Constructive Game Logic

Brandon Bohrer and André Platzer

Logical Systems Lab
Computer Science Department
Carnegie Mellon University

ESOP “2020”
Modal and Constructive Logics Prove Programs

\[ <\alpha>\phi \]
\[ \{\phi\}\alpha\{\psi\} \]

Program Logics

\[ M \rightarrow M' \]
\[ \Gamma \vdash M : \phi \]

Constructive Logics

Applications

Foundations
Modal + Constructive is Underexplored

\[\Gamma \vdash M : \phi\]

Constructive Games

\[M \Rightarrow M'\]

Key Idea: Proofs are Strategies

\[\{\phi\} \alpha \{\psi\}\]

Program Logics

\[\Gamma \vdash M : \phi\]

Constructive Logics

Applications

Foundations
Constructivity Helps Synthesis

Constructive Games

Key Idea: Proofs are Strategies

Program Logics

Applications

Foundations
Foundations and Applications are Broad

D: Degen + Werner
K: Kamide
M: Mamouras
W: Wijesekera + Nerode
T: Today
F: Followup work

High-Level Proof Language
Robot + Simulation Case Studies
Cyber-Physical Systems
Synthesis Tool

Applications
Foundations

Refinement
Big-Step Semantics
Game Logic
Proof terms
Existence + Disjunction Properties
First-Order

Constructive Semantics
Finitary Proofs
Full Constructive Story is Untold

D: Degen + Werner
K: Kamide
M: Mamouras
W: Wijesekera + Nerode
T: Today
F: Followup work

High-Level Proof Language
Robot + Simulation Case Studies
Cyber-Physical Systems
Synthesis Tool

Applications

Foundations

Construcitive Semantics
D K W

Finitary Proofs
D K M

Refinement
M*

Big-Step Semantics

First-Order
M* W

Proof terms
K

Existence + Disjunction Properties
D

Game Logic
Today’s Story is Extensive

D: Degen + Werner
K: Kamide
M: Mamouras
W: Wijesekera + Nerode
T: Today
F: Followup work

High-Level Proof Language
Robot + Simulation
Case Studies
Cyber-Physical Systems
Synthesis Tool

Applications

Foundations

Refinement
M*
Constructive Semantics
D K W

Big-Step Semantics

Game Logic

Proof terms
K

First-Order
M* W

Finitary Proofs
D K M

Existence + Disjunction Properties
D
Follow-Up Story is Even Broader

D: Degen + Werner
K: Kamide
M: Mamouras
W: Wijesekera + Nerode
T: Today
F: Followup work

High-Level Proof Language
Robot + Simulation
Case Studies
Cyber-Physical Systems
Synthesis Tool

Applications

Foundations

Refinement
Construcrive Semantics
Finitary Proofs

Big-Step Semantics
Game Logic
Proof terms
Existence + Disjunction Properties

M* F
D K W T F
D K M T F

M* W T F
Discrete Foundations Built Today

\( T \rightarrow \text{<Discrete>\( p(\mathbb{Q}) \)} \)

\( T: \) Today
\( F: \) Followup work

Applications

Foundations

Realizability + Proof Terms + Small Step
Were Followed by Continuous Systems

F: Today
F: Followup work

F  Game ≤ Program

F  <Discrete+ODE> p(\mathbb{R})

T  <Discrete> p(\mathbb{Q})

Refinements

Type theory + Big Step

Realizability + Proof Terms + Small Step

Applications

Foundations
Whose Proofchecker and Synthesizer were Implemented

T: Today
F: Followup work

Proof Checker → Proof Tree → Executable Code

High-Level Proof Scripts

T: <Discrete> p(\mathbb{Q})

F: Game ≤ Program
F: <Discrete+ODE> p(\mathbb{R})

Refinements
Type theory + Big Step
Realizability + Proof Terms + Small Step

Applications
Foundations
And Applied on Hardware and in Simulation

Robot + Simulation Case Studies

High-Level Proof Scripts → Proof Tree → Executable Code

Proof Checker → Synthesizer

Applications

Foundations

T: Today
F: Followup work

Game ≤ Program

<Discrete + ODE> p(\mathbb{R})

<Discrete> p(\mathbb{Q})

Refinements

Type theory + Big Step

Realizability + Proof Terms + Small Step
Nim is an Introductory Example

\[ \text{NIM} = \begin{cases} \{ \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; \quad ?c > 0 \}; \\ \{ \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; \quad ?c > 0 \}^d \end{cases} \]

*If \( c \in \{0, 2, 3\} \) (mod 4), the first player can achieve \( c \in \{2, 3, 4\} \).

