Constructive Game Logic *

Rose Bohrer[1] and André Platzer[1,2]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
aplatzer@cs.cmu.edu
[2] Fakultät für Informatik, Technische Universität München, München, Germany

Abstract. Game Logic is an excellent setting to study proofs-about-programs via the interpretation of those proofs as programs, because constructive proofs for games correspond to effective winning strategies to follow in response to the opponent's actions. We thus develop *Constructive Game Logic*, which extends Parikh's Game Logic (GL) with constructivity and with first-order programs *à la* Pratt's first-order dynamic logic (DL). Our major contributions include: 1. a novel realizability semantics capturing the adversarial dynamics of games, 2. a natural deduction calculus and operational semantics describing the computational meaning of strategies via proof-terms, and 3. theoretical results including soundness of the proof calculus w.r.t. realizability semantics, progress and preservation of the operational semantics of proofs, and Existential Properties enabling the extraction of computational artifacts from game proofs. Together, these results provide the most general account of a Curry-Howard interpretation for any program logic to date, and the first at all for Game Logic.

Keywords: Game Logic, Constructive Logic, Natural Deduction, Proof Terms

1 Introduction

Two of the most essential tools in theory of programming languages are *program logics*, such as Hoare calculi [29] and dynamic logics [45], and the *Curry-Howard correspondence* [17,31], wherein propositions correspond to types, proofs to functional programs, and proof term normalization to program evaluation. Their intersection, the Curry-Howard interpretation of program logics, has received surprisingly little study. We undertake such a study in the setting of Game Logic (GL) [38], because this leads to novel insights, because the Curry-Howard correspondence can be explained particularly intuitively for games, and because our first-order GL is a superset of common logics such as first-order Dynamic Logic (DL).

Constructivity and program verification have met before: Higher-order constructive logics [16] obey the Curry-Howard correspondence and are used to

* This research was sponsored by the AFOSR under grant number FA9550-16-1-0288. The authors were also funded by the NDSEG Fellowship and Alexander von Humboldt Foundation, respectively.

P. Müller (Ed.): ESOP 2020, LNCS 12075, pp. 84–111, 2020.
https://doi.org/10.1007/978-3-030-44914-8_4

develop verified functional programs. Program logics are also often embedded in constructive proof assistants such as Coq [48], inheriting constructivity from their metalogic. Both are excellent ways to develop verified software, but we study something else.

We study the computational content of a program logic *itself*. Every fundamental concept of computation is expected to manifest in all three of logic, type systems, and category theory [27]. Because dynamics logics (DL's) such as GL have shown that program execution is a first-class construct in modal logic, the theorist has an imperative to explore the underlying notion of computation by developing a constructive GL with a Curry-Howard interpretation.

The computational content of a proof is especially clear in GL, which generalizes DL to programmatic models of zero-sum, perfect-information games between two players, traditionally named Angel and Demon. Both normal-play and misère-play games can be modeled in GL. In classical GL, the diamond modality $\langle\alpha\rangle\phi$ and box modality $[\alpha]\phi$ say that Angel and Demon respectively have a strategy to ensure ϕ is true at the end of α, which is a model of a game. The difference between classical GL and CGL is that classical GL allows proofs that exclude the middle, which correspond to strategies which branch on undecidable conditions. CGL proofs can branch only on decidable properties, thus they correspond to strategies which are *effective* and can be executed by computer. Effective strategies are crucial because they enable the synthesis of code that implements a strategy. Strategy synthesis is itself crucial because even simple games can have complicated strategies, and synthesis provides assurance that the implementation correctly solves the game. A GL strategy resolves the choices inherent in a game: a diamond strategy specifies every move made by the Angel player, while a box strategy specifies the moves the Demon player will make.

In developing *Constructive Game Logic* (CGL), adding constructivity is a deep change. We provide a natural deduction calculus for CGL equipped with proof terms and an operational semantics on the proofs, demonstrating the meaning of strategies as functional programs and of winning strategies as functional programs that are guaranteed to achieve their objective no matter what counterstrategy the opponent follows. While the proof calculus of a constructive logic is often taken as ground truth, we go a step further and develop a realizability semantics for CGL as programs performing winning strategies for game proofs, then prove the calculus sound against it. We adopt realizability semantics in contrast to the winning-region semantics of classical GL because it enables us to prove that CGL satisfies novel properties (Section 8). The proof of our Strategy Property (Theorem 2) constitutes an (on-paper) algorithm that computes a player's (effective) strategy from a proof that they can win a game. This is the key test of constructivity for CGL, which would not be possible in classical GL. We show that CGL proofs have *two* computational interpretations: the operational semantics interpret an arbitrary proof (strategy) as a functional program which reduces to a normal-form proof (strategy), while realizability semantics interpret Angel strategies as programs which defeat arbitrary Demonic opponents.

While CGL has ample theoretical motivation, the practical motivations from synthesis are also strong. A notable line of work on dGL extends first-order GL to hybrid games to verify safety-critical adversarial cyber-physical systems [42]. We have designed CGL to extend smoothly to hybrid games, where synthesis provides the correctness demanded by safety-critical systems and the synthesis of correct monitors of the external world [36].

2 Related Work

This work is at the intersection of game logic and constructive modal logics. Individually, they have a rich literature, but little work has been done at their intersection. Of these, we are the first for GL and the first with a proofs-as-programs interpretation for a full first-order program logic.

Games in Logic. Parikh's propositional GL [38] was followed by coalitional GL [39]. A first-order version of GL is the basis of differential game logic dGL [42] for hybrid games. GL's are unique in their clear delegation of strategy to the *proof* language rather than the *model* language, crucially allowing succinct game specifications with sophisticated winning strategies. Succinct specifications are important: specifications are *trusted* because proving the *wrong theorem* would not ensure correctness. Relatives without this separation include Strategy Logic [15], Alternating-Time Temporal Logic (ATL) [5], CATL [30], Ghosh's SDGL [24], Ramanujam's structured strategies [46], Dynamic-epistemic logics [6,10,49], evidence logics [9], and Angelic Hoare logic [35].

Constructive Modal Logics. A major contribution of CGL is our constructive semantics for games, not to be confused with game semantics [1], which are used to give programs semantics *in terms of* games. We draw on work in semantics for constructive modal logics, of which two main approaches are intuitionistic Kripke semantics and realizability semantics.

An overview of Intuitionistic Kripke semantics is given by Wijesekera [52]. Intuitionistic Kripke semantics are parameterized over worlds, but in contrast to classical Kripke semantics, possible worlds represent what is currently *known* of the state. Worlds are preordered by $w_1 \geq w_2$ when w_1 contains at least the knowledge in w_2. Kripke semantics were used in Constructive Concurrent DL [53], where both the world and knowledge of it change during execution. A key advantage of realizability semantics [37,33] is their explicit interpretation of constructivity as computability by giving a *realizer*, a program which witnesses a fact. Our semantics combine elements of both: Strategies are represented by realizers, while the game state is a Kripke world. Constructive set theory [2] aids in understanding which set operations are permissible in constructive semantics.

Modal semantics have also exploited mathematical structures such as: i) Neighborhood models [8], topological models for spatial logics [7], and temporal logics of dynamical systems [20]. ii) Categorical [3], sheaf [28], and pre-sheaf [23] models. iii) Coalgebraic semantics for classical Propositional Dynamic Logic

(PDL) [19]. While games are known to exhibit algebraic structure [25], such laws are not essential to this work. Our semantics are also notable for the seamless interaction between a constructive Angel and a classical Demon.

CGL is first-order, so we must address the constructivity of operations that inspect game state. We consider rational numbers so that equality is decidable, but our work should generalize to constructive reals [11,13].

Intuitionistic modalities also appear in dynamic-epistemic logic (DEL) [21], but that work is interested primarily in proof-theoretic semantics while we employ realizability semantics to stay firmly rooted in computation. Intuitionistic Kripke semantics have also been applied to multimodal System K with iteration [14], a weak fragment of PDL.

Constructivity and Dynamic Logic. With CGL, we bring to fruition several past efforts to develop constructive dynamic logics. Prior work on PDL [18] sought an Existential Property for Propositional Dynamic Logic (PDL), but they questioned the practicality of their own implication introduction rule, whose side condition is non-syntactic. One of our results is a first-order Existential Property, which Degen cited as an open problem beyond the methods of their day [18]. To our knowledge, only one approach [32] considers Curry-Howard or functional proof terms for a program logic. While their work is a notable precursor to ours, their logic is a weak fragment of PDL without tests, monotonicity, or unbounded iteration, while we support not only PDL but the much more powerful first-order GL. Lastly, we are preceded by Constructive Concurrent Dynamic Logic, [53] which gives a Kripke semantics for Concurrent Dynamic Logic [41], a proper fragment of GL. Their work focuses on an epistemic interpretation of constructivity, algebraic laws, and tableaux. We differ in our use of realizability semantics and natural deduction, which were essential to developing a Curry-Howard interpretation for CGL. In summary, we are justified in claiming to have the first Curry-Howard interpretation with proof terms and Existential Properties for an *expressive* program logic, the first constructive game logic, and the only with first-order proof terms.

While constructive natural deduction calculi map most directly to functional programs, proof terms can be generated for any proof calculus, including a well-known interpretation of classical logic as continuation-passing style [26]. Proof terms have been developed [22] for a Hilbert calculus for dL, a dynamic logic (DL) for hybrid systems. Their work focuses on a provably correct interchange format for classical dL proofs, not constructive logics.

3 Syntax

We define the language of CGL, consisting of terms, games, and formulas. The simplest terms are *program variables* $x, y \in \mathcal{V}$ where $\mathcal{V}$ is the set of variable identifiers. Globally-scoped mutable program variables contain the state of the game, also called the *position* in game-theoretic terminology. All variables and terms are rational-valued ($\mathbb{Q}$); we also write $\mathbb{B}$ for the set of Boolean values $\{0, 1\}$ for false and true respectively.

Definition 1 (Terms). *A* term f, g *is a rational-valued computable function over the game state. We give a nonexhaustive grammar of terms, specifically those used in our examples:*

$$f, g \ ::= \ \cdots \mid q \mid x \mid f + g \mid f \cdot g \mid f/g \mid f \ \mathsf{mod}\ g$$

where $q \in \mathbb{Q}$ *is a rational literal,* x *a program variable,* $f+g$ *a sum,* $f \cdot g$ *a product. Division-with-remainder is intended for use with integers, but we generalize the standard notion to support rational arguments. Quotient* f/g *is integer even when* f *and* g *are non-integer, and thus leaves a rational remainder* $f \ \mathsf{mod}\ g$*. Divisors* g *are assumed to be nonzero.*

A game in CGL is played between a constructive player named Angel and a classical player named Demon. Our usage of the names Angel and Demon differs subtly from traditional GL usage for technical reasons. Our Angel and Demon are asymmetric: Angel is "our" player, who must play constructively, while the "opponent" Demon is allowed to play classically because our opponent need not be a computer. At any time some player is *active*, meaning their strategy resolves all decisions, and the opposite player is called *dormant*. Classical GL identifies Angel with active and Demon with dormant; the notions are distinct in CGL.

Definition 2 (Games). *The set of* games α, β *is defined recursively as such:*

$$\alpha, \beta \ ::= \ ?\phi \mid x := f \mid x := * \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid \alpha^d$$

In the *test game* $?\phi$, the active player wins if they can exhibit a constructive proof that formula ϕ currently holds. If they do not exhibit a proof, the dormant player wins by default and we informally say the active player "broke the rules". In deterministic assignment games $x := f$, neither player makes a choice, but the program variable x takes on the value of a term f. In nondeterministic assignment games $x := *$, the active player picks a value for $x : \mathbb{Q}$. In the choice game $\alpha \cup \beta$, the active player chooses whether to play game α or game β. In the sequential composition game $\alpha; \beta$, game α is played first, then β from the resulting state. In the repetition game α^*, the active player chooses after each repetition of α whether to continue playing, but loses if they repeat α infinitely. Notably, the exact number of repetitions can depend on the dormant player's moves, so the active player need not know, let alone announce, the exact number of iterations in advance. In the dual game α^d, the active player becomes dormant and vice-versa, then α is played. We parenthesize games with braces $\{\alpha\}$ when necessary. Sequential and nondeterministic composition both associate to the right, i.e., $\alpha \cup \beta \cup \gamma \equiv \{\alpha \cup \{\beta \cup \gamma\}\}$. This does not affect their semantics as both operators are associative, but aids in reading proof terms.

