Refining Operational Logics

Joseph C. Osborn
Computational Media
Computer Science
University of California at Santa Cruz
1156 High Street
Santa Cruz, CA 95064
jcosborn@soe.ucsc.edu

Noah Wardrip-Fruin
Computational Media
University of California at Santa Cruz
1156 High Street
Santa Cruz, CA 95064
nwf@soe.ucsc.edu

Michael Mateas
Computational Media
University of California at Santa Cruz
1156 High Street
Santa Cruz, CA 95064
michaelm@soe.ucsc.edu

ABSTRACT

This paper expands on and refines the theoretical framework of operational logics, which simultaneously addresses how games operate at a procedural level and how games communicate these operations to players. In the years since their introduction, operational logics have been applied in domains ranging from game studies to game generation and game modeling languages. To support these uses and to enable new ones, we resolve some standing ambiguities and provide a catalog of key, fundamental operational logics.

Concretely, we provide an explicit and detailed definition of operational logics; specify a set of logics which seems fundamental and suffices to interpret a broad variety of games across several genres; give the first detailed explanation of how exactly operational logics combine; and suggest application domains for which operational logics-based analysis and knowledge representation are especially appropriate.

CCS CONCEPTS

•Computing methodologies → Knowledge representation and reasoning; Ontology engineering; •Applied computing → Media arts;

KEYWORDS

operational logics, playable models, proceduralist readings

ACM Reference format:
Joseph C. Osborn, Noah Wardrip-Fruin, and Michael Mateas. 2017. Refining Operational Logics. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 11 pages.
DOI: 10.1145/3102071.3102107

1 INTRODUCTION

Since their initial development in [23, 24], *operational logics* (OLs) have enjoyed broad use and inspired several approaches to game studies. Besides their direct use in describing games [9], OLs underlie several approaches to understanding how games communicate ideas [3, 21] and a variety of projects in player/game modeling and generation [4, 6, 7, 16–18]. The key move in all these cases has been to step away from considering games as bags of mechanics and towards viewing them as assemblages of abstract operations from diverse logics.

Outside of the game studies literature, in recent years the authors have taught an introductory undergraduate game design class using operational logics as the unifying theme. Each module of the course addresses a different set of operational logics, building up a core set of literacies for interpreting and for designing games. Besides the analysis of games, game-making tools are described in terms of the logics they support and/or foreground, which gives students context for understanding new tools' authorial affordances. Students progress through a variety of tools over the course of the quarter, and the operational logics framework gives them a touchstone for deciding which tool to use and how best to use it.

Although OLs have already proved useful, there remain some ambiguities in the theory and its use despite attempts at clarification [8]. In this work, we will therefore try to support the uses to which OLs have already been put—and possibly enable new uses—by taking on some of these ambiguities, refining the theory of operational logics. Hopefully, this new account will enable more scholars to use operational logics in their work, accounting for phenomena that were previously inaccessible. Specifically, we aim to address these questions:

(1) What operational logics are there besides collision, linking, and resource logics?
(2) Is the number of operational logics small, large, or infinite?
(3) What exactly is an abstract operation?
(4) How do logics communicate their operations to players?
(5) How are operational logics combined?
(6) How are operational logics different from playable models?
(7) Why work from the perspective of operational logics rather than some other theory?

In the following sections, we refine and reconcile previous definitions of operational logics and playable models; provide a catalog of operational logics with some old and new examples; and give a detailed account of the ways in which operational logics can combine together. To conclude, we argue for the utility of operational logics as a knowledge representation both for human and computational purposes.

FDG'17, Hyannis, MA, USA

978-1-4503-5319-9/17/08…$15.00
DOI: 10.1145/3102071.3102107

2 WHAT ARE OPERATIONAL LOGICS?

"Operational logics connect fundamental abstract operations, which determine the state evolution of a system, with how they are understood at a human level [2009]." Ultimately, operational logics provide the raw material from which game mechanics are built. At the same time—and from a different perspective—they tie together low-level and abstract mechanics with the presentation of sensory stimuli to players to enable an understanding of what the game is doing and how it functions. Operational logics are not beneath or above mechanics, but represent a different slice through a game which we feel is often a more useful view on how games function in the space between designers and players. We begin by breaking down and investigating extant definitions of operational logics, explaining each component with small examples; following that, we show some complete operational logics in the new, consolidated terms.

According to Mateas and Wardrip-Fruin, an operational logic is "a packaging of a rhetorical strategy—'an authoring (representational) strategy'—with a process—'supported by abstract processes or lower-level logics'—in order to provide an effective authorial affordance—supporting the specification of 'the behaviors a system must exhibit in order to be understood as representing a specified domain to a specified audience [2009].'" Osborn *et al.* write "operational logics are combinations of *abstract processes* with their *communicative roles* in a game, connected through an ongoing *game state presentation* and supporting a *gameplay experience* [2015]." These are similar but distinct in important ways: the former definition refers to possible lower-level logics and calls out the role of authorship in the logic, whereas the latter emphasizes the continuity over time of the presentation. Both center the combination of a *communicative role* and an *abstract process* so that idealized players can interpret meaningful causal relationships between observed phenomena and supposed underlying computations. We proceed by breaking down the definitions above into their primary components.

A *communicative role* (also called a communicative strategy) describes how the logic must be employed authorially to communicate its operations to players as part of the larger game system. In other words, we can group logics according to families which share the same communicative role, though they might support that role with different processes or state presentations. For example, the generic communicative role of collision detection could be stated as "Virtual objects can touch, and these touches can have consequences." If this is not revealed to players—e.g., if the objects' visual locations fall out of sync with their simulated positions—the role is not fulfilled and the logic falls apart. The communicative role is the phenomenon in terms of which designers author game rules, and which designers hope that players engage with and understand. Of course, such communication does not always succeed for all audiences, but the most common logics are widely (if subconsciously) understood by game-literate players. It is through this understanding that players develop their ability to play: to understand the game-world such that they can take action intentionally and interpret its results, an important part of the experience of agency many games provide [25].