If \( c \equiv 1 \) (mod 4), the second player can maintain \( c \equiv 1 \) (mod 4): \( c > 0 \rightarrow c \mod 4 = 1 \rightarrow [\text{Nim}](c \mod 4 = 1) \).
(Subtraction) Nim is an Introductory Example

\[
\text{Nim} = \left\{ \begin{array}{l}
\{ \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; \ ?c > 0 \}; \\
\{ \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; \ ?c > 0 \}^d \end{array} \right\}^* 
\]

Assign: Choose move
Test: Take turns

- If \( c \in \{0, 2, 3\} \mod 4 \), the first player can achieve \( c \in \{2, 3, 4\} \mod 4 \)
- If \( c \equiv 1 \mod 4 \), the second player can maintain \( c \equiv 1 \mod 4 \):

First player wins
Second player wins
(Subtraction) Nim is an Introductory Example

\[ \text{NIM} = \begin{cases}  
\{ \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0 \};  
\{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0 \}^d  
\end{cases} \]

- **Assign**: \{ \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0 \};
- **Choose move**: \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0 \}^d
- **Test**: \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0
- **Repeat**: \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0
- **Take turns**: \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0

If \( c \in \{0, 2, 3\} \mod 4 \), the first player can achieve \( c \in \{2, 3, 4\} \mod 4 \)

\[ c > 0 \rightarrow c \mod 4 \in \{0, 2, 3\} \rightarrow \langle \text{Nim} \rangle (c \mod 4 = 1) \]

If \( c \equiv 1 \mod 4 \), the second player can maintain \( c \equiv 1 \mod 4 \):

\[ c > 0 \rightarrow c \mod 4 = 1 \rightarrow [\text{Nim}] (c \mod 4 = 1) \]
Nim is an Introductory Example

\[
\text{Nim} = \left\{ \begin{array}{l}
\{ \{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0 \}; \\
\{ c := c - 1 \cup c := c - 2 \cup c := c - 3 \}; ?c > 0 \}^d
\end{array} \right.^*
\]

- If \( c \in \{0, 2, 3\} \pmod{4} \), the first player can achieve \( c \in \{2, 3, 4\} \)

\[
c > 0 \rightarrow c \mod 4 \in \{0, 2, 3\} \rightarrow \langle \text{Nim} \rangle(c \in \{2, 3, 4\})
\]

- If \( c \equiv 1 \pmod{4} \), the second player can maintain \( c \equiv 1 \pmod{4} \):

\[
c > 0 \rightarrow c \mod 4 = 1 \rightarrow [\text{Nim}](c \mod 4 = 1)
\]
Stateful Realizers Define + Play Games

If $X : Region$ then $X \langle \langle \alpha \rangle \rangle : Region$ and $X[\alpha] : Region$.

Regions $X$ defined by $X \subseteq (Realizer \times State) \cup \{T\} \cup \{\bot\}$.

Realizers $a, b, c$ are higher-order, continuation-passing programs.
Stateful Realizers Define + Play Games

If \( X : \text{Region} \) then \( X \langle\langle \alpha \rangle\rangle : \text{Region} \) and \( X[[\alpha]] : \text{Region} \).

Regions \( X \) defined by \( X \subseteq (\text{Realizer} \times \text{State}) \cup \{T\} \cup \{\bot\} \).
Realizers \( a, b, c \) are higher-order, continuation-passing programs.
Example Angelic semantics cases:

\[
X\langle\alpha \cup \beta\rangle = X_{\{0\}}\langle\alpha\rangle \cup X_{\{1\}}\langle\beta\rangle
\]
Stateful Realizers Define + Play Games

If $X : \text{Region}$ then $X \langle \langle \alpha \rangle \rangle : \text{Region}$ and $X[[\alpha]] : \text{Region}$. Regions $X$ defined by $X \subseteq (\text{Realizer} \times \text{State}) \cup \{\top\} \cup \{\bot\}$. Realizers $a, b, c$ are higher-order, continuation-passing programs.