Definition 3 (CGL Formulas). *The set of CGL* formulas ϕ *(also* ψ, ρ*) is given recursively by the grammar:*

$$\phi \ ::= \ \langle\alpha\rangle\phi \mid [\alpha]\phi \mid f \sim g$$

The defining constructs in CGL (and GL) are the modalities $\langle\alpha\rangle\phi$ and $[\alpha]\phi$. These mean that the active or dormant Angel (i.e., constructive) player has a constructive strategy to play α and achieve postcondition ϕ. This paper does not develop the modalities for active and dormant Demon (i.e., classical) players because by definition those cannot be synthesized to executable code. We assume the presence of interpreted comparison predicates $\sim \in \{\leq, <, =, \neq, >, \geq\}$.

The standard connectives of first-order constructive logic can be derived from games and comparisons. Verum ($\mathtt{tt}$) is defined $1 > 0$ and falsum ($\mathtt{ff}$) is $0 > 1$. Conjunction $\phi \wedge \psi$ is defined $\langle ?\phi\rangle\psi$, disjunction $\phi \vee \psi$ is defined $\langle ?\phi \cup ?\psi\rangle\mathtt{tt}$, implication $\phi \rightarrow \psi$ is defined $[?\phi]\psi$, universal quantification $\forall x\, \phi$ is defined $[x := *]\phi$, and existential quantification $\exists x\, \phi$ is defined $\langle x := *\rangle\phi$. As usual in logic, equivalence $\phi \leftrightarrow \psi$ can also be defined $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$. As usual in constructive logics, negation $\neg\phi$ is defined $\phi \rightarrow \mathtt{ff}$, and inequality is defined by $f \neq g \equiv \neg(f = g)$. We will use the derived constructs freely but present semantics and proof rules only for the core constructs to minimize duplication. Indeed, it will aid in understanding of the proof term language to keep the definitions above in mind, because the proof terms for many first-order programs follow those from first-order constructive logic.

For convenience, we also write derived operators where the dormant player is given control of a single choice before returning control to the active player. The *dormant choice* $\alpha \cap \beta$, defined $\{\alpha^d \cup \beta^d\}^d$, says the dormant player chooses which branch to take, but the active player is in control of the subgames. We write ϕ_x^y (likewise for α and f) for the *renaming* of x for y and vice versa in formula ϕ, and write ϕ_x^f for the *substitution* of term f for program variable x in ϕ, if the substitution is admissible (Def. 9 in Section 6).

3.1 Example Games

We demonstrate the meaning and usage of the CGL constructs via examples, culminating in the two classic games of Nim and cake-cutting.

Nondeterministic Programs. Every (possibly nondeterministic) program is also a one-player game. For example, the program $n := 0; \{n := n + 1\}^*$ can nondeterministically sets n to any natural number because Angel has a choice whether to continue after every repetition of the loop, but is not allowed to continue forever. Conversely, games are like programs where the environment (Demon) is adversarial, and the program (Angel) strategically resolves nondeterminism to overcome the environment.

Demonic Counter. Angel's choices often must be *reactive* to Demon's choices. Consider the game $c := 10; \{c := c - 1 \cap c := c - 2\}^*; ?0 \leq c \leq 2$ where Demon repeatedly decreases c by 1 or 2, and Angel chooses when to stop. Angel only wins because she can pass the test $0 \leq c \leq 2$, which she can do by simply repeating the loop until $0 \leq c \leq 2$ holds. If Angel had to decide the loop duration in advance, Demon could force a rules violation by "guessing" the duration and changing his choices of $c := c - 1$ vs. $c := c - 2$.

Coin Toss. Games are perfect-information and do not possess randomness in the probabilistic sense, only (possibilistic) nondeterminism. This standard limitation is shown by attempting to express a coin-guessing game:

$$\{\mathsf{coin} := 0 \cap \mathsf{coin} := 1\}; \{\mathsf{guess} := 0 \cup \mathsf{guess} := 1\}; ?\mathsf{guess} = \mathsf{coin}$$

The Demon player sets the value of a tossed coin, but does so adversarially, not randomly, since strategies in CGL are *pure* strategies. The Angel player has perfect knowledge of coin and can set guess equivalently, thus easily passing the test guess = coin, unlike a real coin toss. Partial information games are interesting future work that could be implemented by limiting the variables visible in a strategy.

Nim. Nim is the standard introductory example of a discrete, 2-player, zero-sum, perfect-information game. We consider misère play (last player loses) for a version of Nim that is also known as the *subtraction game*. The constant Nim defines the game Nim.

$$\textsc{Nim} = \Big\{\big\{\{c := c-1 \cup c := c-2 \cup c := c-3\}; ?c > 0\big\}; \\ \big\{\{c := c-1 \cup c := c-2 \cup c := c-3\}; ?c > 0\big\}^d\Big\}^*$$

The game state consists of a single counter c containing a natural number, which each player chooses ($\cup$) to reduce by 1, 2, or 3 ($c := c - k$). The counter is non-negative, and the game repeats as long as Angel wishes, until some player empties the counter, at which point that player is declared the loser ($?c > 0$).

Proposition 1 (Dormant winning region). *Suppose* $c \equiv 1 \pmod 4$, *Then the dormant player has a strategy to ensure* $c \equiv 1 \pmod 4$ *as an invariant. That is, the following* CGL *formula is valid (true in every state):*

$$c > 0 \to c \mathit{\ mod\ } 4 = 1 \to [\textsc{Nim}^*]\, c \mathit{\ mod\ } 4 = 1$$

This implies the dormant player wins the game because the active player violates the rules once $c = 1$ and no move is valid. We now state the winning region for an active player.

Proposition 2 (Active winning region). *Suppose* $c \in \{0, 2, 3\} \pmod 4$ *initially, and the active player controls the loop duration. Then the active player can achieve* $c \in \{2, 3, 4\}$*:*

$$c > 0 \to c \mathit{\ mod\ } 4 \in \{0, 2, 3\} \to \langle \textsc{Nim}^* \rangle\, c \in \{2, 3, 4\}$$

At that point, the active player will win in one move by setting $c = 1$ which forces the dormant player to set $c = 0$ and fail the test $?c > 0$.

Cake-cutting. Another classic 2-player game, from the study of equitable division, is the cake-cutting problem [40]: The active player cuts the cake in two, then the (initially-)dormant player gets first choice of a piece. This is an optimal protocol for splitting the cake in the sense that the active player is incentivized to split the cake evenly, else the dormant player could take the larger piece. Cake-cutting is also a simple use case for fractional numbers. The constant CC defines the cake-cutting game. Here x is the relative size (from 0 to 1) of the first piece, y is the size of the second piece, a is the size of the active player's piece, and d is the size of dormant player's piece.

$$\begin{aligned}\mathrm{CC} = {} & x := *; ?(0 \leq x \leq 1); y := 1 - x; \\ & \{a := x; d := y \cap a := y; d := x\}\end{aligned}$$

The game is played only once. The active player picks the division of the cake, which must be a fraction $0 \leq x \leq 1$. The dormant player then picks which slice goes to whom.

The active player has a tight strategy to achieve a 0.5 cake share, as stated in Proposition 3.

Proposition 3 (Active winning region). *The following formula is valid:*

$$\langle \mathrm{CC} \rangle\, a \geq 0.5$$

The dormant player also has a computable strategy to achieve exactly 0.5 share of the cake (Proposition 4). Division is fair because each player has a strategy to get their fair 0.5 share.

Proposition 4 (Dormant winning region). *The following formula is valid:*

$$[\mathrm{CC}]\, d \geq 0.5$$

Computability and Numeric Types. Perfect fair division is only achieved for $a, d \in \mathbb{Q}$ because rational equality is decidable. Trichotomy ($a < 0.5 \vee a = 0.5 \vee a > 0.5$) is a tautology, so the dormant player's strategy can inspect the active player's choice of a. Notably, we intend to support constructive reals in future work, for which exact equality is not decidable and trichotomy is not an axiom. Future work on real-valued CGL will need to employ approximate comparison techniques as is typical for constructive reals [11,13,51]. The examples in this section have been proven [12] using the calculus defined in Section 5.

4 Semantics

We now develop the semantics of CGL. In contrast to classical GL, whose semantics are well-understood [38], the major semantic challenge for CGL is capturing the competition between a *constructive* Angel and *classical* Demon. We base our approach on realizability semantics [37,33], because this approach makes the

relationship between constructive proofs and programs particularly clear, and generating programs from CGL proofs is one of our motivations.

Unlike previous applications of realizability, games feature two agents, and one could imagine a semantics with two realizers, one for each of Angel and Demon. However, we choose to use only one realizer, for Angel, which captures the fact that only Angel is restricted to a computable strategy, not Demon. Moreover, a single realizer makes it clear that Angel cannot inspect Demon's strategy, only the game state, and also simplifies notations and proofs. Because Angel is computable but Demon is classical, our semantics has the flavor both of realizability semantics and of a traditional Kripke semantics for programs.

The semantic functions employ *game states* $\omega \in \mathfrak{S}$ where we write $\mathfrak{S}$ for the set of all states. We additionally write $\top, \bot \in \mathfrak{S}$ (not to be confused with formulas $\mathtt{tt}$ and $\mathtt{ff}$) for the pseudo-states $\top$ and $\bot$ indicating that Angel or Demon respectively has won the game early by forcing the other to fail a test. Each $\omega \in \mathfrak{S}$ maps each $x \in \mathcal{V}$ to a value $\omega(x) \in \mathbb{Q}$. We write ω_x^v for the state that agrees with ω except that x is assigned value v where $v \in \mathbb{Q}$.

Definition 4 (Arithmetic term semantics). *A term f is a computable function of the state, so the interpretation $[\![f]\!]\omega$ of term f in state ω is $f(\omega)$.*

4.1 Realizers

To define the semantics of games, we first define realizers, the programs which implement strategies. The language of realizers is a higher-order lambda calculus where variables can range over game states, numbers, or realizers which realize a give proposition ϕ. Gameplay proceeds in continuation-passing style: invoking a realizer returns another realizer which performs any further moves. We describe the typing constraints for realizers informally, and say a is a $\langle\alpha\rangle\phi$-realizer ($a \in \langle\alpha\rangle\phi\,\mathcal{R}\mathbf{z}$) if it provides strategic decisions exactly when $\langle\alpha\rangle\phi$ demands them.

