Alternative Logics, Continued: Constructive, Linear, and Temporal Logic

CS 81

February 9, 2011

Recall: Constructive Logic

The BHK interpretation leads to constructive logic (a.k.a. intuitionistic logic). Essentially, classical logic without the law of the excluded middle.

$$\frac{}{P \vee \neg P}$$

As a consequence, we no longer have

$$\frac{\neg\neg P}{P} \qquad \frac{\begin{array}{c}\neg P\\ \vdots\\ \bot\end{array}}{P}$$

Constructive/Computable Mathematics

Key observations:

- ✓ As long as you limit yourself to constructive reasoning, then you can build *self-consistent* mathematical “worlds” that disagree with classical mathematics
  - E.g., $\forall x \in \mathbb{R}.\ (x = 0) \vee (x \neq 0)$ is neither provable nor refutable.
  - E.g., *every* function $\mathbb{R} \to \mathbb{R}$ is continuous

Constructive/Computable Mathematics

Key observations:

- ✓ As long as you limit yourself to constructive reasoning, then you can build *self-consistent* mathematical "worlds" that disagree with classical mathematics
  - E.g., $\forall x \in \mathbb{R}.\ (x = 0) \vee (x \neq 0)$ is neither provable nor refutable.
  - E.g., *every* function $\mathbb{R} \to \mathbb{R}$ is continuous
- ✓ Such worlds are consistent with what we can actually compute!
  - We can't code up

    ```
    bool equalsZero(real_t x) { ... }
    ```

    that always works, for any reasonable representation of real numbers.
  - Every function of

    ```
    real_t f(real_t x) { ... }
    ```

    is continuous (as long as it's a "total" "function" "on the reals".)

The RZ System

Program developed in collaboration 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.

Upshot: A translation of constructive logic into normal programmer-speak.

Formal Basis: Realizability

Realizability dates back to Stephen 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
  $\forall x \in \mathbb{R}. \exists y \in \mathbb{R}. x = y \times y$
  is its "constructive" content:

```
real_t sqrt(real_t x);
  // forall x:real_t,  x = sqrt(x) * sqrt(x)
```

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

RZ Example: A Decidable Set

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

```
class myset;
bool decide(myset x, myset y);
     // forall a b : ||s||,
     //    if decide a b then
     //       a == b
     //    else
     //       not (a == b)
```

RZ Example: Finitely enumerable sets

The axioms (not shown) imply a data structure for 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) imply a data structure for 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) imply a data structure for 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 to code up a "set" of exact real numbers

RZ Example: Finitely enumerable sets

The axioms (not shown) imply a data structure for 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 to code up a "set" of exact real numbers
- ✓ Only possibility: unordered list, possibly with duplicates (why?)

RZ Example: Finitely enumerable sets

The axioms (not shown) imply a data structure for 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 to code up a "set" of exact real numbers
- ✓ Only possibility: unordered list, possibly with duplicates (why?)
- ✓ Can't reliably compute the sum (not idempotent)

RZ Example: Finitely enumerable sets

The axioms (not shown) imply a data structure for 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 to code up a "set" of exact real numbers
- ✓ Only possibility: unordered 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) imply a data structure for 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 to code up a "set" of exact real numbers
- ✓ Only possibility: unordered 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) imply a data structure for 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 to code up a "set" of exact real numbers
- ✓ Only possibility: unordered 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) imply a data structure for 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 to code up a "set" of exact real numbers
- ✓ Only possibility: unordered 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

And Now for Something Completely Different

Curry-Howard Isomorphism: PL theorists know that rules for type-checking are isomorphic to constructive proof rules.

$$\frac{a : T \qquad b : U}{(a, b) : T \times U} \qquad \frac{e : T \times U}{\mathsf{fst}(e) : T} \qquad \frac{e : T \times U}{\mathsf{snd}(e) : U}$$

$$\frac{P \qquad Q}{P \wedge Q} \qquad \frac{P \wedge Q}{P} \qquad \frac{P \wedge Q}{Q}$$

Practical application: Proofs-as-programs

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
$p \rightarrow (q$ and $r)$

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
$p \rightarrow (q$ and $r)$

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

Compare

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

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

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

If $\$10 \to$ book and $\$10 \to$ movie ticket, then
$\$10 \to$ (book or 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
$p \rightarrow (q$ and $r)$

