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.

\[ P \lor \neg P \]

As a consequence, we no longer have

\[ \neg \neg P \quad \vdash \quad P \quad \vdash \quad P \]
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., \( \forall x \in \mathbb{R}. (x = 0) \lor (x \neq 0) \) 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., \( \forall x \in \mathbb{R}. (x = 0) \lor (x \neq 0) \) 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) \lor 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 Enumerably 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 Enumerably 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 Enumerably 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 \text{max} or \text{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 \text{max} or \text{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.

\[
\begin{align*}
\frac{a : T \quad b : U}{(a, b) : T \times U} & \quad \frac{e : T \times U}{\text{fst}(e) : T} \\
\frac{e : T \times U}{\text{snd}(e) : U}
\end{align*}
\]

\[
\begin{align*}
\frac{P \quad Q}{P \land Q} & \quad \frac{P \land Q}{P} & \quad \frac{P \land Q}{Q}
\end{align*}
\]

Practical application: Proofs-as-programs
Linear Logic: A Logic of Resources
**COMPARE**

If $10 \rightarrow \text{book}$ and $10 \rightarrow \text{movie ticket}$, then $10 + 10 \rightarrow (\text{book and movie ticket})$. 
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
$p \rightarrow (q \text{ and } r)$
**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 $10 \rightarrow (\text{book and movie ticket})$. 
**Compare**

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

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

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

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

\[
\begin{align*}
\quad, A, B \vdash \cdots \quad & \quad, A \otimes B \vdash \cdots \\
\quad, A \vdash \cdots \quad & \quad B \vdash \cdots \quad \quad, A \& B \vdash \cdots \\
\quad, A \& B \vdash \cdots \quad & \quad, A \vdash \cdots \quad \quad, B \vdash \cdots \\
\quad, A \oplus B \vdash \cdots \\
\quad, A \oplus B \vdash \cdots
\end{align*}
\]
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)

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

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

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
\[
    \text{Memory} \to \text{Memory}
\]
or, more accurately,
\[
    \text{Memory} \rightarrow \circ \text{Memory}
\]

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

<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 of time progression](image-url)
ALIENS
EF Aliens
AF Aliens
EG Aliens
AG Aliens
$AG(\text{not Aliens})$
$EF(AG \text{ Aliens})$
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:

\[ M(c) \iff \text{light for the main road is showing color } c. \]

\[ S(c) \iff \text{light for the side road is showing color } c. \]

\[ W \iff \text{sensor is detecting a car waiting on the side road} \]

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>