Example Angelic semantics cases:

$$X \langle \langle \alpha \cup \beta \rangle \rangle = X_{\{0\}} \langle \langle \alpha \rangle \rangle \cup X_{\{1\}} \langle \langle \beta \rangle \rangle$$

- $X \langle \langle ?\phi \rangle \rangle \ni (b, \omega)$ ← $((a, b), \omega) \in X$ and $(a, \omega) \notin [\phi]$  
- $X \langle \langle ?\phi \rangle \rangle \ni \bot$ ← $((a, b), \omega) \in X$ and $(a, \omega) \notin [\phi]$
Stateful Realizers Define + Play Games

If $X : Region$ then $X \langle \langle \alpha \rangle \rangle : Region$ and $X [[\alpha]] : Region$.

Regions $X$ defined by $X \subseteq (Realizer \times State) \cup \{T\} \cup \{\bot\}$.

Realizers $a, b, c$ are higher-order, continuation-passing programs.

Example Angelic semantics cases:

- $X \langle \alpha \cup \beta \rangle = X \langle 0 \rangle \langle \alpha \rangle \cup X \langle 1 \rangle \langle \beta \rangle$
- $X \langle ?\phi \rangle \ni (b, \omega) \leftarrow ((a, b), \omega) \in X$ and $(a, \omega) \in [\phi]$
- $X \langle ?\phi \rangle \ni \bot \leftarrow ((a, b), \omega) \in X$ and $(a, \omega) \not\in [\phi]$
- $X \langle x := f \rangle \ni (a, \omega[x \mapsto f(\omega)]) \leftarrow (a, \omega) \in X$

Won
Lost
Chose $\alpha$
Chose $\beta$
Modify
Stateful Realizers Define + Play Games

If $X : Region$ then $X \langle \alpha \rangle : Region$ and $X[[\alpha]] : Region$.

Regions $X$ defined by $X \subseteq (Realizer \times State) \cup \{T\} \cup \{\bot\}$.

Realizers $a, b, c$ are higher-order, continuation-passing programs.

Example Angelic semantics cases:

$X \langle \alpha \cup \beta \rangle = X_{\langle 0 \rangle} \langle \alpha \rangle \cup X_{\langle 1 \rangle} \langle \beta \rangle$

$X \langle ?\phi \rangle \ni (b, \omega)$  \quad \leftarrow ((a, b), \omega) \in X \text{ and } (a, \omega) \in [\phi]$

$X \langle ?\phi \rangle \ni \bot$  \quad \leftarrow ((a, b), \omega) \in X \text{ and } (a, \omega) \notin [\phi]$

$X \langle x := f \rangle \ni (a, \omega[x \mapsto f(\omega)])$  \quad \leftarrow (a, \omega) \in X$

$X \langle \alpha^d \rangle = X[[\alpha]]$  \quad \text{Modify}
Natural Deduction Makes Proofs Functional Programs

\[
\begin{array}{c}
\langle \cup \rangle E \\
\frac{\Gamma \vdash \langle \alpha \cup \beta \rangle \phi \quad \Gamma, \langle \alpha \rangle \phi \vdash \psi \quad \Gamma, \langle \beta \rangle \phi \vdash \psi}{\Gamma \vdash \psi}
\end{array}
\]

\[
\begin{array}{c}
\langle ? \rangle I \\
\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \langle ? \rangle \psi}
\end{array}
\]

\[
\begin{array}{c}
\langle ? \rangle E_1 \\
\frac{\Gamma \vdash \langle ? \rangle \psi}{\Gamma \vdash \phi}
\end{array}
\]

\[
\begin{array}{c}
\langle ? \rangle E_2 \\
\frac{\Gamma \vdash \langle ? \rangle \psi}{\Gamma \vdash \psi}
\end{array}
\]

\[
\begin{array}{c}
\langle \cup \rangle I_1 \\
\frac{\Gamma \vdash \langle \alpha \rangle \phi}{\Gamma \vdash \langle \alpha \cup \beta \rangle \phi}
\end{array}
\]

\[
\begin{array}{c}
\langle \cup \rangle I_2 \\
\frac{\Gamma \vdash \langle \beta \rangle \phi}{\Gamma \vdash \langle \alpha \cup \beta \rangle \phi}
\end{array}
\]

\[
\begin{array}{c}
[?] I \\
\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash [?] \psi}
\end{array}
\]

\[
\begin{array}{c}
[?] E \\
\frac{\Gamma \vdash [?] \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}
\end{array}
\]
Natural Deduction Can Prove Games

\[
\vdash [\alpha][\beta] \phi \quad \vdash [\alpha; \beta] \phi \quad \vdash [\alpha] \phi^d
\]
Natural Deduction Can Prove Games