An *abstract process* is a specification for how a process operates. For example, the abstract process for collision detection could be stated as, "When the coordinate spaces of two virtual objects intersect, declare the intersection." This specification is agnostic as to the specific algorithm and implementation—the coordinate spaces would be quite different for 2D and 3D games, as well as for games using rough bounding boxes versus pixel-accurate methods. Not all possible implementations of an abstract process may succeed in supporting the communicative role of the logic for all audiences or in all contexts. Candidate implementations must therefore be *e ective* as well as *feasible*: as Mateas and Wardrip-Fruin assert, the definition of the abstract process must be implementable on a computer.

The term "process" is used broadly here and covers both state-altering procedures and predicates; we find it useful to call the obligations a logic places on its implementation *abstract operations*, which we may also call *operators* of the logic. Some abstract operations specify relations or invariants maintained by the logic, while others describe the sorts of queries the logic supports, and still others opportunistically trigger abstract operations when predicates of a logic become true or false.

A *game state presentation* is how players see, hear, and feel the specific behavior of the operational logic in the context of the game. Games' platforms can offer diverse options for presentation: compare vector versus raster graphics, stereoscopic 3D versus flat images, and the proprioceptive experience of motion-controlled versus keyboard-controlled games. Different ways of presenting the game state can require very different data—often called game *assets*—to support presentation of the logic's operations. For example, the only asset specific to the presentation of collision detection in *Pong* is a sound triggered when the ball collides with a wall or paddle. But in a *Call of Duty* game, the presentation of collision detection for bullets and bodies alone may involve many animation and sound assets required to realistically present different types and locations of damage. This is a general trend as the level of detail represented in a game grows finer: more data assets are required for a logic's presentation. The crafting and selection of data is a key way game creators can suggest more concrete audience interpretations, especially for logics that have been established with relatively abstract communicative goals: pattern-matching logics can readily support *cra ing* if the patterns resemble the crafted artifact, and linking logics can naturally evoke *spatial connection* if the nodes and links are styled like a mass transit map.

The *gameplay experience* is what happens when a player encounters the logic. The player engages simultaneously with the sensory experience of the game state presentation, the intentional act of performing actions which change the game state, and the interpretive process of determining a mapping between physical interaction devices and the available actions (in terms of the operational logics' abstract operations). This is where the game's creators hope the communicative role will be fulfilled, and also where players may discover possibilities the designers never intended. Often player understanding and discovery is not immediate, and it may be imperfect, especially in its details—players learn through experimentation, after all. This is especially true on the boundaries between logics.

Even in a simple game such as *Pong* or *Breakout*, the physical reaction triggered when a ball collides with a paddle can differ depending on where along the paddle the ball collides; this basic connection between collision and physics may take time for players to grasp. Nonetheless, the communication that balls and paddles can collide, and that balls bounce off the paddle when this happens, takes place quickly in nearly all initial play sessions (confirming the expectations of game-literate players).

Note that players cannot have effective agency until they know what operational logics are at work: if they cannot interpret the events on the screen as being connected to some OL, such as physics or collision or resource transactions, they have no way of knowing their possible objectives, available actions, or even the system's current state. On the other hand, even a game with inconsistent or nonsensical combinations of instantial assets can still be played effectively. Treanor's discussion of *BurgerTime* [19] drives home the point that whether the player interprets *BurgerTime* as line cookery, construction, the inventive and intellectually rewarding life of the chef, or indeed combat with enormous animated foodstuffs, they can still play and win the game. On the other hand, if a player fails to recognize the logics at work—believing they are controlling the food rather than the chef, or interpreting the chef as a cursor for selecting burger components—their game will end with a very low score.

While *BurgerTime* has many rhetorical interpretations, some sensible and some ludicrous (indeed, the most obvious reading is ridiculous), it is impossible to avoid concluding that the player moves one character and not the others, that touching the egg or hot dog ends the round, that pepper halts the movement of the enemies, and so on. Operational logics communicate this latter kind of cause-and-effect relationship, enabling players to interpret the game state and its change over time.

2.1 Playable Models and Proceduralist Readings

Are all meaningful inferences about a game necessarily tied to operational logics? *BurgerTime* supports a reading of being about cooking, and in fact seems to communicate that cooking is taking place through its instantial assets as well as its system; but an argument from effective agency suggests that the concept of a "cooking logic" is not well-founded. Believing or not believing that cooking is taking place has no impact on one's performance of the game or ability to achieve scoring or survival objectives; on the other hand, it is vital to believe that pepper is a limited resource, that dropping burger parts on enemies scores points, and so on. Operational logics must be understood to be leveraged in play, and players that do not comprehend the logics at work cannot play effectively. In that case, what is this other class of interpretations which admits superficial or inconsistent readings? OLs have previously been discussed in relation to at least two different notions of procedural representation: playable models [11] and the "representational considerations" of proceduralist readings [22].

