# Alternative Logics 1: Linear Logic and Temporal Logic

October 8, 2012

CS 81

# 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
$p \rightarrow (q \text{ and } r)$

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

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

- ✓ Resources cannot be ignored: use or release.
- ✓ All memory allocated must be relinquished
- ✓ All disk files opened must be closed

**Describing Stateful Computation**: A piece of program code like

```
x = x + 1
```

can be modeled as a function

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

# Sample Applications

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

- ✓ Resources cannot be ignored: use or release.
- ✓ All memory allocated must be relinquished
- ✓ All disk files opened must be closed

**Describing Stateful Computation**: A piece of program code like

```
x = x + 1
```

can be modeled as a function

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

but perhaps

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

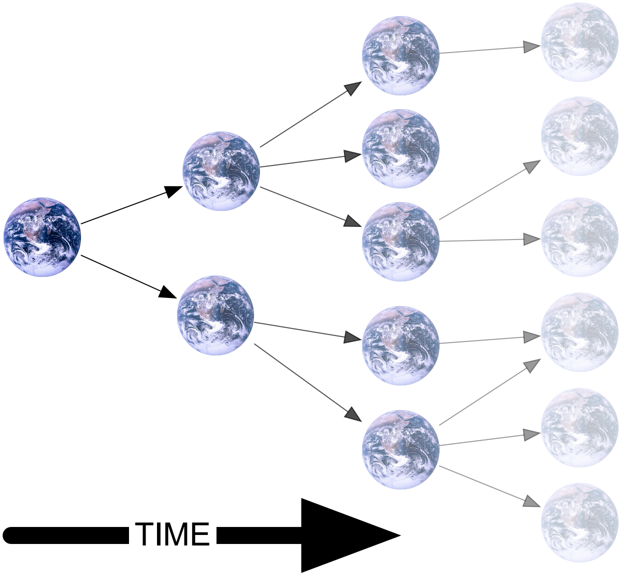
would be more accurate?

Other applications: concurrency, linguistics, ...

# Going the Other Direction

Constructive Logic results from

# Modal Logic: Logics of Possible Worlds

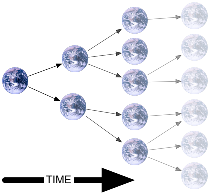


# Propositions About Aliens

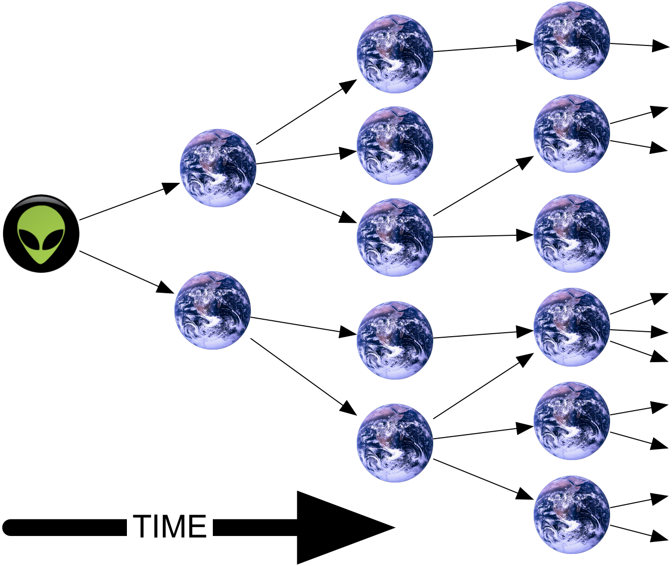
- ✓ Aliens are among us (right now).
- ✓ Aliens will always be among us.
- ✓ Aliens will *eventually* be among us.
- ✓ Aliens could *eventually* be always among us.
- ✓ Aliens will always be among us.
- ✓ If faster-than-light travel is invented, aliens will *eventually* be among us.

# New Quantifiers

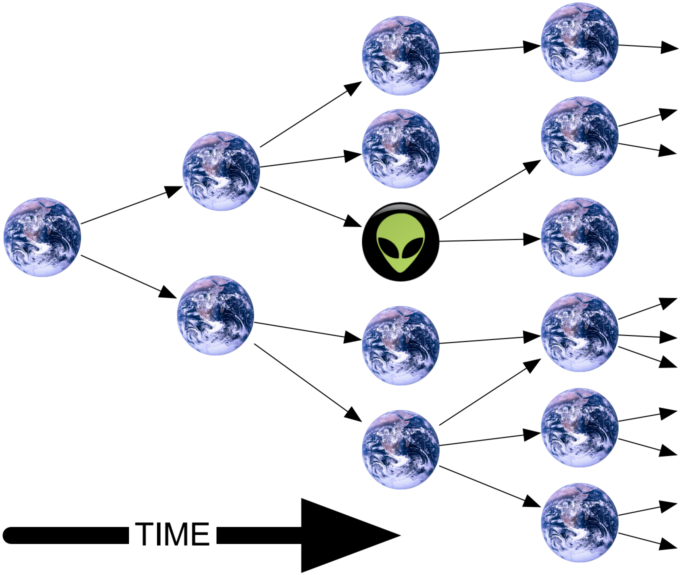
| | Eventually True | Invariably True |
|---|---|---|
| Some Path | EF $p$<br>Possibly | EG $p$<br>Potentially Always |
| All Paths | AF $p$<br>Inevitably | AG $p$<br>Invariably |



