Applied Logics

Chris Stone

September 28, 2010

Outline

- How can there be more than one logic?
  Why would you want more than one?
- Constructive Logic
  - Research idea: RZ
- Linear Logic
- Temporal Logic
  - Research idea: CSL

Computable Mathematics

Traditional CS is closely connected to *discrete* mathematics

Data is finite (though arbitrarily large)

- Lists
- Trees
- Hash tables
- Bigints (arbitrarily large integers)

Computable Mathematics

Traditional CS is closely connected to *discrete* mathematics

Data is finite (though arbitrarily large)

- Lists
- Trees
- Hash tables
- Bigints (arbitrarily large integers)

But what if you want to compute with

- Real numbers
- Infinite sequences and limits
- Integration and differentiation
- Arbitrary continuous functions
- Arbitrary open (or closed, or compact) subsets of the plane
- Differential equations

Problems with Floating-Point: Limited Range

Floating-point numbers are a *finite* subset of the *rationals*, i.e,

$$\pm(2^{24}+m)2^{n} \qquad \begin{cases} 0 \leq m < 2^{24} \\ -140 \leq n \leq 103 \end{cases}$$

Problems with Floating-Point: Limited Range

Floating-point numbers are a *finite* subset of the *rationals*, i.e,

$$\pm(2^{24}+m)2^n \qquad \begin{cases} 0 \le m < 2^{24} \\ -140 \le n \le 103 \end{cases}$$

$$\begin{gathered}
340282346638528859811704183484516925440 \\
34028232\underline{6356119256160033759537265639424} \\
\vdots \\
1/2^{126} \\
+0 \\
-0 \\
1/2^{126} \\
\vdots \\
-340282326356119256160033759537265639424 \\
-340282346638528859811704183484516925440
\end{gathered}$$

Problems with Floating-Point: Approximate Answers

Round-off and cancellation matter!

```
//  1/1 + 1/2 + ... + 1/(N-1)
float sum1 = 0.0;
for (int n = 1; n < N; ++n)
   sum1 += 1.0 / float(n);

//  1/(N-1) + ... + 1/2 + 1/1
float sum2 = 0.0;
for (int n = N-1; n > 0; --n)
   sum2 += 1.0 / float(n);
```

Problems with Floating-Point: Approximate Answers

Round-off and cancellation matter!

```
//  1/1 + 1/2 + ... + 1/(N-1)
float sum1 = 0.0;
for (int n = 1; n < N; ++n)
   sum1 += 1.0 / float(n);

//  1/(N-1) + ... + 1/2 + 1/1
float sum2 = 0.0;
for (int n = N-1; n > 0; --n)
   sum2 += 1.0 / float(n);
```

When $\mathtt{N} = 10^8$, $\mathtt{sum1} = 15.4037$ and $\mathtt{sum2} = 18.8079$

Some Topics of Computable Mathematics

Some Topics of Computable Mathematics

1. How can a computer represent
   - "Real" real numbers
   - "Real" complex numbers?
   - Infinite sequences of reals?
   - Differentiable functions?
   - Arbitrary open (or closed, or compact) subsets of the plane?
   - ...

Some Topics of Computable Mathematics

1. How can a computer represent
   - "Real" real numbers
   - "Real" complex numbers?
   - Infinite sequences of reals?
   - Differentiable functions?
   - Arbitrary open (or closed, or compact) subsets of the plane?
   - ...
2. What operations are computable?
   - Multiplication of reals?
   - Square root of a complex?
   - Finding a zero of a function?
   - Integral of a function over a closed interval?
   - Intersection of two open sets?
   - ...

Some Topics of Computable Mathematics

1. How can a computer represent
   - "Real" real numbers
   - "Real" complex numbers?
   - Infinite sequences of reals?
   - Differentiable functions?
   - Arbitrary open (or closed, or compact) subsets of the plane?
   - ...
2. What operations are computable?
   - Multiplication of reals?
   - Square root of a complex?
   - Finding a zero of a function?
   - Integral of a function over a closed interval?
   - Intersection of two open sets?
   - ...
3. Can we do these efficiently (at least in practice)?

Representing Real Numbers

What does it mean for a computer to "have" a *real* number?

Representing Real Numbers

What does it mean for a computer to “have” a *real* number?

What is a reasonable representation for a “real” *real* number?

Representing Real Numbers

What does it mean for a computer to "have" a *real* number?

What is a reasonable representation for a "real" *real* number?

First idea: infinite sequence of decimal digits

$$\begin{aligned} 1/2 &= [5, 0, 0, 0, 0, 0, 0, \ldots] \\ 1/30 &= [0, 3, 3, 3, 3, 3, 3, 3, \ldots] \end{aligned}$$

Representing Real Numbers

What does it mean for a computer to "have" a *real* number?

What is a reasonable representation for a "real" *real* number?

First idea: infinite sequence of decimal digits

$$\begin{array}{rcl} 1/2 & = & [5,0,0,0,0,0,0,\ldots] \\ 1/30 & = & [0,3,3,3,3,3,3,3,\ldots] \end{array}$$

In practice, we'd produce digits *on demand*, e.g.,

$$\texttt{real} \quad \equiv \quad \texttt{nat} \rightarrow \{0,\ldots,9\}$$

Critique

The operation "Divide by 10" is computable! (How?)

$$1, 4, 1, 5, 9, 2, 6, 5, 3, 5, \ldots$$

Critique

The operation "Divide by 10" is computable! (How?)

$$1, 4, 1, 5, 9, 2, 6, 5, 3, 5, \ldots$$

The operation "Multiply by 3" is not!

$$
\begin{array}{lcl}
0,\ 3,\ 1 & \times\ 3 = & 0,\ 9,\ 3 \\
0,\ 3,\ 3,\ 1 & \times\ 3 = & 0,\ 9,\ 9,\ 3 \\
0,\ 3,\ 3,\ 3,\ 1 & \times\ 3 = & 0,\ 9,\ 9,\ 9,\ 3 \\
\hline
0,\ 3,\ 9 & \times\ 3 = & 1,\ 1,\ 7 \\
0,\ 3,\ 3,\ 9 & \times\ 3 = & 1,\ 0,\ 1,\ 7 \\
0,\ 3,\ 3,\ 3,\ 9 & \times\ 3 = & 1,\ 0,\ 0,\ 1,\ 7
\end{array}
$$

Critique

The operation "Divide by 10" is computable! (How?)

$$1, 4, 1, 5, 9, 2, 6, 5, 3, 5, \ldots$$

The operation "Multiply by 3" is not!

$$\begin{array}{lcl}
0,\ 3,\ 1 & \times\ 3 = & 0,\ 9,\ 3 \\
0,\ 3,\ 3,\ 1 & \times\ 3 = & 0,\ 9,\ 9,\ 3 \\
0,\ 3,\ 3,\ 3,\ 1 & \times\ 3 = & 0,\ 9,\ 9,\ 9,\ 3 \\
\hline
0,\ 3,\ 9 & \times\ 3 = & 1,\ 1,\ 7 \\
0,\ 3,\ 3,\ 9 & \times\ 3 = & 1,\ 0,\ 1,\ 7 \\
0,\ 3,\ 3,\ 3,\ 9 & \times\ 3 = & 1,\ 0,\ 0,\ 1,\ 7
\end{array}$$

Suppose you check 100 digits and see 0, 3, 3, 3, 3, . . .
What's the *first* digit of the output?

