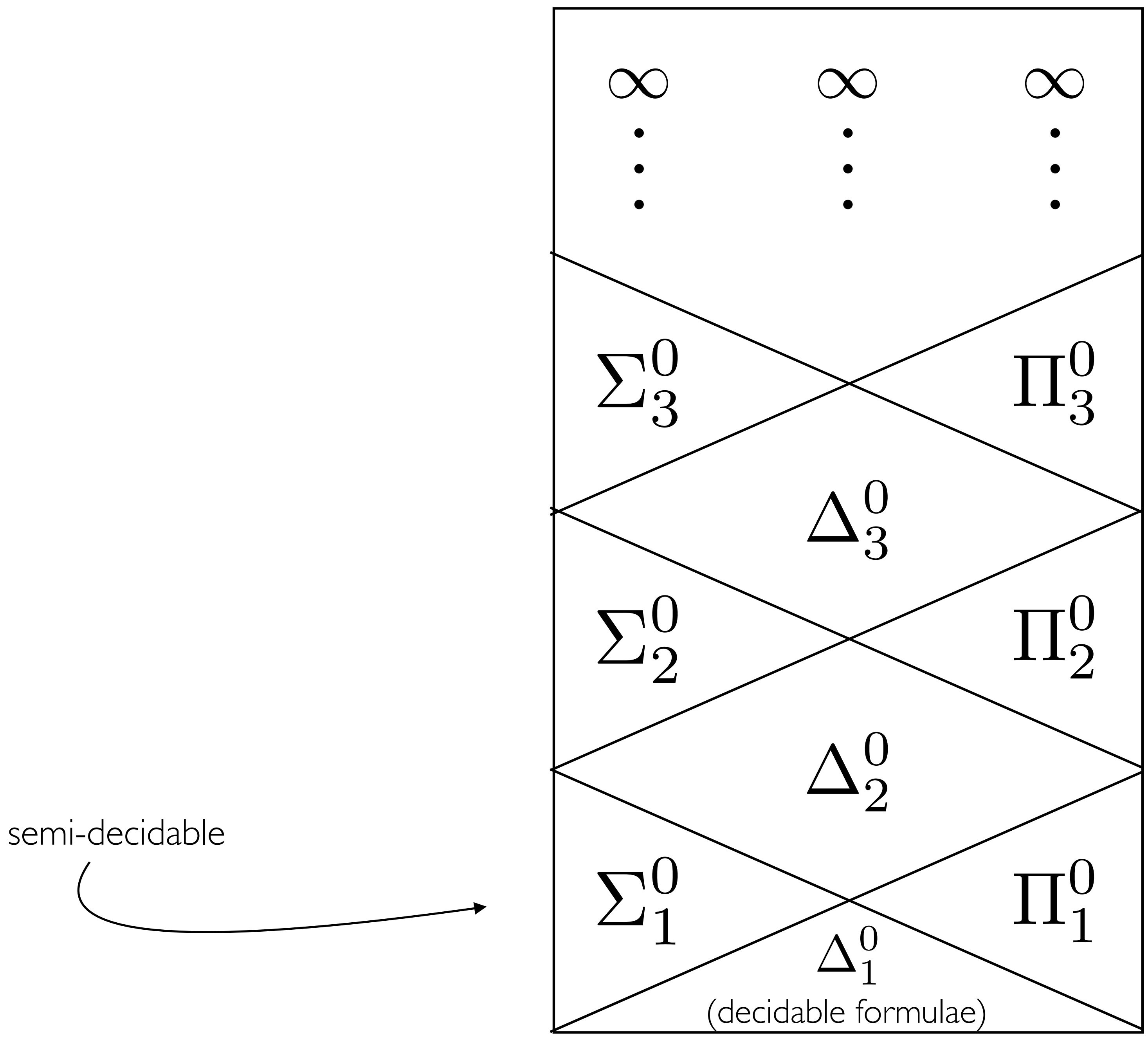
Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

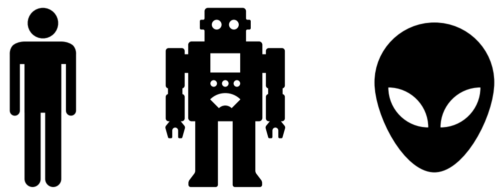
$\mathcal{A}^r\mathcal{H}$ (Arithmetic Hierarchy)



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

Arithmetic Hierarchy, Part I

Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

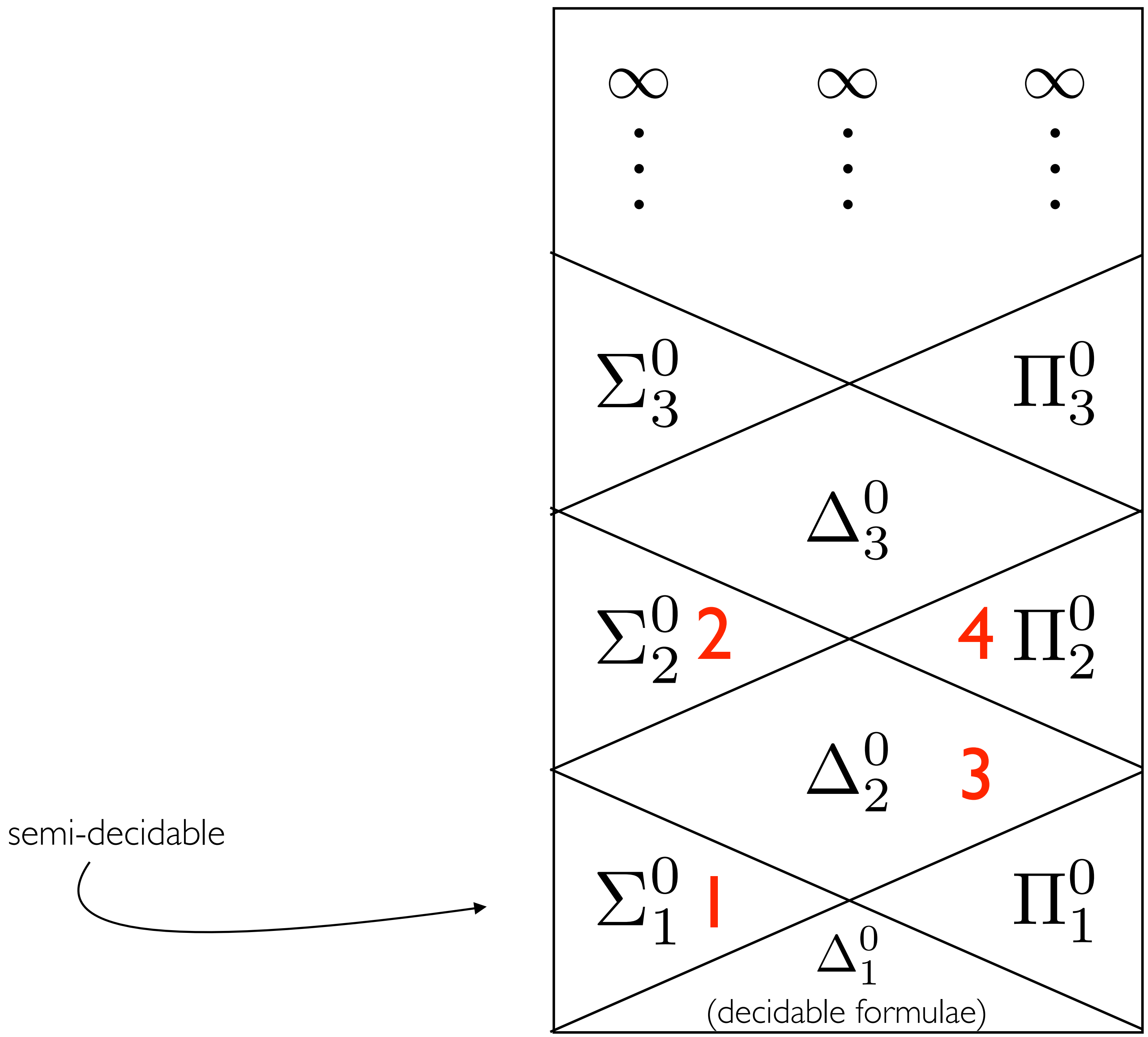
$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

Let R be a Turing-decidable (= decidable, *simpliciter*) dyadic relation. Where is the set:

$$\{x : \exists y R(x, y)\},$$

1 2 3 or 4?

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



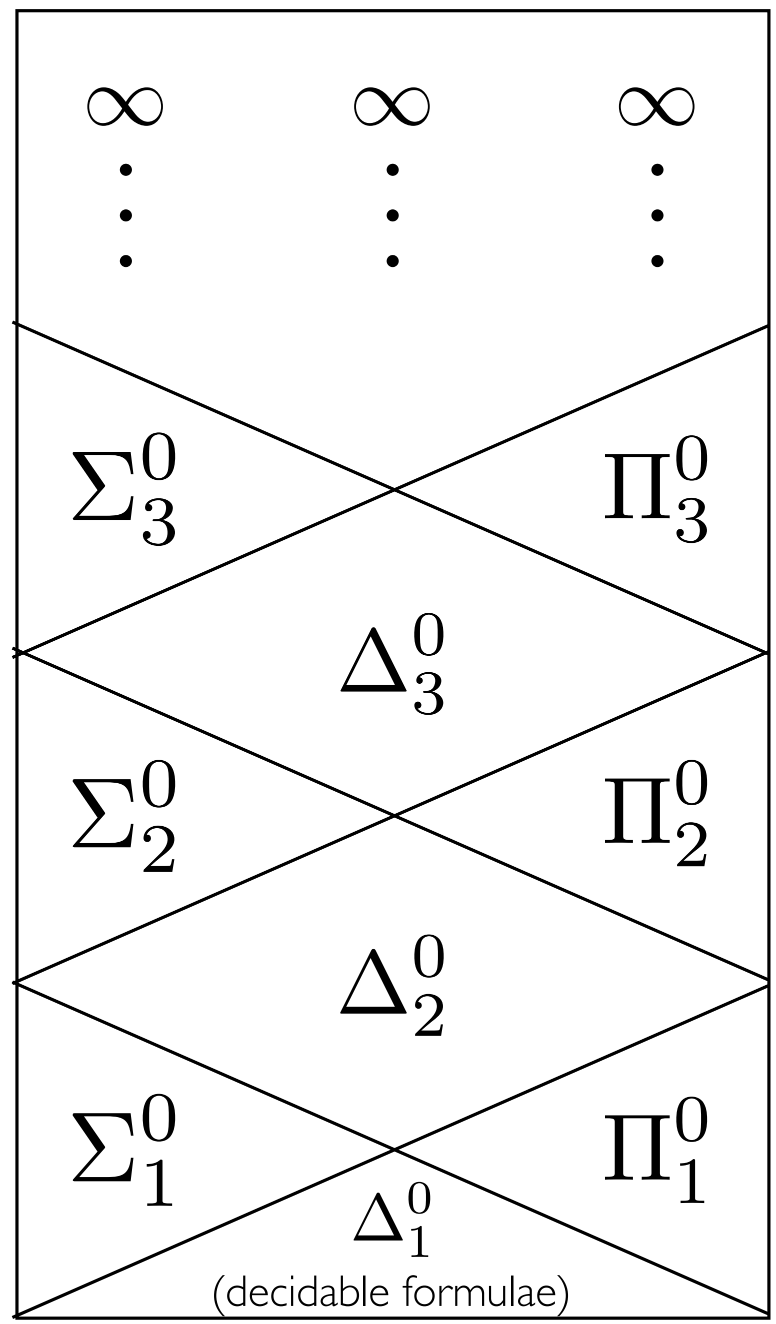
$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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

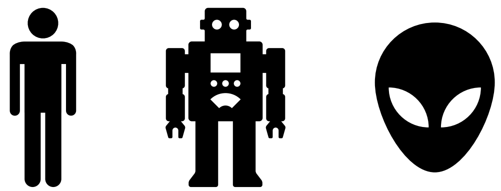
semi-decidable



$$\Delta^0_1 = \Sigma^0_1 \cap \Pi^0_1$$

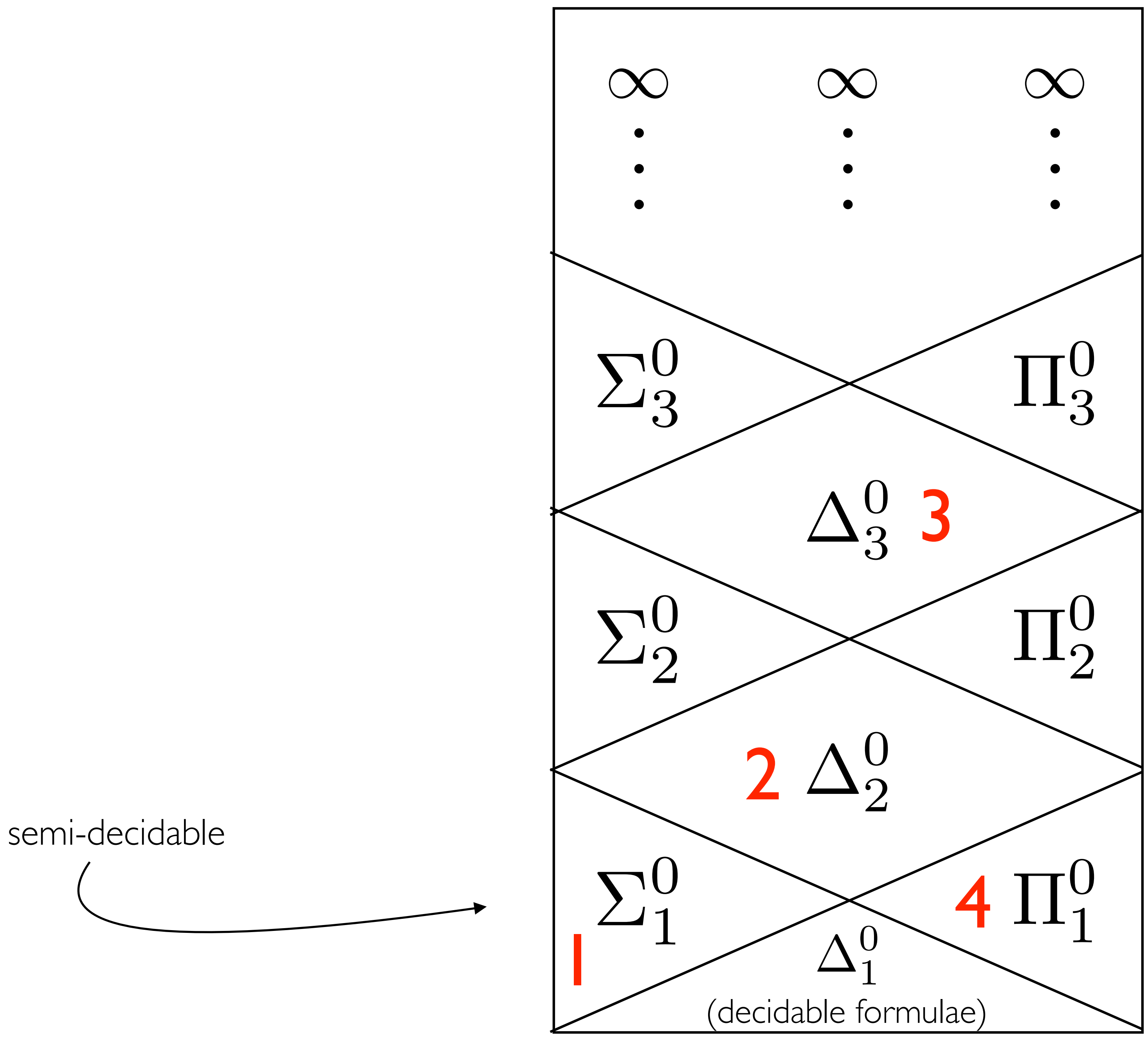
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k(\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k'(\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



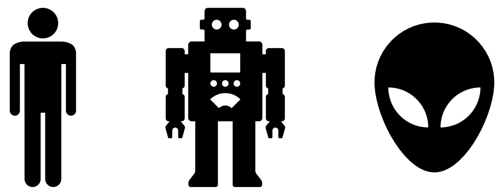
Let R be a Turing-decidable (= decidable, *simpliciter*) dyadic relation. Where is the set:

$$\{x : \forall y R(x, y)\},$$

1 2 3 or 4?

$$\Delta^0_1 = \Sigma^0_1 \cap \Pi^0_1$$

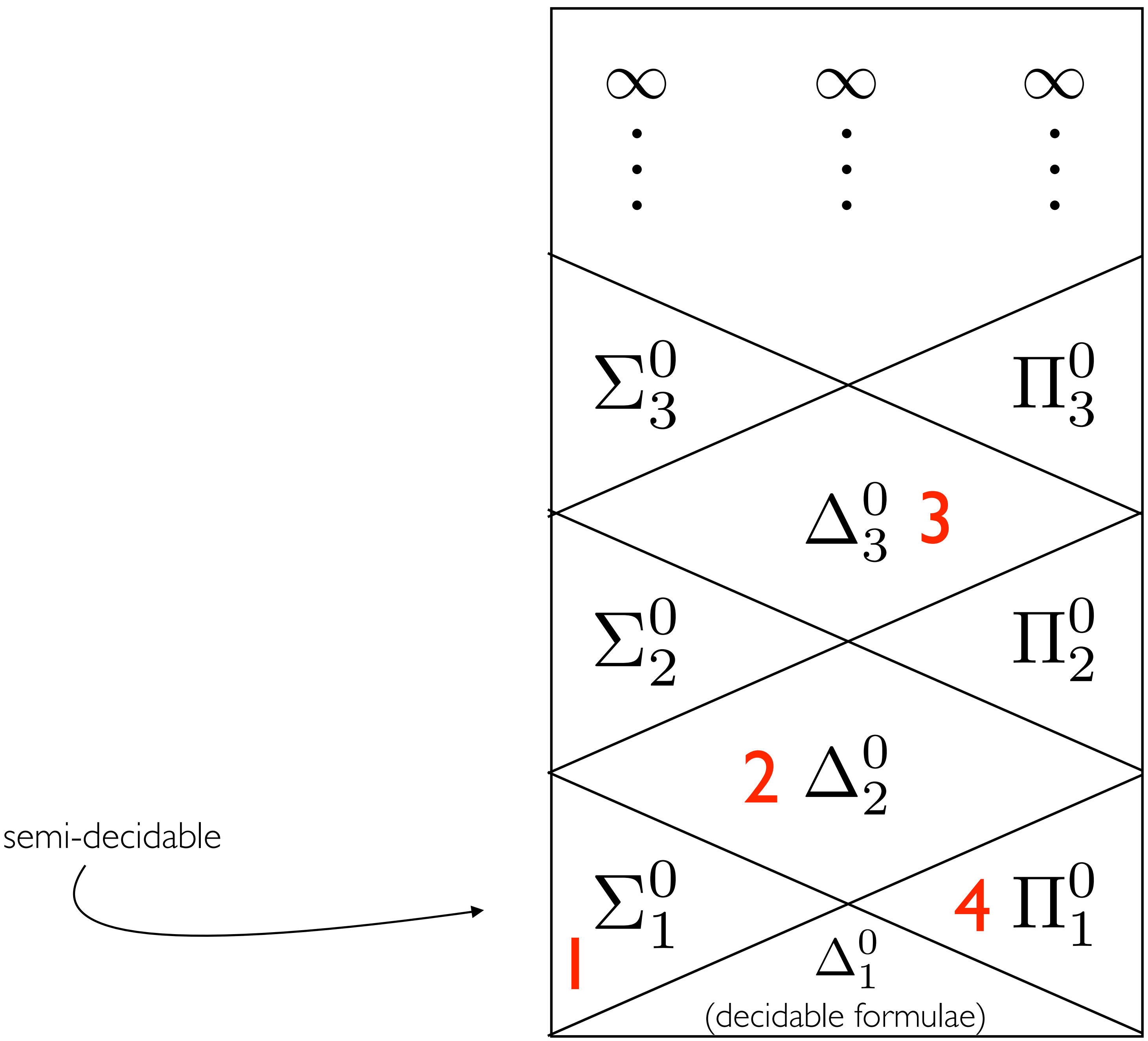
Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

Let R be a Turing-decidable (= decidable, *simpliciter*) dyadic relation. Where is the set:

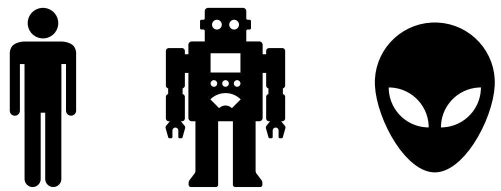
$$\{x : \forall y R(x, y)\},$$

1 2 3 or 4?