"Playable models compose structuring information with varying operational logics to support the player in incremental exploration, intention formation, and interpretation" of a concept or system [11]. The authors distinguish between "fully playable" and "trivial" models. Playability seems to be used to measure both the model's complexity and the degree to or depth at which players come to understand the system and form plans using it. How are these models different from operational logics, beyond comprising combinations of them? First, playable models rely more strongly (but not exclusively) on cultural/social knowledge. A playable model of combat needs the player to know what violence is, and a playable model of securities exchange can only function for players familiar with property and commerce (although with this background, even an abstract game of squares and circles or randomly generated stock ticker symbols would be sufficient). Second, a playable model can be realized using a variety of different combinations of logics, whereas operational logics are themselves committed to particular abstract processes. Finally, one can attempt to construct a playable models for nearly any natural or social phenomenon, whereas operational logics seem tied to questions of how players interpret low-level interactions with a system.

Each playable model is a lens through which a group of phenomena can be seen to represent a concept or system. For example, the model of combat synthesizes a variety of somewhat independent systems into a coherent argument. We might imagine reusing the same systems, perhaps with different instantial assets, to represent dating and romance—but here players would be able to independently form the idea that they are engaging in combat and this *doubling* would be an argument in and of itself: the claim that *love is a ba le eld.* Whereas an operational logic can be reused in different contexts and still be recognized—merely refined and not altered—models are much more contingent. It is one thing if a resource logic is used to govern interactions with a weapons shop in a game; but if social interactions with non-player characters also hinge on the exchange of money or gifts, these characters quickly come to feel like vending machines or shops themselves.

Apart from playable models, we have the kinds of meanings that are often the target of proceduralist readings. Treanor *et al.* provide examples of interpretations (evaluative, volitional, and rhetorical) afforded by graphical logics in the context of *Kaboom!* [20]. This is a more general type of *aboutness* which is often inextricable from cultural knowledge. Moreover, these meanings are inherently subjective and open to interpretation and argument. This is not meant as a value judgement; rather, the goal of their project is to address how games make rhetorical claims, rather than how players come to understand the workings of game systems. Proceduralist readings are one *strategy* (among many) for making sense of games, which happen to have been historically rooted in OLs.

Operational logic interpretations, distinct from both of these, tend to halt at the level of cultural knowledge and determining their presence or absence is less sensitive to argumentation. Some types of *literacy* may be required to recognize or engage with a logic, and of course these come from cultural experiences, but a (blurry) line can be drawn between this kind of mechanical engagement and more metaphorical interpretation. Looking to film, it's the distinction that can be drawn between montage theory and the trope of the training montage: the former is a logic of the juxtaposition of instants of time, while the latter suggests personal growth over a period of time using the former. While we could argue that a particular montage does not function as a training montage, we

have specific conditions—the fulfillment of a communicative role—that allow us to claim with some measure of certainty whether a sequence of shots is a montage at all. The ways that metaphor and cultural knowledge are used in the communicative role of a logic are usually relatively simple and straightforward. If interpretation begins to make more complex metaphorical mappings, this is a signal that one is moving out of the domain of the logic and into a different type of meaning. Thematically, *Pong* isn't *about* collision—it can be *interpreted as* tennis due to its *playable model* of reflection and momentum provided mainly by a *collision logic*.

Operational logics do present instantial assets to players, as described earlier; but the OL functions and supports players' agency even if the player's avatar is a hockey puck shooting coffee cups at swans. This combination of assets is nonsensical from the standpoint of playable models (neither coffee cups nor hockey pucks are implicated in the swan ecosystem) and from proceduralist readings (while the swan wants to avoid the coffee cup, we can't argue in good faith that the swan is a metonym for caffeine-intolerant consumers and that the hockey puck represents coffee culture). But a player could play and do well at this game, understanding its physics, goals, and behaviors. All that matters from an operational logics perspective is that the objects' appearances do not confound the communicative role: if our coffee cup projectiles had a visual appearance vastly larger or smaller than their actual collision bounds, or if the enemy's sprite were utterly indistinguishable from the background, players would not be able to make sense of the game whatsoever.

If the operations of a logic are not communicated well—i.e., if the communicative role is not fulfilled—then the authorial affordances of the logic do not successfully translate into affordances for the players. They are in effect trying to play a different game from the one they are actually interacting with; there is a *bug* either in the realization of the abstract process or in the game state presentation (or perhaps the player does not possess the required literacies). The *Breakout* player who does not realize that the paddle has five segments will see the ball's movement as random, when in fact no chance logic is at work. If the ball occasionally passes through the paddle due to a bug, players can only choose to accept the bug or to invent an explanation to assign a cause to the phenomenon. We can distinguish partial understanding (not knowing about the segments) from the inability to form an understanding due to an inconsistent system.

We also have the case of full understanding: not only when the player knows the rules as the designer does, but when the player understands the logics at work *better* than the designer. Imagine a player who understands how pseudo-random numbers are generated within a game—for example, that the frame count modulo 60 is used to generate numbers. This player can manipulate luck as they please, while still engaging with the notion that a chance logic is at work.

The purpose of this distinction is to shift the infinity of concepts a game can be about squarely into the domain of playable models and other forms of interpretation and, hopefully, away as much as possible from operational logics. This is a vital precondition for our project of cataloging and classification.

2.2 How Operational Logics Communicate

Compared to abstract operations, the communicative roles and game state presentations of operational logics are relatively undertheorized. While the translation from abstract operations to implemented code can be fairly natural, most examples of communicative roles and presentations have been either very specific (e.g., how a health bar shows the state of the health resource) or extremely broad (e.g., two-dimensional shapes on a screen standing in for colliding objects); this is true even in the discussion above! It is clear that the role and state presentation of the logic are tightly connected, just as the role and abstract operations are tightly connected. A concrete presentation must effectively reveal the workings of a logic in the same way that an implementation of the abstract processes must effectively enact them.