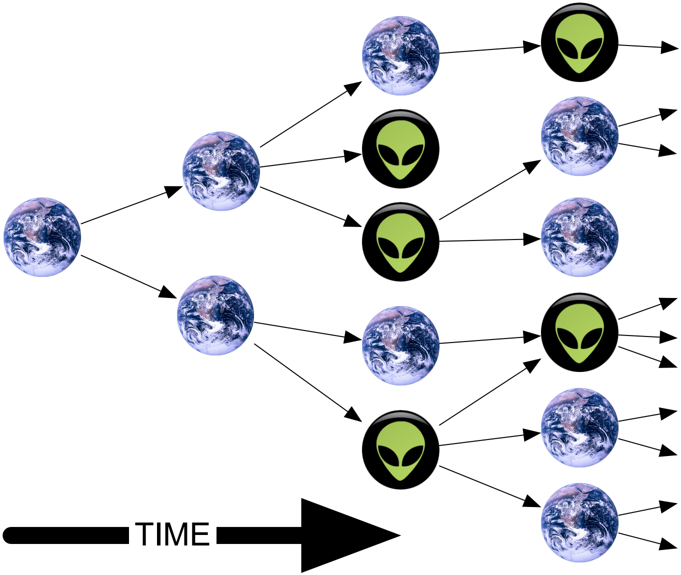
# Aliens



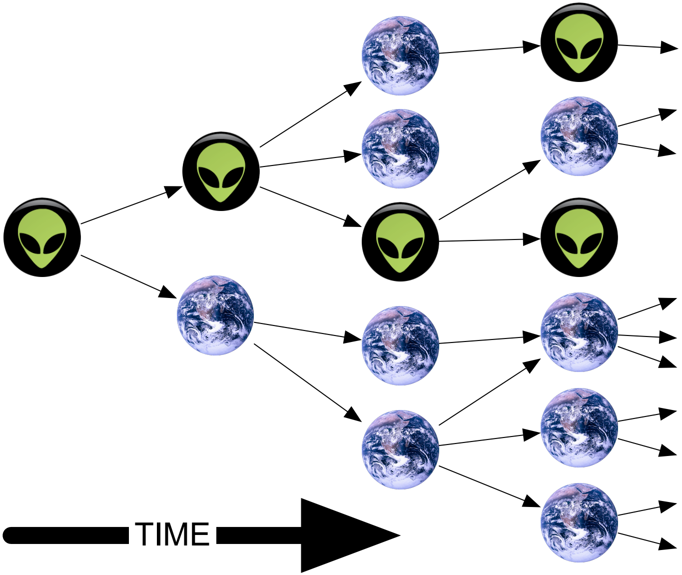
# EF Aliens



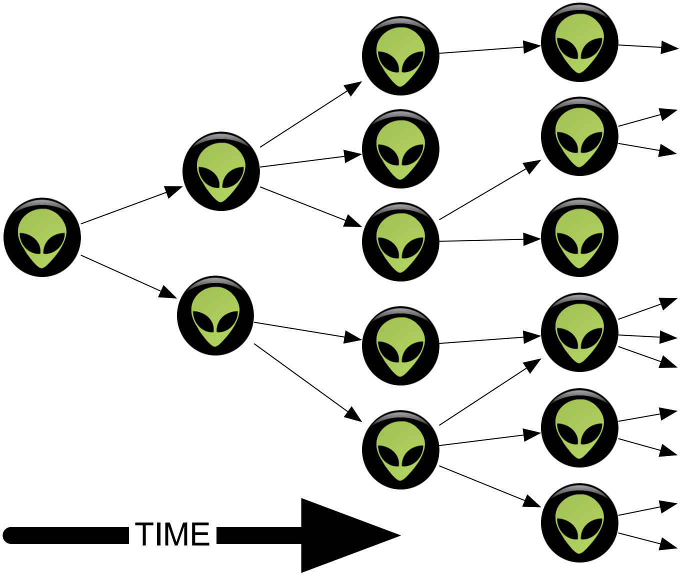
# AF Aliens



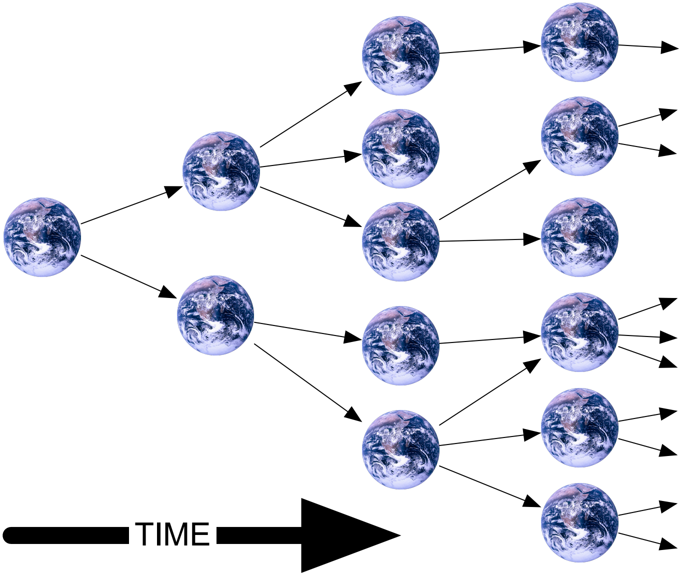
# EG Aliens



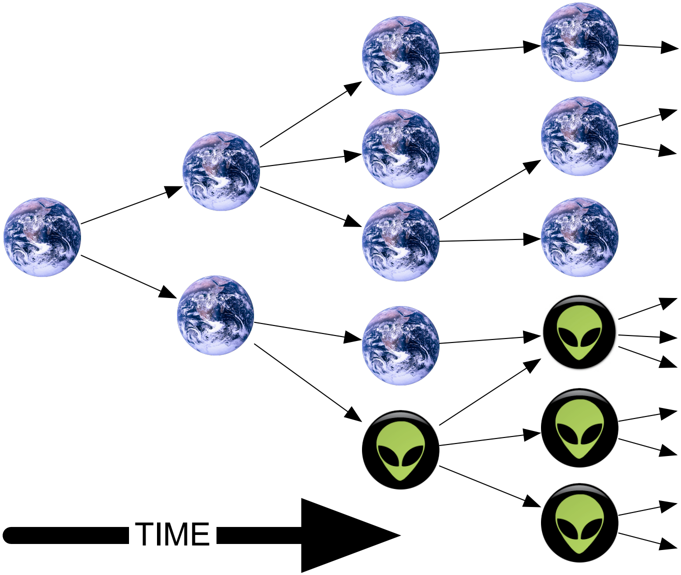
# AG Aliens



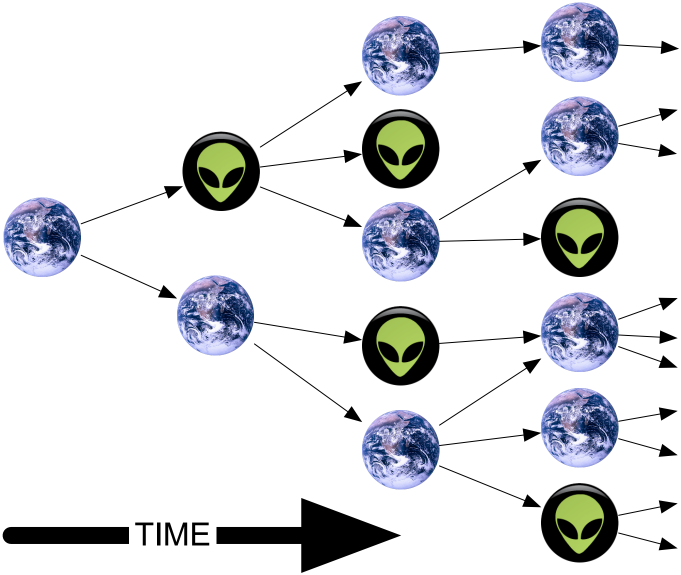
# AG(not Aliens)



# EF(AG Aliens)



# AG(EF Aliens)



| | Eventually True | Invariably True |
|---|---|---|
| Some Path | EF $p$<br>Possibly | EG $p$<br>Potentially Always |
| All Paths | AF $p$<br>Inevitably | AG $p$<br>Invariably |

- ✓ It is possible that this computer is about to crash

| | Eventually True | Invariably True |
|---|---|---|
| Some Path | EF $\mathbf{p}$<br>Possibly | EG $\mathbf{p}$<br>Potentially Always |
| All Paths | AF $\mathbf{p}$<br>Inevitably | AG $\mathbf{p}$<br>Invariably |