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

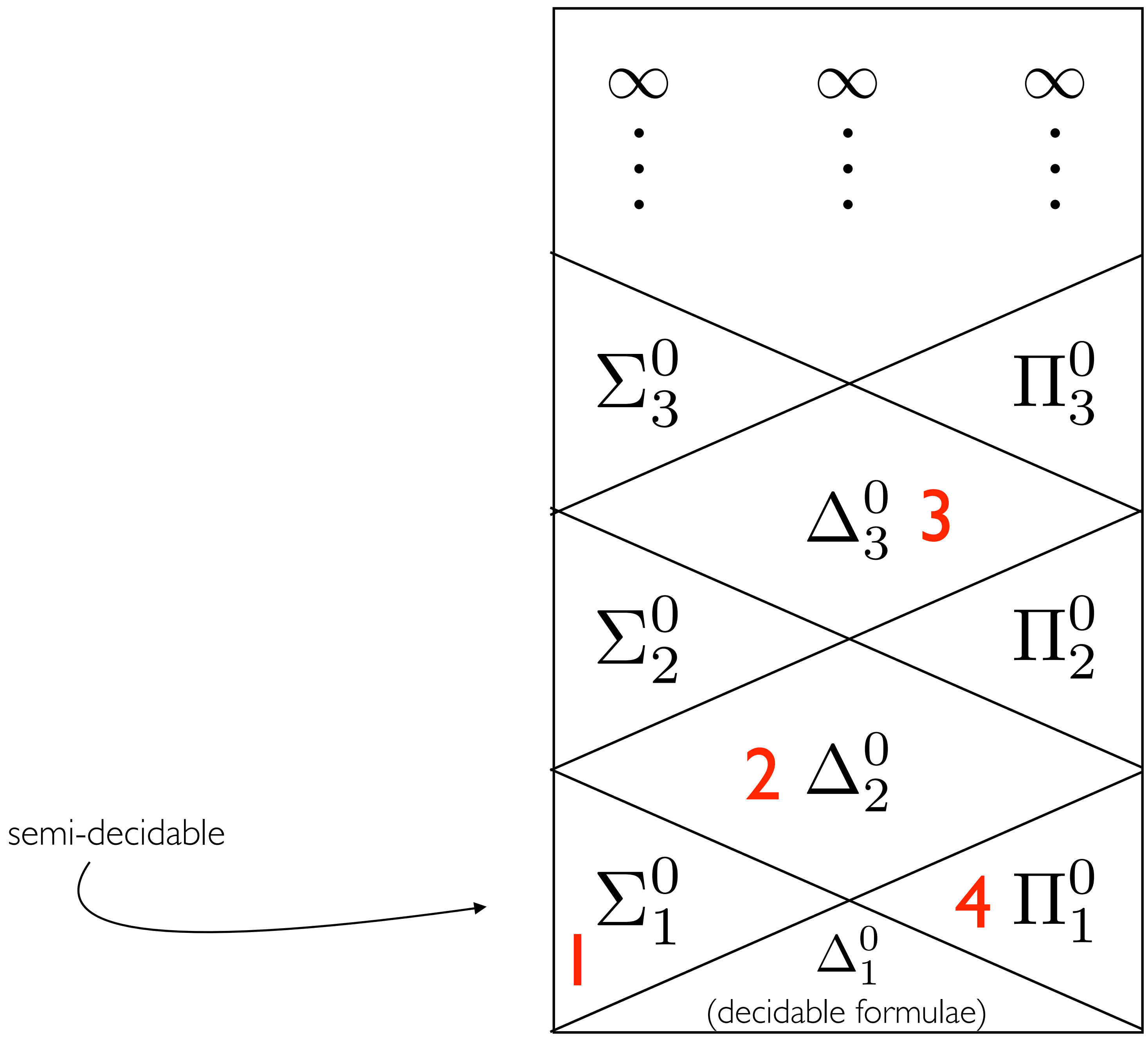
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



Let R be a Turing-decidable (= decidable, *simpliciter*) dyadic relation. Where is the set:

$$\{x : \forall y R(x, y)\},$$

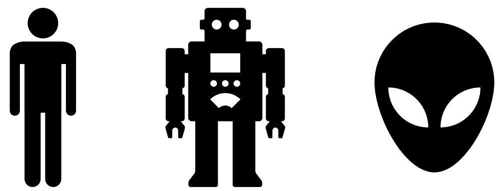
1 2 3 or 4?

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

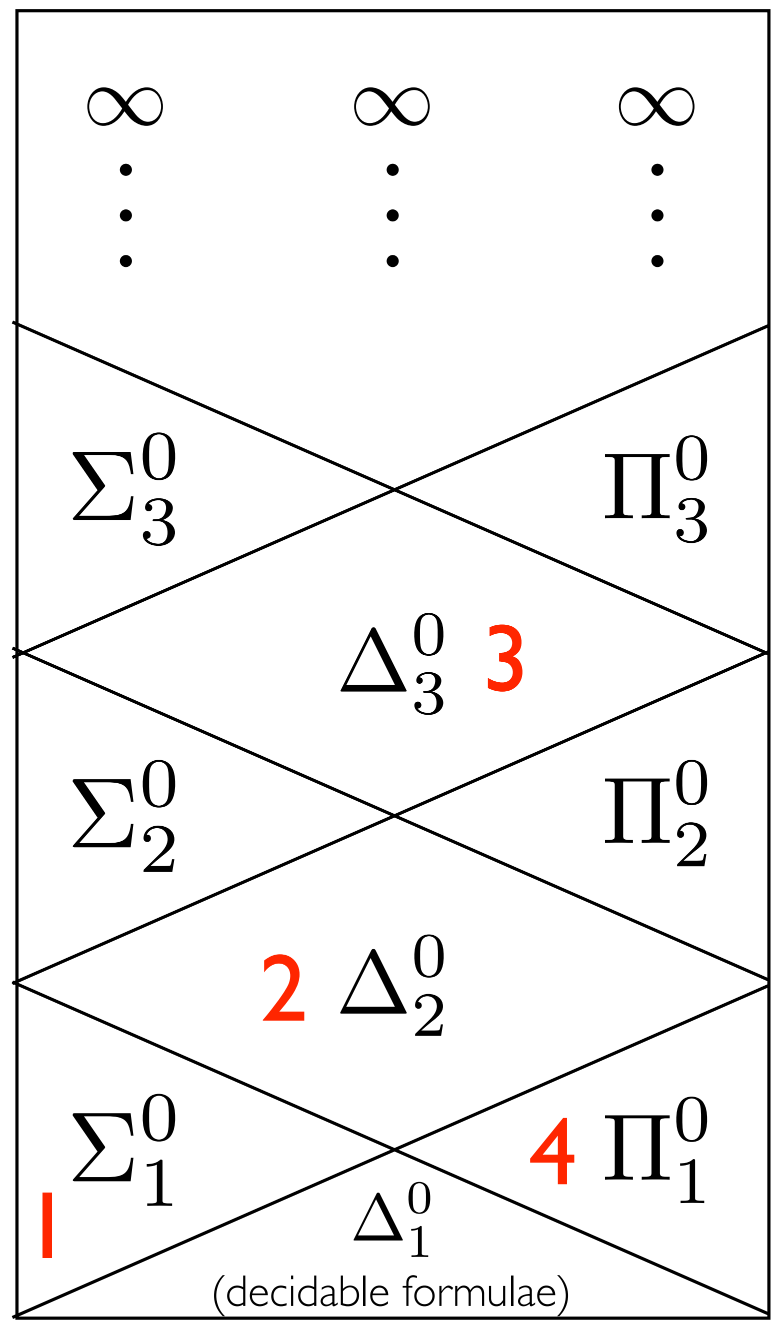
Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

$$\textbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



semi-decidable

Let R be a Turing-decidable (= decidable, *simpliciter*) dyadic relation. Where is the set:

$$\{x : \forall y R(x, y)\},$$

1 2 3 or 4?

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

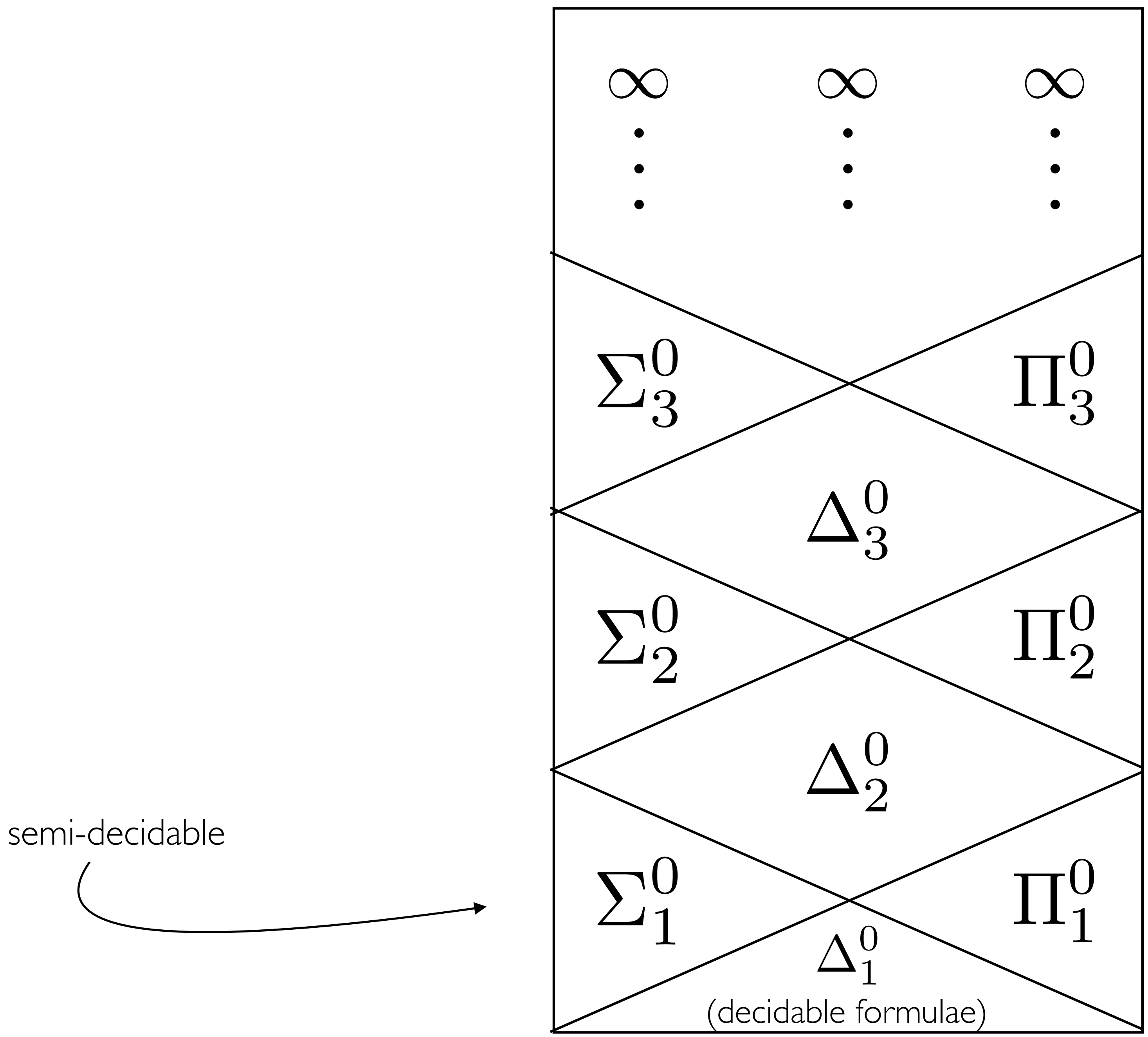
$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

Arithmetic Hierarchy, Part I

Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



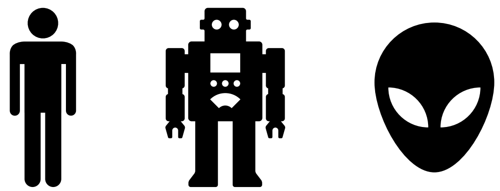
$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

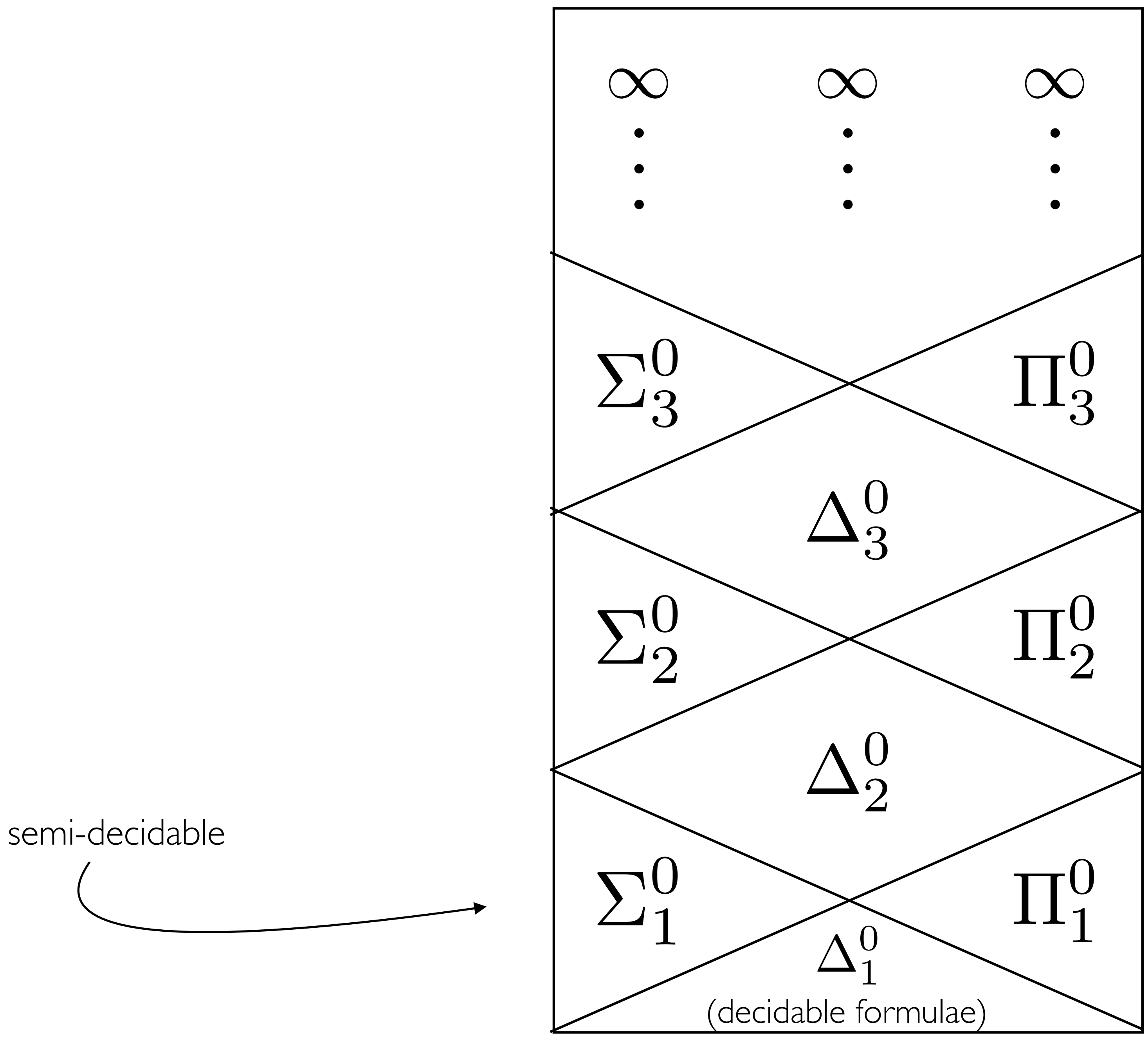
Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k(\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k'(\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

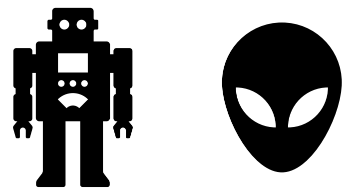
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