Definition 5 (Realizers). *The syntax of realizers $a, b, c \in \mathcal{R}\mathbf{z}$ (where $\mathcal{R}\mathbf{z}$ is the set of all realizers) is defined coinductively:*

$$a, b, c ::= x \mid () \mid (a, b) \mid \pi_L(a) \mid \pi_R(a) \mid (\lambda\omega : \mathfrak{S}.\ a(\omega)) \mid (\Lambda x : \mathbb{Q}.\ a)$$
$$\mid (\Lambda x : \phi\,\mathcal{R}\mathbf{z}.\ a) \mid a\,v \mid a\,b \mid a\,\omega \mid \mathtt{if}\ (f(\omega))\ a\ \mathtt{else}\ b$$

where x is a program (or realizer) variable and f is a term over the state ω. The Roman a, b, c should not be confused with the Greek α, β, γ which range over games. Realizers have access to the game state ω, expressed by lambda realizers $(\lambda\omega : \mathfrak{S}.\ a(\omega))$ which, when applied in a state ν, compute the realizer a with ν substituted for ω. State lambdas λ are distinguished from propositional and first-order lambdas Λ. The unit realizer $()$ makes no choices and is understood as a unit tuple. Units $()$ realize $f \sim g$ because *rational* comparisons, in contrast to real comparisons, are decidable. Conditional strategic decisions are realized by $\mathtt{if}\ (f(\omega))\ a\ \mathtt{else}\ b$ for computable function $f : \mathfrak{S} \to \mathbb{B}$, and execute a if f returns truth, else b. Realizer $(\lambda\omega : \mathfrak{S}.\ f(\omega))$ is a $\langle\alpha \cup \beta\rangle\phi$-realizer if $f(\omega) \in (\{0\} \times \langle\alpha\rangle\phi\,\mathcal{R}\mathbf{z}) \cup (\{1\} \times \langle\beta\rangle\phi\,\mathcal{R}\mathbf{z})$ for all ω. The first component determines

which branch is taken, while the second component is a continuation which must be able to play the corresponding branch. Realizer $(\lambda\omega : \mathfrak{S}.\ f(\omega))$ can also be a $\langle x := * \rangle\phi$-realizer, which requires $f(\omega) \in \mathbb{Q} \times (\phi\,\mathcal{R}\mathbf{z})$ for all ω. The first component determines the value of x while the second component demonstrates the postcondition ϕ. The pair realizer (a, b) realizes both Angelic tests $\langle ?\phi \rangle\psi$ and dormant choices $[\alpha \cup \beta]\phi$. It is identified with a pair of realizers: $(a, b) \in \mathcal{R}\mathbf{z} \times \mathcal{R}\mathbf{z}$.

A dormant realizer waits and remembers the active Demon's moves, because they typically inform Angel's strategy once Angel resumes action. The first-order realizer $(\Lambda x : \mathbb{Q}.\ b)$ is a $[x := *]\phi$-realizer when b_x^v is a ϕ-realizer for every $v \in \mathbb{Q}$; Demon tells Angel the desired value of x, which informs Angel's continuation b. The higher-order realizer $(\Lambda x : \phi\,\mathcal{R}\mathbf{z}.\ b)$ realizes $[?\phi]\psi$ when b_x^c realizes ψ for every ϕ-realizer c. Demon announces the realizer for ϕ which Angel's continuation b may inspect. Tuples are inspected with projections $\pi_L(a)$ and $\pi_R(a)$. A lambda is inspected by applying arguments $a\,\omega$ for state-lambdas, $a\,v$ for first-order, and $a\,b$ for higher-order. Realizers for sequential compositions $\langle \alpha; \beta \rangle\phi$ (likewise $[\alpha; \beta]\phi$) are $\langle \alpha \rangle\langle \beta \rangle\phi$-realizers: first α is played, and in every case the continuation must play β before showing ϕ. Realizers for repetitions α^* are streams containing α-realizers, possibly infinite by virtue of coinductive syntax. Active loop realizer $\mathsf{ind}(x.\ a)$ is the least fixed point of the equation $b = [b/x]a$, i.e., x is a recursive call which must be invoked only in accordance with some well-order. We realize dormant loops with $\mathsf{gen}(a,\ x.b,\ x.c)$, coinductively generated from initial value a, update b, and post-step c with variable x for current generator value.

Active loops must terminate, so $\langle \alpha^* \rangle\phi$-realizers are constructed inductively using any well-order on states. Dormant loops must be played as long as the opponent wishes, so $[\alpha^*]\phi$-realizers are constructed coinductively, with the invariant that ϕ has a realizer at every iteration.

4.2 Formula and Game Semantics

A state ω paired with a realizer a that continues the game is called a *possibility*. A *region* (written X, Y, Z) is a set of possibilities. We write $[\![\phi]\!] \subseteq \phi\,\mathcal{R}\mathbf{z} \times \mathfrak{S}$ for the region which realizes formula ϕ. A formula ϕ is *valid* iff some a uniformly realizes every state, i.e., $\{a\} \times \mathfrak{S} \subseteq [\![\phi]\!]$. A sequent $\Gamma \vdash \phi$ is *valid* iff the formula $\bigwedge \Gamma \to \phi$ is valid, where $\bigwedge \Gamma$ is the conjunction of all assumptions in Γ.

The game semantics are region-oriented, i.e., they process possibilities in bulk, though Angel commits to a strategy from the start. The region $X\langle\!\langle \alpha \rangle\!\rangle : \wp(\mathcal{R}\mathbf{z} \times \mathfrak{S})$ is the union of all end regions of game α which arise when active Angel commits to an element of X, then Demon plays adversarially. In $X[\![\alpha]\!] : \wp(\mathcal{R}\mathbf{z} \times \mathfrak{S})$ Angel is the *dormant* player, but it is still Angel who commits to an element of X and Demon who plays adversarially. Recall that pseudo-states $\top$ and $\bot$ represent early wins by each Angel and Demon, respectively. The definitions below implicitly assume $\bot, \top \notin X$, they extend to the case $\bot \in X$ (likewise $\top \in X$) using the equations $(X \cup \{\bot\})[\![\alpha]\!] = X[\![\alpha]\!] \cup \{\bot\}$ and $(X \cup \{\bot\})\langle\!\langle \alpha \rangle\!\rangle = X\langle\!\langle \alpha \rangle\!\rangle \cup \{\bot\}$. That is, if Demon has already won by forcing an Angel violation initially, any remaining game can be skipped with an immediate Demon victory, and vice-versa. The game semantics exploit the *Angelic* projections $Z_{\langle 0 \rangle}, Z_{\langle 1 \rangle}$

and *Demonic* projections $Z_{[0]}, Z_{[1]}$, which represent binary decisions made by a constructive Angel and a classical Demon, respectively. The Angelic projections, which are defined $Z_{\langle 0\rangle} = \{(\pi_R(a),\omega) \mid \pi_L(a)(\omega) = 0, (a,\omega) \in Z\}$ and $Z_{\langle 1\rangle} = \{(\pi_R(a),\omega) \mid \pi_L(a)(\omega) = 1, (a,\omega) \in Z\}$, filter by which branch Angel chooses with $\pi_L(a)(\omega) \in \mathbb{B}$, then project the remaining strategy $\pi_R(a)$. The Demonic projections, which are defined $Z_{[0]} \equiv \{(\pi_L(a),\omega) \mid (a,\omega) \in Z\}$ and $Z_{[1]} \equiv \{(\pi_R(a),\omega) \mid (a,\omega) \in Z\}$, contain the same states as Z, but project the realizer to tell Angel which branch Demon took.

Definition 6 (Formula semantics). $[\![\phi]\!] \subseteq \mathcal{R}\mathbf{z} \times \mathfrak{S}$ *is defined as:*

$$\begin{aligned}(\mathbb{O},\omega) &\in [\![f \sim g]\!] \text{ iff } [\![f]\!]\omega \sim [\![g]\!]\omega\\ (a,\omega) &\in [\![\langle\alpha\rangle\phi]\!] \text{ iff } \{(a,\omega)\}\langle\!\langle\alpha\rangle\!\rangle \subseteq ([\![\phi]\!] \cup \{\top\})\\ (a,\omega) &\in [\![[\alpha]\phi]\!] \text{ iff } \{(a,\omega)\}[\![\alpha]\!] \subseteq ([\![\phi]\!] \cup \{\top\})\end{aligned}$$

Comparisons $f \sim g$ defer to the term semantics, so the interesting cases are the game modalities. Both $[\alpha]\phi$ and $\langle\alpha\rangle\phi$ ask whether Angel wins α by following the given strategy, and differ only in whether Demon vs. Angel is the active player, thus in both cases *every* Demonic choice must satisfy Angel's goal, and early Demon wins are counted as Angel losses.

Definition 7 (Angel game forward semantics). *We inductively define the region* $X\langle\!\langle\alpha\rangle\!\rangle : \wp(\mathcal{R}\mathbf{z} \times \mathfrak{S})$ *in which* α *can end when active Angel plays* X*:*

$$\begin{aligned}X\langle\!\langle ?\phi\rangle\!\rangle &= \{(\pi_R(a),\omega) \mid (\pi_L(a),\omega) \in [\![\phi]\!] \text{ for some } (a,\omega) \in X\ \}\\ &\quad \cup \{\bot \mid (\pi_L(a),\omega) \notin [\![\phi]\!] \text{ for all } (a,\omega) \in X\ \}\\ X\langle\!\langle x := f\rangle\!\rangle &= \{(a, \omega_x^{[\![f]\!]\omega}) \mid (a,\omega) \in X\}\\ X\langle\!\langle x := *\rangle\!\rangle &= \{(\pi_R(a), \omega_x^{\pi_L(a)(\omega)}) \mid (a,\omega) \in X\}\\ X\langle\!\langle \alpha;\beta\rangle\!\rangle &= (X\langle\!\langle\alpha\rangle\!\rangle)\langle\!\langle\beta\rangle\!\rangle\\ X\langle\!\langle \alpha \cup \beta\rangle\!\rangle &= X_{\langle 0\rangle}\langle\!\langle\alpha\rangle\!\rangle \cup X_{\langle 1\rangle}\langle\!\langle\beta\rangle\!\rangle\\ X\langle\!\langle \alpha^*\rangle\!\rangle &= \bigcap\{Z_{\langle 0\rangle} \subseteq \mathcal{R}\mathbf{z} \times \mathfrak{S} \mid X \cup (Z_{\langle 1\rangle}\langle\!\langle\alpha\rangle\!\rangle) \subseteq Z\}\\ X\langle\!\langle \alpha^d\rangle\!\rangle &= X[\![\alpha]\!]\end{aligned}$$

Definition 8 (Demon game forward semantics). *We inductively define the region* $X[\![\alpha]\!] : \wp(\mathcal{R}\mathbf{z} \times \mathfrak{S})$ *in which* α *can end when dormant Angel plays* X*:*

$$\begin{aligned}X[\![?\phi]\!] &= \{(a\,b,\omega) \mid (a,\omega) \in X, (b,\omega) \in [\![\phi]\!], \text{ some } b \in \mathcal{R}\mathbf{z}\}\\ &\quad \cup \{\top \mid (a,\omega) \in X, \text{ but no } (b,\omega) \in [\![\phi]\!]\}\\ X[\![x := f]\!] &= \{(a, \omega_x^{[\![f]\!]\omega}) \mid (a,\omega) \in X\}\\ X[\![x := *]\!] &= \{(a\,r, \omega_x^r) \mid r \in \mathbb{Q}\}\\ X[\![\alpha;\beta]\!] &= (X[\![\alpha]\!])[\![\beta]\!]\\ X[\![\alpha \cup \beta]\!] &= X_{[0]}[\![\alpha]\!] \cup X_{[1]}[\![\beta]\!]\\ X[\![\alpha^*]\!] &= \bigcap\{Z_{[0]} \subseteq \mathcal{R}\mathbf{z} \times \mathfrak{S} \mid X \cup (Z_{[1]}[\![\alpha]\!]) \subseteq Z\}\\ X[\![\alpha^d]\!] &= X\langle\!\langle\alpha\rangle\!\rangle\end{aligned}$$

Angelic tests $?\phi$ end in the current state ω with remaining realizer $\pi_R(a)$ if Angel can realize ϕ with $\pi_L(a)$, else end in $\bot$. Angelic deterministic assignments consume no realizer and simply update the state, then end. Angelic nondeterministic assignments $x := *$ ask the realizer $\pi_L(a)$ to compute a new value for x from the current state. Angelic compositions $\alpha;\beta$ first play α, then β from the resulting state using the resulting continuation. Angelic choice games $\alpha \cup \beta$ use the Angelic projections to decide which branch is taken according to $\pi_L(a)$. The realizer $\pi_R(a)$ may be reused between α and β, since $\pi_R(a)$ could just invoke $\pi_L(a)$ if it must decide which branch has been taken. This definition of Angelic choice (corresponding to constructive disjunction) captures the reality that realizers in CGL, in contrast with most constructive logics, are entitled to observe a game state, but they must do so in computable fashion.