- ✓ It is possible that this computer is about to crash
- ✓ It is possible that this computer is about to crash

| | Eventually True | Invariably True |
|---|---|---|
| Some Path | EF $\mathbf{p}$ <br> Possibly | EG $\mathbf{p}$ <br> Potentially Always |
| All Paths | AF $\mathbf{p}$ <br> Inevitably | AG $\mathbf{p}$ <br> Invariably |

- ✓ It is possible that this computer is about to crash
- ✓ It is possible that this computer is about to crash
- ✓ It is always possible that this computer is about to crash

| | Eventually True | Invariably True |
|---|---|---|
| Some Path | EF $\mathbf{p}$ | EG $\mathbf{p}$ |
| | Possibly | Potentially Always |
| All Paths | AF $\mathbf{p}$ | AG $\mathbf{p}$ |
| | Inevitably | Invariably |

- ✓ It is possible that this computer is about to crash
- ✓ It is possible that this computer is about to crash
- ✓ It is always possible that this computer is about to crash
- ✓ Eventually this computer will crash.

| | Eventually True | Invariably True |
|---|---|---|
| Some Path | EF $\mathbf{p}$<br>Possibly | EG $\mathbf{p}$<br>Potentially Always |
| All Paths | AF $\mathbf{p}$<br>Inevitably | AG $\mathbf{p}$<br>Invariably |

- ✓ It is possible that this computer is about to crash
- ✓ It is possible that this computer is about to crash
- ✓ It is always possible that this computer is about to crash
- ✓ Eventually this computer will crash.
- ✓ This computer will never crash.

| | Eventually True | Invariably True |
| --- | --- | --- |
| Some Path | EF $\mathbf{p}$<br>Possibly | EG $\mathbf{p}$<br>Potentially Always |
| All Paths | AF $\mathbf{p}$<br>Inevitably | AG $\mathbf{p}$<br>Invariably |

- ✓ It is *possible* that this computer is about to crash
- ✓ It is *possible* that this computer is about to crash
- ✓ It is *always possible* that this computer is about to crash
- ✓ *Eventually* this computer will crash.
- ✓ This computer will *never* crash.
- ✓ If I *ever* pour Coke on the keyboard, the computer will *eventually* crash

# 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 $p$<br>Possibly | EG $p$<br>Potentially Always |
| All Paths | AF $p$<br>Inevitably | AG $p$<br>Invariably |

# 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 ($3 < x \leqslant 4$)
  - "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

# Current Practice for Shared-Memory

Preemptive Scheduling

- ✓ Each thread gets the entire computer for $N$ ms.
- ✓ Programs may be interrupted at any point.
- ✓ Compatible with one CPU or many.
- ✓ Requires defensive programming!

# Current Practice for Shared-Memory

Preemptive Scheduling

- ✓ Each thread gets the entire computer for N ms.
- ✓ Programs may be interrupted at any point.
- ✓ Compatible with one CPU or many.
- ✓ Requires defensive programming!

```
int x = 0;
```

```
r1 = x;
x = r1 + 1;
```

```
r2 = x;
x = r2 + 1;
```

# Current Practice for Shared-Memory

Preemptive Scheduling

- ✓ Each thread gets the entire computer for N ms.
- ✓ Programs may be interrupted at any point.
- ✓ Compatible with one CPU or many.
- ✓ Requires defensive programming!

```
int x = 0;

r1 = x;
x = r1 + 1;
                r2 = x;
                x = r2 + 1;

x == 2
```

# Current Practice for Shared-Memory

Preemptive Scheduling

- ✓ Each thread gets the entire computer for N ms.
- ✓ Programs may be interrupted at any point.
- ✓ Compatible with one CPU or many.
- ✓ Requires defensive programming!

```
int x = 0;
```

```
                          r2 = x;
                          x = r2 + 1;
r1 = x;
x = r1 + 1;
```

```
x == 2
```

# Current Practice for Shared-Memory

Preemptive Scheduling

- ✓ Each thread gets the entire computer for N ms.
- ✓ Programs may be interrupted at any point.
- ✓ Compatible with one CPU or many.
- ✓ Requires defensive programming!

```
int x = 0;

r1 = x;
                    r2 = x;
x = r1 + 1;
                    x = r2 + 1;

x == 1
```

# Puzzle

What are the possible final values of x, if each thread loops 10 times?

```
int x = 0;
```

```
for (int i=0; i<10; ++i) {
   r1 = x;
   x = r1 + 1;
}
```

```
for (int j=0; j<10; ++j) {
   r2 = x;
   x = r2 + 1;
}
```