Better Representations of Reals

As just a few examples among many

- Infinite sequences of *signed* digits (e.g., -9 to 9)
- Rapidly converging Cauchy Sequences:

  $$\{\ a \in \mathbb{Q}^{\mathbb{N}} \mid \forall 0 \leq i < j.\ |a_i - a_j| < 2^i\ \}$$

  (even better: use dyadic rationals!)
- Converging infinite sequences of open intervals (with rational endpoints)

Key Results

Given “good” representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, $\ldots$.

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, ...
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.

Key Results

Given “good” representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, $\ldots$.
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.
4. Every computable function $\mathbb{R} \to \mathbb{R}$ is *continuous*

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, $\ldots$.
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.
4. Every computable function $\mathbb{R} \to \mathbb{R}$ is *continuous*
5. The floor function $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{N}$ is *not* computable.

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, ...
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.
4. Every computable function $\mathbb{R} \to \mathbb{R}$ is *continuous*
5. The floor function $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{N}$ is *not* computable.
6. Every computable function $\mathbb{R} \to \mathbb{N}$ is constant.

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, ...
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.
4. Every computable function $\mathbb{R} \to \mathbb{R}$ is *continuous*
5. The floor function $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{N}$ is *not* computable.
6. Every computable function $\mathbb{R} \to \mathbb{N}$ is constant.
7. There is no computable operator that finds a zero of every continuous $f : [0, 1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$.

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, ...
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.
4. Every computable function $\mathbb{R} \to \mathbb{R}$ is *continuous*
5. The floor function $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{N}$ is *not* computable.
6. Every computable function $\mathbb{R} \to \mathbb{N}$ is constant.
7. There is no computable operator that finds a zero of every continuous $f : [0,1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$.
8. Integrals on finite intervals are generally computable; derivatives of computable functions may not be.

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, ...
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.
4. Every computable function $\mathbb{R} \to \mathbb{R}$ is *continuous*
5. The floor function $\lfloor\cdot\rfloor : \mathbb{R} \to \mathbb{N}$ is *not* computable.
6. Every computable function $\mathbb{R} \to \mathbb{N}$ is constant.
7. There is no computable operator that finds a zero of every continuous $f : [0,1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$.
8. Integrals on finite intervals are generally computable; derivatives of computable functions may not be.

Key Results

Given "good" representations of reals:

1. The basic operations ($+$, $-$, $\times$, $\div$) are computable.
2. So are square-root, exponential, sine, ...
3. Comparisons ($=$, $<$, $\leq$, etc.) on $\mathbb{R}$ are *not* computable.
4. Every computable function $\mathbb{R} \to \mathbb{R}$ is *continuous*
5. The floor function $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{N}$ is *not* computable.
6. Every computable function $\mathbb{R} \to \mathbb{N}$ is constant.
7. There is no computable operator that finds a zero of every continuous $f : [0, 1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$.
8. Integrals on finite intervals are generally computable; derivatives of computable functions may not be.

These are the same results that one can get by constructive reasoning.

Why Work with Constructive Logic?

- *Every* classical proof can be turned into a constructive proof
  - Just need to stick some $\neg\neg$'s.

Why Work with Constructive Logic?

- *Every* classical proof can be turned into a constructive proof
  - Just need to stick some $\neg\neg$'s.
  - Instead of proving a solution exists, you prove that not all values can be non-solutions.

Why Work with Constructive Logic?

- *Every* classical proof can be turned into a constructive proof
  - Just need to stick some $\neg\neg$'s.
  - Instead of proving a solution exists, you prove that not all values can be non-solutions.

Why Work with Constructive Logic?

- *Every* classical proof can be turned into a constructive proof
  - Just need to stick some $\neg\neg$'s.
  - Instead of proving a solution exists, you prove that not all values can be non-solutions.

- Compare:

Theorem
Every function $f : [0, 1] \to \mathbb{R}$ is continuous

Theorem
Every computable function f from the computable reals between 0 and 1 to the computable reals is continuous with a computable modulus of continuity.

Why Work with Constructive Logic?

- *Every* classical proof can be turned into a constructive proof
  - Just need to stick some $\neg\neg$'s.
  - Instead of proving a solution exists, you prove that not all values can be non-solutions.

- Compare:

> **Theorem**
> *Every function $f : [0,1] \to \mathbb{R}$ is continuous*

> **Theorem**
> *Every computable function f from the computable reals between 0 and 1 to the computable reals is continuous with a computable modulus of continuity.*

- Close connection between computable and constructive.

The RZ System

(Program developed with Andrej Bauer, University of Ljubljana)

- Input: Specifications in constructive logic
  - Sets, functions, predicates exist
  - Certain (constructive) axioms hold
- Output: Interfaces describing code
  - Representations must exist
  - Functions operating on these representations must exist
  - They must satisfy certain (classical!) properties.

Formal Basis: Realizability

Realizability dates back to Kleene, as an attempt to interpret constructive logic classically.

Embarrassingly short summary of a rich topic:

- A realizer of a mathematical object
  (e.g., set, vector, tuple, group, smooth manifold)
  is an implementation.
- A realizer of a mathematical proposition
  (e.g., $\forall n \in \mathbb{N}.\ \forall x \in \mathbb{R}.\ \exists y \in \mathbb{R}.\ |x - y^2| < \frac{1}{n}$)
  is its "constructive" content.

Realizers are tedious to describe by hand. Hence, RZ.

RZ Example: A Decidable Set

```
Parameter s : Set.
Axiom decide:
   forall a b : s, (a = b) \/ not (a = b).
```

```
type s
val decide : s -> s -> bool
     // forall a b : ||s||,
     //    if decide a b then
     //      a =s= b
     //    else
     //      not (a =s= b)
```

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: "finite sets" of items without decidable equality.

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: "finite sets" of items without decidable equality.

- Suppose you want a "set" of exact real numbers

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: "finite sets" of items without decidable equality.

- Suppose you want a "set" of exact real numbers
- Only possibility: a list, possibly with duplicates (why?)

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite “sets” supporting:

- The empty set
- A way to add an element to a set
- A “fold” or “reduce” operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: “finite sets” of items without decidable equality.

- Suppose you want a “set” of exact real numbers
- Only possibility: a list, possibly with duplicates (why?)
- Can’t reliably compute the sum (not idempotent)

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: "finite sets" of items without decidable equality.

- Suppose you want a "set" of exact real numbers
- Only possibility: a list, possibly with duplicates (why?)
- Can't reliably compute the sum (not idempotent)
- Can't reliably compute the size

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: "finite sets" of items without decidable equality.

- Suppose you want a "set" of exact real numbers
- Only possibility: a list, possibly with duplicates (why?)
- Can't reliably compute the sum (not idempotent)
- Can't reliably compute the size
- Can't bound the size (as a deterministic function of the *set*)

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: "finite sets" of items without decidable equality.