Recall that the communicative role is what we must explain to support an interpretation that the logic is present. The presentation maps the underlying game state into sensory phenomena to support that role. Every game takes its own approach to that mapping for each of its logics, fulfilling their respective communicative roles—exactly as every game has its own concrete implementations of operational logics' abstract processes. Just as we can recognize the abstract operations of resource transactions independently of their concrete implementation in different games, we can discuss approaches to game state presentation that are common across games. Resource logics, for example, are often presented by placing numbers next to icons and/or textual labels, or animating a floating number near the resource pool whose quantity of contained resources has just changed.

Every logic can be identified with a somewhat open-ended set of presentation strategies, although these necessarily depend on the literacies of the target audience. Often, these strategies may overlap in some ways with those of other logics. As an example, the two-dimensional combat game *Worms* expresses both its collision and physics logics by drawing all simulated characters and terrain on the same plane, so that visual overlaps and perceived movement reflect simulated collisions and physical dynamics. When characters in *Worms* are injured, the amount of damage is displayed as a floating number originating at the character's head and rising upwards while fading out. The resource logic leverages some of the same *channels* used by the collision and physics logics' game state presentation.

Ultimately, game state presentation can only be defined completely when logics are composed into a game. It is the composition proper that provides the set of communication channels used to fulfill the communicative role according to the appropriate presentation strategies. We save a complete discussion of how operational logics compose for Sec. 4.

3 ENUMERATING OPERATIONAL LOGICS

The most commonly cited operational logics in the literature are collision, resource, and linking logics. Additionally, terms like *textual logics* and *graphical logics* were included in the first formulations of operational logics. Readers of earlier texts on operational logics may have concluded that these five terms comprise the whole universe of operational logics. Occasionally, systems emerge that posit new operational logics [9], but the canonicity of these examples

is not completely clear. We know from Mateas and Wardrip-Fruin that the preconditions of communicative intent and implementable process exclude large classes of candidate operational logics [2009], but we have no standard catalog of logics. In fact, we do not even know whether the number of logics is small (fewer than ten), moderate (a few dozen), large (several hundred), or infinite! Answering this question is a central goal of this paper.

We argue that this uncertainty is essentially a category error. "Collision logic" is not a single, monolithic entity, but a family of operational logics which all possess sufficient communicative roles, abstract operations, etc to function as collision logics. "Physics logics" could address momentum, gravity, wind-shear, or other physical behaviors; each pairing of the operation simulating that system with the human-visible phenomena which communicate it could be seen as a tiny operational logic in the family of physics logics, working in concert to form one specific larger physics logic (or, equivalently, the larger physics logic could be seen as comprising a superset of the abstract operations and game state presentations of the constituent physical behaviors). Likewise, "graphical logics" are exactly those logics whose game state presentations are primarily graphical, and by convention refers to the composition of collision, physics, character-state, and simple resource logics.

Mateas and Wardrip-Fruin exemplify their definition of operational logics with a *random event resource management logic* which merely parameterizes a resource logic with random chance; this seems to immediately open the door to an unbounded number of logics, all on equal footing with each other. This reduces the power of operational logics as an underpinning for game mechanics, because it seems that nearly every mechanic could be phrased as having a corresponding special-cased operational logic. It is more useful to separate the *random event* part from the *resource management* part, yielding a resource logic and a *chance logic*; this also removes the dependency on another specific logic's presence. While it is certainly possible to delineate an unbounded number of logics, we prefer to focus on *families* of logics, generally grouped by their abstract operations and communicative roles. While the number of *concrete* logics is limited only by human imagination, the number of *families* is potentially amenable to enumeration.

Guided by the utility of *graphical logics* as an umbrella term to characterize what is the same and what is different between arcade games and other games featuring colliding characters, we cataloged the main families of operational logics by examining particular games and genres [10] (some entries are reproduced in Fig. 2). Our catalog describes, for each logic, its main components: the abstract process and commonly-seen abstract operations, the communicative role, and conventional game state presentation strategies. We must stress that this is preliminary work biased towards home console games of the 1980s and 1990s. Future work could perform a grounded analysis or some other rigorous approach to sampling games and coding them in terms of which logics are present and in what ways. So far, we have identified fifteen prominent operational logics that appear widely (see Fig. 1); as we identify new ones they will be added to the catalog.

We assembled our catalog by playing and analyzing games, accounting for high-level concepts like *power-ups* or *turn scheduling* in terms of existing and candidate operational logics. We consider

Camera	Chance	Character-State
Collision	Control	Game Mode
Linking	Persistence	Physics
Progression	Recombinatory	Resource
Selection	Spatial Matching	Temporal Matching

Figure 1: Fifteen key operational logics

both the abstract operations that could be used to implement a concept and the communicative roles that must be fulfilled to obtain a satisfactory reading. This approach mirrors that taken in [11] to identify how models are built up from groups of operational logics' abstract operations.

We chose a set of around two dozen home console games from 1984-1998, with a few additions to include some rhythm games. For the initial set of logics in our catalog, we tried to address a breadth of loose (and admittedly ill-defined and overlapping) genre categories: action, fighting, puzzle, rhythm, role-playing, simulation, sports, and strategy. Games were selected for their notability and for diversity within genre groupings. The full list is available alongside the catalog [10]; these specific games are not intended to form any kind of canon or authoritative resource, only to act as a starting point for seeding the catalog.