\[
\begin{align*}
\text{[;]I} & : & \Gamma \vdash [\alpha][\beta] \phi & \quad & \text{[:=]I} & : & \Gamma, (x = f^\gamma_x) \vdash \phi \\
\quad & : & \Gamma \vdash [\alpha; \beta] \phi & \quad & \quad & \Gamma \vdash [x := f^\gamma_x] \phi \\
\text{[*]I} & : & \Gamma \vdash \psi, \psi \vdash [\alpha] \psi & \quad & \quad & \Gamma \vdash [\alpha^*] \phi \\
\quad & : & \Gamma \vdash [\alpha^*] \phi \\
\text{⟨*⟩I} & : & \Gamma \vdash \varphi, M_0 = M \succ 0 \vdash \langle \alpha \rangle (\varphi \land M_0 \succ M) & \quad & \quad & \varphi, M = 0 \vdash \phi \\
\quad & : & \Gamma \vdash \langle \alpha^* \rangle \phi
\end{align*}
\]
Proof Calculus is Sound

Theorem (Soundness of proof calculus)

Every provable sequent \((\Gamma \vdash \phi)\) is valid.
Proof Calculus is Sound

Theorem (Soundness of proof calculus)
Every provable sequent \((\Gamma \vdash \phi)\) is valid.

Lemma (Arithmetic-term substitution)
If \(\Gamma \vdash \phi\) then \(\sigma(\Gamma) \vdash \sigma(\phi)\) for admissible substitutions \(\sigma\).

Lemma (Coincidence)
The semantics of formula \(\phi\) depends only on free variables of \(\phi\).

Lemma (Bound effect)
Only bound variables of game \(\alpha\) are modified by execution.
Proofs Are *Imperative Programs*

**Lemma (Weak Existence Property)**

If $\Gamma \vdash (\exists x : Q \phi)$, there exists $f : State \to Q$ which witnesses $\phi$.

**Lemma (Weak Disjunction Property)**

If $\Gamma \vdash \phi \lor \psi$ there exists $f : State \to Bool$ which chooses a branch of $\phi \lor \psi$. In each case, $\phi$ or $\psi$ has a realizer.
Proofs Are *Imperative Programs*

**Lemma (Weak Existence Property)**

*If* $\Gamma \vdash (\exists x : Q \phi)$, *there exists* $f : \text{State} \rightarrow Q$ *which witnesses* $\phi$.

**Lemma (Weak Disjunction Property)**

*If* $\Gamma \vdash \phi \lor \psi$ *there exists* $f : \text{State} \rightarrow \text{Bool}$ *which chooses a branch of* $\phi \lor \psi$. *In each case,* $\phi$ *or* $\psi$ *has a realizer.*

**Theorem (Strategy Property for Angel's Turn)**

*If* $\Gamma \vdash \langle \alpha \rangle \phi$, *there exists a realizer that wins* $\langle \langle \alpha \rangle \rangle$ *with goal* $\phi$ *assuming* $\Gamma$ *initially.*

**Theorem (Strategy Property for Demon’s Turn)**

*If* $\Gamma \vdash [\alpha] \phi$, *there exists a realizer that wins* $[[\alpha]]$ *with goal* $\phi$ *assuming* $\Gamma$ *initially.*
Realizability Reduces Constructivity to Soundness

Lemma (Weak Existence Property)
If $\Gamma \vdash (\exists x : Q \phi)$, there exists $f : \text{State} \rightarrow Q$ which witnesses $\phi$.

Lemma (Weak Disjunction Property)
If $\Gamma \vdash \phi \lor \psi$ there exists $f : \text{State} \rightarrow \text{Bool}$ which chooses a branch of $\phi \lor \psi$. In each case, $\phi$ or $\psi$ has a realizer.

Theorem (Strategy Property for Angel’s Turn)
If $\Gamma \vdash \langle \alpha \rangle \phi$, there exists a realizer that wins $\langle \langle \alpha \rangle \rangle$ with goal $\phi$ assuming $\Gamma$ initially.

Theorem (Strategy Property for Demon’s Turn)
If $\Gamma \vdash [\alpha] \phi$, there exists a realizer that wins $[[\alpha]]$ with goal $\phi$ assuming $\Gamma$ initially.