- Suppose you want a "set" of exact real numbers
- Only possibility: a list, possibly with duplicates (why?)
- Can't reliably compute the sum (not idempotent)
- Can't reliably compute the size
- Can't bound the size (as a deterministic function of the *set*)
- *Can* compute the `max` or `min`

RZ Example: Finitely enumerable sets

The axioms (not shown) yield a data structure for finite "sets" supporting:

- The empty set
- A way to add an element to a set
- A "fold" or "reduce" operation, for binary operators that are
  - Commutative (`f(x,y) = f(y,x)`)
  - Idempotent (`f(x,x) = x`)

Interpretation: "finite sets" of items without decidable equality.

- Suppose you want a "set" of exact real numbers
- Only possibility: a list, possibly with duplicates (why?)
- Can't reliably compute the sum (not idempotent)
- Can't reliably compute the size
- Can't bound the size (as a deterministic function of the *set*)
- *Can* compute the `max` or `min`
- *Can* compute empty/non-empty

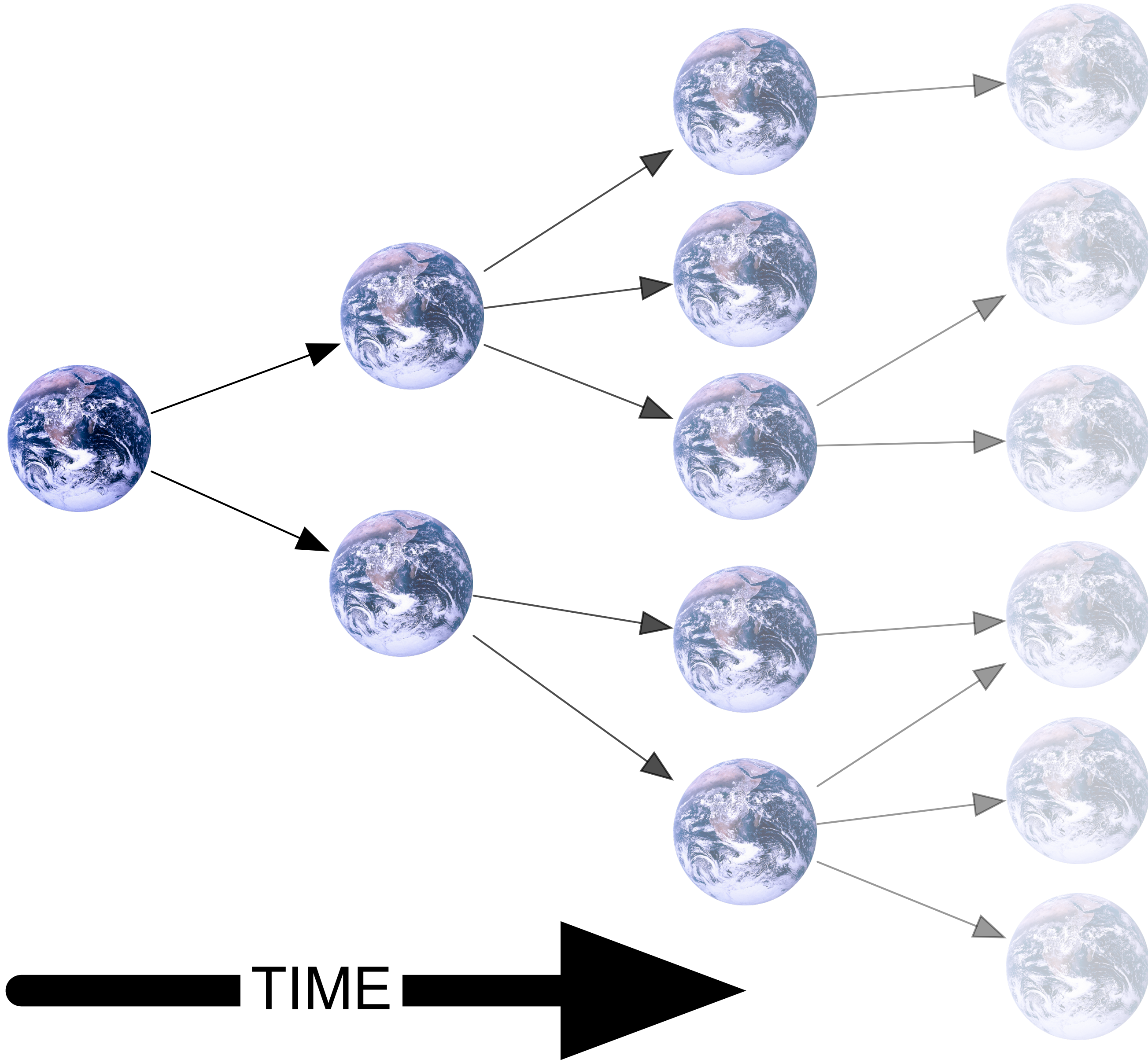
RZ Example: Exact Reals

work by Andrej Bauer and Istok Kavkler

- Axiomatize the (constructive) real numbers
- Run this through RZ
- Hand-write implementations that match the interfaces

Goal: An efficient implementation of exact real arithmetic with a strong theoretical basis.