3.1 Fundamental Logics

Generally, we identify logics by going from perception down to abstract operations: reasoning from observed behavior towards an underlying process. Where possible, we have tried to explain observed phenomena in terms of previously-known logics; if these explanations were awkward or did not reflect both the abstract process *and* the communicative role of the candidate logic, we acquiesced and formalized it as a new logic. While a logic or combination of logics can suffice to implement an abstract process of another logic, this can feel abstract and allegorical even in the best case. For example, a full resource pool inhibits the addition of new resources in a similar way as a collision logic inhibits the movement of objects into occupied space; but this blockage is hard to interpret successfully as *collision*, being more adequately interpreted as *saturation* or *exceeding capacity*. To detect a redundancy, fulfilling the abstract process is important; but we only have a truly redundant set if the communicative role—broadly construed—can be readily expressed by other logics.

It is not, however, the case that every notion we might wish to communicate to players has a corresponding operational logic. We set aside the primarily culturally-informed concepts which support interpretative frameworks such as proceduralist readings along with the complex playable models which are evidently built out of multiple logics: combat, for example, or cooking. After this step, we still have to apply some selection process to determine if a concept is or is not backed by a novel logic. We want, as much as possible, a parsimonious catalog. We would prefer to work with a collision logic, but not a logic of space; a physics logic, but not a logic of driving or running; a resource logic, but not a logic of growth. While collision logics may be used to model space, and physics logics may be used to communicate running, and resource increase may be used to model personal growth, the latter set of concepts do not

Name	**Collision Logics**
Communicative role	Virtual objects can touch, and these touches can have consequences.
Abstract process	Detection of overlaps between subsets of entities and the automatic triggering of reactions when these occur.
Abstract operations	Determine or alter which entities may collide with each other. Determine which entities are overlapping. Determine or alter the size of an entity. Separate the positions of two or more entities such that they do not overlap Whenever one of the above predicates is or becomes true or false, trigger an operation of this or another logic.
Presentation	Shapes, images, or 3-D models for each entity whose dimensions match those of the corresponding entity, projected on a plane. Audiovisual effects when a collision occurs. The presence or absence of text indicating whether two entities are in physical contact.
Required concepts	Entities and their positions; space.
Provided concepts	Collision, overlapping, the extents of objects.

Name	**Persistence Logics**
Communicative role	Some things in the world stay the same across sessions or between rooms or when scrolled offscreen, while others change or reset.
Abstract process	Determining which entities, assets, and variables are *in* a particular *scope* (e.g., saved-game scope, current-level scope, visible-area scope) and persisting or resetting those as the scope changes.
Abstract operations	Determine or alter which persistence scopes are active. Set, clear, or restore the facts associated with a particular scope. Trigger an action of another logic when scopes change, are saved, or are restored.
Presentation	Give audiovisual feedback when scopes change or when variables have been reset. When the player is about to leave a scope, give textual descriptions of what data will be preserved and what will be lost.
Required concepts	Abstract objects and facts governed by scoping rules.
Provided concepts	Active and inactive scopes, saves, and resets.

Name	**Resource Logics**
Communicative role	Generic or specific resources can be created, destroyed, converted, or transferred between abstract or concrete locations.
Abstract process	Enacting resource transactions within or between locations and triggering actions when conditions are met.
Abstract operations	Check if a transaction involving some units of certain resources is possible for some given locations. Perform a resource transaction. Perform an action of another logic when a transaction occurs. Perform an action of another logic when the quantity of resources within a location crosses a threshold.
Presentation	Indicating resource types by icon or text, locations by distinct regions of the screen, and resource quantity by numeric labels. Trigger an audiovisual effect at the instant a transaction occurs. Intertwined type and quantity representation, e.g., ten units of resource R appear as ten icons. Transactions described as lists of prerequisite resources and quantities and lists of product resources and quantities, graphically dimmed if insufficient.
Required concepts	Resource types and locations, including a way to count resources of a type in a location and a way to add/remove resources.
Provided concepts	Resource transactions and aggregate quantities.

Name	**Character-State Logics**
Communicative role	Virtual agents can act in different ways at different times.
Abstract process	Governing the finite, discrete states of a set of game characters or other entities, and the automatic triggering of reactions when states change.
Abstract operations	Determine or alter the discrete state of an entity according to a given transition system. When an entity's discrete state changes, perform some operators of this or another logic. Synchronize state changes between several entities.
Presentation	Change a character's sprite, animation, or visual effects according to current state. Trigger an audiovisual effect at the instant the state changes. Textual or iconic depictions of the character's state.
Required concepts	Entities and condition predicates for making transitions.
Provided concepts	Entities in/out of a state; state transition events.

Figure 2: Four example catalog entries.

qualify as *primary* or *fundamental* logics: we can find other logics that model space, running, or growth equally well using completely different game state presentations and abstract processes. The "growth" of an object increasing in size is quite different from the "growth" of a character's strength statistic increasing. While we could imagine a category of *growth logics* which include collision logics, resource logics, linking logics, et cetera, this is best left as a grouping by possible interpretive affordances (similarly, *graphical* and *textual logics* are groupings of related logics).

Consider, for example, how *vision* works in three games: *e Legend of Zelda: A Link to the Past*, *DOOM*, and *Metal Gear Solid*. In the first and third, we generally see a rectangular region around our avatar which stops at linked room boundaries; this region slides around as we move, like a camera on a rig might. *DOOM*, on the other hand, provides a first-person view from our avatar's perspective; the camera does move along with our movements, but in a much more direct and immediate way, as if it were hand-held or helmet-mounted.

These games model several distinct kinds of sight. Besides the rectangular area in view of the camera, *Zelda* has some dark rooms which are lit locally by the hero's lantern and globally by lighting wall or floor sconces; the former, local light functions as a kind of polygonal overlay, whereas the latter acts more like a timed resource whose quantity controls illumination. Moreover, enemies in *Zelda* tend to ignore Link until he comes close or enters a region of space just ahead of their current position and orientation. Enemies also have a kind of sight more akin to the lantern's light than to the panning camera. This type of vision is a defining characteristic of *Metal Gear Solid*, and avoiding being seen is a key consideration during play. Functionally, what an enemy can see (or not) is more like *DOOM*'s perspective camera than it is like the panning camera used in scrolling through the space of a level.