Repetition Semantics. In any GL, the challenge in defining the semantics of repetition games α^* is that the number of iterations, while finite, can depend on both players' actions and is thus not known in advance, while the DL-like semantics of α^* as the finite reflexive, transitive closure of α gives an advance-notice semantics. Classical GL provides the no-advance-notice semantics as a fixed point [38], and we adopt the fixed point semantics as well. The Angelic choice whether to stop ($Z_{\langle 0\rangle}$) or iterate the loop ($Z_{\langle 1\rangle}$) is analogous to the case for $\alpha \cup \beta$.

Duality Semantics. To play the dual game α^d, the active and dormant players switch roles, then play α. In *classical* GL, this characterization of duality is interchangeable with the definition of α^d as the game that Angel wins exactly when it is impossible for Angel to lose. The characterizations are *not* interchangeable in CGL because the Determinacy Axiom (all games have winners) of GL is not valid in CGL:

Remark 1 (Indeterminacy). Classically equivalent determinacy axiom schemata $\neg\langle\alpha\rangle\neg\phi \to [\alpha]\phi$ and $\langle\alpha\rangle\neg\phi \vee [\alpha]\phi$ of classical GL are not valid in CGL, because they imply double negation elimination.

Remark 2 (Classical duality). In classical GL, Angelic dual games are characterized by the axiom schema $\langle\alpha^d\rangle\phi \leftrightarrow \neg\langle\alpha\rangle\neg\phi$, which is not valid in in CGL. It is classically interdefinable with $\langle\alpha^d\rangle \leftrightarrow [\alpha]\phi$.

The determinacy axiom is not valid in CGL, so we take $\langle\alpha^d\rangle \leftrightarrow [\alpha]\phi$ as primary.

4.3 Demonic Semantics

Demon wins a Demonic test by presenting a realizer b as evidence that the precondition holds. If he cannot present a realizer (i.e., because none exists), then the game ends in $\top$ so Angel wins by default. Else Angel's higher-order realizer a consumes the evidence of the pre-condition, i.e., Angelic strategies are entitled to depend (computably) on *how* Demon demonstrated the precondition. Angel can check that Demon passed the test by executing b. The Demonic repetition game

α^* is defined as a fixed-point [42] with Demonic projections. Computationally, a winning invariant for the repetition is the witness of its winnability.

The remaining cases are innocuous by comparison. Demonic deterministic assignments $x := f$ deterministically store the value of f in x, just as Angelic assignments do. In demonic nondeterministic assignment $x := *$, Demon chooses to set x to *any* value. When Demon plays the choice game $\alpha \cup \beta$, Demon chooses classically between α and β. The dual game α^d is played by Demon becoming dormant and Angel become active in α.

Semantics Examples. The realizability semantics of games are subtle on a first read, so we provide examples of realizers. In these examples, the state argument ω is implicit, and we refer to $\omega(x)$ simply as x for brevity.

Recall that $[?\phi]\psi$ and $\phi \to \psi$ are equivalent. For any ϕ, the identity function $(\Lambda x : \phi \, \mathcal{R}\mathbf{z}.\ x)$ is a $\phi \to \phi$-realizer: for every ϕ-realizer x which Demon presents, Angel can present the same x as evidence of ϕ. This confirms expected behavior per propositional constructive logic: the identity function is the proof of self-implication.

In example formula $\langle x := *^d; \{x := x \cup x := -x\}\rangle x \geq 0$, Demon gets to set x, then Angel decides whether to negate x in order to make it nonnegative. It is realized by $\Lambda x : \mathbb{Q}.\ (($`if` $(x < 0)\ 1$ `else` $0), ())$: Demon announces the value of x, then Angel's strategy is to check the sign of x, taking the right branch when x is negative. Each branch contains a deterministic assignment which consumes no realizer, then the postcondition $x \geq 0$ has trivial realizer `()`.

Consider the formula $\langle \{x := x + 1\}^*\rangle x > y$, where Angel's winning strategy is to repeat the loop until $x > y$, which will occur as x increases. The realizer is $\mathsf{ind}(w.\ ($`if` $(x > y)\ (0, ())$ `else` $(1, w), ()))$, which says that Angel stops the loop if $x > y$ and proves the postcondition with a trivial strategy. Else Angel continues the loop, whose body consumes no realizer, and supplies the inductive call w to continue the strategy inductively.

Consider the formula $[?x > 0; \{x := x + 1\}^*]\exists y\,(y \leq x \wedge y > 0)$ for a subtle example. Our strategy for Angel is to record the initial value of x in y, then maintain a proof that $y \leq x$ as x increases. This strategy is represented by $\Lambda w : (x > 0)\, \mathcal{R}\mathbf{z}.\ \mathsf{gen}((x, ((), w)),\ z.(\pi_L(z), ((), \pi_R(\pi_R(z)))),\ z.z)$. That is, initially Demon announces a proof w of $x > 0$. Angel specifies the initial element of the realizer stream by witnessing $\exists y\,(y \leq x \wedge y > 0)$ with $c_0 = (x, ((), w))$, where the first component instantiates $y = x$, the trivial second component indicates that $y \leq y$ trivially, and the third component reuses w as a proof of $y > 0$. Demon can choose to repeat the loop arbitrarily. When Demon demands the k'th repetition, z is bound to c_{k-1} to compute $c_k = (\pi_L(z), ((), \pi_R(\pi_R(z))))$, which plays the next iteration. That is, at each iteration Angel witnesses $\exists y\,(y \leq x \wedge y > 0)$ by assigning the same value (stored in $\pi_L(z)$) to y, reproving $y \leq x$ with `()`, then reusing the proof (stored in $\pi_R(\pi_R(z))$) that $y > 0$.

5 Proof Calculus

Having settled on the meaning of a game in Section 4, we proceed to develop a calculus for proving CGL formulas syntactically. The goal is twofold: the practical motivation, as always, is that when verifying a concrete example, the realizability semantics provide a notion of ground truth, but are impractical for proving large formulas. The theoretical motivation is that we wish to expose the computational interpretation of the modalities $\langle\alpha\rangle\phi$ and $[\alpha]\phi$ as the types of the players' respective winning strategies for game α that has ϕ as its goal condition. Since CGL is constructive, such a strategy constructs a proof of the postcondition ϕ.

To study the computational nature of proofs, we write proof terms explicitly: the main proof judgement $\Gamma \vdash M : \phi$ says proof term M is a proof of ϕ in context Γ, or equivalently a proof of sequent $(\Gamma \vdash \phi)$. We write M, N, O (sometimes A, B, C) for arbitrary proof terms, and p, q, ℓ, r, s, g for *proof variables*, that is variables that range over proof terms of a given proposition. In contrast to the assignable *program variables*, the proof variables are given their meaning by substitution and are scoped locally, not globally. We adapt propositional proof terms such as pairing, disjoint union, and lambda-abstraction to our context of game logic. To support first-order games, we include first-order proof terms and new terms for features: dual, assignment, and repetition games.

We now develop the calculus by starting with standard constructs and working toward the novel constructs of CGL. The assumptions p in Γ are named, so that they may appear as variable proof-terms p. We write Γ_x^y and M_x^y for the renaming of program variable x to y and vice versa in context Γ or proof term M, respectively. Proof rules for state-modifying constructs explicitly perform renamings, which both ensures they are applicable as often as possible and also ensures that references to proof variables support an intuitive notion of lexical scope. Likewise Γ_x^f and M_x^f are the substitutions of term f for program variable x. We use distinct notation to substitute proof terms for proof variables while avoiding capture: $[N/p]M$ substitutes proof term N for proof variable p in proof term M. Some proof terms such as pairs prove both a diamond formula and a box formula. We write $\langle M, N\rangle$ and $[M, N]$ respectively to distinguish the terms or $\langle\!\langle M, N\rangle\!\rangle$ to treat them uniformly. Likewise we abbreviate $\langle\!\langle\alpha\rangle\!\rangle\phi$ when the same rule works for both diamond and box modalities, using $[\![\alpha]\!]\phi$ to denote its dual modality. The proof terms $\langle x := f_x^y \text{ in } p.\ M\rangle$ and $[x := f_x^y \text{ in } p.\ M]$ introduce an auxiliary ghost variable y for the old value of x, which improves completeness without requiring manual ghost steps.

The propositional proof rules of CGL are in Fig. 1. Formula $[?\phi]\psi$ is constructive implication, so rule [?]E with proof term $M\ N$ eliminates M by supplying an N that proves the test condition. Lambda terms $(\lambda p : \phi.\ M)$ are introduced by rule [?]I by extending the context Γ. While this rule is standard, it is worth emphasizing that here p is a *proof variable* for which a proof term (like N in [?]E) may be substituted, and that the *game state* is untouched by [?]I. Constructive disjunction (between the branches $\langle\alpha\rangle\phi$ and $\langle\beta\rangle\phi$) is the choice $\langle\alpha \cup \beta\rangle\phi$. The introduction rules for injections are $\langle\cup\rangle$I1 and $\langle\cup\rangle$I2, and case-analysis is performed with rule $\langle\cup\rangle$E, with two branches that prove a common consequence

$$\langle\cup\rangle\text{E}\ \frac{\Gamma \vdash A : \langle\alpha\cup\beta\rangle\phi \quad \Gamma, \ell : \langle\alpha\rangle\phi \vdash B : \psi \quad \Gamma, r : \langle\beta\rangle\phi \vdash C : \psi}{\Gamma \vdash \langle \mathsf{case}\ A\ \mathsf{of}\ \ell \Rightarrow\ B \mid r \Rightarrow\ C\rangle : \psi}$$

$$\langle\cup\rangle\text{I1}\ \frac{\Gamma \vdash M : \langle\alpha\rangle\phi}{\Gamma \vdash \langle \ell \cdot M\rangle : \langle\alpha\cup\beta\rangle\phi} \qquad [\cup]\text{E1}\ \frac{\Gamma \vdash M : [\alpha\cup\beta]\phi}{\Gamma \vdash [\pi_1 M] : [\alpha]\phi}$$

$$\langle\cup\rangle\text{I2}\ \frac{\Gamma \vdash M : \langle\beta\rangle\phi}{\Gamma \vdash \langle r \cdot M\rangle : \langle\alpha\cup\beta\rangle\phi} \qquad [\cup]\text{E2}\ \frac{\Gamma \vdash M : [\alpha\cup\beta]\phi}{\Gamma \vdash [\pi_2 M] : [\beta]\phi}$$

$$[\cup]\text{I}\ \frac{\Gamma \vdash M : [\alpha]\phi \quad \Gamma \vdash N : [\beta]\phi}{\Gamma \vdash [M, N] : [\alpha\cup\beta]\phi} \qquad \text{hyp}\ \frac{}{\Gamma, p : \phi \vdash p : \phi}$$

$$\langle?\rangle\text{I}\ \frac{\Gamma \vdash M : \phi \quad \Gamma \vdash N : \psi}{\Gamma \vdash \langle M, N\rangle : \langle?\phi\rangle\psi} \qquad \langle?\rangle\text{E1}\ \frac{\Gamma \vdash M : \langle?\phi\rangle\psi}{\Gamma \vdash \langle \pi_1 M\rangle : \phi}$$

$$[?]\text{I}\ \frac{\Gamma, p : \phi \vdash M : \psi}{\Gamma \vdash (\lambda p : \phi.\ M) : [?\phi]\psi} \qquad \langle?\rangle\text{E2}\ \frac{\Gamma \vdash M : \langle?\phi\rangle\psi}{\Gamma \vdash \langle \pi_2 M\rangle : \psi}$$

$$[?]\text{E}\ \frac{\Gamma \vdash M : [?\phi]\psi \quad \Gamma \vdash N : \phi}{\Gamma \vdash (M\ N) : \psi}$$

Fig. 1. CGL proof calculus: Propositional rules

from each disjunct. The cases $\langle?\phi\rangle\psi$ and $[\alpha\cup\beta]\phi$ are conjunctive. Conjunctions are introduced by $\langle?\rangle$I and $[\cup]$I as pairs, and eliminated by $\langle?\rangle$E1, $\langle?\rangle$E2, $[\cup]$E1, and $[\cup]$E2 as projections. Lastly, rule hyp says formulas in the context hold by assumption.