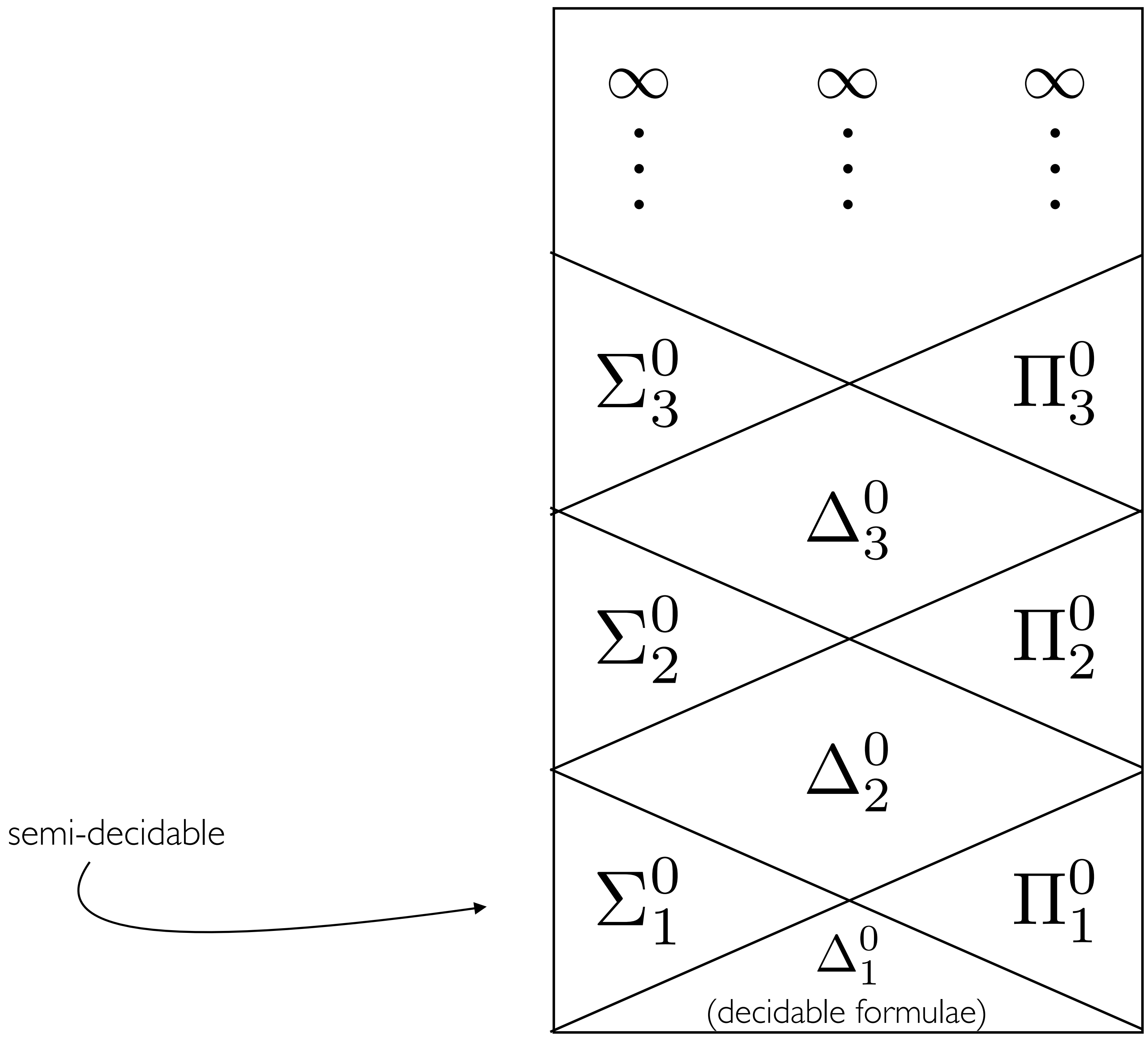
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

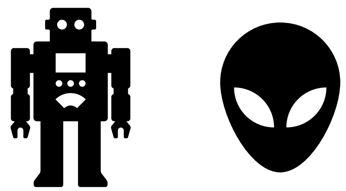
Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

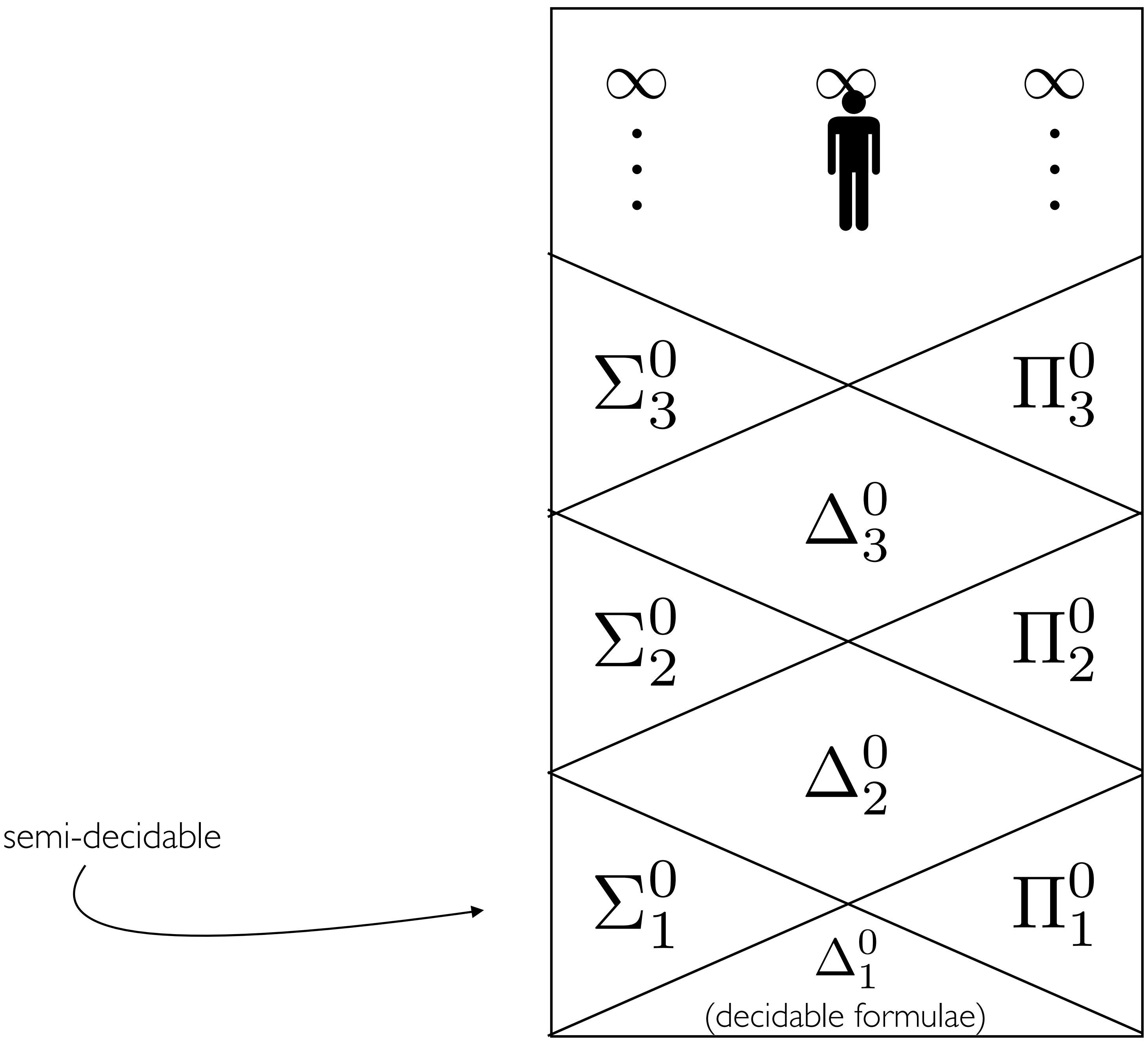
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

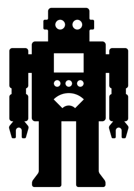
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

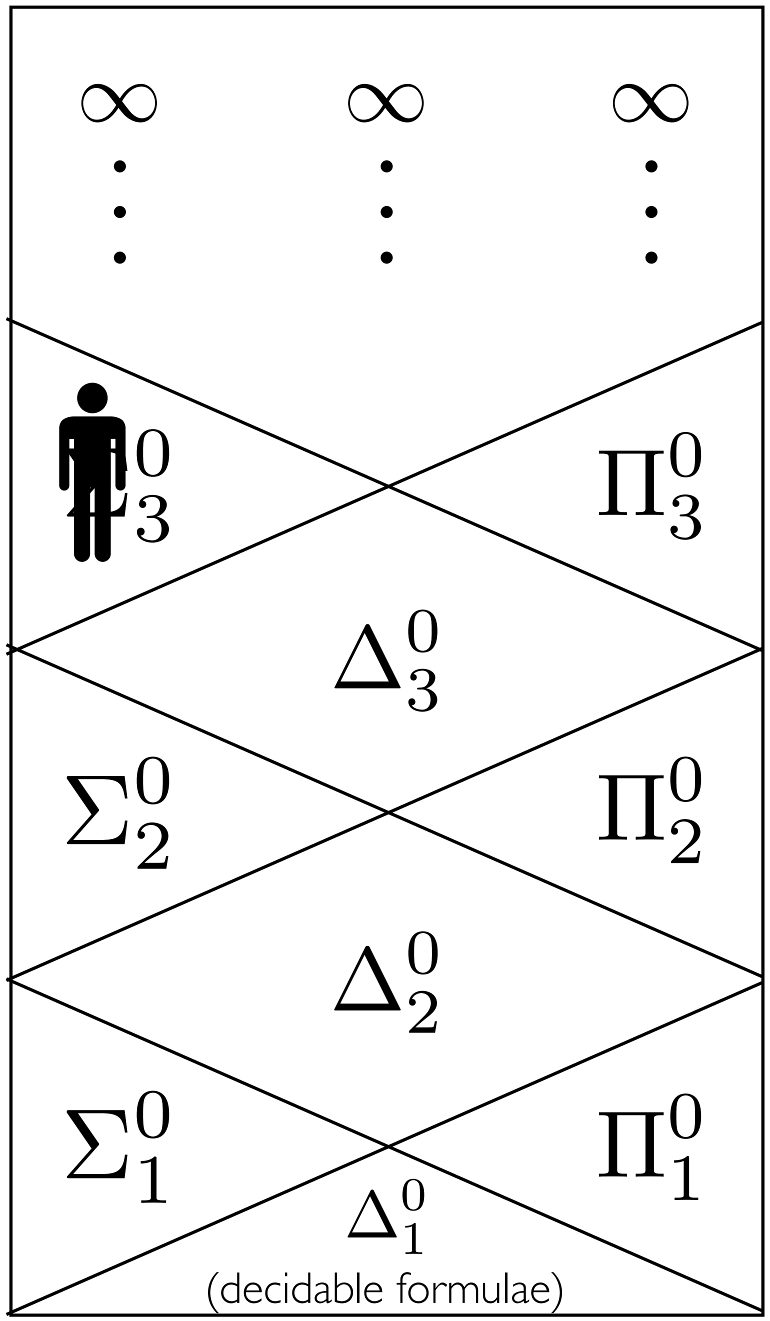
Arithmetic Hierarchy, Part I




Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k(\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k'(\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

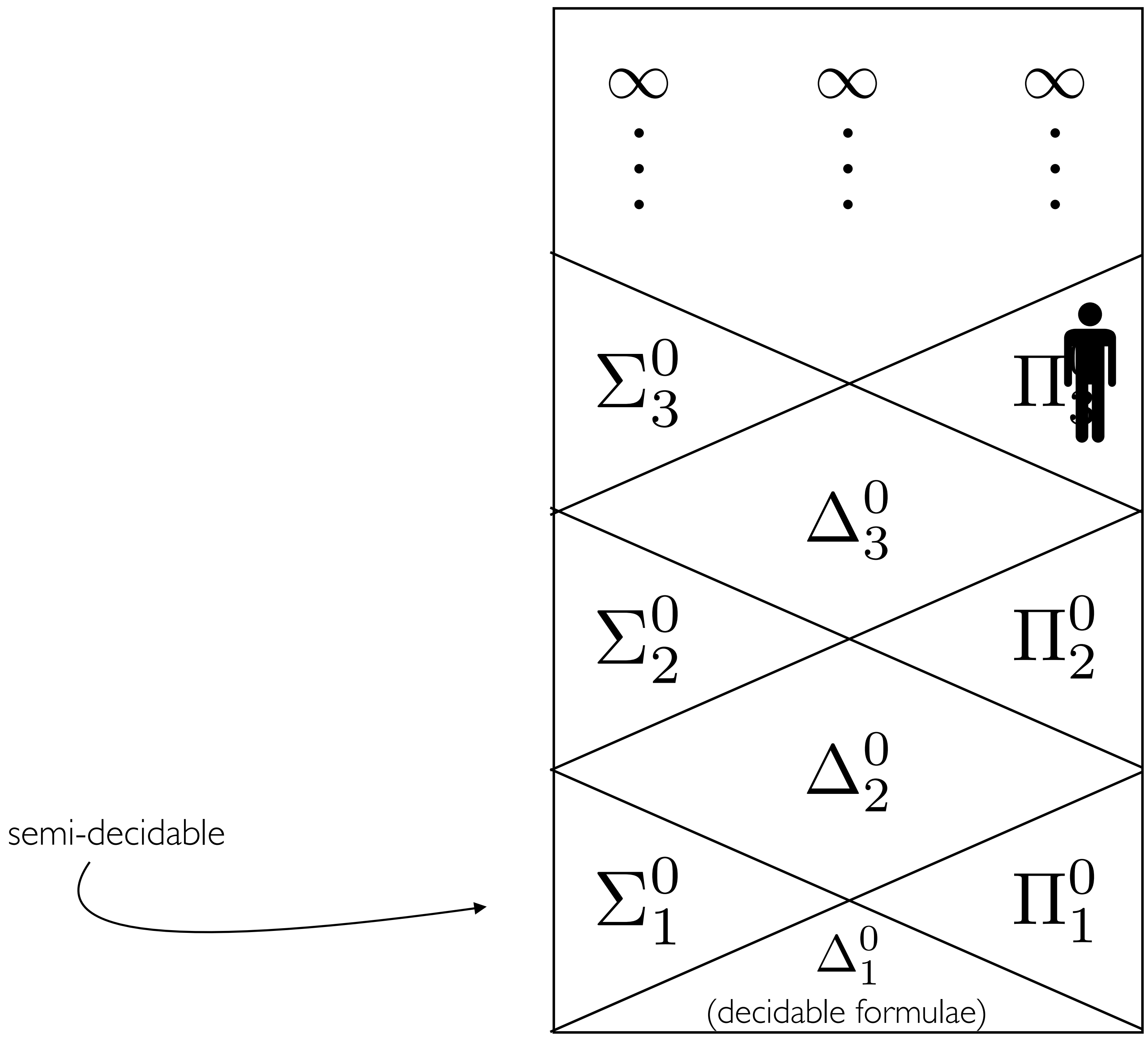
$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

Arithmetic Hierarchy, Part I

Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

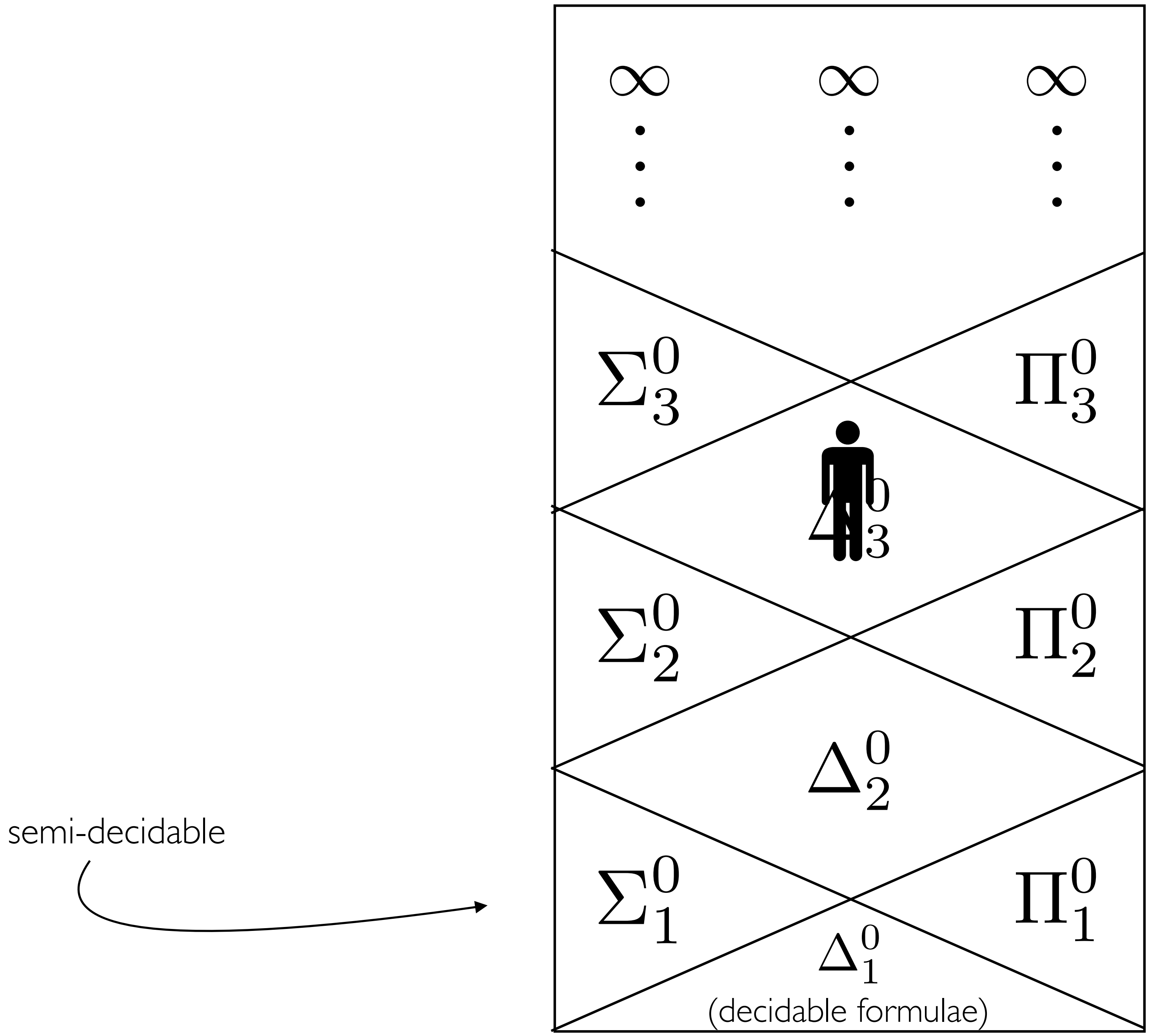
From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

Arithmetic Hierarchy, Part I

Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k(\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k'(\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

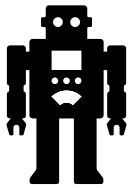
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