It is clear that vision is a concept that games can and do model in a variety of ways. We can imagine many implementations (colliding with vision volumes, being on the other side of a closed or open door, being in shadows or in a spotlight, drawing or not drawing a sprite) and ways to communicate that status (changes in character behavior or background music). There is, however, a key distinction to be drawn between, on the one hand, the communicative role of indicating that agents may or may not see other agents in particular situations; and on the other hand, communicating to a player that *they* are seeing part of a larger environment.

We could hypothesize a single *vision* logic, but we have seen that this is too open-ended and does not have one clear communicative role. Then, why not have both a *vision* logic and a *camera* logic, separating out the role of showing who can see what and the role of showing the player a sub-region of the level? Neither candidate logic depends significantly on cultural knowledge, which is promising; but the former is readily explained in terms of e.g., collision or resource logics, while the latter does not have such a clean decomposition. A moving camera could be expressed, in some cases, as a stage which is moving around under the lens; but it is strange to think of the *DOOM* avatar as a stationary object in a rapidly-moving jumble of walls. We are forced to posit a *camera logic*, one of whose uses might even be the modeling of vision (e.g., in a multiplayer game).

To generalize from this example, we ask two questions when attempting to explain a modeled concept and determine whether it justifies the introduction of a new operational logic: First, does this candidate logic have a distinctive role? Second, does the candidate logic admit a usefully concrete implementation? Splitting out cameras from other types of vision is an example of distinguishing based on the first question; but what does it mean to ask if a candidate logic has an implementable or usefully concrete abstract process?

We often see *scheduling* in games: for example, the turn order of a role-playing game or the alternation between players in a turn-taking game. But the abstract process this suggests—"determine who goes next and whose turn it is, and give the current player control"—is at once too generic and too specific. A good operational logic has a process which is abstract but puts constraints on possible implementations, and at the same time does not over-constrain its set of abstract operations and limit the contexts in which the logic can be used (recall that abstract operations describe the sorts of game state transitions that the logic enables). A scheduling logic phrased as above excludes the possibility of simultaneous turns, and if it expands any further to incorporate that it morphs into a version of the more general *control logic* ("different entities are controlled by different inputs at different times"). For an example of applying both considerations, consider menus in games. Their abstract process is "select from some possibly hierarchical options and trigger actions based on those options" and their communicative role is that the player has many options of which they may choose one. These are already so close to each other that something seems amiss; moreover, the candidate *menu logic* seems to step on both *selection logics* (governing which of a set of items is currently designated as selected) and control logics, overlapping with them while also overly specializing them to the case of menus. We also see many different kinds of menus in games: we could press a keyboard key to activate a menu item, or move a cursor to select and then press a key to confirm, or move the player character into one or another doorway or region of space to activate a menu item.

We see from this breadth of implementations (in some cases engaging totally different logics) that menus are not likely to have their own operational logic; or, rather, that the explanatory power of menu logics is poor. The candidate "menu logic" fails to have a distinctive role.

These questions also allow us to distinguish, for example, game-mode logics (characterized by state machines) from character-state logics (also characterized by state machines). Although their abstract processes are similar in the sense that both transition some entity through a state transition system, they have completely distinct communicative roles.

4 HOW DO OPERATIONAL LOGICS COMPOSE?

Mateas and Wardrip-Fruin assumed that there were low- and high-level operational logics, with the latter being strictly built out of the former; playable models were not strongly positioned with respect to this hierarchy [2009]. Treanor *et al.* explicitly distinguished logics from rhetorical argumentation and illustrated how the latter could be justified in terms of the former [2011]. Osborn *et al.* showed an example of how multiple logics could form a playable model when taken together [2015]. These works established *that* logics compose and *for what purposes*, but not *in what ways*. We argue that operational logics do not combine in a simple hierarchy; in fact, they compose in a variety of cross-cutting, overlapping, highly contextual, and complex ways.

Fortunately, the composition of operational logics is not totally unstructured. We identify three main points of contact or sharing between logics: communicative channels, operational integration,

and structural synthesis. Every composition of multiple logics (e.g., a game, genre, or game-making tool) must define the semantics of each of these interfaces. Even if the same logics are deployed, different choices for this glue can result in very different games.

For example, a side-scrolling beat-em-up may use a collision logic to determine damage, a character-state logic to determine character behaviors, and a resource logic to track the health of each character. Here, the display of characters' health bars over their heads (as opposed to putting them in corners of the screen) is a shared *communicative channel* (to wit, the character's embodied sprite); the triggering of resource transactions in response to collisions is an *operational integration*; and the precise, frame-by-frame data-flow and mapping between a character's position in the world, its current character-state, and the positions of its hit-boxes and hurt-boxes (damaging and vulnerable collision areas) is a natural example of *structural synthesis*.

The composition of operational logics opens up four key questions:

(1) How do the logics fulfill their communicative role, i.e., how do they enact their game state presentation?
(2) Where logics overlap and depend on each other, often in cyclic ways, how is this condition expressed in the rules and resolved at runtime?
(3) What is the game state, exactly, and how is that initialized and managed across time?
(4) When logics demand different parts of the game state, or have different sorts of inputs and outputs or world models, how is this mapping done?

Question one is addressed by communicative channels; the second is the subject of operational integration. The last two concerns are the domain of structural synthesis.