Theorem (Soundness of proof calculus)
Every provable sequent $(\Gamma \vdash \phi)$ is valid.
Proofs Terms Show *Functional Interpretation*

- Interpret explicit proof syntax as pure functional program
- Modal separation: proof about program $\approx$ monadic program
- Application: normalize proofs to simplify further processing
Proofs Terms Show *Functional* Interpretation

- Interpret explicit proof syntax as pure functional program
- Modal separation: proof about program \( \approx \) monadic program
- Application: normalize proofs to simplify further processing

**Definition (Proof term grammar)**

\[
M, N, O ::= (\lambda p : \phi. \ M) \ | \ \langle M, N \rangle \ | \ \langle \ell \cdot M \rangle \ | \ \langle r \cdot M \rangle \\
| (M \ rep p : \psi. \ N \ in \ O) \\
| \langle \iota \ M \rangle \ | \ \langle \text{yield} \ M \rangle \ | \ \langle x := f^P_x \ in \ p. \ M \rangle
\]
Proofs Terms Show *Functional* Interpretation

- Interpret explicit proof syntax as pure functional program
- Modal separation: proof about program $\approx$ monadic program
- Application: normalize proofs to simplify further processing

**Definition (Proof term grammar)**

$$M, N, O ::= (\lambda p : \phi. M) \mid \langle M, N \rangle \mid \langle \ell \cdot M \rangle \mid \langle r \cdot M \rangle$$

$$\mid (M \text{ rep } p : \psi. N \text{ in } O)$$

$$\mid \langle \nu M \rangle \mid \langle \text{yield } M \rangle \mid \langle x := f^Y_x \text{ in } p. M \rangle$$
Probes Terms Show *Functional* Interpretation

- Interpret explicit proof syntax as pure functional program
- Modal separation: proof about program $\approx$ monadic program
- Application: normalize proofs to simplify further processing

Definition (Proof term grammar)

$$M, N, O ::= (\lambda p : \phi. \ M) \mid \langle M, N \rangle \mid \langle \ell \cdot M \rangle \mid \langle r \cdot M \rangle \mid (M \text{ rep } p : \psi. \ N \text{ in } O) \mid \langle \iota M \rangle \mid \langle \text{yield } M \rangle \mid \langle x := f^y_{\times} \text{ in } p. \ M \rangle$$
Proof Terms Execute By Simplifying

Definition (Operational semantics)
$M \rightarrow M'$ if $M$ reduces to $M'$ in one step.

Definition (Normal forms)
Normal proof terms $M$ consist of canonical forms and case analyses.
Proof Terms Execute By Simplifying

Definition (Operational semantics)

\[ M \rightarrow M' \text{ if } M \text{ reduces to } M' \text{ in one step.} \]

Definition (Normal forms)

Normal proof terms \( M \) consist of canonical forms and case analyses.

Lemma (Progress)

If \( \vdash M : \phi \), then either \( M \) is normal or \( M \rightarrow M' \) for some \( M' \).

Lemma (Preservation)

If \( \vdash M : \phi \) and \( M \rightarrow^* M' \), then \( \vdash M' : \phi \).
Propositional Connectives are an Example

\[ \lambda \phi \beta \quad (\lambda p : \phi. \ M) \ N \mapsto [N/p]M \]

\[ \pi_1 \S \quad M \mapsto M' \]

\[ [\pi_1 M] \mapsto [\pi_1 M'] \]

\[ [\pi_1] \C \quad [\pi_1 \langle \text{case } M \text{ of } \ell \Rightarrow N \mid r \Rightarrow O \rangle] \mapsto \langle \text{case } M \text{ of } \ell \Rightarrow [\pi_1 N] \mid r \Rightarrow [\pi_1 O] \rangle \]

\[ \pi_2 \S \quad M \mapsto M' \]

\[ [\pi_2 M] \mapsto [\pi_2 M'] \]

\[ [\pi_2] \C \quad [\pi_2 \langle \text{case } M \text{ of } \ell \Rightarrow N \mid r \Rightarrow O \rangle] \mapsto \langle \text{case } M \text{ of } \ell \Rightarrow [\pi_2 N] \mid r \Rightarrow [\pi_2 O] \rangle \]
These Foundations Have Been Built On

\[ <\text{Discrete} + \text{ODE}> p(\mathbb{R}) \quad \text{Type theory + Big Step} \]

\[ <\text{Discrete}> p(\mathbb{Q}) \quad \text{Realizability + Proof Terms + Small Step} \]

\[ \text{Game} \leq \text{Program} \quad \text{Refinements} \]

http://www.cs.cmu.edu/~bbohrer/