We now begin considering non-propositional rules, starting with the simplest ones. The majority of the rules in Fig. 2, while thoroughly useful in proofs,

$$\langle*\rangle\text{C}\ \frac{\Gamma \vdash A : \langle\alpha^*\rangle\phi \quad \Gamma, s : \phi \vdash B : \psi \quad \Gamma, g : \langle\alpha\rangle\langle\alpha^*\rangle\phi \vdash C : \psi}{\Gamma \vdash \langle \mathsf{case}_*\ A\ \mathsf{of}\ s \Rightarrow\ B \mid g \Rightarrow C\rangle : \psi}$$

$$\text{M}\ \frac{\Gamma \vdash M : \langle\alpha\rangle\phi \quad \Gamma_{\frac{\boldsymbol{y}}{\mathrm{BV}(\alpha)}}, p : \phi \vdash N : \psi}{\Gamma \vdash M \circ_p N : \langle\alpha\rangle\psi}$$

$$[*]\text{E}\ \frac{\Gamma \vdash M : [\alpha^*]\phi}{\Gamma \vdash [\mathsf{unroll}\ M] : \phi \wedge [\alpha][\alpha^*]\phi} \qquad \langle\!\langle d\rangle\!\rangle\text{I}\ \frac{\Gamma \vdash M : [\![\alpha]\!]\phi}{\Gamma \vdash \langle\!\langle \mathsf{yield}\ M\rangle\!\rangle : \langle\!\langle \alpha^d\rangle\!\rangle\phi}$$

$$\langle*\rangle\text{S}\ \frac{\Gamma \vdash M : \phi}{\Gamma \vdash \langle \mathsf{stop}\ M\rangle : \langle\alpha^*\rangle\phi} \qquad [*]\text{R}\ \frac{\Gamma \vdash M : \phi \wedge [\alpha][\alpha^*]\phi}{\Gamma \vdash [\mathsf{roll}\ M] : [\alpha^*]\phi}$$

$$\langle*\rangle\text{G}\ \frac{\Gamma \vdash M : \phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi}{\Gamma \vdash \langle \mathsf{go}\ M\rangle : \langle\alpha^*\rangle\phi} \qquad \langle\!\langle;\rangle\!\rangle\text{I}\ \frac{\Gamma \vdash M : \langle\!\langle\alpha\rangle\!\rangle\langle\!\langle\beta\rangle\!\rangle\phi}{\Gamma \vdash \langle\!\langle \iota\ M\rangle\!\rangle : \langle\!\langle\alpha;\beta\rangle\!\rangle\phi}$$

Fig. 2. CGL proof calculus: Some non-propositional rules

are computationally trivial. The repetition rules ($[*]$E,$[*]$R) fold and unfold the notion of repetition as iteration. The rolling and unrolling terms are named in analogy to the *iso-recursive* treatment of recursive types [50], where an explicit operation is used to expand and collapse the recursive definition of a type.

Rules $\langle*\rangle$C,$\langle*\rangle$S,$\langle*\rangle$G are the destructor and injectors for $\langle\alpha^*\rangle\phi$, which are similar to those for $\langle\alpha\cup\beta\rangle\phi$. The duality rules ($\langle\!\langle d\rangle\!\rangle$I) say the dual game is proved by proving the game where roles are reversed. The sequencing rules ($\langle\!\langle;\rangle\!\rangle$I) say a

sequential game is played by playing the first game with the goal of reaching a state where the second game is winnable.

Among these rules, monotonicity M is especially computationally rich. The notation $\Gamma\frac{\boldsymbol{y}}{\mathrm{BV}(\alpha)}$ says that in the second premiss, the assumptions in Γ have all bound variables of α (written $\mathrm{BV}(\alpha)$) renamed to fresh variables $\boldsymbol{y}$ for completeness. In practice, Γ usually contains some assumptions on variables that are not bound, which we wish to access without writing them explicitly in ϕ. Rule M is used to execute programs right-to-left, giving shorter, more efficient proofs. It can also be used to derive the Hoare-logical sequential composition rule, which is frequently used to reduce the number of case splits. Note that like every GL, CGL is subnormal, so the modal modus ponens axiom K and Gödel generalization (or necessitation) rule G are not sound, and M takes over much of the role they usually serve. On the surface, M simply says games are monotonic: a game's goal proposition may freely be replaced with a weaker one. From a computational perspective, Section 7 will show that rule M can be (lazily) eliminated. Moreover, M is an *admissible* rule, one whose instances can all be derived from existing rules. When proofs are written right-to-left with M, the normalization relation translates them to left-to-right normal proofs. Note also that in checking $M \circ_p N$, the context Γ has the bound variables α renamed freshly to some $\boldsymbol{y}$ within N, as required to maintain soundness across execution of α.

Next, we consider *first-order* rules, i.e., those which deal with first-order programs that modify *program* variables. The first-order rules are given in Fig. 3. In $\langle{:}*\rangle$E, $\mathrm{FV}(\psi)$ are the *free variables* of ψ, the variables which can influence its meaning. Nondeterministic assignment provides quantification over rational-

$$
\begin{array}{lll}
[\![:=]\!]\mathrm{I} & \dfrac{\Gamma\frac{y}{x}, p : (x = f\frac{y}{x}) \vdash M : \phi}{\Gamma \vdash [\![x := f\frac{y}{x} \text{ in } p.\ M]\!] : [\![x := f]\!]\phi} & (y \text{ fresh}) \\[2ex]
\langle{:}*\rangle\mathrm{I} & \dfrac{\Gamma\frac{y}{x}, p : (x = f\frac{y}{x}) \vdash M : \phi}{\Gamma \vdash \langle f\frac{y}{x} :* p.\ M\rangle : \langle x := *\rangle\phi} & (y, p \text{ fresh},\ f \text{ comp.}) \\[2ex]
\langle{:}*\rangle\mathrm{E} & \dfrac{\Gamma \vdash M : \langle x := *\rangle\phi \quad \Gamma\frac{y}{x}, p : \phi \vdash N : \psi}{\Gamma \vdash \mathsf{unpack}(M, py.\ N) : \psi} & (y \text{ fresh},\ x \notin \mathrm{FV}(\psi)) \\[2ex]
[{:}*]\mathrm{I} & \dfrac{\Gamma\frac{y}{x} \vdash M : \phi}{\Gamma \vdash (\lambda x : \mathbb{Q}.\ M) : [x := *]\phi} & (\text{y fresh}) \\[2ex]
[{:}*]\mathrm{E} & \dfrac{\Gamma \vdash M : [x := *]\phi}{\Gamma \vdash (M\ f) : \phi_x^f} & (\phi_x^f \text{ admiss.})
\end{array}
$$

Fig. 3. CGL proof calculus: first-order games

valued *program* variables. Rule [:*]I is universal, with proof term $(\lambda x : \mathbb{Q}.\ M)$. While this notation is suggestive, the difference vs. the function proof term $(\lambda p : \phi.\ M)$ is essential: the proof term M is checked (resp. evaluated) in a state where the program variable x has changed from its initial value. For soundness, [:*]I renames x to fresh program variable y throughout context Γ, written $\Gamma\frac{y}{x}$. This means that M can freely refer to all facts of the full context, but they

now refer to the state as it was before x received a new value. Elimination [:∗]E then allows instantiating x to a term f. Existential quantification is introduced by ⟨:∗⟩I whose proof term $\langle f\frac{y}{x} :* p.\ M\rangle$ is like a dependent pair plus bound renaming of x to y. The witness f is an arbitrary computable term, as always. We write $\langle f_{\frac{}{x}} :* M\rangle$ for short when y is not referenced in M. It is eliminated in ⟨:∗⟩E by unpacking the pair, with side condition $x \notin \mathrm{FV}(\psi)$ for soundness. The assignment rules ⟦:=⟧I do not quantify, per se, but always update x to the value of the term f, and in doing so introduce an assumption that x and f (suitably renamed) are now equal. In ⟨:∗⟩I and ⟦:=⟧I, program variable y is fresh.

$$\langle*\rangle\mathrm{I}\ \frac{\Gamma \vdash A : \varphi \qquad p : \varphi, q : \mathcal{M}_0 = \mathcal{M} \succ \mathbf{0} \vdash B : \langle\alpha\rangle(\varphi \wedge \mathcal{M}_0 \succ \mathcal{M}) \qquad p : \varphi, q : \mathcal{M} = \mathbf{0} \vdash C : \phi}{\Gamma \vdash \mathsf{for}(p : \varphi(\mathcal{M}) = A; q.\, B; C)\{\alpha\} : \langle\alpha^*\rangle\phi}\ (\mathcal{M}_0 \text{ fresh})$$

$$[*]\mathrm{I}\ \frac{\Gamma \vdash M : J \qquad p : J \vdash N : [\alpha]J \qquad p : J \vdash O : \phi}{\Gamma \vdash (M\ \mathsf{rep}\ p : J.\ N\ \mathsf{in}\ O) : [\alpha^*]\phi} \qquad \mathrm{FP}\ \frac{\Gamma \vdash A : \langle\alpha^*\rangle\phi \qquad s : \phi \vdash B : \psi \qquad g : \langle\alpha\rangle\psi \vdash C : \psi}{\Gamma \vdash FP(A, s.\ B, g.\ C) : \psi}$$

$$\mathrm{split}\ \Gamma \vdash (\mathsf{split}\ [f, g]\ ) : f \leq g \vee f > g$$

Fig. 4. CGL proof calculus: loops

The looping rules in Fig. 4, especially ⟨∗⟩I, are arguably the most sophisticated in CGL. Rule ⟨∗⟩I provides a strategy to repeat a game α until the postcondition ϕ holds. This is done by exhibiting a convergence predicate φ and termination metric $\mathcal{M}$ with terminal value $\mathbf{0}$ and well-ordering $\succ$. Proof term A shows φ holds initially. Proof term B guarantees $\mathcal{M}$ decreases with every iteration where $\mathcal{M}_0$ is a fresh metric variable which is equal to $\mathcal{M}$ at the antecedent of B and is never modified. Proof term C allows any postcondition ϕ which follows from convergence $\varphi \wedge \mathcal{M} = \mathbf{0}$. Proof term $\mathsf{for}(p : \varphi(\mathcal{M}) = A; q.\, B; C)\{\alpha\}$ suggests the computational interpretation as a for-loop: proof A shows the convergence predicate holds in the initial state, B shows that each step reduces the termination metric while maintaining the predicate, and C shows that the postcondition follows from the convergence predicate upon termination. The game α repeats until convergence is reached ($\mathcal{M} = \mathbf{0}$). By the assumption that metrics are well-founded, convergence is guaranteed in finitely (but arbitrarily) many iterations.

A naïve, albeit correct, reading of rule ⟨∗⟩I says $\mathcal{M}$ is literally some term f. If lexicographic or otherwise non-scalar metrics should be needed, it suffices to interpret φ and $\mathcal{M}_0 \succ \mathcal{M}$ as formulas over several scalar variables.