4.1 Communicative Channels

The simplest way for logics to overlap is by sharing *communicative channels*. This can be seen as a kind of semantic multiplexing, where affordances offered by one logic can be leveraged by another. Of course, operational logics must be defined independently of their possible relations to other logics, so it falls on the composition of logics to define how one logic's communicative role and game state presentation strategy are fulfilled in the context of another's.

Games with distinct characters or sprites, e.g., from character-state or collision logics, commonly share the space around the sprite as a channel. Regardless of how the sprites are positioned in the world, additional regions of the screen might be allocated for individual characters (for example the status displays in a fighting or role-playing game). Channels can also be duplicated or parameterized: games with split-screen multiplayer provide several distinct copies of the same configuration of channels (one for each player). Ultimately, what pixels get drawn to the screen (or which tokens are positioned around a board) has to be defined somewhere, and that display has to incorporate information from every logic. This type of composition is always present, even if the individual logics interact in no other ways.

Compared to operational logics, graphical and textual design are relatively well-understood. All the laws of composition and information design apply, and a complete discussion of how a designer may choose to enact the game state presentation of a logic is outside the scope of this project. Suffice it to say that each logic has its own universe of discourse—the colliders of a collision logic, the resource quantities and types of a resource logic—and it falls to designers and programmers to decide upon and realize a game's eventual sensory output, consistent with the communicative roles and game state presentation of the involved operational logics. These two aspects of operational logics are a bridge to these other, well-established disciplines.

4.2 Operational Integration

Sometimes, an operational logic may need to make a decision based on a fact determined in another logic, or it may need to trigger an action drawn from its own operators or those of another logic. Examples include a character who is injured when collisions occur, an enemy that changes behavior when the player comes close, or a resource transaction which is only available when a specially timed sequence of button inputs occurs. We call this kind of composition *operational* in the sense that it has to do with combining the abstract operations of several logics.

The example catalog entries in Fig. 2 include, in their abstract operations, phrases like "trigger an operation of this or another logic." These are explicit sites of composition where logics can integrate their operators together.

One way to think about this type of composition is that every logic defines a set of logical predicates and a set of actions: questions that can be asked about the logic's view of the game world and changes in the world that this logic is competent to make. Operational integrations are a type of composition that can be defined almost exclusively at this level, mostly agnostic of the underlying implementation of the integration. While their concrete semantics—how data about character positions are used to determine whether character behavior changes should take place—are up to the implementation, they at least *have* a clear implementation-independent abstract semantics. Communicative channels and structural synthesis, on the other hand, admit much less space between the implementation and the specification.

This operational integration might be implemented by a blackboard where the programmatic systems implementing logics write their calculated facts and desired actions, through object-oriented messaging and observation, via forward-chaining inference, or by an event propagation mechanism. If the game specifies, for example, an ordering among logics—that movements are resolved before collisions, and the collision handling code is processed before determining the applications of a spatial pattern matching logic—then shared variables could suffice to implement the operational integration.

4.3 Structural Synthesis

We have accounted for how operational logics' concepts are presented to players via (possibly shared) communicative channels and how mechanics can be built out of several logics through operational integrations. But we have glossed over how concepts like characters and space—which are required and provided by a variety of logics—are eventually resolved and unified. This is different from referring to a black-box predicate (are two characters touching?)

or triggering an opaque action (perform a resource transaction): a character-state logic and a physics logic need to somehow refer to the same set of characters. *Structural synthesis* defines how game state flows between operational logics, how it is mapped between their different internal representations, and so on.

In the catalog entries above, one pair of fields has so far gone unexplained: required and provided concepts. This is a kind of ontological free variable, a placeholder for terms which can't be defined within the logic itself or which could be converted into the native terms of other logics (in the style of modular action languages [5]). Consider, for example, graphical logic games comprising collision, physics, character-state, and resource logics. A *game character* is not just a collision volume, nor just a point mass, nor just a state machine, nor just a resource pool. The same character *identity* has to be consistent across all of these logics. The physics logic repositions the point mass subject to the constraints of the collider, and with velocity and acceleration determined by the state machine; if a projectile hits the collider, the corresponding resource pool must have health resources removed; and so on. Some of these integrations are operational, but all assume an external *mapping* to place them into the same conceptual framework.

Crafting systems in games are often designed as simple resource transactions: take three of item A and combine them with two of item B to make one of item C. When inventories are also described in terms of item types and quantities, the mapping seems natural: a movement from one resource pool to another. On the other hand, many games feature inventories with a spatial embedding—multiple packs and satchels, with items taking up different numbers and arrangements of slots, where items may form stacks of bounded size—implicating e.g., a collision or linking logic alongside resource logics in the determination of what items are available for crafting. In cases like these, the resource logic behind the crafting system must have a way of counting resources of a given type in a given abstract location, and a way of inserting any newly created or transferred resources into an abstract location. The resource transaction should be disallowed, for example, if the pack has no room for an object with the product's shape. This mapping is (and must be) specific to the concrete composition in question, and it is a question of structural data-flow. How is the game state (opaque and inscrutable to individual logics and operators) projected out onto the terms of a particular logic or set of operations? And how are the effects of these operations merged back in to the global state? What is the game state, exactly, and how is it set up and managed across time?

For example, in arcade games of the sort produced by Game-o-Matic, little bundles of arbitrary, isolated mechanics are glued together into characters [22]. The game-making toolkit Game Maker emphasizes essentially the same logics as Game-o-Matic, but its characters are instead described as coherent sets of condition/reaction pairs.