Modal Logic: Logics of Possible Worlds

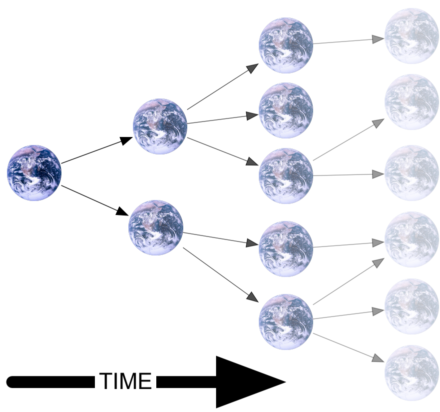


Propositions About Aliens

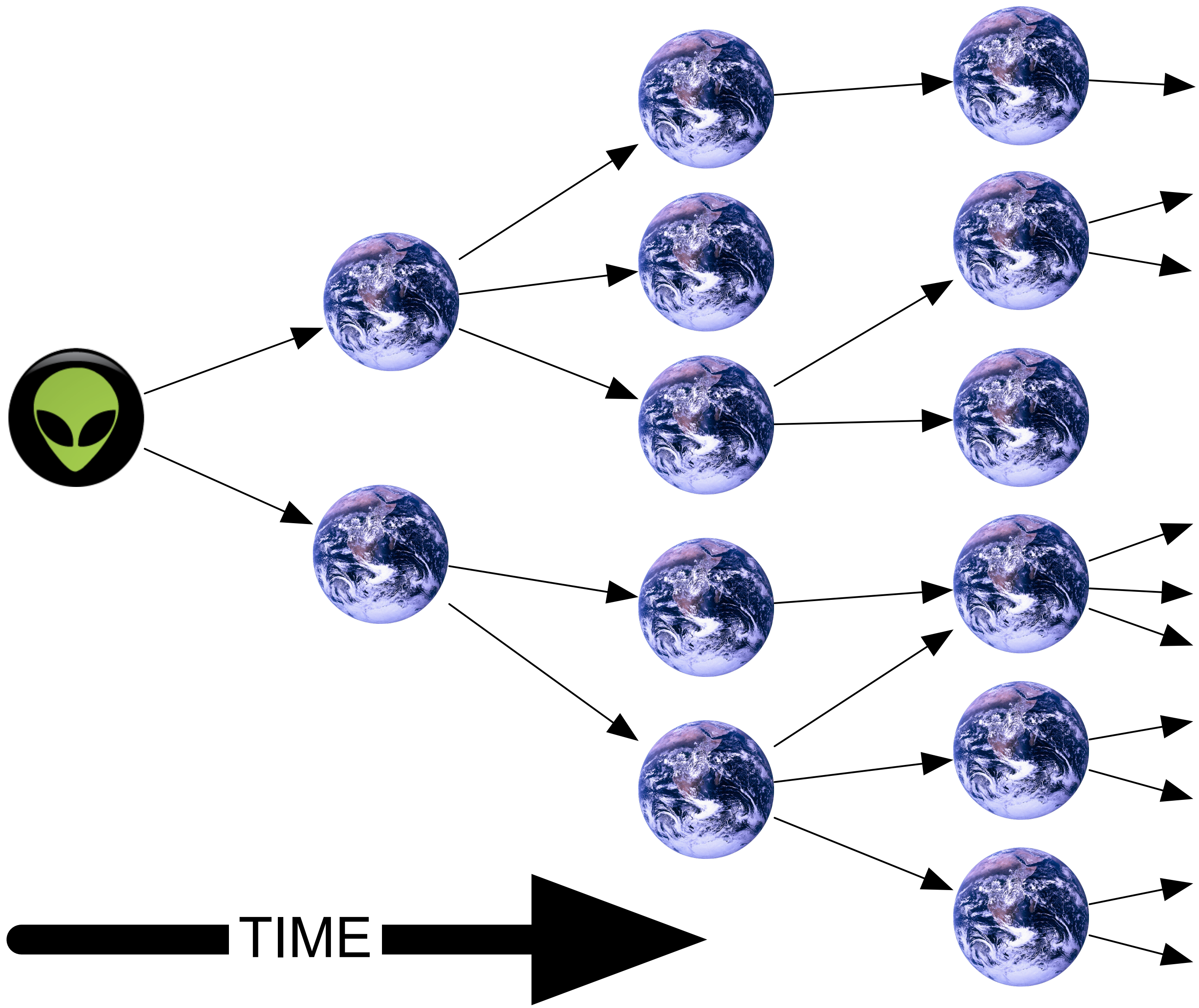
- Aliens are among us (now).
- Aliens may eventually be among us.
- Aliens might always be among us.
- Aliens could eventually be always among us.
- Aliens will always be among us.
- If faster-than-light travel is invented, aliens will eventually be among us.

New Quantifiers

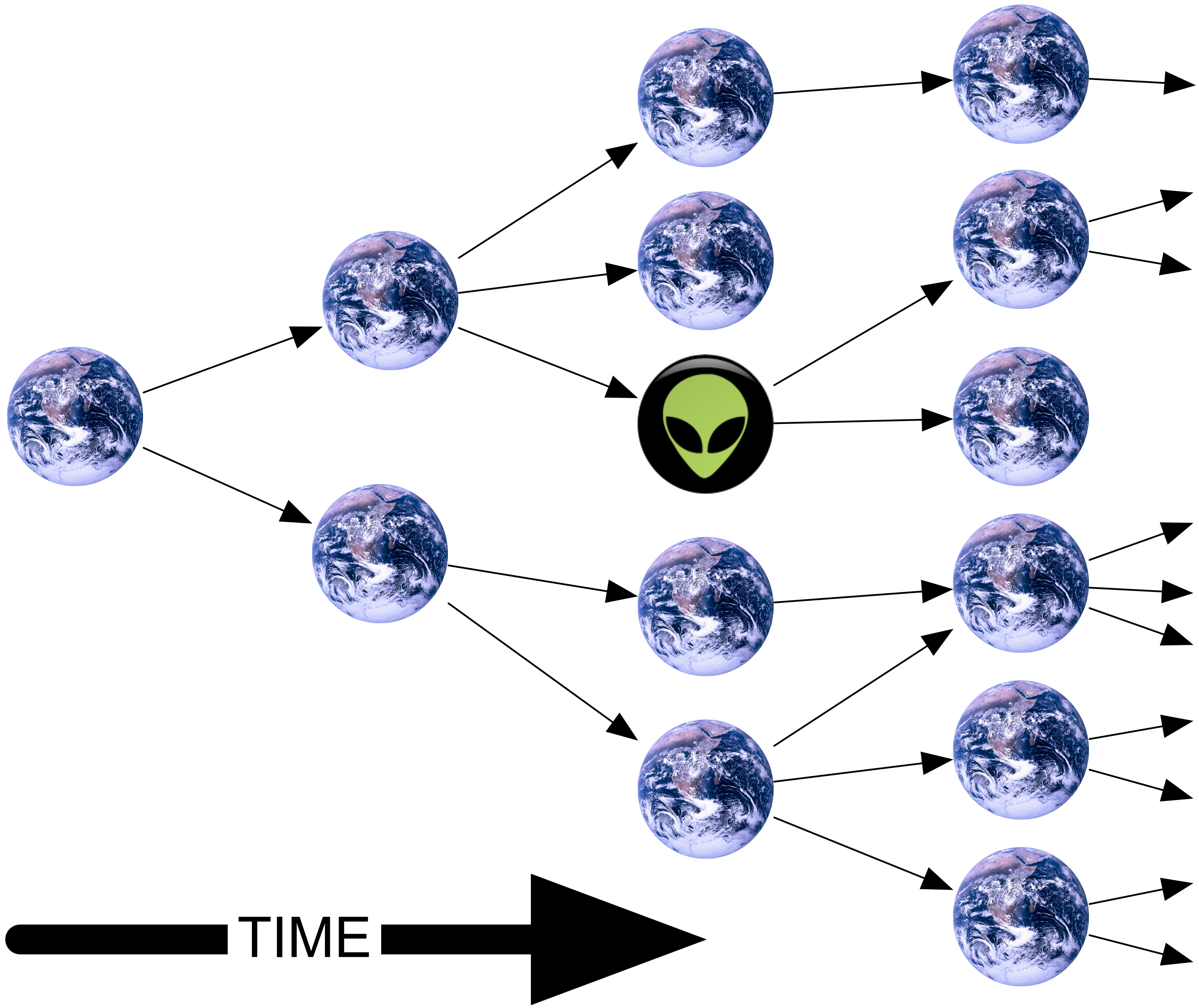
	Eventually True	Invariably True
Some Path	EF p Possibly	EG p Potentially Always
All Paths	AF p Inevitably	AG p Invariably