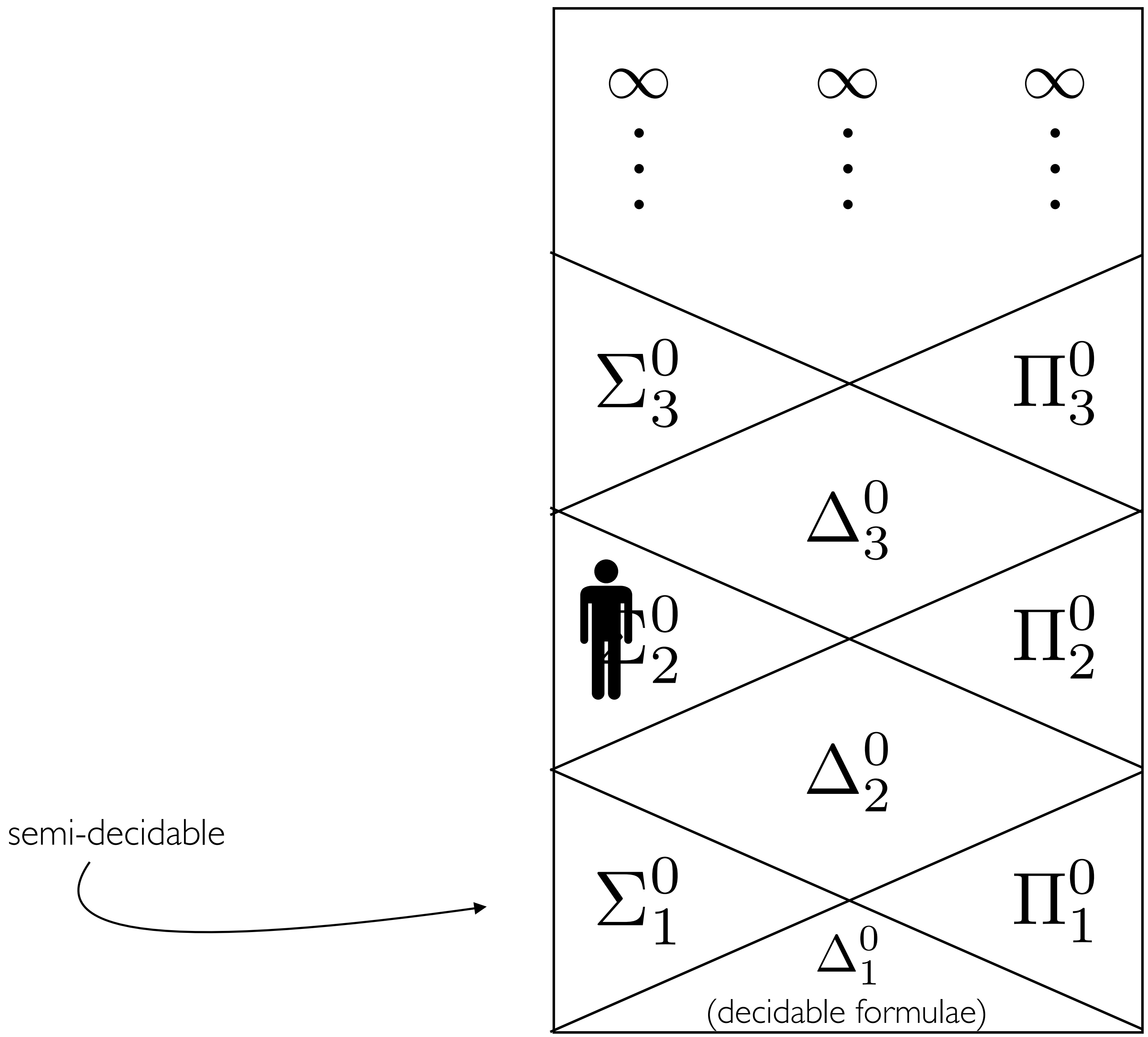
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

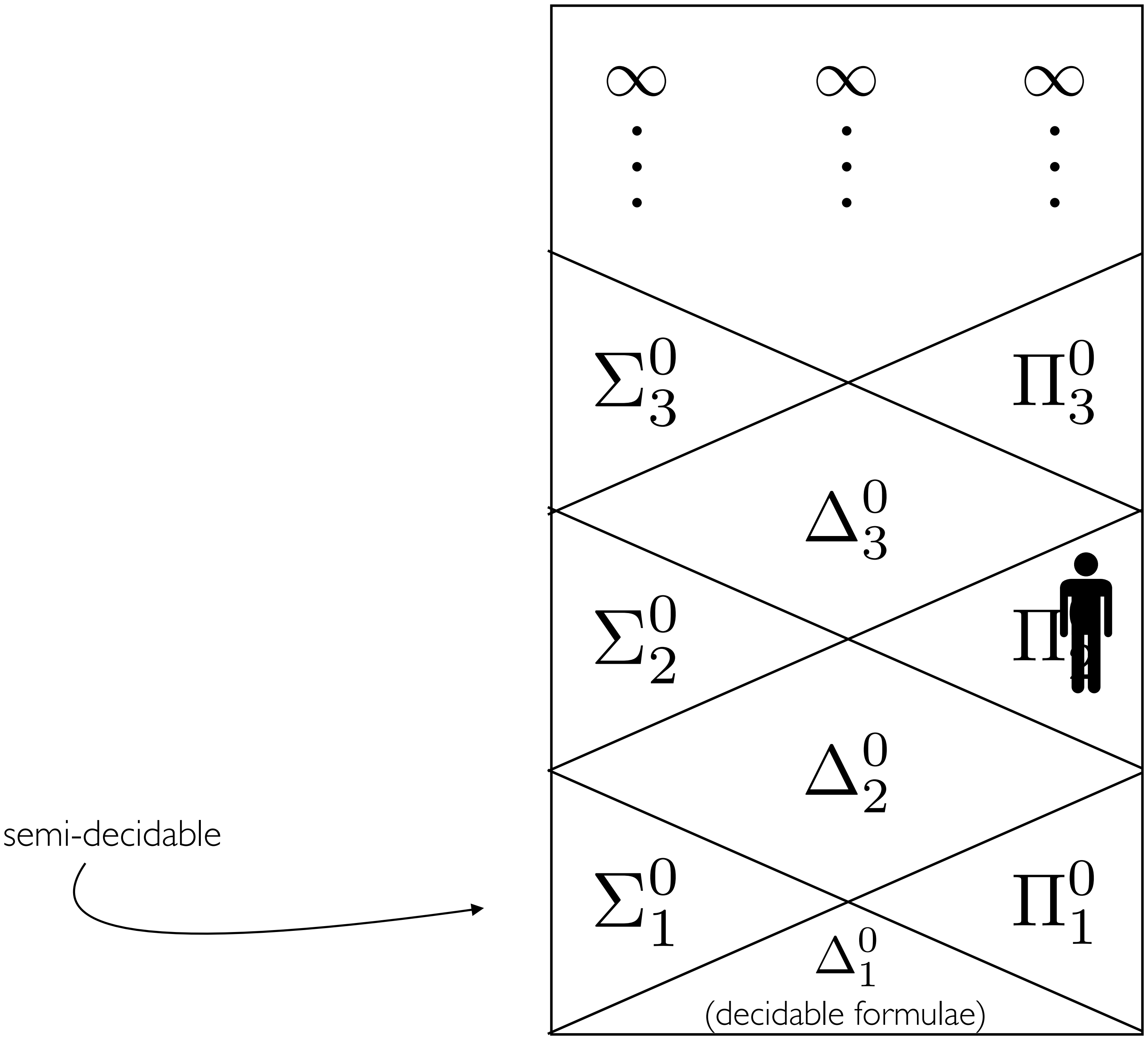
From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

Arithmetic Hierarchy, Part I

Can you see the carryover from PH?

$$\textbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v[\exists k(\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k'(\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

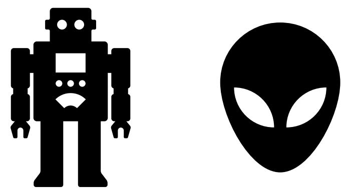
Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

Arithmetic Hierarchy, Part I

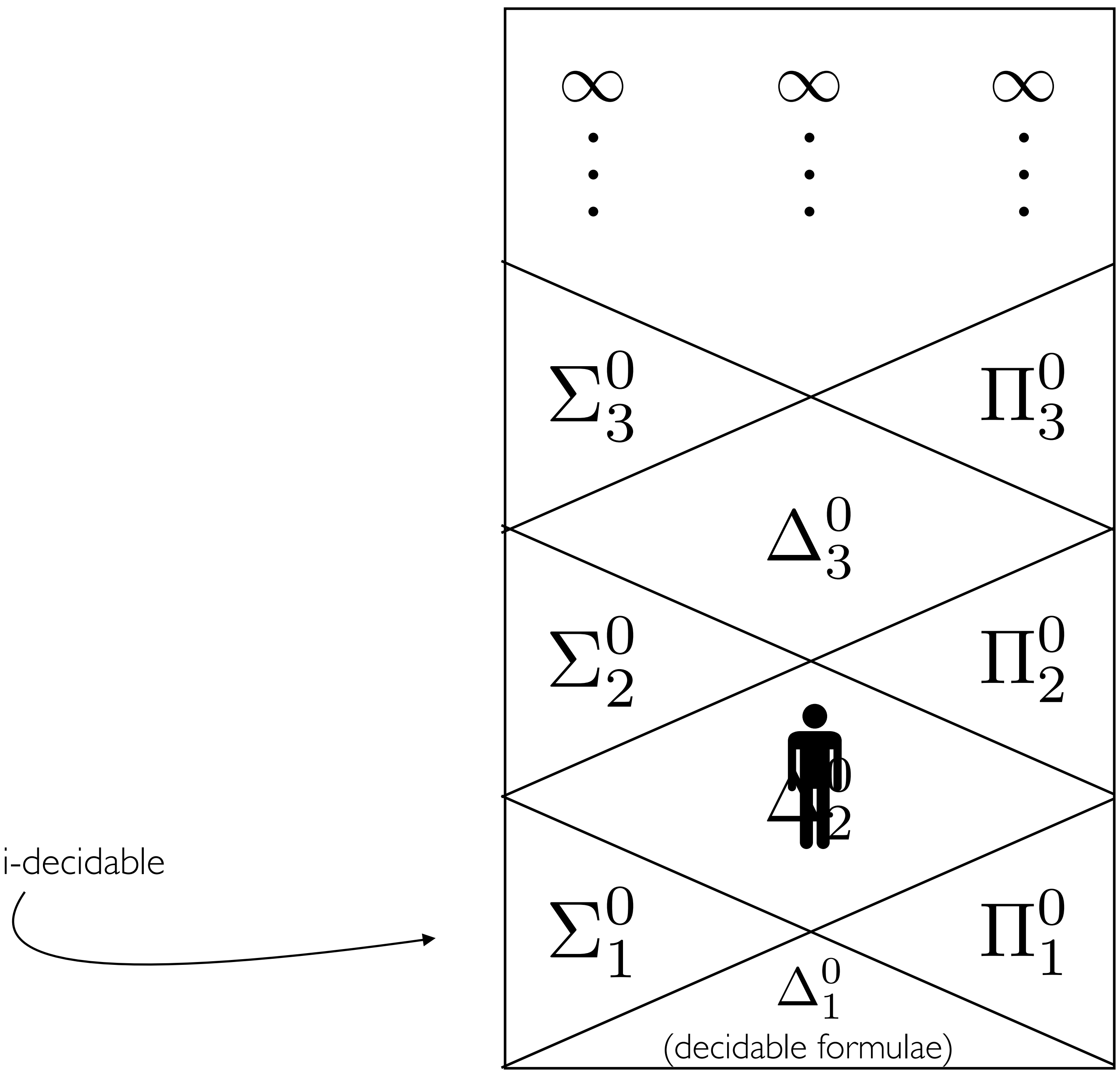
Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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

semi-decidable



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

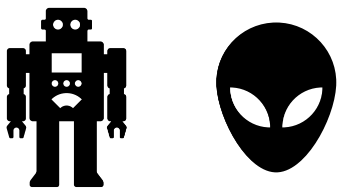
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

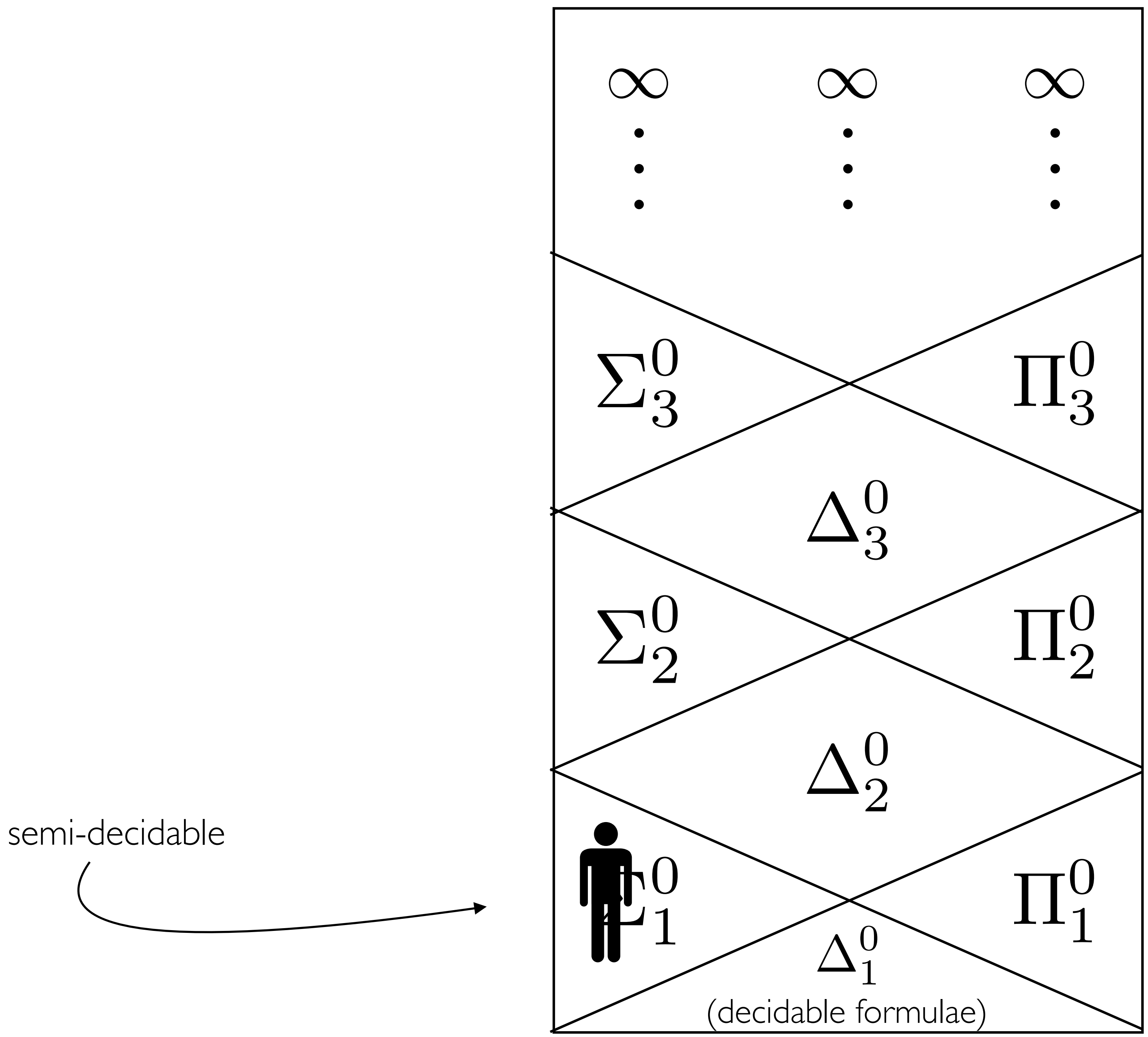
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

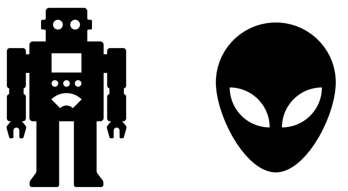
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! ...

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

Arithmetic Hierarchy, Part I

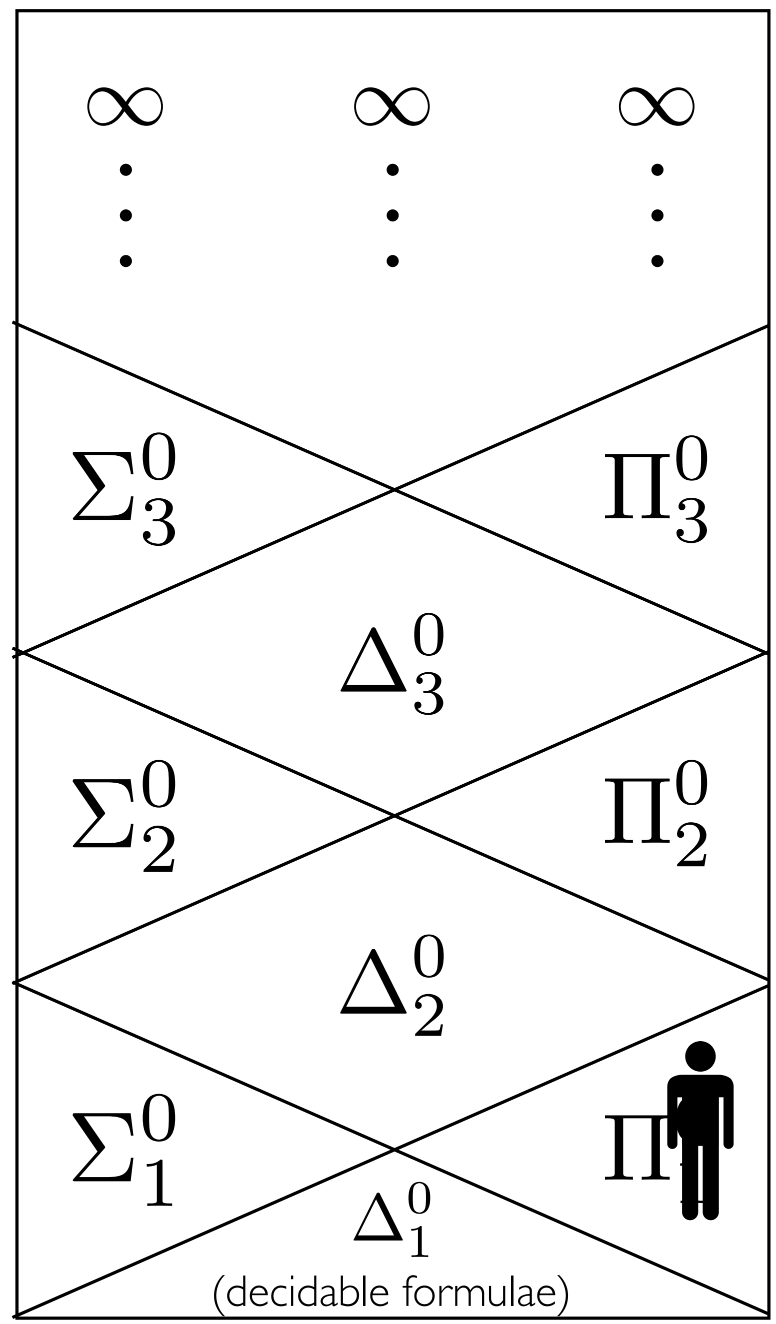


Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v[\exists k(\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k'(\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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

semi-decidable



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

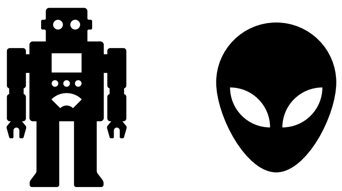
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

