Alternative Logics 1: Linear Logic and Temporal Logic

October 8, 2012
CS 81
Linear Logic: A Logic of Resources
If $10 \rightarrow \text{book}$ and $10 \rightarrow \text{movie ticket}$, then 
$10 + 10 \rightarrow (\text{book and movie ticket})$. 
**Compare**

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

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

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

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
$10 \rightarrow (\text{book and movie ticket})$. 


**Compare**

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

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

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

If $10 \rightarrow \text{book and } 10 \rightarrow \text{movie ticket}$, then 
$10 \rightarrow (\text{book or movie ticket})$?
**COMPARE**

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

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

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

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

[If first prize in a raffle is a book; second prize a ticket] winning-number $\rightarrow (\text{book or movie ticket})$. 
**Some Linear Connectives**

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

\[
\begin{array}{c}
\frac{}{\cdot, A, B \vdash \cdots} \\
\frac{}{\cdot, A \otimes B \vdash \cdots}
\end{array}
\]

\[
\begin{array}{c}
\frac{}{\cdot, A \vdash \cdots} \\
\frac{}{\cdot, A \& B \vdash \cdots}
\end{array} \quad \begin{array}{c}
\frac{}{\cdot, B \vdash \cdots} \\
\frac{}{\cdot, A \& B \vdash \cdots}
\end{array}
\]

\[
\begin{array}{c}
\frac{}{\cdot, A \vdash \cdots} \\
\frac{}{\cdot, B \vdash \cdots} \\
\frac{}{\cdot, A \oplus B \vdash \cdots}
\end{array}
\]

\[
\begin{array}{c}
\frac{}{\cdot, A \vdash \cdots} \\
\frac{}{\cdot, A \& B \vdash \cdots}
\end{array}
\]

\[
\begin{array}{c}
\frac{}{\cdot, A \vdash \cdots} \\
\frac{}{\cdot, B \vdash \cdots} \\
\frac{}{\cdot, A \oplus B \vdash \cdots}
\end{array}
\]
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)

\[ D \otimes D \otimes D \otimes D \otimes D \rightarrow H \otimes C \otimes !F \otimes (O \& S) \otimes (P \oplus I) \]
**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)

\[
\begin{align*}
D \otimes D & \otimes D \otimes D \otimes D \\
\Rightarrow & \\
H \otimes C \otimes !F \otimes (O \& S) \otimes (P \oplus I)
\end{align*}
\]

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

Memory → 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

Memory → Memory

but perhaps

Memory → 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

<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Some Path</strong></td>
<td>$\text{EF } p$</td>
<td>$\text{EG } p$</td>
</tr>
<tr>
<td></td>
<td>Possibly</td>
<td>Potentially Always</td>
</tr>
<tr>
<td><strong>All Paths</strong></td>
<td>$\text{AF } p$</td>
<td>$\text{AG } p$</td>
</tr>
<tr>
<td></td>
<td>Inevitably</td>
<td>Invariably</td>
</tr>
</tbody>
</table>

![Diagram showing the flow of time with quantifiers and paths]
ALIENS
EF Aliens
AF Aliens
EG Aliens
AG Aliens
AG(not Aliens)
$\text{EF}(\text{AG Aliens})$
AG(EF Aliens)
<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some Path</td>
<td>EF $p$</td>
<td>EG $p$</td>
</tr>
<tr>
<td></td>
<td>Possibly</td>
<td>Potentially Always</td>
</tr>
<tr>
<td>All Paths</td>
<td>AF $p$</td>
<td>AG $p$</td>
</tr>
<tr>
<td></td>
<td>Inevitably</td>
<td>Invariably</td>
</tr>
</tbody>
</table>

✓ It is possible that this computer is about to crash
<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some Path</td>
<td>EF $p$</td>
<td>EG $p$</td>
</tr>
<tr>
<td></td>
<td>Possibly</td>
<td>Potentially Always</td>
</tr>
<tr>
<td>All Paths</td>
<td>AF $p$</td>
<td>AG $p$</td>
</tr>
<tr>
<td></td>
<td>Inevitably</td>
<td>Invariably</td>
</tr>
</tbody>
</table>

✓ It is possible that this computer is about to crash
✓ It is possible that this computer is about to crash
<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Some Path</strong></td>
<td>EF $p$</td>
<td>EG $p$</td>
</tr>
<tr>
<td></td>
<td>Possibly</td>
<td>Potentially Always</td>
</tr>
<tr>
<td><strong>All Paths</strong></td>
<td>AF $p$</td>
<td>AG $p$</td>
</tr>
<tr>
<td></td>
<td>Inevitably</td>
<td>Invariably</td>
</tr>
</tbody>
</table>

- ✓ 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
### Temporal Logic

<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some Path</td>
<td>$\text{EF } p$</td>
<td>$\text{EG } p$</td>
</tr>
<tr>
<td></td>
<td>Possibly</td>
<td>Potentially Always</td>
</tr>
<tr>
<td>All Paths</td>
<td>$\text{AF } p$</td>
<td>$\text{AG } p$</td>
</tr>
<tr>
<td></td>
<td>Inevitably</td>
<td>Invariably</td>
</tr>
</tbody>
</table>

- ✓ 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.
<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some Path</td>
<td>EF $p$</td>
<td>EG $p$</td>
</tr>
<tr>
<td></td>
<td>Possibly</td>
<td>Potentially Always</td>
</tr>
<tr>
<td>All Paths</td>
<td>AF $p$</td>
<td>AG $p$</td>
</tr>
<tr>
<td></td>
<td>Inevitably</td>
<td>Invariably</td>
</tr>
</tbody>
</table>

✓ 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.
<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some Path EF $p$</td>
<td></td>
<td>EG $p$</td>
</tr>
<tr>
<td>Possibly</td>
<td></td>
<td>Potentially Always</td>
</tr>
<tr>
<td>All Paths AF $p$</td>
<td></td>
<td>AG $p$</td>
</tr>
<tr>
<td>Inevitably</td>
<td></td>
<td>Invariably</td>
</tr>
</tbody>
</table>

✓ 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:

- \( M(c) \iff \) light for the main road is showing color \( c \).
- \( S(c) \iff \) light for the side road is showing color \( c \).
- \( W \iff \) 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.

<table>
<thead>
<tr>
<th></th>
<th>Eventually True</th>
<th>Invariably True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some Path</td>
<td>( EF p )</td>
<td>( EG p )</td>
</tr>
<tr>
<td></td>
<td>Possibly</td>
<td>Potentially Always</td>
</tr>
<tr>
<td>All Paths</td>
<td>( AF p )</td>
<td>( AG p )</td>
</tr>
<tr>
<td></td>
<td>Inevitably</td>
<td>Invariably</td>
</tr>
</tbody>
</table>
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
**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!

```plaintext
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!

```c
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!

```java
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!

```c
int x = 0;

r1 = x;

x = r1 + 1;

r2 = x;

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;
}
```