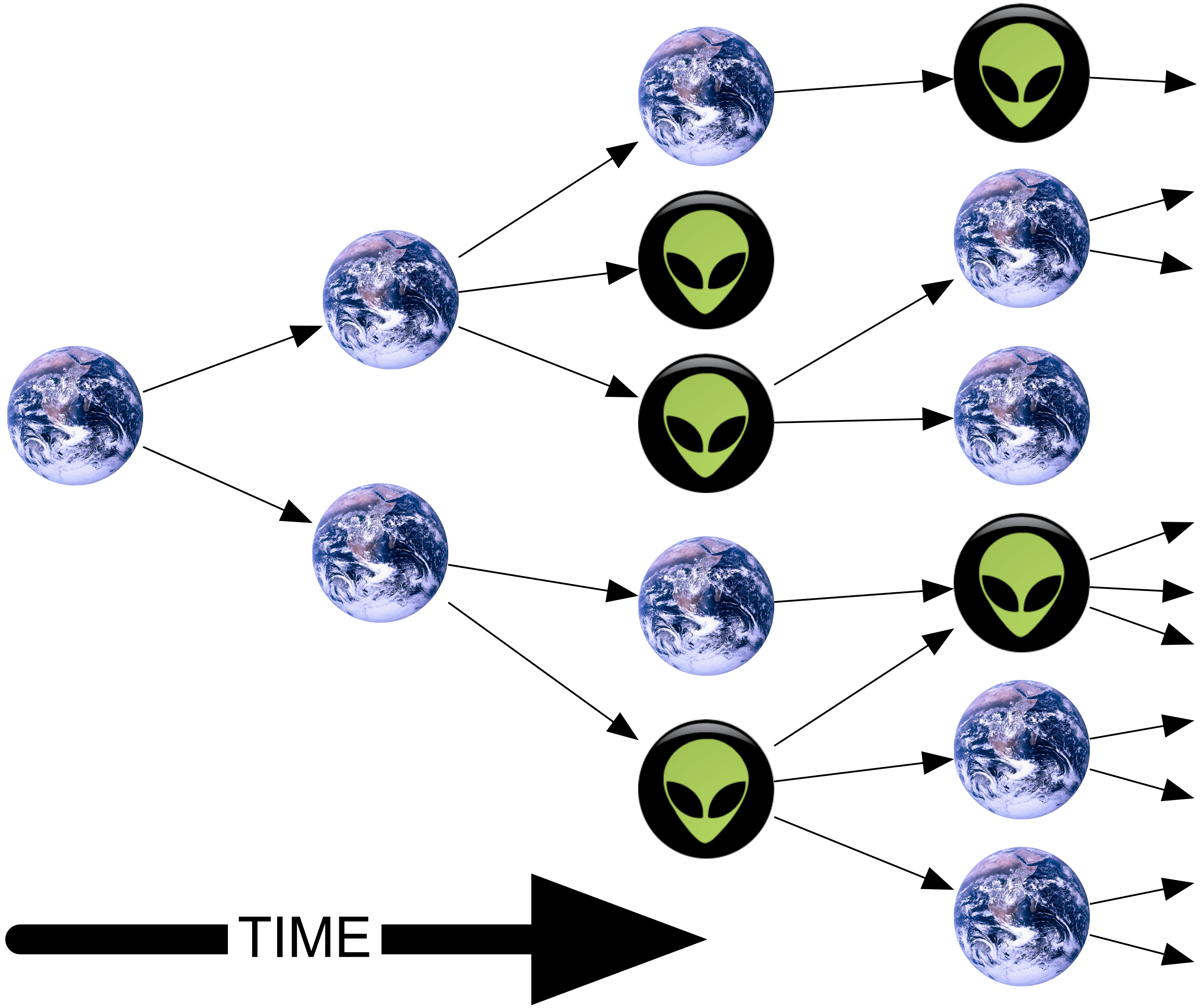
Aliens



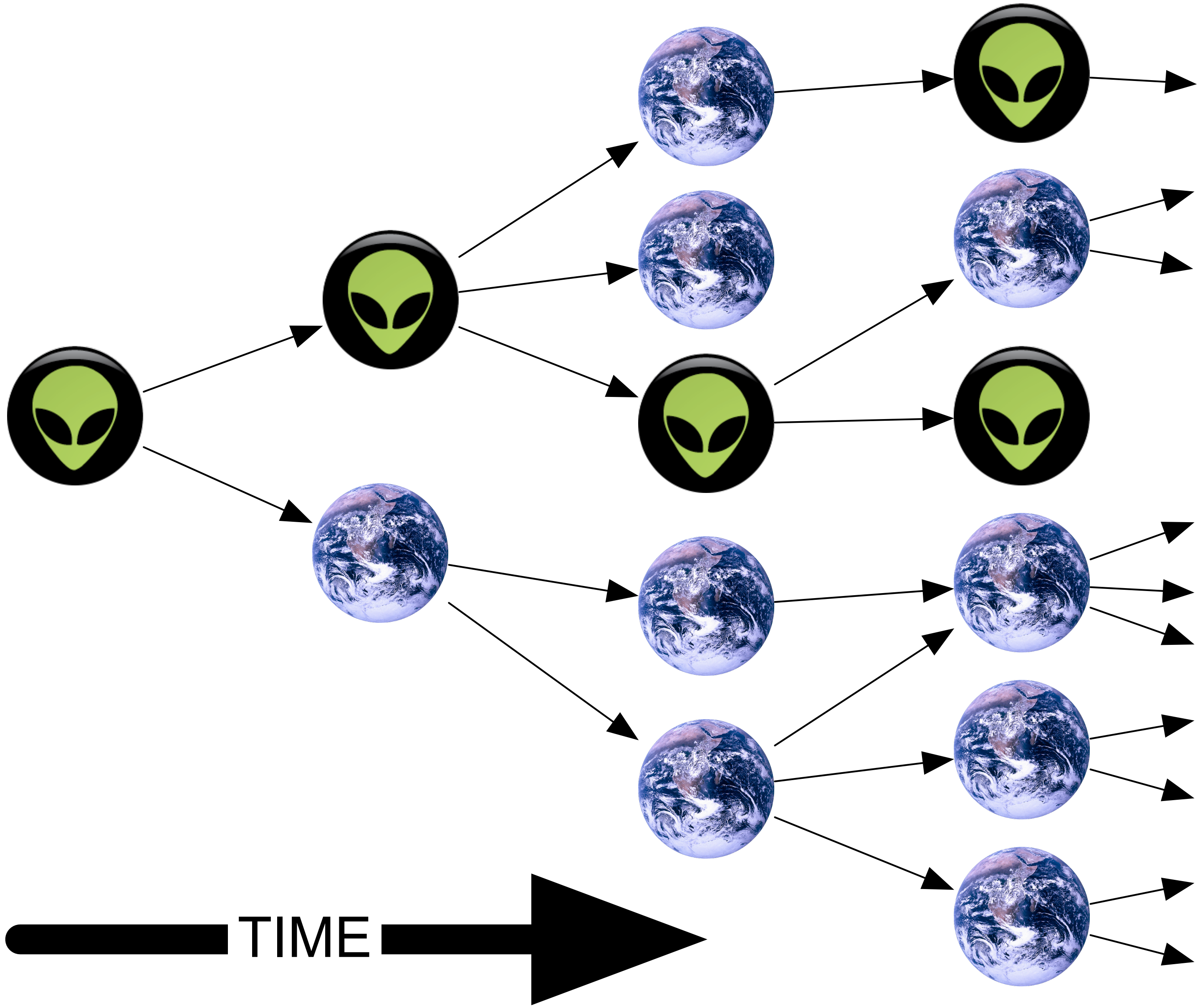
EF *Aliens*