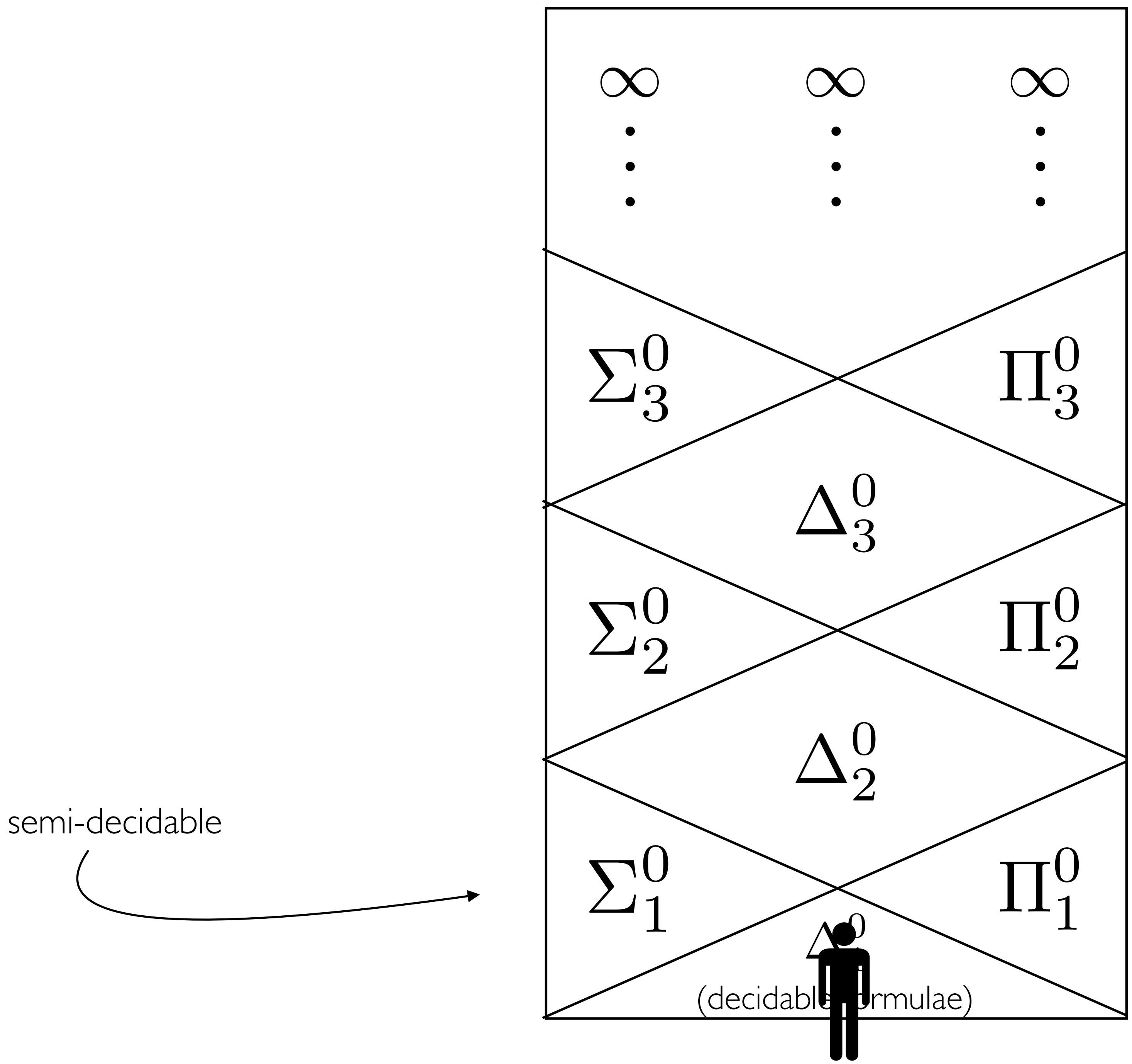
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

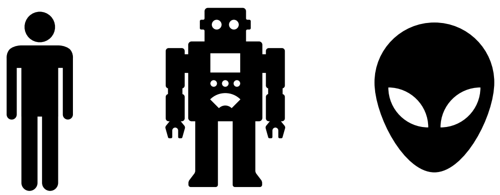
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

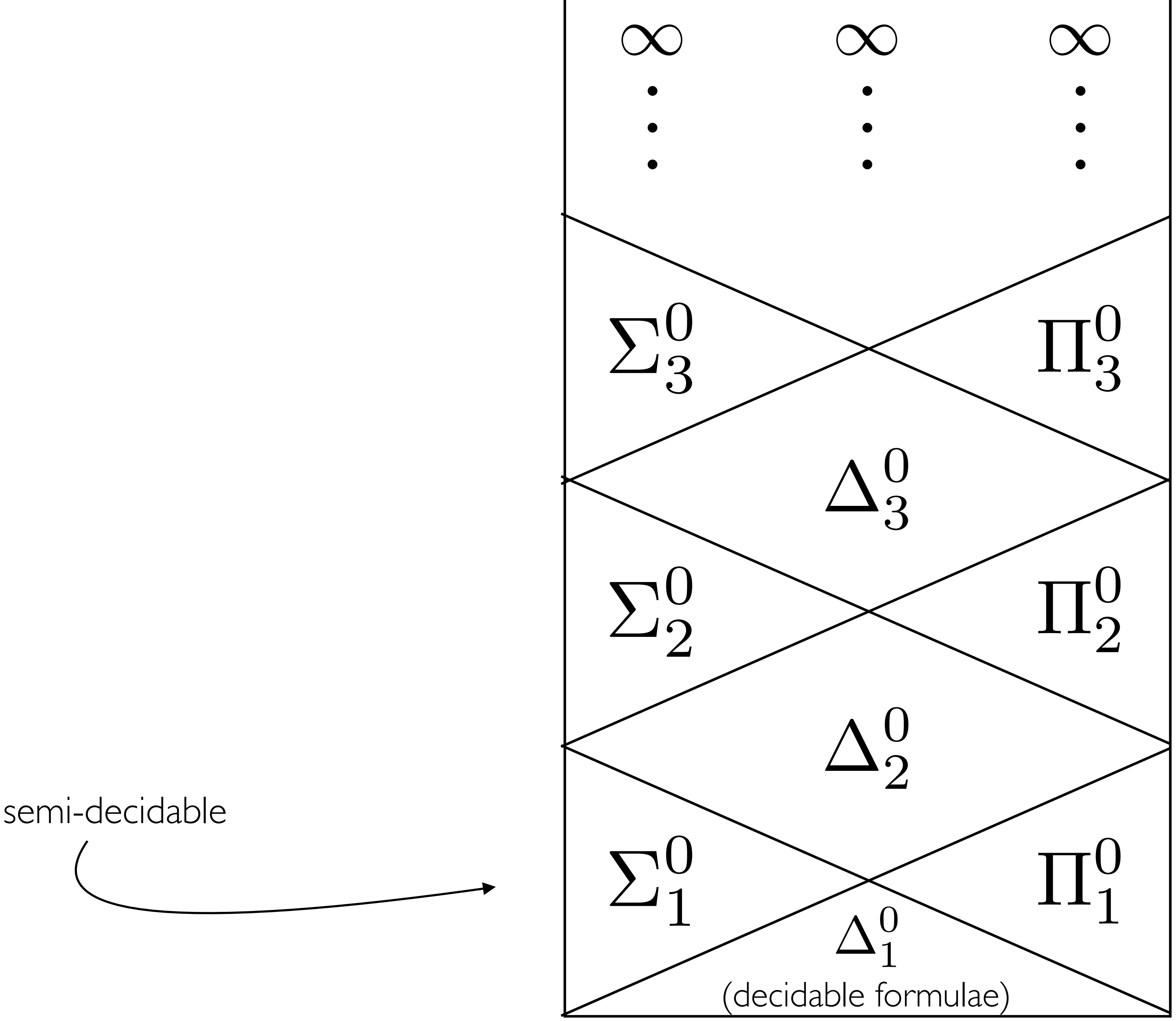
Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

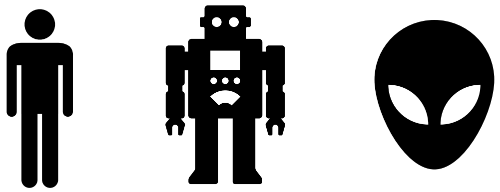
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

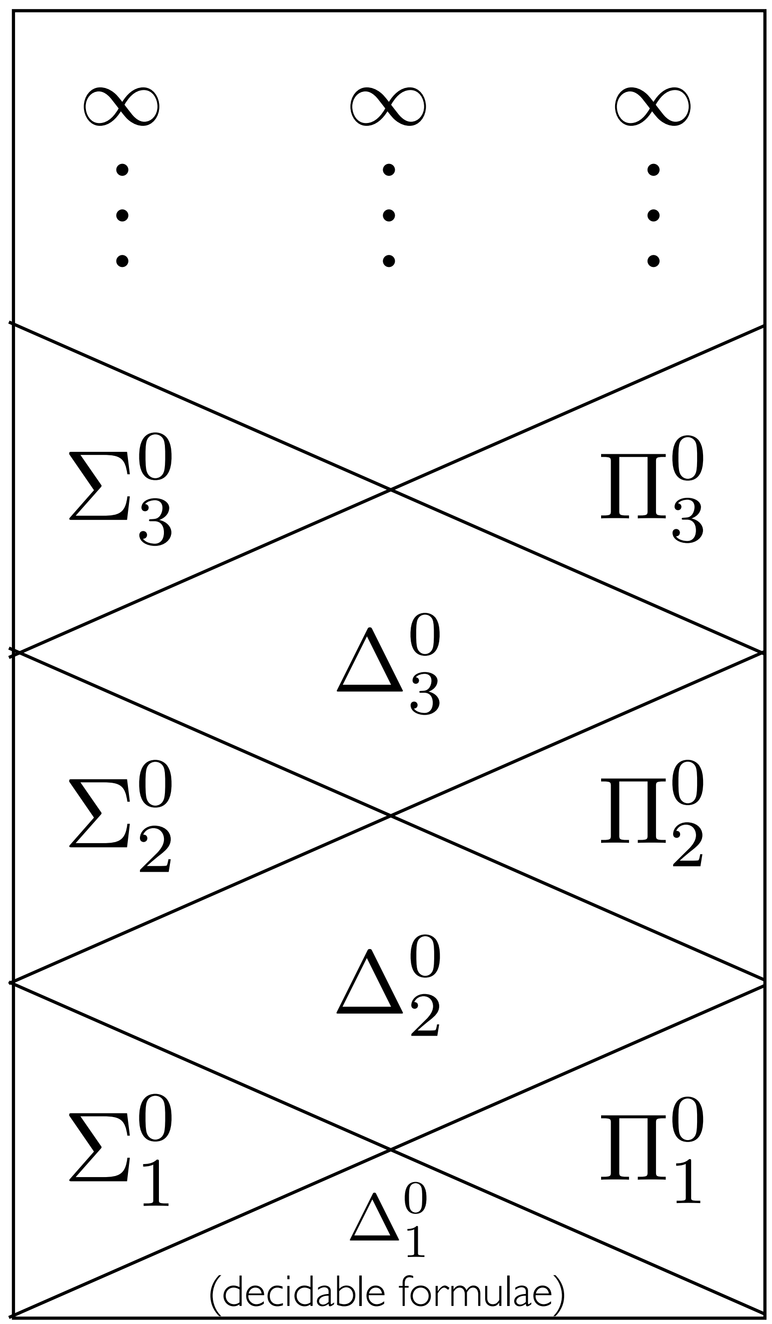
Arithmetic Hierarchy, Part I

Can you see the carryover from PH?



$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



semi-decidable

$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

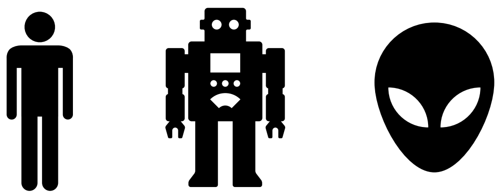
$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

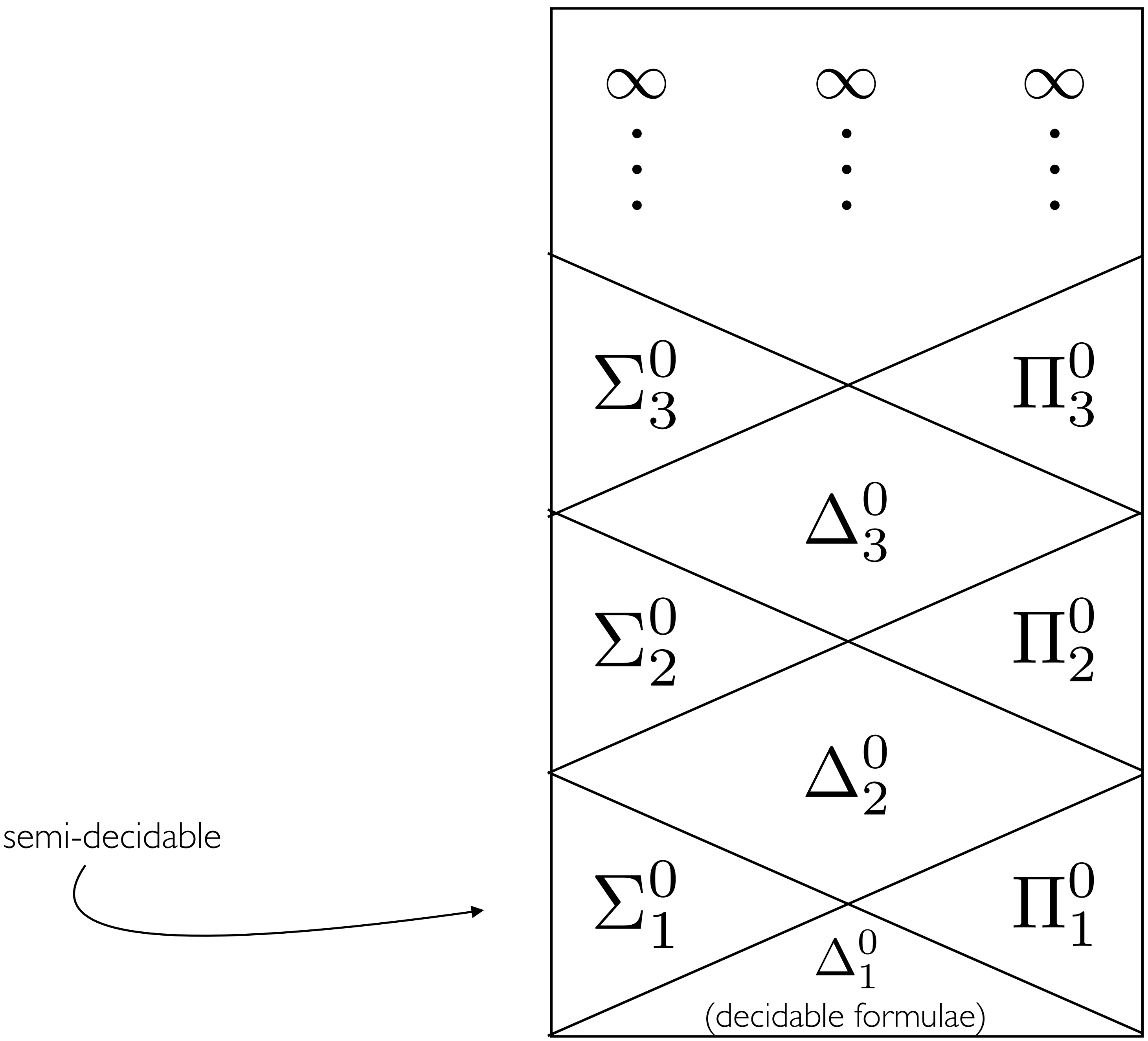
Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u\rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u\rangle : v, k')]$$

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



$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

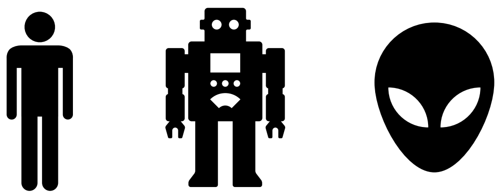
$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

$$\Delta^0_1 = \Sigma^0_1 \cap \Pi^0_1$$

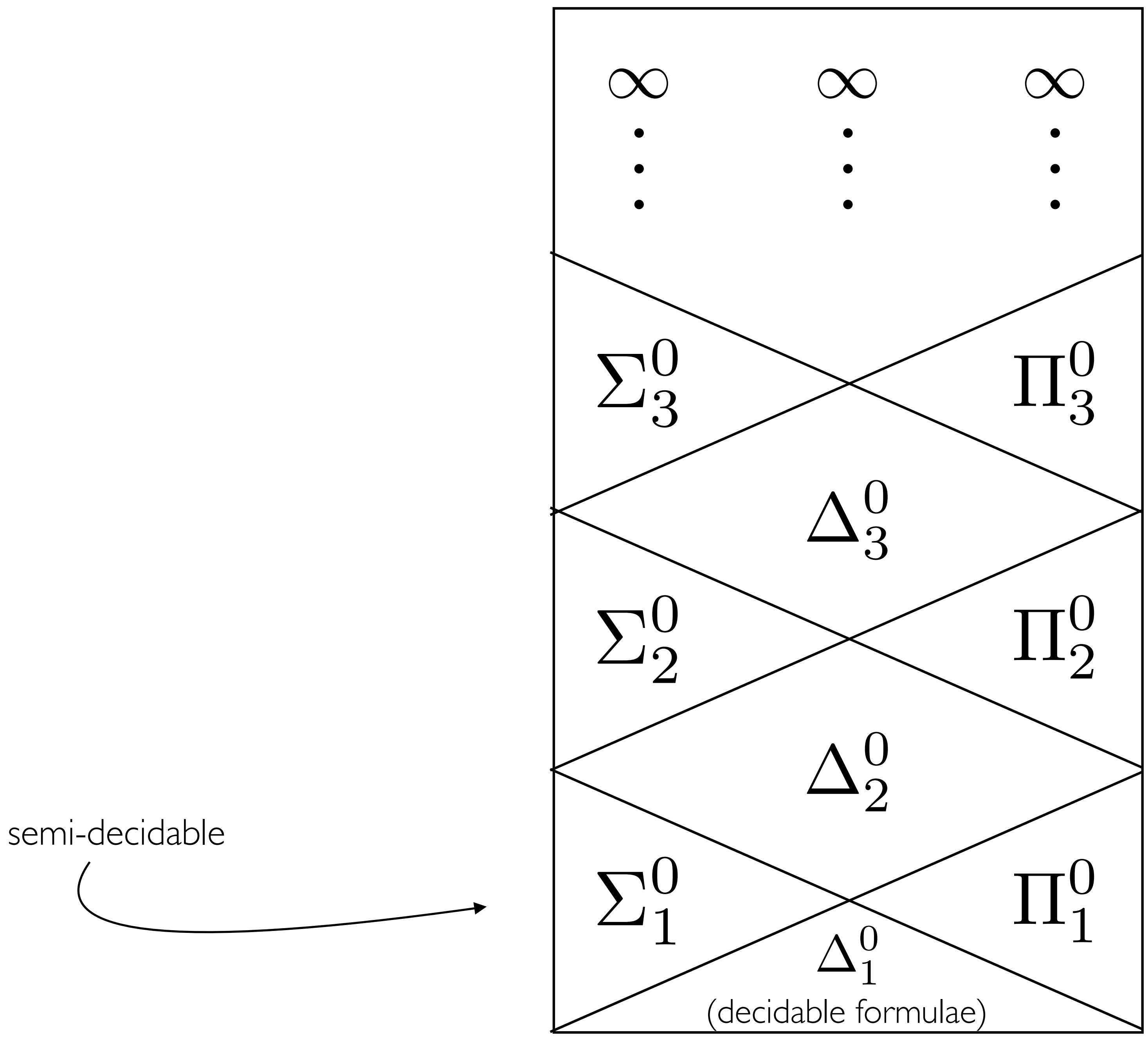
Arithmetic Hierarchy, Part I



Can you see the carryover from PH?

$$\mathbf{2SAMEFUNC} := \{\mathfrak{m}_1, \mathfrak{m}_2 : \forall u \forall v [\exists k (\langle \mathfrak{m}_1, u \rangle : v, k \leftrightarrow \exists k' (\langle \mathfrak{m}_2, u \rangle : v, k')]$$

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



$$\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$$

$$x \in \Sigma_i \text{ iff } \exists R\ \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \forall \text{ if } i \text{ even}; Q_i = \exists \text{ if } i \text{ odd})$$

$$x \in \Pi_i \text{ iff } \exists R\ \forall y_1 \exists y_2 \cdots Q_i y_i R(x, y_1, y_2, \ldots, y_i)$$
$$(Q_i = \exists \text{ if } j \text{ even}; Q_i = \forall \text{ if } j \text{ odd})$$

Try your hand at classifying! …

From Kleene: The set to be classified, $\mathscr{K}$, consists of all those inputs to a given Turing machine $\mathfrak{m}$ that results in this machine halting after some number of steps.