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

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

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

Some 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*

$$\frac{\_\_\_\_, A, B \vdash \cdots}{\_\_\_\_, A \otimes B \vdash \cdots}$$

$$\frac{\_\_\_\_, A \vdash \cdots}{\_\_\_\_, A \& B \vdash \cdots} \qquad \frac{\_\_\_\_ B \vdash \cdots}{\_\_\_\_, A \& B \vdash \cdots}$$

$$\frac{\_\_\_\_, A \vdash \cdots \qquad \_\_\_\_, B \vdash \cdots}{\_\_\_\_, A \oplus B \vdash \cdots}$$

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)

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

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

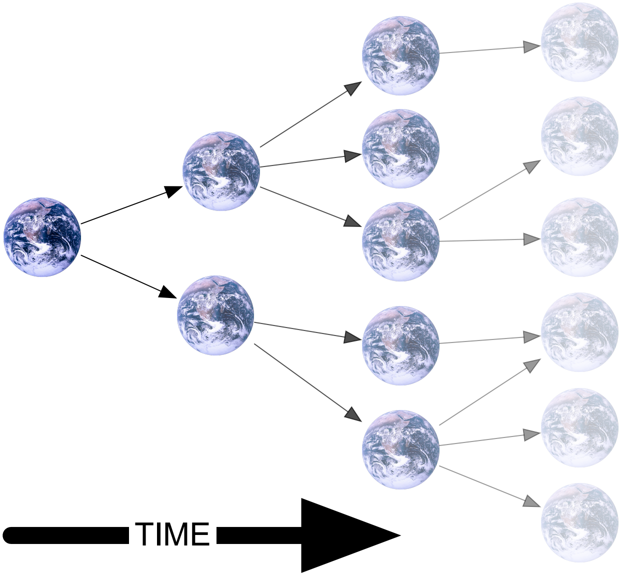
$$\text{Memory} \rightarrow \text{Memory}$$

or, more accurately,

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

Other applications: concurrency, linguistics, ...

Modal Logic: Logics of Possible Worlds

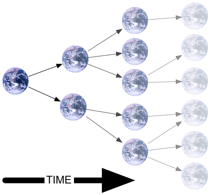


Propositions About Aliens

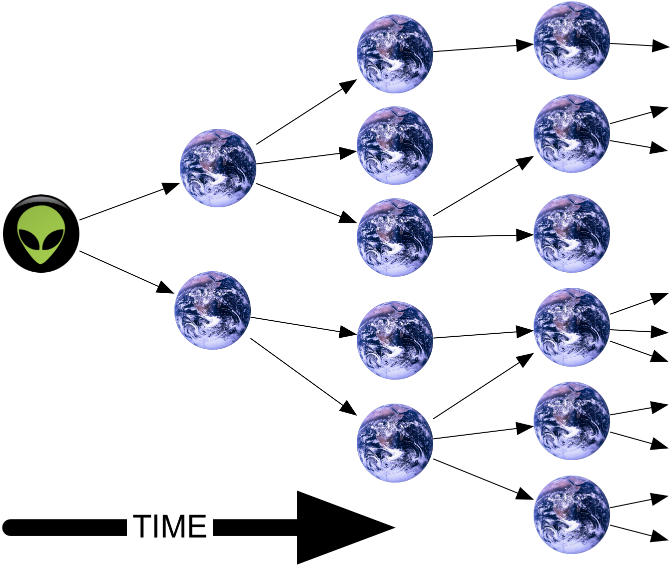
- ✓ Aliens are among us (right now).
- ✓ Aliens will always be among us.
- ✓ Aliens will *eventually* be among us.
- ✓ Aliens could *eventually* be always among us.
- ✓ Aliens will always be among us.
- ✓ If faster-than-light travel is invented, aliens will *eventually* be among us.

New Quantifiers

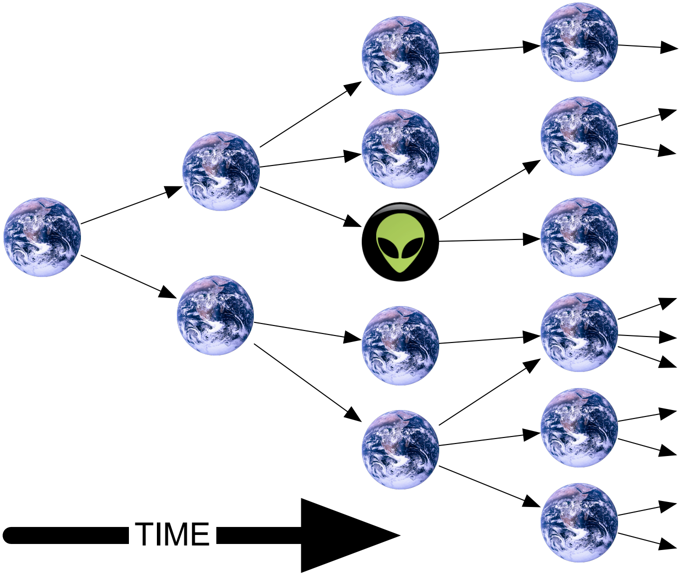
	Eventually True	Invariably True
Some Path	EF p	EG p
	Possibly	Potentially Always
All Paths	AF p	AG p
	Inevitably	Invariably