Rule FP says $\langle\alpha^*\rangle\phi$ is a least pre-fixed-point. That is, if we wish to show a formula ψ holds now, we show that ψ is any pre-fixed-point, then it must hold as it is no lesser than ϕ. Rule [∗]I is the well-understood induction rule for loops, which applies as well to repeated games. Premiss O ensures [∗]I supports any provable postcondition, which is crucial for eliminating M in Lemma 7. The elimination form for $[\alpha^*]\phi$ is simply [∗]E. Like any program logic, reasoning in CGL consists of first applying program-logic rules to decompose a program until the

program has been entirely eliminated, then applying first-order logic principles at the leaves of the proof. The *constructive* theory of rationals is undecidable because it can express the undecidable [47] *classical* theory of rationals. Thus facts about rationals require proof in practice. For the sake of space and since our focus is on program reasoning, we defer an axiomatization of rational arithmetic to future work. We provide a (non-effective!) rule FO which says valid first-order formulas are provable.

$$\text{FO}\ \frac{\Gamma \vdash M : \rho}{\Gamma \vdash \mathsf{FO}[\phi](M) : \phi}\quad (\text{exists } a \text{ s.t. } \{a\} \times \mathfrak{S} \subseteq [\![\rho \to \phi]\!],\ \rho, \phi \text{ F.O.})$$

An effective special case of FO is split (Fig. 4), which says all term comparisons are decidable. Rule split can be generalized to decide termination metrics ($\mathcal{M} = \mathbf{0} \vee \mathcal{M} \succ \mathbf{0}$). Rule iG says the value of term f can be remembered in fresh ghost variable x:

$$\text{iG}\ \frac{\Gamma, p : x = f \vdash M : \phi}{\Gamma \vdash \mathsf{Ghost}[x = f](p.\ M) : \phi}\quad (x \text{ fresh except free in } M, p \text{ fresh})$$

Rule iG can be defined using arithmetic and with quantifiers:

$$\mathsf{Ghost}[x = f](p.\ M) \equiv (\lambda x : \mathbb{Q}.\ (\lambda p : (x = f).\ M))\ f\ (\mathsf{FO}[f = f]())$$

What's Novel in the CGL Calculus? CGL extends first-order reasoning with game reasoning (sequencing [32], assignments, iteration, and duality). The combination of first-order reasoning with game reasoning is synergistic: for example, repetition games are known to be more expressive than repetition systems [42]. We give a new natural-deduction formulation of monotonicity. Monotonicity is admissible and normalization translates monotonicity proofs into monotonicity-free proofs. In doing so, normalization shows that right-to-left proofs can be (lazily) rewritten as left-to-right. Additionally, first-order games are rife with changing state, and soundness requires careful management of the context Γ. The extended version [12] uses our calculus to prove the example formulas.

6 Theory: Soundness

Full versions of proofs outlined in this paper are given in the extended version [12]. We have introduced a proof calculus for CGL which can prove winning strategies for Nim and CC. For any new proof calculus, it is essential to convince ourselves of our soundness, which can be done within several prominent schools of thought. In proof-theoretic semantics, for example, the proof rules are taken as the ground truth, but are validated by showing the rules obey expected properties such as harmony or, for a sequent calculus, cut-elimination. While we will investigate proof terms separately (Section 8), we are already equipped to show soundness by direct appeal to the realizability semantics (Section 4), which we take as an independent notion of ground truth. We show soundness of CGL

proof rules against the realizability semantics, i.e., that every provable natural-deduction sequent is valid. An advantage of this approach is that it explicitly connects the notions of provability and computability! We build up to the proof of soundness by proving lemmas on structurality, renaming and substitution.

Lemma 1 (Structurality). *The structural rules W, X, and C are admissible, i.e., the conclusions are provable whenever the premisses are provable.*

$$\text{W}\ \frac{\Gamma \vdash M : \phi}{\Gamma, p : \psi \vdash M : \phi} \qquad \text{X}\ \frac{\Gamma, p : \phi, q : \psi \vdash M : \rho}{\Gamma, q : \psi, p : \phi \vdash M : \rho} \qquad \text{C}\ \frac{\Gamma, p : \phi, q : \phi \vdash M : \rho}{\Gamma, p : \phi \vdash [p/q]M : \rho}$$

Proof summary. Each rule is proved admissible by induction on M. Observe that the only premisses regarding Γ are of the form $\Gamma(p) = \phi$, which are preserved under weakening. Premisses are trivially preserved under exchange because contexts are treated as sets, and preserved modulo renaming by contraction as it suffices to have *any* assumption of a given formula, regardless its name. The context Γ is allowed to vary in applications of the inductive hypothesis, e.g., in rules that bind program variables. Some rules discard Γ in checking the subterms inductively, in which case the IH need not be applied at all. □

Lemma 2 (Uniform renaming). *Let $M\frac{y}{x}$ be the renaming of program variable x to y (and vice-versa) within M, even when neither x nor y is fresh. If $\Gamma \vdash M : \phi$ then $\Gamma\frac{y}{x} \vdash M\frac{y}{x} : \phi\frac{y}{x}$.*

Proof summary. Straightforward induction on the structure of M. Renaming within proof terms (whose definition we omit as it is quite tedious) follows the usual homomorphisms, from which the inductive cases follow. In the case that M is a proof variable z, then $\left(\Gamma\frac{y}{x}\right)(z) = \Gamma(z)\frac{y}{x}$ from which the case follows. The interesting cases are those which modify program variables, e.g., $\langle z := f\frac{w}{z} \text{ in } p.\ M\rangle$. The bound variable z is renamed to $z\frac{y}{x}$, while the auxiliary variable w is α-varied if necessary to maintain freshness. Renaming then happens recursively in M. □

Substitution will use proofs of coincidence and bound effect lemmas.

Lemma 3 (Coincidence). *Only the free variables of an expression influence its semantics.*

Lemma 4 (Bound effect). *Only the bound variables of a game are modified by execution.*

Summary. By induction on the expression, in analogy to [43]. □

Definition 9 (Term substitution admissibility). *For simplicity, we say ϕ_x^f (likewise for context Γ, term f, game α, and proof term M) is admissible if ϕ binds neither x nor free variables of f.*

The latter condition can be relaxed in practice [44] to requiring ϕ does not mention x under bindings of free variables.

Lemma 5 (Arithmetic-term substitution). *If $\Gamma \vdash M : \phi$ and the substitutions Γ^f_x, M^f_x, and ϕ^f_x are admissible, then $\Gamma^f_x \vdash M^f_x : \phi^f_x$.*

Summary. By induction on M. Admissibility holds recursively, and so can be assumed at each step of the induction. For non-atomic M that bind no variables, the proof follows from the inductive hypotheses. For M that bind variables, we appeal to Lemma 3 and Lemma 4. □

Just as arithmetic terms are substituted for program variables, proof terms are substituted for proof variables.

Lemma 6 (Proof term substitution). *Let $[N/p]M$ substitute N for p in M, avoiding capture. If $\Gamma, p : \psi \vdash M : \phi$ and $\Gamma \vdash N : \psi$ then $\Gamma \vdash [N/p]M : \phi$.*

Proof. By induction on M, appealing to renaming, coincidence, and bound effect. When substituting N for p into a term that binds program variables such as $\langle z := f\frac{y}{z} \text{ in } q.\ M\rangle$, we avoid capture by renaming within occurrences of N in the recursive call, i.e., $[N/p]\langle z := f\frac{y}{z} \text{ in } q.\ M\rangle = \langle z := f\frac{y}{z} \text{ in } q.\ [N\frac{z}{y}/p]M\rangle$, preserving soundness by Lemma 2. □

Soundness of the proof calculus exploits renaming and substitution.

Theorem 1 (Soundness of proof calculus). *If $\Gamma \vdash M : \phi$ then $(\Gamma \vdash \phi)$ is valid. As a special case for empty context $\cdot$, if $\cdot \vdash M : \phi$, then ϕ is valid.*

Proof summary. By induction on M. Modus ponens case $A\ B$ reduces to Lemma 6. Cases that bind program variables, such as assignment, hold by Lemma 5 and Lemma 2. Rule W is employed when substituting under a binder. □

We have now shown that the CGL proof calculus is sound, the *sine qua non* condition of any proof system. Because soundness was w.r.t. a realizability semantics, we have shown CGL is constructive in the sense that provable formulas correspond to realizable strategies, i.e., imperative programs executed in an adversarial environment. We will revisit constructivity again in Section 8 from the perspective of proof terms as *functional* programs.

7 Operational Semantics

The Curry-Howard interpretation of games is not complete without exploring the interpretation of proof simplification as normalization of functional programs. To this end, we now introduce a structural operational semantics for CGL proof terms. This semantics provides a view complementary to the realizability semantics: not only do provable formulas correspond to realizers, but proof terms can be directly executed as functional programs, resulting in a *normal* proof term. The chief subtlety of our operational semantics is that in contrast to realizer execution, proof simplification is a static operation, and thus does not inspect game state. Thus the normal form of a proof which branches on the game state is, of necessity, also a proof which branches on the game state. This static-dynamic

phase separation need not be mysterious: it is analogous to the monadic phase separation between a functional program which returns an imperative command vs. the execution of the returned command. While the primary motivation for our operational semantics is to complete the Curry-Howard interpretation, proof normalization is also helpful when implementing software tools which process proof artifacts, since code that consumes a normal proof is in general easier to implement than code that consumes an arbitrary proof.

The operational semantics consist of two main judgments: M normal says that M is a normal form, while $M \mapsto M'$ says that M reduces to term M' in one step of evaluation. A normal proof is allowed a case operation at the top-level, either $\langle \mathsf{case}\ A\ \mathsf{of}\ \ell \Rightarrow\ B \mid r \Rightarrow\ C \rangle$ or $\langle \mathsf{case}_*\ A\ \mathsf{of}\ s \Rightarrow\ B \mid g \Rightarrow C \rangle$. Normal proofs M without state-casing are called *simple*, written M simp. The requirement that cases are top-level ensures that proofs which differ only in where the case was applied share a common normal form, and ensures that β-reduction is never blocked by a case interceding between introduction-elimination pairs. Top-level case analyses are analogous to case-tree normal forms in lambda calculi with coproducts [4]. Reduction of proof terms is eager.

Definition 10 (Normal forms). *We say M is* simple*, written M* simp*, if eliminators occur only under binders. We say M is* normal*, written M* normal*, if M* simp *or M has shape $\langle \mathsf{case}\ A\ \mathsf{of}\ \ell \Rightarrow\ B \mid r \Rightarrow\ C \rangle$ or $\langle \mathsf{case}_*\ A\ \mathsf{of}\ s \Rightarrow\ B \mid g \Rightarrow C \rangle$ where A is a term such as $(\mathsf{split}\ [f, g]\ M)$ that inspects the state. Subterms B and C need not be normal since they occur under the binding of ℓ or r (resp. s or g).*

That is, a normal term has no top-level beta-redexes, and state-dependent cases are top-level. We consider rules $[*]$R, $[:*]$I, $[?]$I, and $\langle\!\![:=]\!\!\rangle$I binding. Rules such as $\langle * \rangle$I have multiple premisses but bind only one. While $[*]$R does not introduce a proof variable, it is rather considered binding to prevent divergence, which is in keeping with a coinductive understanding of formula $[\alpha^*]\phi$. If we did not care whether terms diverge, we could have made $[*]$R non-binding.

For the sake of space, this section focuses on the β-rules (Fig. 5). The full calculus, given in the extended version [12], includes structural and commuting-conversion rules, as well as what we call *monotonicity conversion* rules: a proof term $M \circ_p N$ is simplified by structural recursion on M. The capture-avoiding substitution of M for p in N is written $[M/p]N$ (Lemma 6). The propositional cases $\lambda\phi\beta$, $\lambda\beta$, caseβL, caseβR, $\pi_1\beta$, and $\pi_2\beta$ are standard reductions for applications, cases, and projections. Projection terms $\pi_1 M$ and $\pi_2 M$ should not be confused with projection realizers $\pi_L(a)$ and $\pi_R(a)$. Rule unpackβ makes the witness of an existential available in its client as a ghost variable.