The set to be classified is the set of all pairs of programs P_1 and P_2 s.t. both compute exactly the same functions.

Arithmetic Hierarchy, Part I

What about …
computing over $\mathbb{R}$?!

What about … computing over $\mathbb{R}$?!

I

Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

Theoretical Computer Science 374 (2007) 277–290

Theoretical Computer Science

www.elsevier.com/locate/tcs

A new conceptual framework for analog computation

Jerzy Mycka[a], José Félix Costa[b,*]

[a] *Institute of Mathematics, University of Maria Curie-Skłodowska, Lublin, Poland*
[b] *Department of Mathematics, I.S.T., Universidade Técnica de Lisboa, Lisboa, Portugal*

Received 3 February 2005; received in revised form 26 December 2006; accepted 15 January 2007

Communicated by F. Cucker

Abstract

In this paper we show how to explore the classical theory of computability using the tools of Analysis: A differential scheme is substituted for the classical recurrence scheme and a limit operator is substituted for the classical minimization. We show that most relevant problems of computability over the non-negative integers can be dealt with over the reals: elementary functions are computable, Turing machines can be simulated, the hierarchy of non-computable functions can be represented (the classical halting problem being solvable at some level). The most typical concepts in Analysis become natural in this framework. The most relevant question is posed: Can we solve open problems of classical computability and computational complexity using, as Popper says, the toolbox of Analysis?

Keywords: Recursive function theory over the reals; Analog computation; Dynamical systems; Dynamical systems capable of universal computation

What about … computing over $\mathbb{R}$?!

||

arXiv.org > math > arXiv:math/9808093

Search... | All fields | Search

Help | Advanced Search

Mathematics > Logic

[Submitted on 21 Aug 1998]

Infinite Time Turing Machines

Joel David Hamkins, Andy Lewis

We extend in a natural way the operation of Turing machines to infinite ordinal time, and investigate the resulting supertask theory of computability and decidability on the reals. The resulting computability theory leads to a notion of computation on the reals and concepts of decidability and semi-decidability for sets of reals as well as individual reals. Every Pi^1_1 set, for example, is decidable by such machines, and the semi-decidable sets form a portion of the Delta^1_2 sets. Our oracle concept leads to a notion of relative computability for reals and sets of reals and a rich degree structure, stratified by two natural jump operators.

Comments:	57 pages, 4 figures, to appear in the Journal of Symbolic Logic
Subjects:	**Logic (math.LO)**
MSC classes:	03D30; 03D60
Cite as:	arXiv:math/9808093 **[math.LO]**
	(or arXiv:math/9808093v1 **[math.LO]** for this version)

Submission history

From: Joel David Hamkins [view email]

[v1] Fri, 21 Aug 1998 02:52:43 UTC (148 KB)

Download:

- PDF
- PostScript
- Other formats

(license)

Current browse context:
math

< prev | next >

new | recent | 9808

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

5 blog links (what is this?)

Export Bibtex Citation

Bookmark

What about … computing over $\mathbb{R}$?!

.org > math > arXiv:math/9808093

Search... All fie

Help | Advanced Search

thematics > Logic

mitted on 21 Aug 1998]

finite Time Turing Machines

David Hamkins, Andy Lewis

We extend in a natural way the operation of Turing machines to infinite ordinal time, and investigate the resulting supertask theory of computability and decidability on the reals. The resulting computability theory leads to a notion of computation on the reals and concepts of decidability and semi-decidability for sets of reals as well as individual reals. Every Pi^1_1 set, for example, is decidable by such machines, and the semi-decidable sets form a portion of the Delta^1_2 sets. Our oracle concept leads to a notion of relative computability for reals and sets of reals and a rich degree structure, stratified by two natural jump operators.

nments:	57 pages, 4 figures, to appear in the Journal of Symbolic Logic
jects:	**Logic (math.LO)**
classes:	03D30; 03D60
as:	arXiv:math/9808093 **[math.LO]**
	(or arXiv:math/9808093v1 **[math.LO]** for this version)

bmission history

n: Joel David Hamkins [view email]

Fri, 21 Aug 1998 02:52:43 UTC (148 KB)

Dow

- PDF
- Post
- Othe

(license)

Current
math

< prev

new | rec

Referen

- NASA
- Goog
- Sema

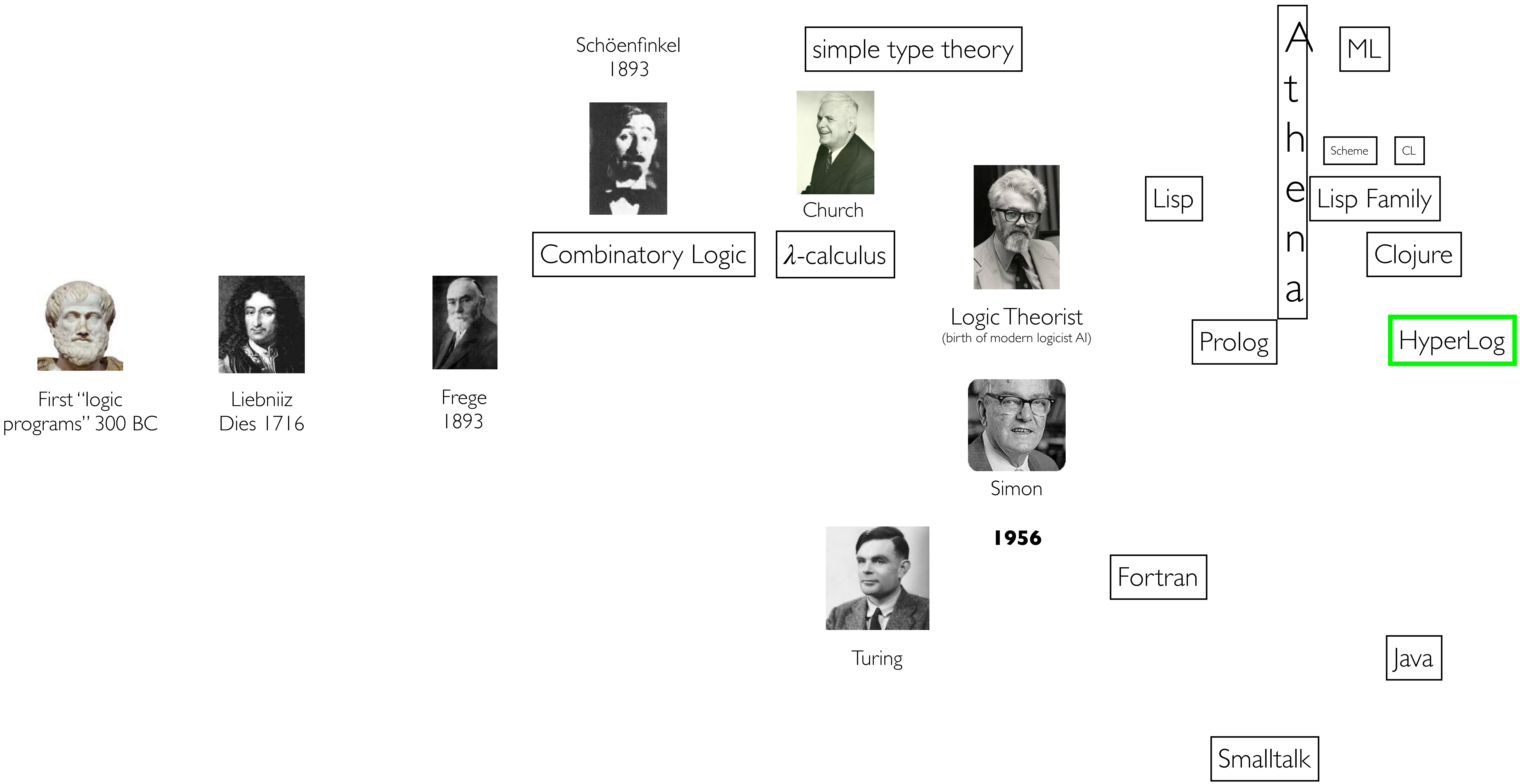
5 blog

Export

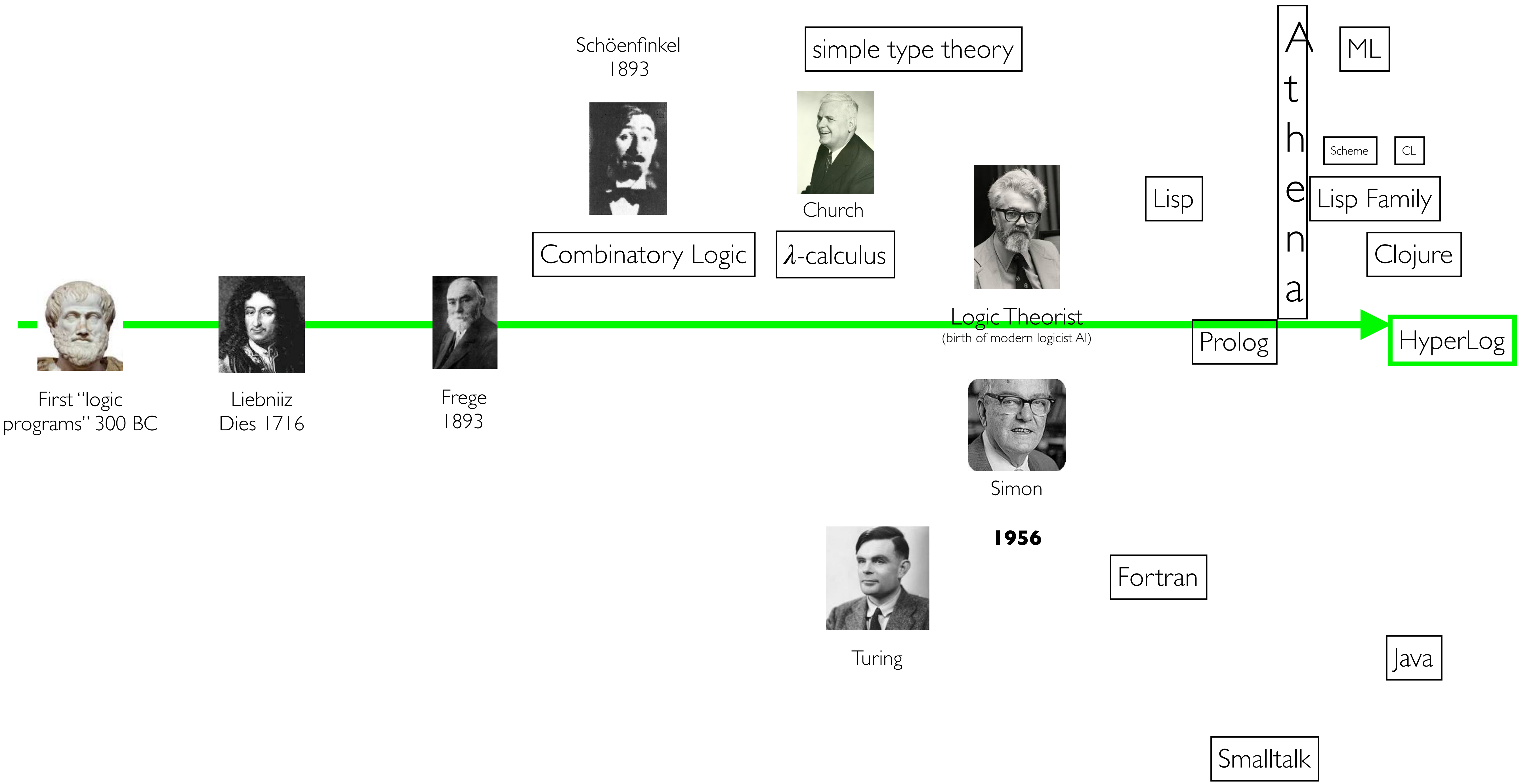
Bookm

Introducing HyperLog .5 …

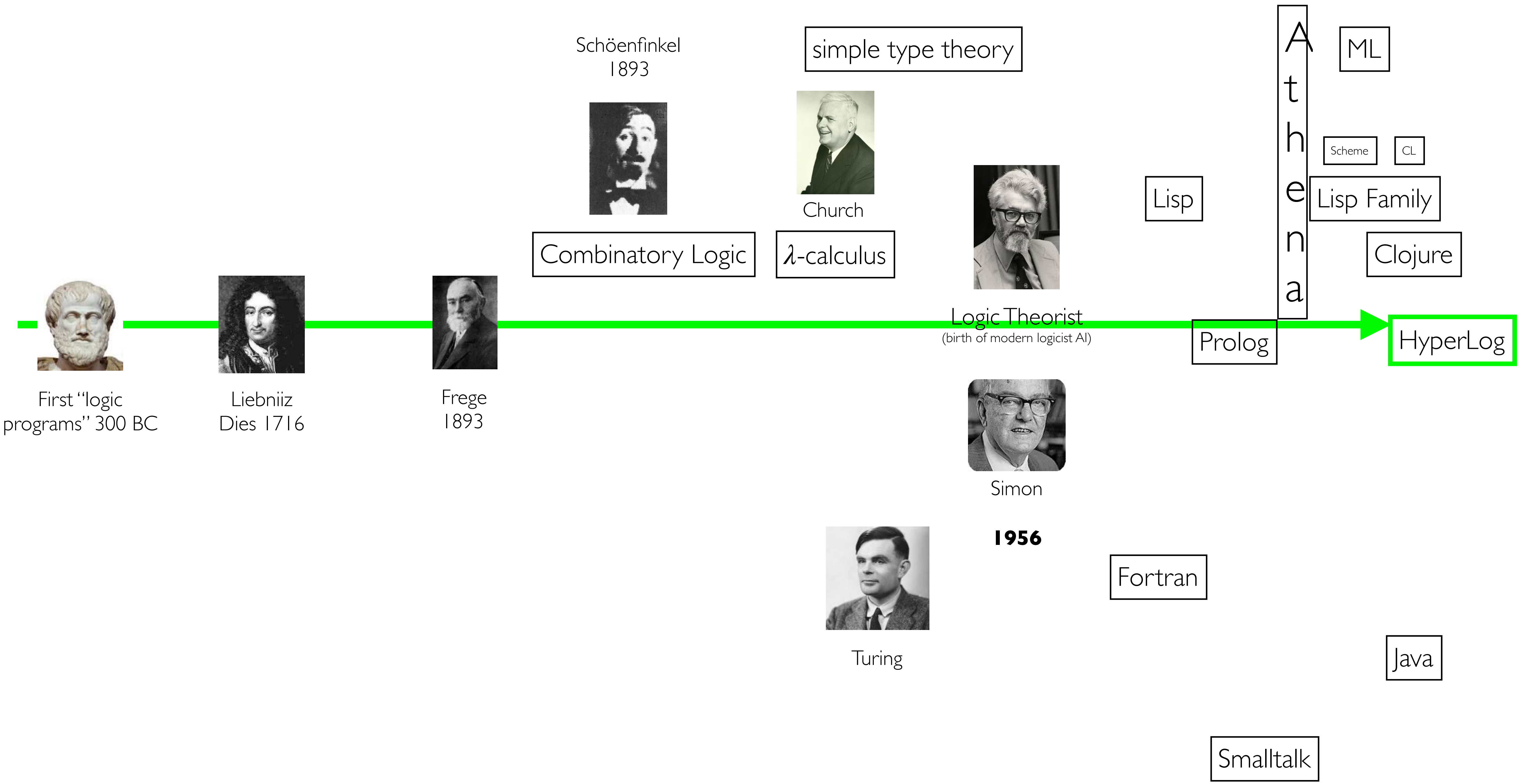
HyperLog: Historico-logico-programming Landscape