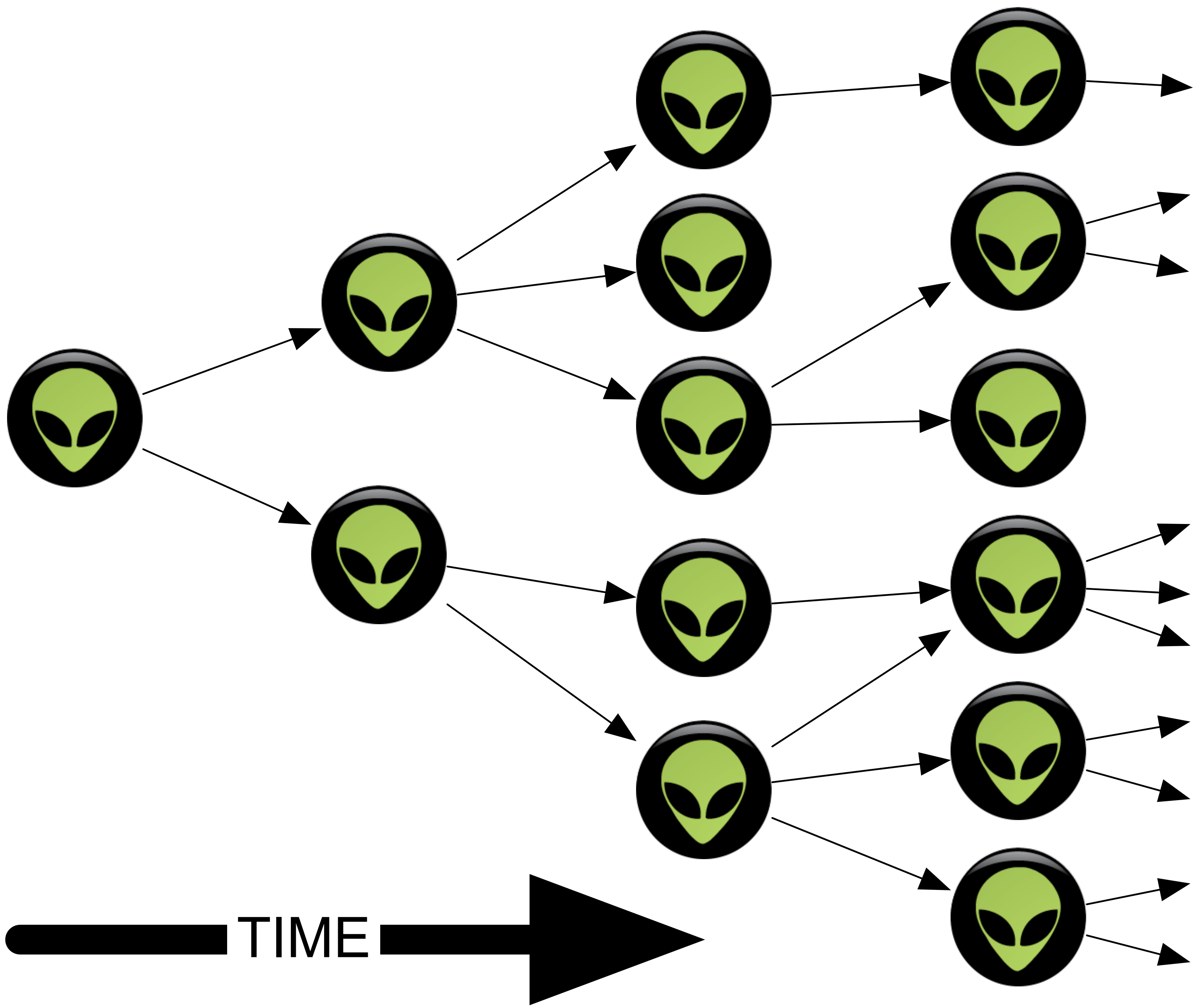
AF *Aliens*



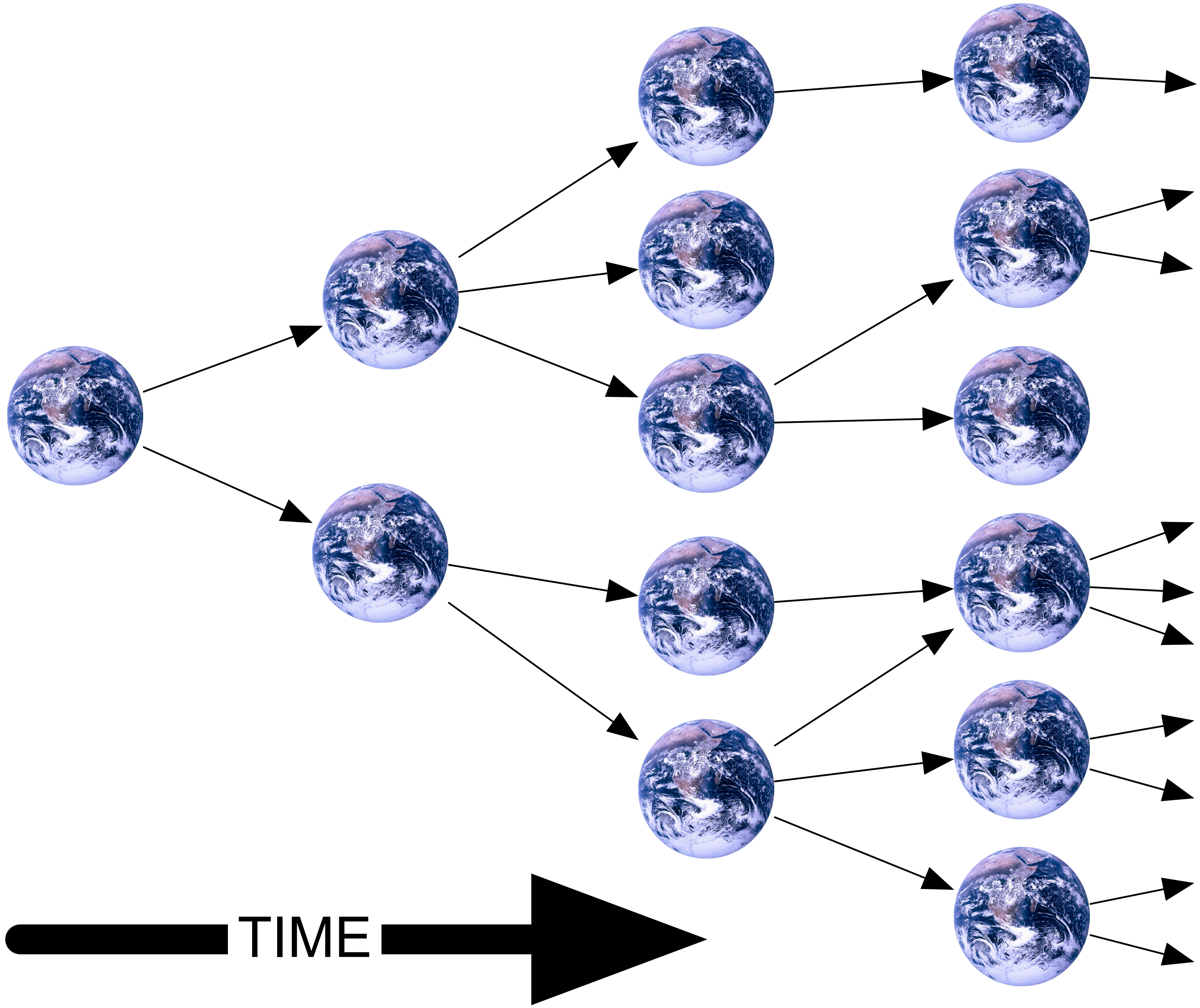
EG *Aliens*