Rule FPβ, repβ, and forβ reduce introductions and eliminations of loops. Rule FPβ, which reduces a proof $FP(A, s.\ B, g.\ C)$ says that if α^* has already terminated according to A, then B proves the postcondition. Else the inductive step C applies, but every reference to the IH g is transformed to a recursive application of FP. If A uses only $\langle * \rangle$S and $\langle * \rangle$G, then $FP(A, s.\ B, g.\ C)$ reduces to a simple term, else if A uses $\langle * \rangle$I, then $FP(A, s.\ B, g.\ C)$ reduces to a case. Rule repβ says loop induction $(M\ \mathsf{rep}\ p : J.\ N\ \mathsf{in}\ O)$ reduces to a delayed pair

$$\lambda\phi\beta\ (\lambda p:\phi.\ M)\ N \mapsto [N/p]M \qquad \mathsf{case}\beta\mathrm{L}\ \langle\!\langle \mathsf{case}\ \langle\!\langle \ell\cdot A\rangle\!\rangle\ \mathsf{of}\ \ell \Rightarrow\ B \mid r \Rightarrow\ C\rangle\!\rangle \mapsto [A/\ell]B$$
$$\lambda\beta\ (\lambda x:\mathbb{Q}.\ M)\ f \mapsto M^f_x \qquad \mathsf{case}\beta\mathrm{R}\ \langle\!\langle \mathsf{case}\ \langle\!\langle r\cdot A\rangle\!\rangle\ \mathsf{of}\ \ell \Rightarrow\ B \mid r \Rightarrow\ C\rangle\!\rangle \mapsto [A/r]C$$
$$\pi_1\beta\ \langle\!\langle \pi_1\langle\!\langle M,N\rangle\!\rangle\rangle\!\rangle \mapsto M \qquad \mathsf{unroll}\beta\ [\mathsf{unroll}\ [\mathsf{roll}\ M]] \mapsto M$$
$$\pi_2\beta\ \langle\!\langle \pi_2\langle\!\langle M,N\rangle\!\rangle\rangle\!\rangle \mapsto N$$
$$\mathrm{unpack}\beta\ \mathsf{unpack}(\langle f\tfrac{y}{x} :* q.\ M\rangle, py.\ N) \mapsto (\mathsf{Ghost}[x = f\tfrac{y}{x}](q.\ [M/p]N))\tfrac{x}{y}$$
$$\mathrm{FP}\beta\ FP(D,s.\ B,g.\ C) \mapsto (\langle \mathsf{case}_*\ D\ \mathsf{of}\ s \Rightarrow\ B \mid g \Rightarrow [(g\circ_z FP(z,s.\ B,g.\ C))/g]C\rangle)$$
$$\mathrm{rep}\beta\ (M\ \mathsf{rep}\ p:J.\ N\ \mathsf{in}\ O) \mapsto [\mathsf{roll}\ \langle M, ([M/p]N)\circ_q (q\ \mathsf{rep}\ p:J.\ N\ \mathsf{in}\ O)\rangle]$$
$$\mathrm{for}\beta\ \mathsf{for}(p:\varphi(\mathcal{M}) = A; q.\ B; C)\{\alpha\} \mapsto$$
$$\langle \mathsf{case\ split}\ [\mathcal{M}, 0]\ \ \mathsf{of}$$
$$\ell \Rightarrow\ \langle \mathsf{stop}\ [(A,\ell)/(p,q)]C\rangle$$
$$\mid r \Rightarrow\ \mathsf{Ghost}[\mathcal{M}_0 = \mathcal{M}](rr.\ \langle \mathsf{go}\ (([A,\langle rr, r\rangle/p,q]B)\circ_t (\mathsf{for}(p:\varphi(\mathcal{M}) = \pi_1 t; q.\ B; C)\{\alpha\}))\rangle)\rangle\rangle$$

Fig. 5. Operational semantics: β-rules

of the "stop" and "go" cases, where the "go" case first shows $[\alpha]J$, for loop invariant J, then expands $J \to [\alpha^*]\phi$ in the postcondition. Note the laziness of [roll] is essential for normalization: when $(M\ \mathsf{rep}\ p:J.\ N\ \mathsf{in}\ O)$ is understood as a coinductive proof, it is clear that normalization would diverge if repβ were applied indefinitely. Rule forβ for $\mathsf{for}(p:\varphi(\mathcal{M}) = A; q.\ B; C)\{\alpha\}$ checks whether the termination metric $\mathcal{M}$ has reached terminal value $\mathbf{0}$. If so, the loop $\langle\mathsf{stop}\rangle$'s and A proves it has converged. Else, we remember $\mathcal{M}$'s value in a ghost term $\mathcal{M}_0$, and $\langle\mathsf{go}\rangle$ forward, supplying A and $\langle r, rr\rangle$ to satisfy the preconditions of inductive step B, then execute the loop $\mathsf{for}(p:\varphi(\mathcal{M}) = \pi_1 t; q.\ B; C)\{\alpha\}$ in the postcondition. Rule forβ reflects the fact that the exact number of iterations is state dependent.

We discuss the structural, commuting conversion, and monotonicity conversion rules for left injections as an example, with the full calculus in [12]. Structural rule $\ell\cdot$S evaluates term M under an injector. Commuting conversion rule $\langle\!\langle\ell\cdot\rangle\!\rangle$C normalizes an injection of a case to a case with injectors on each branch. Monotonicity conversion rule $\langle\!\langle\ell\cdot\rangle\!\rangle\circ$ simplifies a monotonicity proof of an injection to an injection of a monotonicity proof.

$$\ell\cdot\mathrm{S}\ \frac{M \mapsto M'}{\langle\!\langle \ell\cdot M\rangle\!\rangle \mapsto \langle\!\langle \ell\cdot M'\rangle\!\rangle}$$
$$\langle\!\langle\ell\cdot\rangle\!\rangle\mathrm{C}\ \langle\!\langle \ell\cdot\langle \mathsf{case}\ A\ \mathsf{of}\ p \Rightarrow\ B \mid q \Rightarrow\ C\rangle\rangle\!\rangle \mapsto \langle \mathsf{case}\ A\ \mathsf{of}\ p \Rightarrow\ \langle\!\langle \ell\cdot B\rangle\!\rangle \mid q \Rightarrow\ \langle\!\langle \ell\cdot C\rangle\!\rangle\rangle$$
$$\langle\!\langle\ell\cdot\rangle\!\rangle\circ\ \langle\!\langle \ell\cdot M\rangle\!\rangle\circ_p N \mapsto \langle\!\langle \ell\cdot (M\circ_p N)\rangle\!\rangle$$

Fig. 6. Operational semantics: structural, commuting conversion, monotonicity rules

8 Theory: Constructivity

We now complete the study of CGL's constructivity. We validate the operational semantics on proof terms by proving that progress and preservation hold, and

thus the CGL proof calculus is sound as a type system for the functional programming language of CGL proof terms.

Lemma 7 (Progress). *If $\cdot \vdash M : \phi$, then either M is normal or $M \mapsto M'$ for some M'.*

Summary. By induction on the proof term M. If M is an introduction rule, by the inductive hypotheses the subterms are well-typed. If they are all simple, then M simp. If some subterm (not under a binder) steps, then M steps by a structural rule. Else some subterm is an irreducible case expression not under a binder, it lifts by the commuting conversion rule. If M is an elimination rule, structural and commuting conversion rules are applied as above. Else by Def. 10 the subterm is an introduction rule, and M reduces with a β-rule. Lastly, if M has form $A \circ_x B$ and A simp, then by Def. 10 A is an introduction form, thus reduced by some monotonicity conversion rule. □

Lemma 8 (Preservation). *Let $\mapsto^*$ be the reflexive, transitive closure of the $\mapsto$ relation. If $\cdot \vdash M : \phi$ and $M \mapsto^* M'$, then $\cdot \vdash M' : \phi$*

Summary. Induct on the derivation $M \mapsto^* M'$, then induct on $M \mapsto M'$. The β cases follow by Lemma 6 (for base constructs), and Lemma 6 and Lemma 2 (for assignments). C-rules and $\circ$-rules lift across binders, soundly by W. S-rules are direct by IH. □

We gave two understandings of proofs in CGL, as imperative strategies and as functional programs. We now give a final perspective: CGL proofs support synthesis in principle, one of our main motivations. Formally, the Existential Property (EP) and Disjunction Property (DP) justify synthesis [18] for existentials and disjunctions: whenever an existential or disjunction has a proof, then we can compute some instance or disjunct that has a proof. We state and prove an EP and DP for CGL, then introduce a Strategy Property, their counterpart for synthesizing strategies from game modalities. It is important to our EP that terms are arbitrary computable functions, because more simplistic term languages are often too weak to witness the existentials they induce.

Example 1 (Rich terms help). Formulas over polynomial terms can have non-polynomial witnesses.

Let $\phi \equiv (x = y \land x \geq 0) \lor (x = -y \land x < 0)$. Then $f = |x|$ witnesses $\exists y : \mathbb{Q}\ \phi$.

Lemma 9 (Existential Property). *If $\Gamma \vdash M : (\exists x : \mathbb{Q}\ \phi)$ then there exists a term f and realizer b such that for all $(a, \omega) \in [\![\bigwedge \Gamma]\!]$, we have $(b\,a, \omega_x^{f(\omega)}) \in [\![\phi]\!]$.*

Proof. By Theorem 1, the sequent $(\Gamma \vdash \exists x : \mathbb{Q}\ \phi)$ is valid. Since $(a, \omega) \in [\![\bigwedge \Gamma]\!]$, then by the definition of sequent validity, there exists a common realizer c such that $(c\,a, \omega) \in [\![\exists x : \mathbb{Q}\ \phi]\!]$. Now let $f = \pi_L(c\,a)$ and $b = \pi_R(c\,a)$ and the result is immediate by the semantics of existentials. □

Disjunction strategies can depend on the state, so naïve DP does not hold.

Example 2 (Naïve DP). When $\Gamma \vdash M : (\phi \vee \psi)$ there need not be N such that $\Gamma \vdash N : \phi$ or $\Gamma \vdash N : \psi$.

Consider $\phi \equiv x > 0$ and $\psi \equiv x < 1$. Then $\cdot \vdash \mathsf{split}\ [x, 0]\ () : (\phi \vee \psi)$, but neither $x < 1$ nor $x > 0$ is valid, let alone provable.

Lemma 10 (Disjunction Property). *When $\Gamma \vdash M : \phi \vee \psi$ there exists realizer b and computable f, s.t. for every ω and a such that $(a, \omega) \in [\![\bigwedge \Gamma]\!]$, either $f(\omega) = 0$ and $(\pi_L(b), \omega) \in [\![\phi]\!]$, else $f(\omega) = 1$ and $(\pi_R(b), \omega) \in [\![\psi]\!]$.*

Proof. By Theorem 1, the sequent $\Gamma \vdash \phi \vee \psi$ is valid. Since $(a, \omega) \in [\![\bigwedge \Gamma]\!]$, then by the definition of sequent validity, there exists a common realizer c such that $(c\,a, \omega) \in [\![\phi \vee \psi]\!]$. Now let $f = \pi_L(c\,a)$ and $b = \pi_R(c\,a)$ and the result is immediate by the semantics of disjunction. □

Following the same approach, we generalize to a Strategy Property. In CGL, strategies are represented by realizers, which implement every computation made throughout the game. Thus, to show provable games have computable winning strategies, it suffices to exhibit realizers.

Theorem 2 (Active Strategy Property). *If $\Gamma \vdash M : \langle\alpha\rangle\phi$, then there exists a realizer b such that for all ω and realizers a such that $(a, \omega) \in [\![\bigwedge \Gamma]\!]$, then $\{(b\,a, \omega)\}\langle\!\langle\alpha\rangle\!\rangle \subseteq [\![\phi]\!] \cup \{\top\}$.*

Theorem 3 (Dormant Strategy Property). *If $\Gamma \vdash M : [\alpha]\phi$, then there exists a realizer b such that for all ω and realizers a such that $(a, \omega) \in [\![\bigwedge \Gamma]\!]$, then $\{(b\,a, \omega)\}[\![\alpha]\!] \subseteq [\![\phi]\!] \cup \{\top\}$.*

Summary. From proof term M and Theorem 1, we have a realizer for formula $\langle\alpha\rangle\phi$ or $[\alpha]\phi$, respectively. We proceed by induction on α: the realizer $b\,a$ contains all realizers applied in the inductive cases composed with their continuations that prove ϕ in each base case. □

While these proofs, especially EP and DP, are short and direct, we note that this is by design: the challenge in developing CGL is not so much the proofs of this section, rather these proofs become simple because we adopted a realizability semantics. The challenge was in developing the semantics and adapting the proof calculus and theory to that semantics.