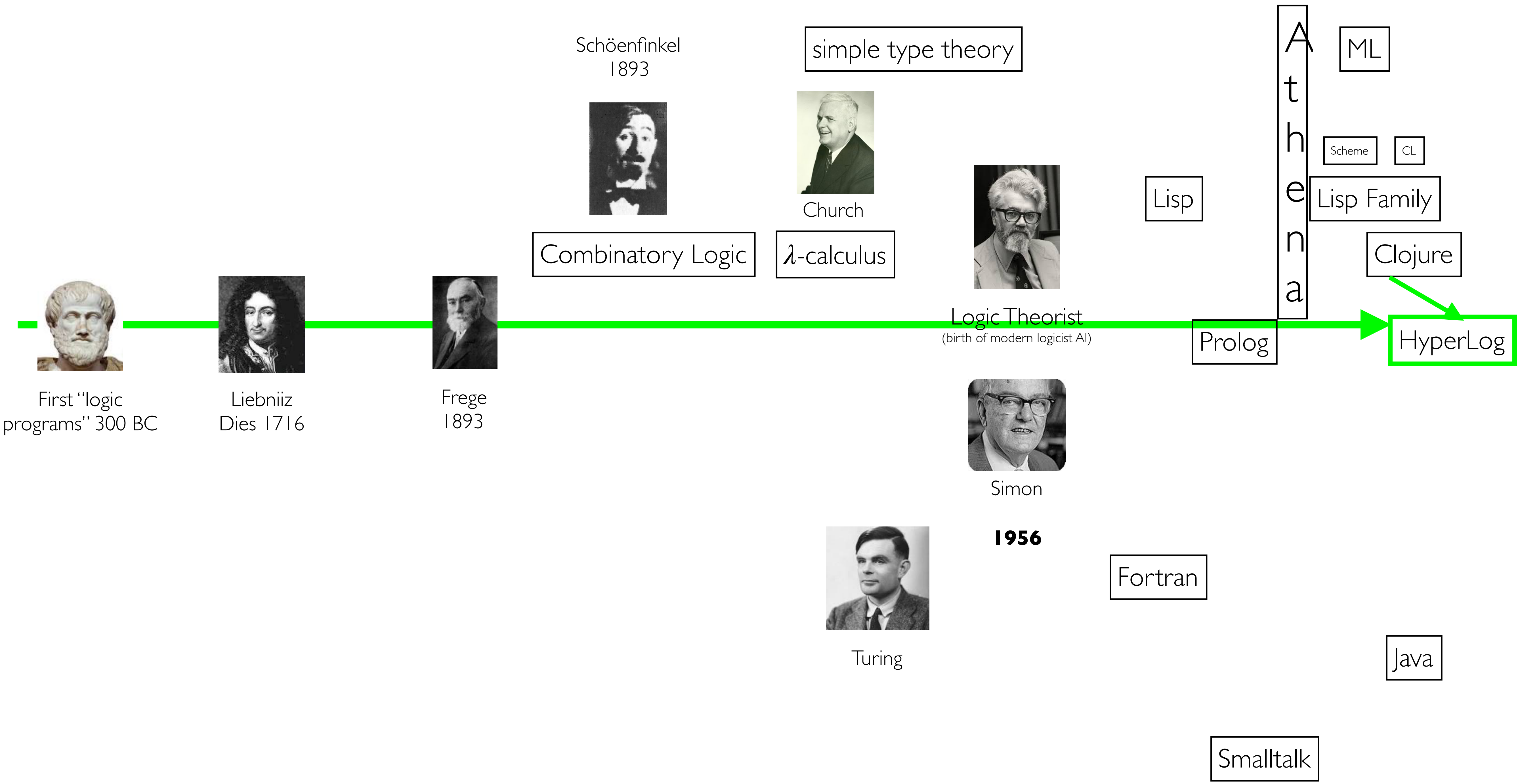
HyperLog:
Historico-logico-programming Landscape



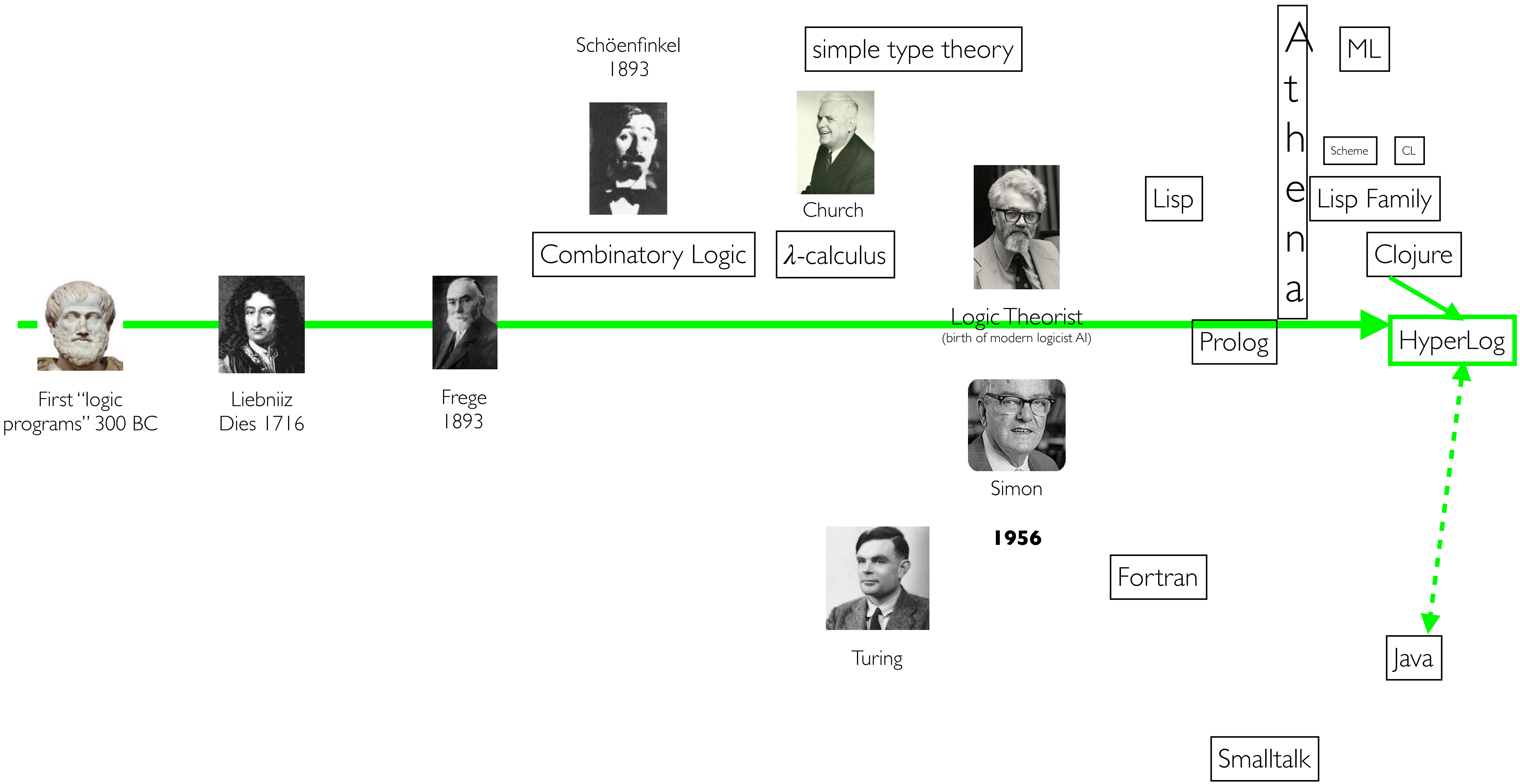
HyperLog:
Historico-logico-programming Landscape



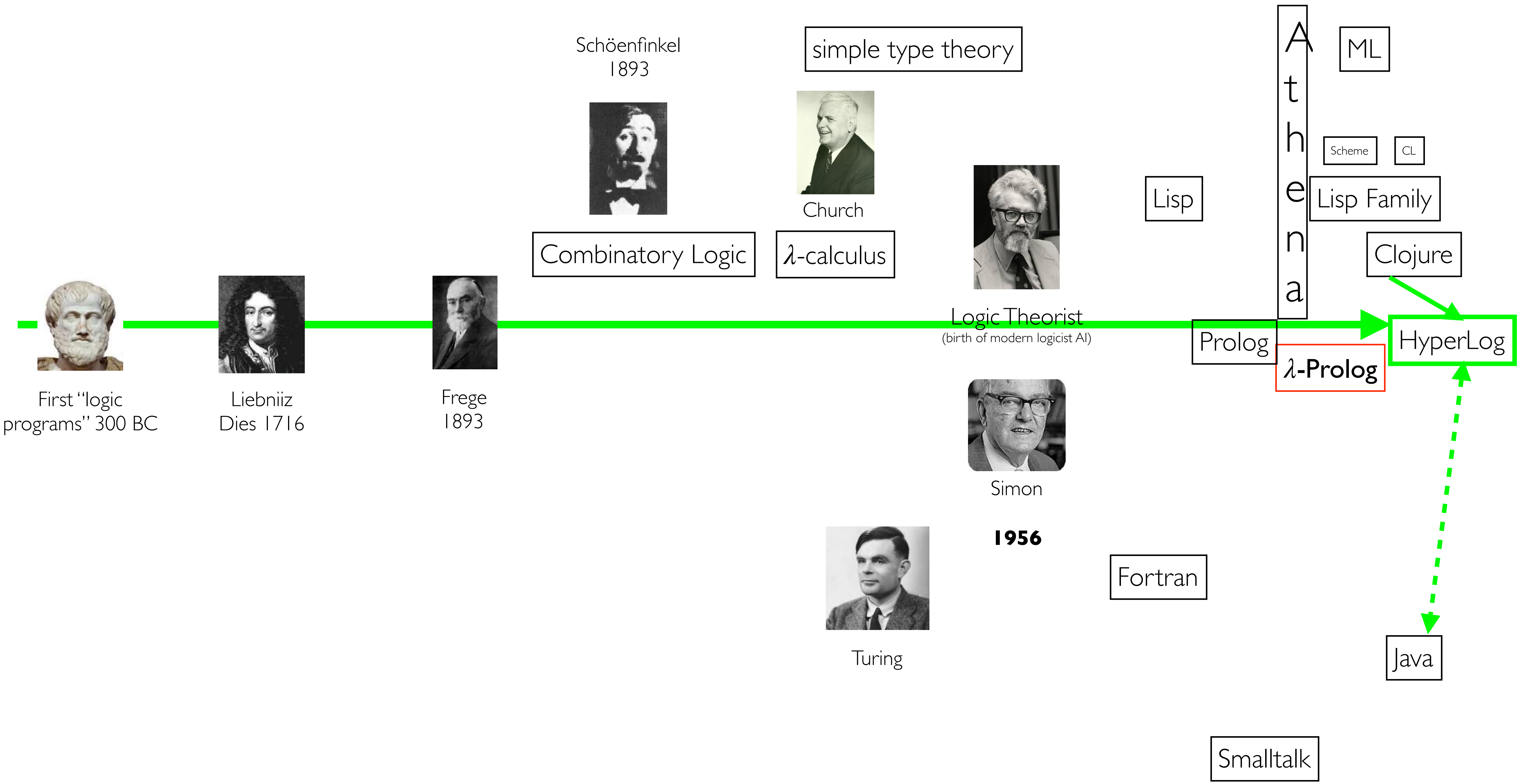
HyperLog: Historico-logico-programming Landscape



HyperLog:
Historico-logico-programming Landscape



HyperLog:
Historico-logico-programming Landscape



Thinking as Computation

Hector J. Levesque
Dept. of Computer Science
University of Toronto

Thinkin

Constants and variables

A Prolog *constant* must start with a *lower case* letter, and can then be followed by any number of letters, underscores, or digits.

A constant may also be a *quoted-string*: any string of characters (except a single quote) enclosed within single quotes.

So the following are all legal constants:

```
sue opp_sex mamboNumber5 'Who are you?'
```

A Prolog *variable* must start with an *upper case* letter, and can then be followed by any number of letters, underscores, or digits.

So the following are all legal variables:

```
X P1 MyDog The_biggest_number Variable_27b
```

Prolog also has *numeric* terms, which we will return to later.

Atomic sentences

The atomic sentences or *atoms* of Prolog have the following form:

$$predicate(term_1, \ldots, term_k)$$

where the predicate is a constant and the terms are either constants or variables.

Note the punctuation:

- immediately after the predicate, there must be a *left parenthesis*;
- between each term, there must be a *comma*;
- immediately after the last term, there must be a *right parenthesis*.

The number of terms k is called the *arity* of the predicate.
If $k = 0$, the parentheses can be left out.

Conditional sentences

The conditional sentences of Prolog have the following form:

$$\textit{head} \;\; \text{:–} \;\; \textit{body}_1, \ldots, \textit{body}_n$$

where the head and each element of the body is an atom.

Note the punctuation:

- immediately after the head, there must be a *colon* then *hyphen*, `:-`.
- between each element of the body, there must be a *comma*.

If $n = 0$, the `:-` should be omitted.

> In other words, an atomic sentence is just a conditional sentence where the body is empty!

Atomic sentences

Conditional sentences

Prolog programs

Prolog programs are simply *knowledge bases* consisting of atomic and conditional sentences as before, but with a sightly different notation.

Here is the “family” example as a Prolog program:

family.pl

```
% This is the Prolog version of the family example

child(john,sue).   child(john,sam).
child(jane,sue).   child(jane,sam).
child(sue,george).   child(sue,gina).

male(john).   male(sam).    male(george).
female(sue).  female(jane).  female(june).

parent(Y,X) :- child(X,Y).
father(Y,X) :- child(X,Y), male(Y).
opp_sex(X,Y) :- male(X), female(Y).
opp_sex(Y,X) :- male(X), female(Y).
grand_father(X,Z) :- father(X,Y), parent(Y,Z).
```

Now let’s look at all the pieces in detail . . .

Prolog Problems

massachusetts institute of technology — artificial intelligence laboratory

Certified Computation

Konstantine Arkoudas

AI Memo 2001-007 April 30, 2001

`unify`, which are by far the two most complicated parts of the system. We only need to trust our five primitive methods. This becomes evident when we ask Athena to produce the relevant certificates. For instance, if we ask Athena to produce the certificate for the method call

```
(!unify (Cons (== s t) Nil))
```

we will obtain the exact same proof that was given in page 17, which only uses the primitive inference rules of our logic.

1.4 Comparison with other approaches

As we mentioned earlier, the idea of using deduction for computational purposes has been around for a long time. There are several methodologies predating DPLs that can be used for certified computation. In this section we will compare DPLs to logic programming languages and to theorem proving systems of the HOL variety.

Comparison with logic programming

The notion of "programming with logic" was a seminal idea, and its introduction and subsequent popularization by Prolog was of great importance in the history of computing. Although logic programming languages can be viewed as platforms for certified computation, they have little in common with DPLs. DPLs are languages for writing proofs and proof strategies. By contrast, in logic programming users do not write proofs; they only write assertions. The inference mechanism that is used for deducing the consequences of those assertions is fixed and sequestered from the user: linear resolution in the case of Prolog, some higher-order extension thereof in the case of higher-order logic programming languages [9, 2], and so on. This rigidity can be unduly constraining. It locks the user into formulating every problem in terms of the same representation (Horn clauses, or higher-order hereditary Harrop clauses [10], etc.) and the same inference method, even when those are not the proper tools to use. For instance, how does one go about proving De Morgan's laws in Prolog? How does one derive $\neg(\exists\, x)\, \neg P(x)$ from the assumption $(\forall\, x)\, P(x)$? Moreover, how does one write a schema that does this for any given x and P? How about higher-order equational rewriting or semantic tableaux? Although in principle more or less everything could be simulated in Prolog, for many purposes such a simulation would be formidably cumbersome.

A related problem is lack of extensibility. Users have no way of extending the underlying inference mechanism so as to allow the system to prove more facts or different types of facts.

The heart of the issue is how much control the user should have over proof construction. In logic programming the proof-search algorithm is fixed, and users are discouraged from tampering with it (e.g., by using impure control operators or clever clause reorderings). Indeed, strong logic programming advocates maintain that the user should have no control at all over proof construction. The user should simply enter a set of assertions, sit back, and let the system deduce the desired consequences. Advocates of weak logic programming allow

massachusetts ins

Certifie

Konstantin

AI Memo 2001-007

unify, which are by far the two most complicated parts of the system. We only need to trust our five primitive methods. This becomes evident when we ask Athena to produce the relevant certificates. For instance, if we ask Athena to produce the certificate for the method call

```
(!unify (Cons (== s t) Nil))
```

we will obtain the exact same proof that was given in page 17, which only uses the primitive inference rules of our logic.

1.4 Comparison with other approaches

As we mentioned earlier, the idea of using deduction for computational purposes has been around for a long time. There are several methodologies predating DPLs that can be used for certified computation. In this section we will compare DPLs to logic programming languages and to theorem proving systems of the HOL variety.

Comparison with logic programming

The notion of "programming with logic" was a seminal idea, and its introduction and subsequent popularization by Prolog was of great importance in the history of computing. Although logic programming languages can be viewed as platforms for certified computation, they have little in common with DPLs. DPLs are languages for writing proofs and proof strategies. By contrast, in logic programming users do not write proofs; they only write assertions. The inference mechanism that is used for deducing the consequences of those assertions is fixed and sequestered from the user: linear resolution in the case of Prolog, some higher-order extension thereof in the case of higher-order logic programming languages [9, 2], and so on. This rigidity can be unduly constraining. It locks the user into formulating every problem in terms of the same representation (Horn clauses, or higher-order hereditary Harrop clauses [10], etc.) and the same inference method, even when those are not the proper tools to use. For instance, how does one go about proving De Morgan's laws in Prolog? How does one derive $\neg(\exists\, x)\, \neg P(x)$ from the assumption $(\forall\, x)\, P(x)$? Moreover, how does one write a schema that does this for any given x and P? How about higher-order equational rewriting or semantic tableaux? Although in principle more or less everything could be simulated in Prolog, for many purposes such a simulation would be formidably cumbersome.

A related problem is lack of extensibility. Users have no way of extending the underlying inference mechanism so as to allow the system to prove more facts or different types of facts.

The heart of the issue is how much control the user should have over proof construction. In logic programming the proof-search algorithm is fixed, and users are discouraged from tampering with it (e.g., by using impure control operators or clever clause reorderings). Indeed, strong logic programming advocates maintain that the user should have no control at all over proof construction. The user should simply enter a set of assertions, sit back, and let the system deduce the desired consequences. Advocates of weak logic programming allow

massachusetts ins

Certifie

Konstantin

AI Memo 2001-007

λ-Prolog

`unify`, which are by far the two most complicated parts of the system. We only need to trust our five primitive methods. This becomes evident when we ask Athena to produce the relevant certificates. For instance, if we ask Athena to produce the certificate for the method call

```
(!unify (Cons (== s t) Nil))
```

we will obtain the exact same proof that was given in page 17, which only uses the primitive inference rules of our logic.

1.4 Comparison with other approaches

As we mentioned earlier, the idea of using deduction for computational purposes has been around for a long time. There are several methodologies predating DPLs that can be used for certified computation. In this section we will compare DPLs to logic programming languages and to theorem proving systems of the HOL variety.