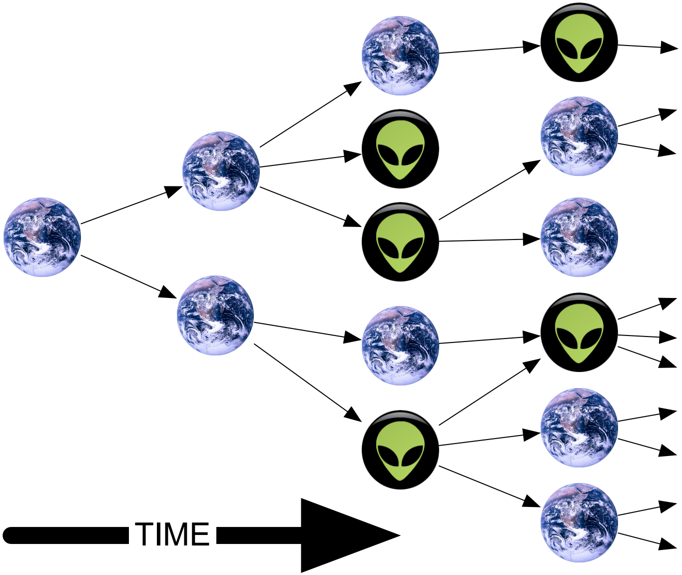
Aliens



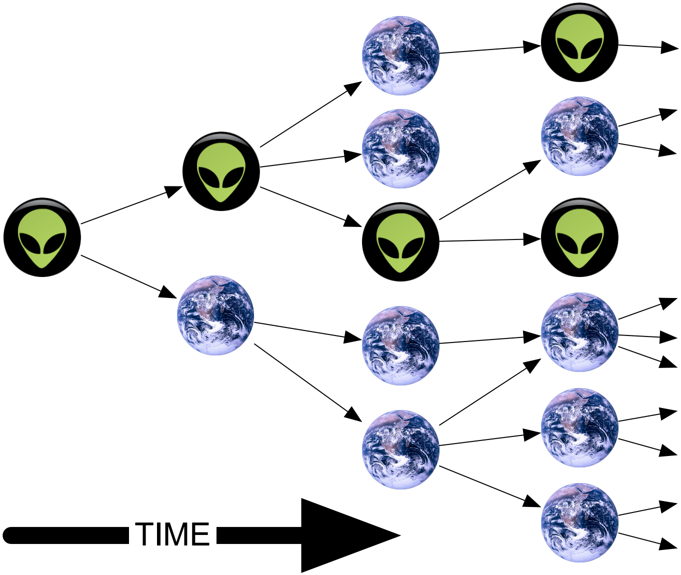
EF *Aliens*



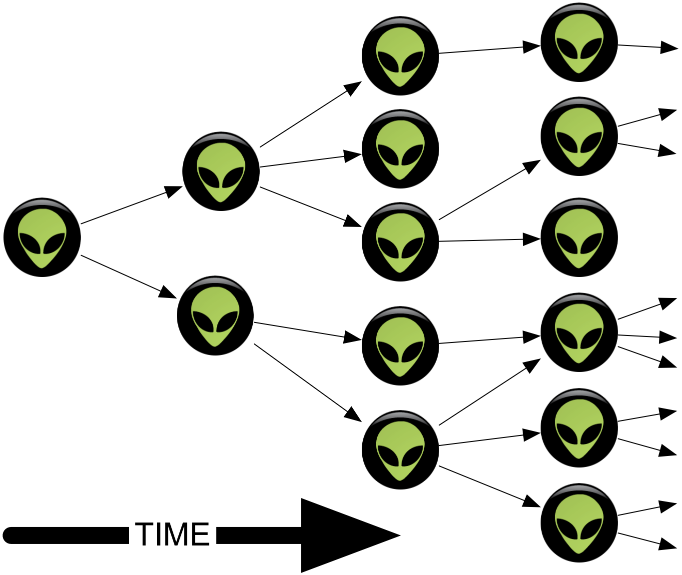
AF *Aliens*



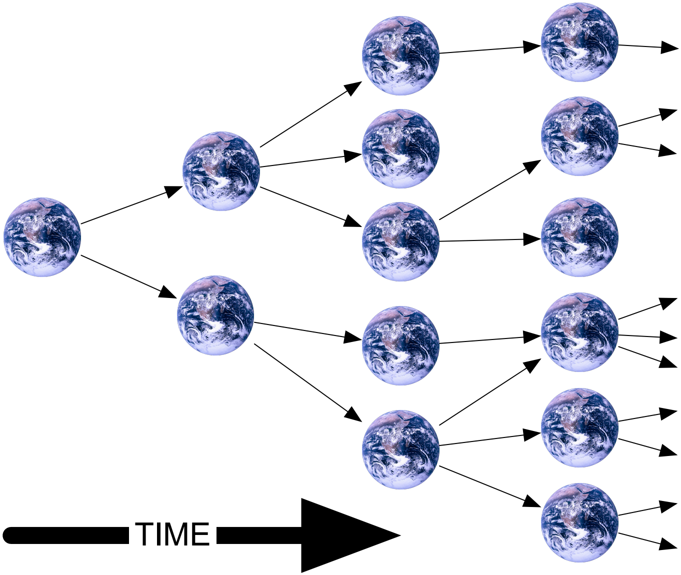
EG *Aliens*



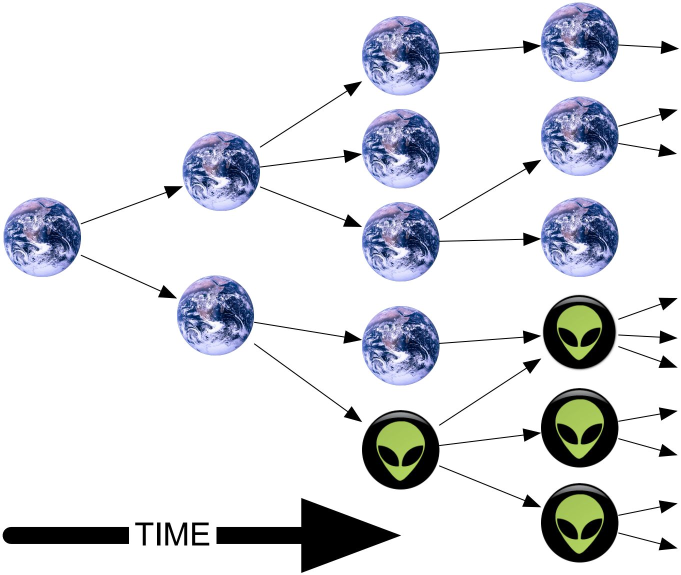
AG *Aliens*



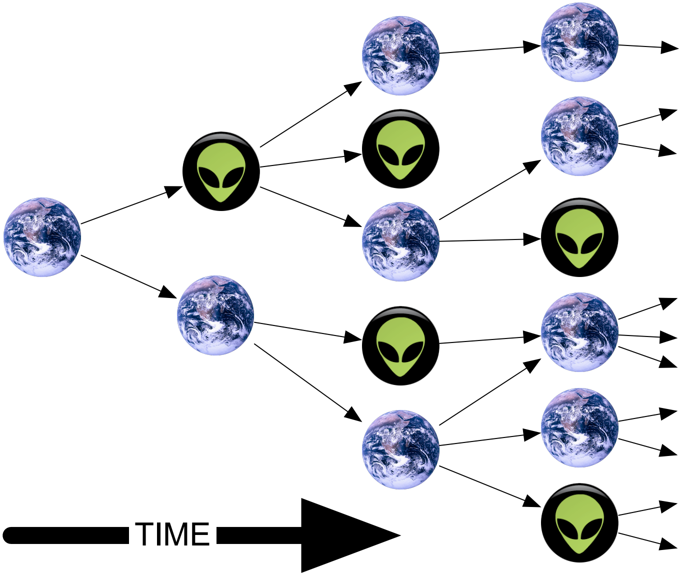
AG(not $Aliens$)



EF(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

Exercise

Controller for a traffic light:

$M(c) \longleftrightarrow$ light for the main road is showing color c.

$S(c) \longleftrightarrow$ light for the side road is showing color c.

$W \longleftrightarrow$ sensor is detecting a car waiting on the side road

---

1. The main road and side road never show green at the same time.
2. Whenever a car waits on the side road, the side-road light will eventually turn green.
3. Regardless of what happens (e.g., on the side road), the main-road light can never become permanently stuck on red.

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