9 Conclusion and Future Work

In this paper, we developed a Constructive Game Logic CGL, from syntax and realizability semantics to a proof calculus and operational semantics on the proof terms. We developed two understandings of proofs as programs: semantically, every proof of game winnability corresponds to a realizer which computes the game's winning strategy, while the language of proof terms is also a functional

programming language where proofs reduce to their normal forms according to the operational semantics. We completed the Curry-Howard interpretation for games by showing Existential, Disjunction, and Strategy properties: programs can be synthesized that decide which instance, disjunct, or moves are taken in existentials, disjunctions, and games. In summary, we have developed the most comprehensive Curry-Howard interpretation of any program logic to date, for a much more expressive logic than prior work [32]. Because CGL contains constructive Concurrent DL and first-order DL as strict fragments, we have provided a comprehensive Curry-Howard interpretation for them in one fell swoop. The key insights behind CGL should apply to the many dynamic and Hoare logics used in verification today.

Synthesis is the immediate application of CGL. Motivations for synthesis include security games [40], concurrent programs with demonic schedulers (Concurrent Dynamic Logic), and control software for safety-critical cyber-physical systems such as cars and planes. In general, any kind of software program which must operate correctly in an adversarial environment can benefit from game logic verification. The proofs of Theorem 2 and Theorem 3 constitute an (on-paper) algorithm which performs synthesis of guaranteed-correct strategies from game proofs. The first future work is to implement this algorithm in code, providing much-needed assurance for software which is often mission-critical or safety-critical. This paper focused on discrete CGL with one numeric type simply because any further features would distract from the core features. Real applications come from many domains which add features around this shared core.

The second future work is to extend CGL to hybrid games, which provide compelling applications from the domain of adversarial cyber-physical systems. This future work will combine the novel features of CGL with those of the classical logic dGL. The primary task is to define a constructive semantics for differential equations and to give constructive interpretations to the differential equation rules of dGL. Previous work on formalizations of differential equations [34] suggests differential equations can be treated constructively. In principle, existing proofs in dGL might happen to be constructive, but this does not obviate the present work. On the contrary, once a game logic proof is shown to fall in the constructive fragment, our work gives a correct synthesis guarantee for it too!

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Inf. Comput. **163**(2), 409–470 (2000), https://doi.org/10.1006/inco.2000.2930
2. Aczel, P., Gambino, N.: The generalised type-theoretic interpretation of constructive set theory. J. Symb. Log. **71**(1), 67–103 (2006), https://doi.org/10.2178/jsl/1140641163
3. Alechina, N., Mendler, M., de Paiva, V., Ritter, E.: Categorical and Kripke semantics for constructive S4 modal logic. In: Fribourg, L. (ed.) CSL. LNCS, vol. 2142, pp. 292–307. Springer (2001), https://doi.org/10.1007/3-ν540-ν44802-ν0_21
4. Altenkirch, T., Dybjer, P., Hofmann, M., Scott, P.J.: Normalization by evaluation for typed lambda calculus with coproducts. In: LICS. pp. 303–310. IEEE Computer Society (2001), https://doi.org/10.1109/LICS.2001.932506

5. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5), 672–713 (2002), https://doi.org/10.1145/585265.585270
6. van Benthem, J.: Logic of strategies: What and how? In: van Benthem, J., Ghosh, S., Verbrugge, R. (eds.) Models of Strategic Reasoning - Logics, Games, and Communities, LNCS, vol. 8972, pp. 321–332. Springer (2015), https://doi.org/10.1007/978-ν3-ν662-ν48540-ν8_10
7. van Benthem, J., Bezhanishvili, G.: Modal logics of space. In: Aiello, M., Pratt-Hartmann, I., van Benthem, J. (eds.) Handbook of Spatial Logics, pp. 217–298. Springer (2007), https://doi.org/10.1007/978-ν1-ν4020-ν5587-ν4_5
8. van Benthem, J., Bezhanishvili, N., Enqvist, S.: A propositional dynamic logic for instantial neighborhood models. In: Baltag, A., Seligman, J., Yamada, T. (eds.) Logic, Rationality, and Interaction - 6th International Workshop, LORI 2017, Sapporo, Japan, September 11-14, 2017, Proceedings. LNCS, vol. 10455, pp. 137–150. Springer (2017), https://doi.org/10.1007/978-ν3-ν662-ν55665-ν8_10
9. van Benthem, J., Pacuit, E.: Dynamic logics of evidence-based beliefs. Studia Logica **99**(1-3), 61–92 (2011), https://doi.org/10.1007/s11225-ν011-ν9347-νx
10. van Benthem, J., Pacuit, E., Roy, O.: Toward a theory of play: A logical perspective on games and interaction. Games (2011), https://doi.org/10.3390/g2010052
11. Bishop, E.: Foundations of constructive analysis (1967)
12. Bohrer, R., Platzer, A.: Constructive hybrid games. CoRR **abs/2002.02536** (2020), https://arxiv.org/abs/2002.02536
13. Bridges, D.S., Vita, L.S.: Techniques of constructive analysis. Springer (2007)
14. Celani, S.A.: A fragment of intuitionistic dynamic logic. Fundam. Inform. **46**(3), 187–197 (2001), http://content.iospress.com/articles/fundamenta-νinformaticae/fi46-ν3-ν01
15. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR. LNCS, Springer (2007), https://doi.org/10.1007/978-ν3-ν540-ν74407-ν8_5
16. Coquand, T., Huet, G.P.: The calculus of constructions. Inf. Comput. **76**(2/3), 95–120 (1988), https://doi.org/10.1016/0890-ν5401(88)90005-ν3
17. Curry, H., Feys, R.: Combinatory logic. In: Heyting, A., Robinson, A. (eds.) Studies in logic and the foundations of mathematics. North-Holland (1958)
18. Degen, J., Werner, J.: Towards intuitionistic dynamic logic. Logic and Logical Philosophy **15**(4), 305–324 (2006). https://doi.org/10.12775/LLP.2006.018
19. Doberkat, E.: Towards a coalgebraic interpretation of propositional dynamic logic. CoRR **abs/1109.3685** (2011), http://arxiv.org/abs/1109.3685
20. Fernández-Duque, D.: The intuitionistic temporal logic of dynamical systems. Log. Methods in Computer Science **14**(3) (2018), https://doi.org/10.23638/LMCS-ν14(3:3)2018
21. Frittella, S., Greco, G., Kurz, A., Palmigiano, A., Sikimic, V.: A proof-theoretic semantic analysis of dynamic epistemic logic. J. Log. Comput. **26**(6), 1961–2015 (2016), https://doi.org/10.1093/logcom/exu063
22. Fulton, N., Platzer, A.: A logic of proofs for differential dynamic logic: Toward independently checkable proof certificates for dynamic logics. In: Avigad, J., Chlipala, A. (eds.) CPP. pp. 110–121. ACM (2016). https://doi.org/10.1145/2854065.2854078
23. Ghilardi, S.: Presheaf semantics and independence results for some non-classical first-order logics. Arch. Math. Log. **29**(2), 125–136 (1989), https://doi.org/10.1007/BF01620621
24. Ghosh, S.: Strategies made explicit in dynamic game logic. Workshop on Logic and Intelligent Interaction at ESSLLI pp. 74 –81 (2008)

25. Goranko, V.: The basic algebra of game equivalences. Studia Logica **75**(2), 221–238 (2003), https://doi.org/10.1023/A:1027311011342
26. Griffin, T.: A formulae-as-types notion of control. In: Allen, F.E. (ed.) POPL. pp. 47–58. ACM Press (1990), https://doi.org/10.1145/96709.96714
27. Harper, R.: The holy trinity (2011), https://web.archive.org/web/20170921012554/http://existentialtype.wordpress.com/2011/03/27/the-νholy-νtrinity/
28. Hilken, B.P., Rydeheard, D.E.: A first order modal logic and its sheaf models
29. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969). https://doi.org/10.1145/363235.363259
30. van der Hoek, W., Jamroga, W., Wooldridge, M.J.: A logic for strategic reasoning. In: Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M.J. (eds.) AAMAS. ACM (2005), https://doi.org/10.1145/1082473.1082497
31. Howard, W.A.: The formulae-as-types notion of construction. To HB Curry: essays on combinatory logic, lambda calculus and formalism **44**, 479–490 (1980)
32. Kamide, N.: Strong normalization of program-indexed lambda calculus. Bull. Sect. Logic Univ. Łódź **39**(1-2), 65–78 (2010)
33. Lipton, J.: Constructive Kripke semantics and realizability. In: Moschovakis, Y. (ed.) Logic from Computer Science. pp. 319–357. Springer (1992). https://doi.org/10.1007/978-1-4612-2822-6_13
34. Makarov, E., Spitters, B.: The Picard algorithm for ordinary differential equations in Coq. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP. LNCS, vol. 7998. Springer (2013), https://doi.org/10.1007/978-ν3-ν642-ν39634-ν2_34
35. Mamouras, K.: Synthesis of strategies using the Hoare logic of angelic and demonic nondeterminism. Log. Methods Computer Science **12**(3) (2016), https://doi.org/10.2168/LMCS-ν12(3:6)2016
36. Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Form. Methods Syst. Des. **49**(1), 33–74 (2016). https://doi.org/10.1007/s10703-016-0241-z, special issue of selected papers from RV'14
37. van Oosten, J.: Realizability: A historical essay. Mathematical Structures in Computer Science **12**(3), 239–263 (2002), https://doi.org/10.1017/S0960129502003626
38. Parikh, R.: Propositional game logic. In: FOCS. pp. 195–200. IEEE (1983), https://doi.org/10.1109/SFCS.1983.47
39. Pauly, M.: A modal logic for coalitional power in games. J. Log. Comput. **12**(1), 149–166 (2002), https://doi.org/10.1093/logcom/12.1.149
40. Pauly, M., Parikh, R.: Game logic - an overview. Studia Logica **75**(2), 165–182 (2003), https://doi.org/10.1023/A:1027354826364
41. Peleg, D.: Concurrent dynamic logic. J. ACM **34**(2), 450–479 (1987), https://doi.org/10.1145/23005.23008
42. Platzer, A.: Differential game logic. ACM Trans. Comput. Log. **17**(1), 1:1–1:51 (2015). https://doi.org/10.1145/2817824
43. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reas. **59**(2), 219–265 (2017). https://doi.org/10.1007/s10817-016-9385-1
44. Platzer, A.: Uniform substitution for differential game logic. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR. LNCS, vol. 10900, pp. 211–227. Springer (2018). https://doi.org/10.1007/978-3-319-94205-6_15
45. Pratt, V.R.: Semantical considerations on floyd-hoare logic. In: FOCS. pp. 109–121. IEEE (1976). https://doi.org/10.1109/SFCS.1976.27

46. Ramanujam, R., Simon, S.E.: Dynamic logic on games with structured strategies. In: Brewka, G., Lang, J. (eds.) Knowledge Representation. pp. 49–58. AAAI Press (2008), http://www.aaai.org/Library/KR/2008/kr08-ν006.php
47. Robinson, J.: Definability and decision problems in arithmetic. J. Symb. Log. **14**(2), 98–114 (1949), https://doi.org/10.2307/2266510
48. The Coq development team: The Coq proof assistant reference manual (2019), https://coq.inria.fr/
49. Van Benthem, J.: Games in dynamic-epistemic logic. Bulletin of Economic Research **53**(4), 219–248 (2001)
50. Vanderwaart, J., Dreyer, D., Petersen, L., Crary, K., Harper, R., Cheng, P.: Typed compilation of recursive datatypes. In: Shao, Z., Lee, P. (eds.) Proceedings of TLDI'03: 2003 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, New Orleans, Louisiana, USA, January 18, 2003. pp. 98–108. ACM (2003), https://doi.org/10.1145/604174.604187
51. Weihrauch, K.: Computable Analysis - An Introduction. Texts in Theoretical Computer Science. An EATCS Series, Springer (2000), https://doi.org/10.1007/978-ν3-ν642-ν56999-ν9
52. Wijesekera, D.: Constructive modal logics I. Ann. Pure Appl. Logic **50**(3), 271–301 (1990), https://doi.org/10.1016/0168-ν0072(90)90059-νB
53. Wijesekera, D., Nerode, A.: Tableaux for constructive concurrent dynamic logic. Ann. Pure Appl. Logic (2005), https://doi.org/10.1016/j.apal.2004.12.001