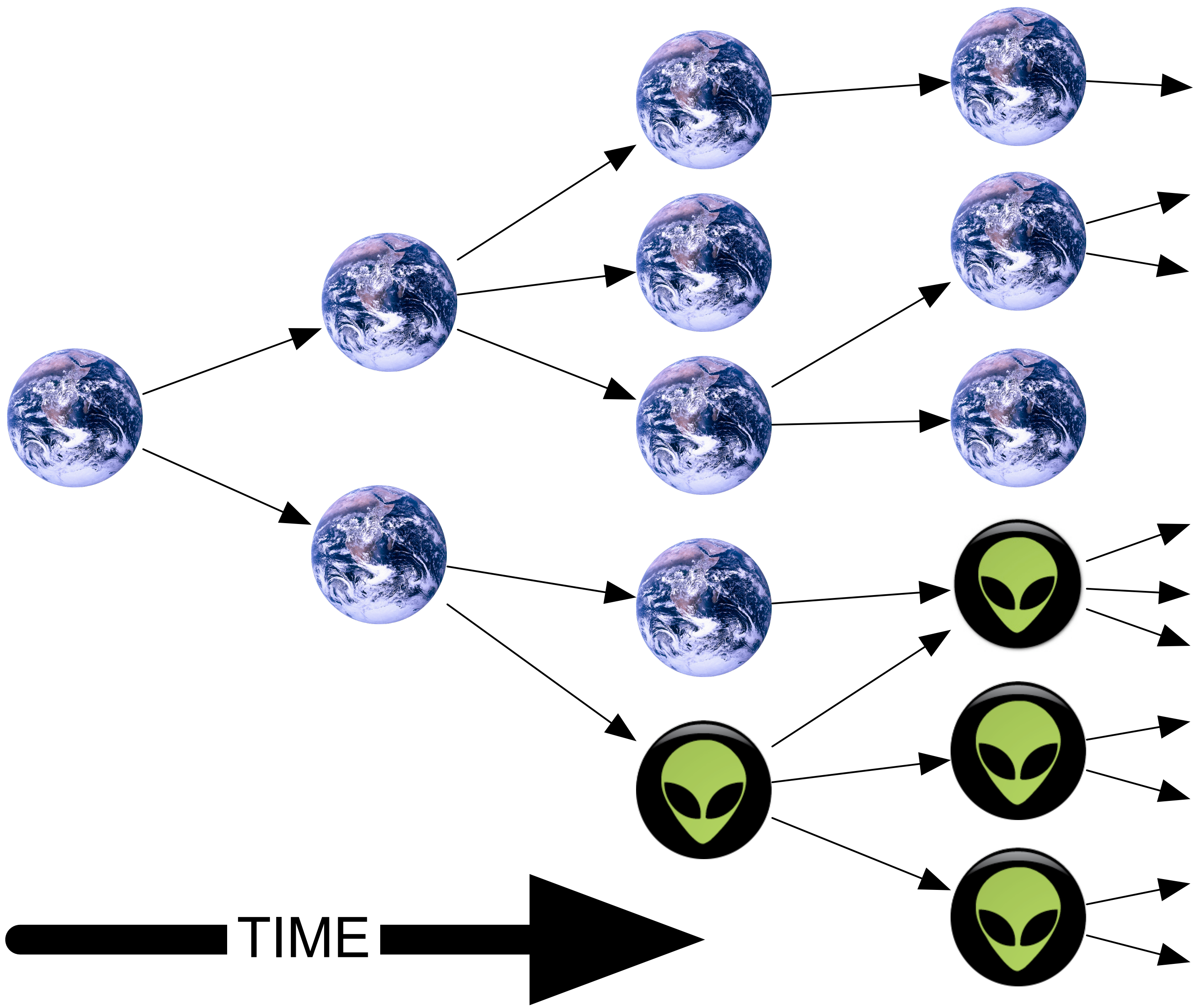
AG *Aliens*



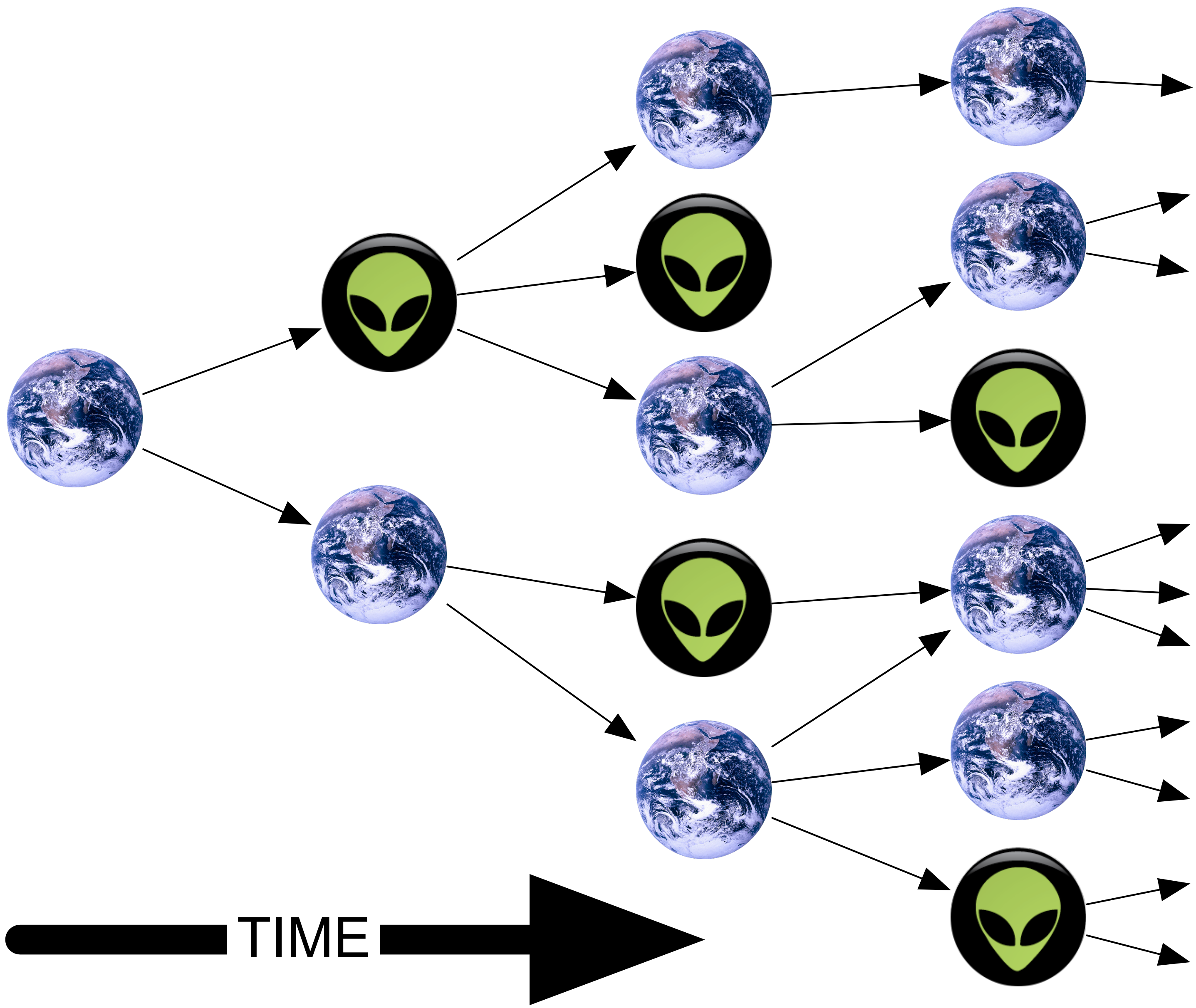
AG(not $Aliens$)