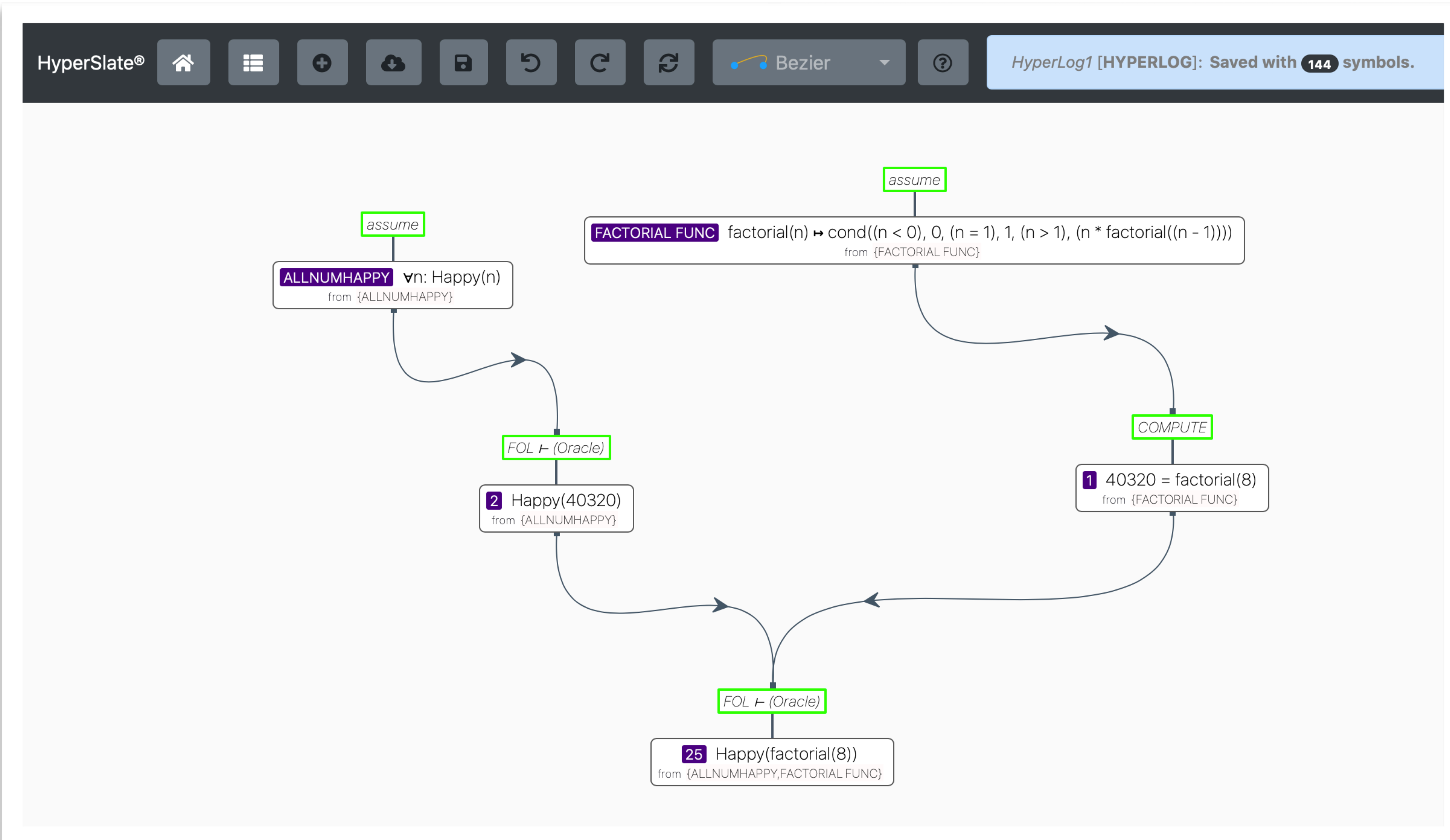
Comparison with logic programming

The notion of "programming with logic" was a seminal idea, and its introduction and subsequent popularization by Prolog was of great importance in the history of computing. Although logic programming languages can be viewed as platforms for certified computation, they have little in common with DPLs. DPLs are languages for writing proofs and proof strategies. By contrast, in logic programming users do not write proofs; they only write assertions. The inference mechanism that is used for deducing the consequences of those assertions is fixed and sequestered from the user: linear resolution in the case of Prolog, some higher-order extension thereof in the case of higher-order logic programming languages [9, 2], and so on. This rigidity can be unduly constraining. It locks the user into formulating every problem in terms of the same representation (Horn clauses, or higher-order hereditary Harrop clauses [10], etc.) and the same inference method, even when those are not the proper tools to use. For instance, how does one go about proving De Morgan's laws in Prolog? How does one derive $\neg(\exists\, x)\, \neg P(x)$ from the assumption $(\forall\, x)\, P(x)$? Moreover, how does one write a schema that does this for any given x and P? How about higher-order equational rewriting or semantic tableaux? Although in principle more or less everything could be simulated in Prolog, for many purposes such a simulation would be formidably cumbersome.

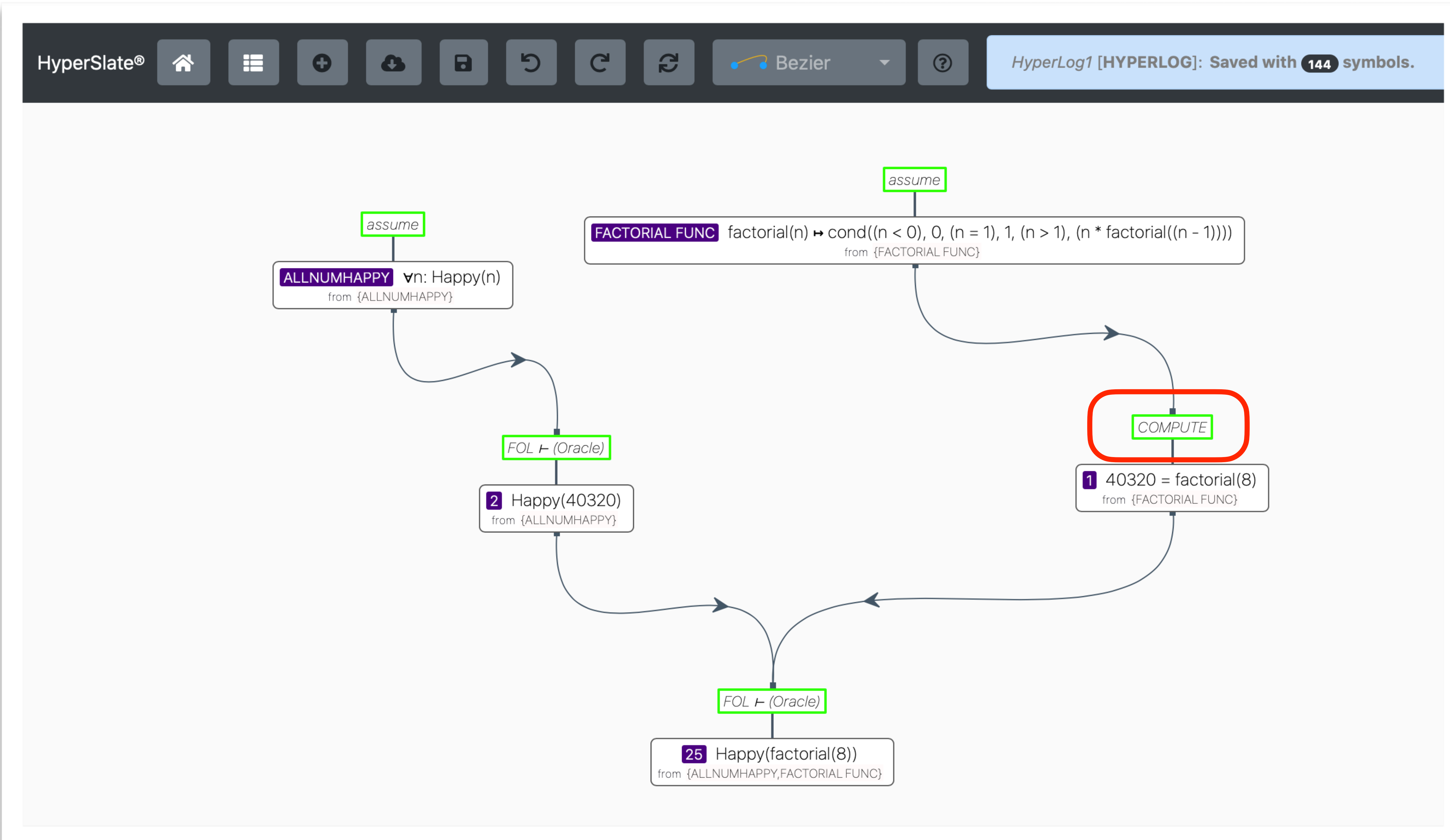
A related problem is lack of extensibility. Users have no way of extending the underlying inference mechanism so as to allow the system to prove more facts or different types of facts.

The heart of the issue is how much control the user should have over proof construction. In logic programming the proof-search algorithm is fixed, and users are discouraged from tampering with it (e.g., by using impure control operators or clever clause reorderings). Indeed, strong logic programming advocates maintain that the user should have no control at all over proof construction. The user should simply enter a set of assertions, sit back, and let the system deduce the desired consequences. Advocates of weak logic programming allow

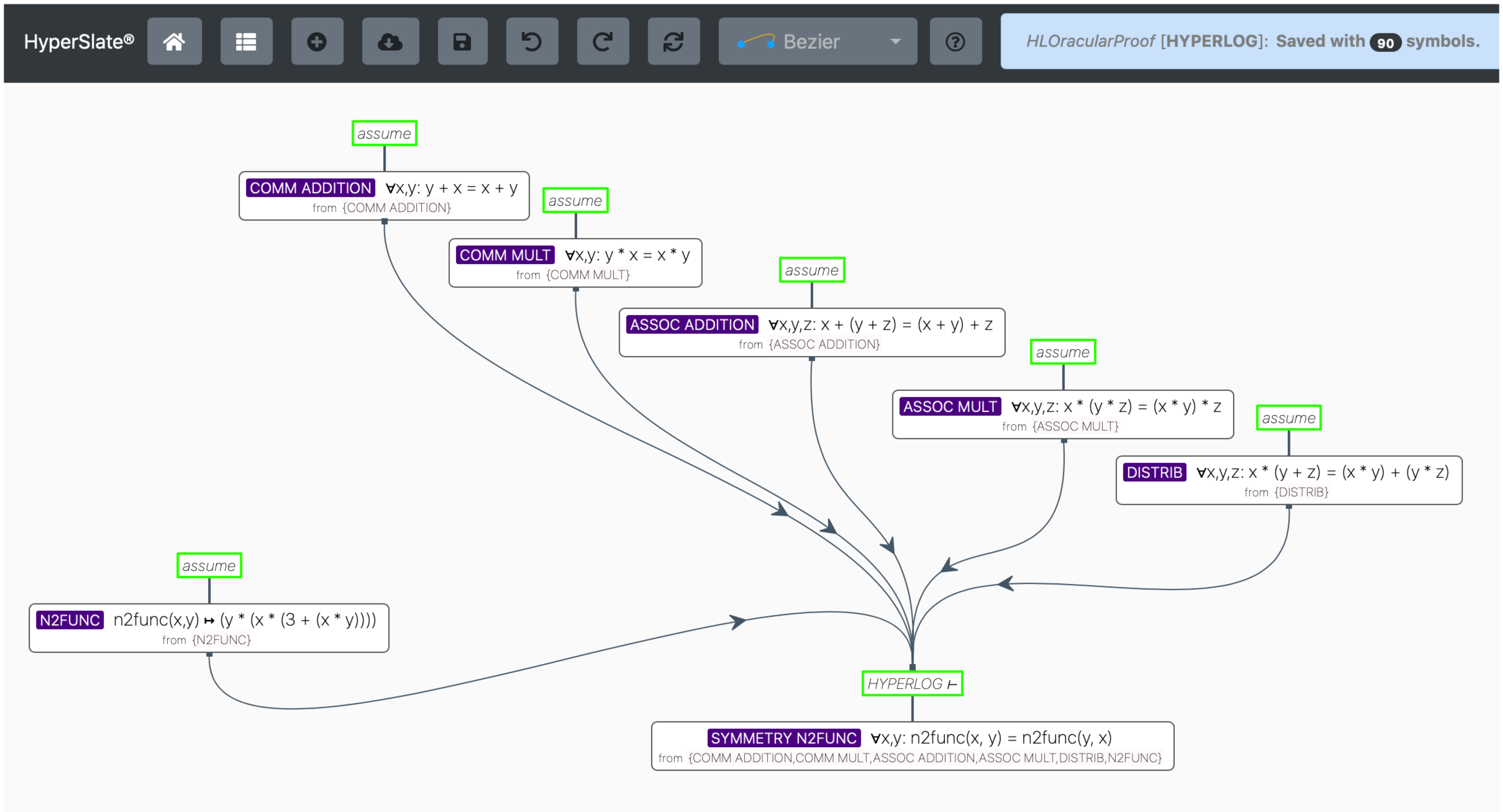
The Factorial of 8 is Happy!



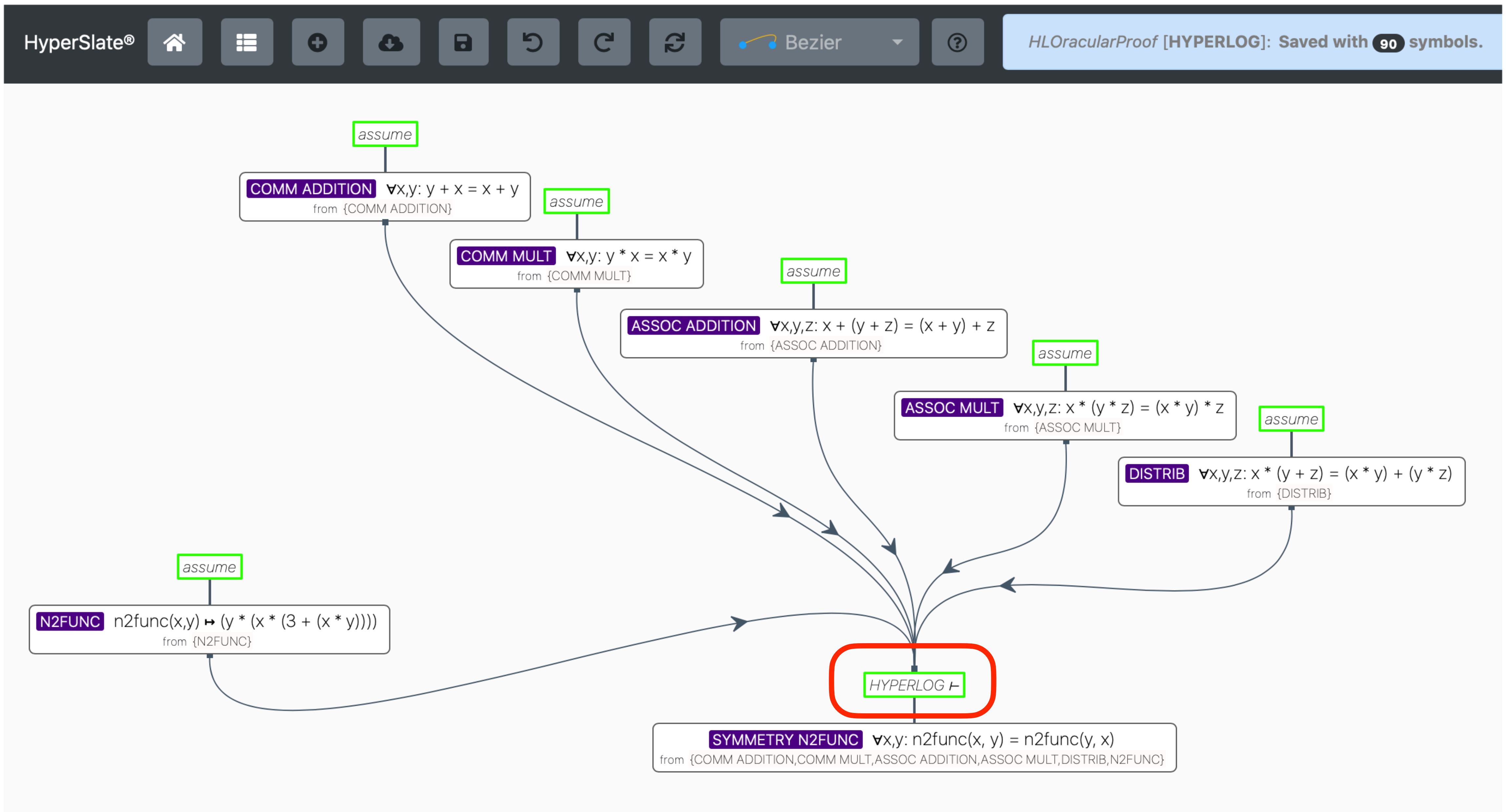
The Factorial of 8 is Happy!



A New Oracle!


HyperSlate®
Bezier
HLOracularProof [HYPERLOG]: Saved with 90 symbols.
assume
COMM ADDITION ∀x,y: y + x = x + y
from {COMM ADDITION}
assume
COMM MULT ∀x,y: y * x = x * y
from {COMM MULT}
assume
ASSOC ADDITION ∀x,y,z: x + (y + z) = (x + y) + z
from {ASSOC ADDITION}
assume
ASSOC MULT ∀x,y,z: x * (y * z) = (x * y) * z
from {ASSOC MULT}
assume
DISTRIB ∀x,y,z: x * (y + z) = (x * y) + (y * z)
from {DISTRIB}
assume
N2FUNC n2func(x,y) ↦ (y * (x * (3 + (x * y))))
from {N2FUNC}
HYPERLOG ⊢
SYMMETRY N2FUNC ∀x,y: n2func(x, y) = n2func(y, x)
from {COMM ADDITION,COMM MULT,ASSOC ADDITION,ASSOC MULT,DISTRIB,N2FUNC}

A New Oracle!



HyperSlate®
Bezier
HLOracularProof [HYPERLOG]: Saved with 90 symbols.
assume
COMM ADDITION ∀x,y: y + x = x + y
from {COMM ADDITION}
assume
COMM MULT ∀x,y: y * x = x * y
from {COMM MULT}
assume
ASSOC ADDITION ∀x,y,z: x + (y + z) = (x + y) + z
from {ASSOC ADDITION}
assume
ASSOC MULT ∀x,y,z: x * (y * z) = (x * y) * z
from {ASSOC MULT}
assume
DISTRIB ∀x,y,z: x * (y + z) = (x * y) + (y * z)
from {DISTRIB}
assume
N2FUNC n2func(x,y) ↦ (y * (x * (3 + (x * y))))
from {N2FUNC}
HYPERLOG ⊢
SYMMETRY N2FUNC ∀x,y: n2func(x, y) = n2func(y, x)
from {COMM ADDITION,COMM MULT,ASSOC ADDITION,ASSOC MULT,DISTRIB,N2FUNC}

Available in HyperLog .5

```
"letfn",
"=", "cond",
"and", "or", "not", "not=",
"+", "-", "*", "/", "quot", "rem", "mod",
"inc", "dec", "max" , "min", "+'", "-'", "*'", "inc'", "dec'",
"==", "<", ">", "<=", ">=", "compare",
"zero?", "pos?", "neg?", "even?", "odd?",
"number?", "rational?", "integer?", "ratio?", "decimal?", "float?",
"double?", "int?", "nat-int?", "neg-int?", "pos-int?",
"count", "get", "subs", "compare",
"clojure.string/join", "clojure.string/escape", "clojure.string/split",
"clojure.string/split-lines", "clojure.string/replace", "clojure.string/replace-first",
"reverse", "index-of", "last-index-of", "str"
```

Making Morally  Machines

Selmer Bringsjord ∧ Naveen Sundar Govindarajulu ∧ John Licato

Making Morally  **Machines**

Selmer Bringsjord ∧ Naveen Sundar Govindarajulu ∧ John Licato

er løsningen, med nok penger!