$\mathsf{EF}(\mathsf{AG}\ Aliens)$



AG(EF $Aliens$)



Model Checkers (Spin, SMV, Uppaal,. . .)

Particularly successful in hardware verification and protocol verification.

- Start with finite-state systems
  - Explicit states ($n = 3$) vs. symbolic states ($3 < x \leq 4$)
  - "Finite" includes millions or billions of states, or more
  - Transitions can be bits of computer code

- Automatically verify properties, using
  - Exhaustive search of reachable states (as necessary)
  - Clever representations (e.g., BDDs)
  - Abstractions

- Need a language of properties: usually some form of temporal logic

CSL: Checkable Sequence Language

with Bob Keller, Heather Justice (HMC '09), Daniel Furlong (HMC '11), Andrew Carter (HMC '12), Yu-Wen Tung (JPL)

- Spacecraft have constraints:
  - "Don't run both heaters at the same time"
  - "Never open the hatch during the launch phase"
  - "The antenna must be recalibrated at least once per hour"
  - "Don't power the thruster for more than one hour"

CSL: Checkable Sequence Language

with Bob Keller, Heather Justice (HMC '09), Daniel Furlong (HMC '11), Andrew Carter (HMC '12), Yu-Wen Tung (JPL)

- Spacecraft have constraints:
  - "Don't run both heaters at the same time"
  - "Never open the hatch during the launch phase"
  - "The antenna must be recalibrated at least once per hour"
  - "Don't power the thruster for more than one hour"

- Current JPL strategy: simulate each command sequence
  - Ignores nondeterminism/unknowns/failure modes

CSL: Checkable Sequence Language

with Bob Keller, Heather Justice (HMC '09), Daniel Furlong (HMC '11), Andrew Carter (HMC '12), Yu-Wen Tung (JPL)

- Spacecraft have constraints:
  - "Don't run both heaters at the same time"
  - "Never open the hatch during the launch phase"
  - "The antenna must be recalibrated at least once per hour"
  - "Don't power the thruster for more than one hour"
- Current JPL strategy: simulate each command sequence
  - Ignores nondeterminism/unknowns/failure modes
- CSL: a (Java-like) domain-specific language designed
  1. for spacecraft simulations
  2. to be model-checkable

Linear Logic: A Logic of Resources

Compare

If $\$10 \rightarrow$ book and $\$10 \rightarrow$ movie ticket, then
$\$10 + \$10 \rightarrow$ (book and movie ticket).

Compare

If $\$10 \rightarrow$ book and $\$10 \rightarrow$ movie ticket, then
$\$10 + \$10 \rightarrow$ (book and movie ticket).

If $p \rightarrow q$ and $p \rightarrow r$, then
$p \rightarrow (q \text{ and } r)$

Compare

$$\text{If } \$10 \to \text{book and } \$10 \to \text{movie ticket, then}$$
$$\$10 + \$10 \to (\text{book and movie ticket}).$$

$$\text{If } p \to q \text{ and } p \to r, \text{ then}$$
$$p \to (q \text{ and } r)$$

$$\text{If } \$10 \to \text{book and } \$10 \to \text{movie ticket, then}$$
$$\$10 \to (\text{book and movie ticket}).$$

Compare

If $\$10 \rightarrow$ book and $\$10 \rightarrow$ movie ticket, then
$\$10 + \$10 \rightarrow$ (book and movie ticket).

If $p \rightarrow q$ and $p \rightarrow r$, then
$p \rightarrow (q \text{ and } r)$

If $\$10 \rightarrow$ book and $\$10 \rightarrow$ movie ticket, then
$\$10 \rightarrow$ (book and movie ticket).

[If first prize in a raffle is a book; second prize a ticket]
winning-ticket $\rightarrow$ (book or movie ticket).

Linear Connectives

$p \otimes q$	Both p and q simultanously
$p \,\&\, q$	One of p or q (your choice)
$p \oplus q$	One of p or q (not your choice)
$p \multimap q$	q follows if I use p exactly once
$!p$	Zero or more copies of p, as needed

An Example (due to Patrick Lincoln)

Fixed-Price Menu: $5

Hamburger

Coke

Fries (All-you-can-eat)

Onion Soup or Salad

Dessert of the Day (Pie or Ice Cream)

An Example (due to Patrick Lincoln)

Fixed-Price Menu: $5

Hamburger

Coke

Fries (All-you-can-eat)

Onion Soup or Salad

Dessert of the Day (Pie or Ice Cream)

$$D \otimes D \otimes D \otimes D \otimes D$$

$$\multimap$$

$$H \otimes C \otimes !F \otimes (O \,\&\, S) \otimes (P \oplus I)$$

An Example (due to Patrick Lincoln)

Fixed-Price Menu: $5
Hamburger
Coke
Fries (All-you-can-eat)
Onion Soup or Salad
Dessert of the Day (Pie or Ice Cream)

$$D \otimes D \otimes D \otimes D \otimes D$$
$$\multimap$$
$$H \otimes C \otimes !F \otimes (O \mathbin{\&} S) \otimes (P \oplus I)$$

Note: The US Government gets to assume $!D$. You don't.

Sample Applications

Linear type systems for programming languages: make "proper" use of resources or your program won't compile/run:

- Memory used must be relinquished
- Disk files opened must be closed
- Resources cannot be ignored: use or release.

Describing Stateful Computation: A piece of program code like

```
x = x + 1
```

can be modeled as a function

$$\text{Memory} \to \text{Memory}$$

or, more accurately,

$$\text{Memory} \multimap \text{Memory}$$

Other applications: concurrency, linguistics, ...