These trade-offs do not only happen in game engines, but in individual games. If a character moves too far from an active enemy, that enemy might *despawn* or reset; from an operational integration standpoint it does not matter whether the enemy continues to exist in the world, but the actual game state might reuse the memory associated with that enemy for some other purpose. If persistence logics are used to determine what characters might be active for the purpose of low-level collision checking or state updating, that is an example of structural synthesis; so is the use of a linking logic to help determine which region of 2D space a camera is scrolling over. Operational integrations and communicative channel sharing are, eventually, built on top of this structural synthesis.

Camera	Interval logics
Chance	Bayesian graphical models
Character-state	Networks of finite state machines
Collision	Qualitative/quantitative spatial constraints
Control	Action logics, event calculi
Game mode	Hierarchical finite state machines
Linking	Second-order logic over fixed graphs
Persistence	Separation logic, nominal logic
Physics	Switched systems of differential equations
Progression	Modal logics
Recombinatory	Production systems, automata
Resource	Multiset rewriting, numerical transition systems
Selection	Monadic second-order logic
Spatial Matching	Term rewriting
Temporal Matching	(Metric) temporal logic

Table 1: Key operational logics and formal correspondents

5 APPLICATIONS

At this point, we have a complete account of operational logics: their constituent definitions, a (partial) catalog of fundamental logics, and a description of the ways in which they compose, both at the level of abstract semantics and concrete implementations. Why go to the effort of constructing this conceptual apparatus? What is the benefit of an operational logics approach? In fact, we have already seen cases where a more developed theory of operational logics has proved useful. We divide these applications into two rough categories: human interpretation and machinic knowledge representation.

What do humans get out of operational logics? The utility of operational logics in game studies has already been shown in the works cited in our introduction. Hopefully, this new account will enable more scholars to use operational logics in their work, accounting for phenomena that were previously inaccessible to the theory.

Operational logics are also a useful basis for knowledge representation in and around games, mainly in the areas of game modeling and reverse-engineering. Several projects in this area were cited earlier, but now we have the theoretical foundation that helps explain why they were successful. The key move in these projects has been to step away from considering games as bags of mechanics and towards viewing them as assemblages of abstract operations from diverse logics, with suitable formal mappings. Formal logic has clear applications to games—both the linear-logic-based Ceptre and the answer set semantics [14, 15] have been used to great effect in design and prototyping. We therefore include a provisional mapping between operational logics and well-understood formal logics (Tab. 1), for potential integration in a logical framework such as Satisfiability Modulo Theories [2]. This mapping aims to be as natural as possible, preserving the qualitative and human-relevant aspects of the communicative roles while finding concrete semantics for the abstract processes. This has clear applications in game design

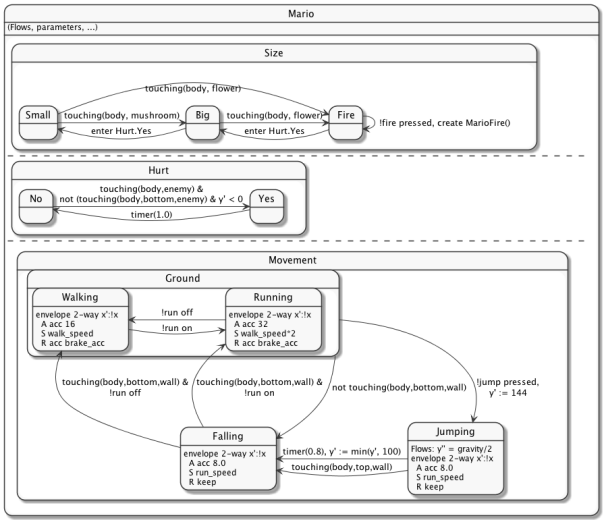

Figure 3: (Simplified) Super Mario as a hybrid automaton.

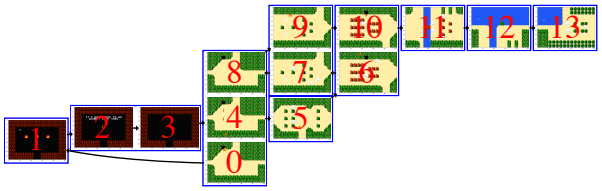

Figure 4: Hypothesized structure of the first few rooms of *The Legend of Zelda* (reproduced from [12]).

verification and a potential further generalization of the automated proceduralist readings work above.

If we have a well-defined way to compose operational logics and a well-defined way to compose formal logics, we could potentially bridge these to obtain modular, compositional specification and verification of game designs. As an example of the benefits of this approach, the authors have seen productive connections between the theory of hybrid automata (a combination of finite state machines and switched systems of differential equations) [1] and graphical logic games; this has led to work both in modeling languages (as in Fig. 3) and in reverse-engineering game mechanics [16], recovering hybrid automata specifications from game characters [17], and automatically mapping game levels (as in Fig. 4) from observations of game play [12].

Finally, we also see applications to AI general videogame playing: the GVG-AI competition specifically targets graphical logic games, and recent years' progress on general videogame playing has focused more on exploring causal and relational aspects of these games' collision logics [13]. We believe that an expanded operational logics approach could lead to even more generalized approaches, moving from the domain of specific mechanics into general combinations of abstract operations.

Operational logics are a versatile and powerful formalism addressing both perception and simulation. We hope that this work and the catalog of logics encourage their broader use both by scholars and by designers of game-making tools, modeling languages, and other computational systems. Games are more than just loose collections of homogeneous mechanics, and operational logics yield a clear alternative representation.

REFERENCES

[1] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. 1993. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*. Springer.

[2] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. 2009. Satisfiability Modulo Theories. *Handbook of satis ability* 185 (2009), 825–885.

[3] Ian Bogost. 2007. *Persuasive games: e expressive power of videogames*. Mit Press.

[4] Eurico Doirado and Carlos Martinho. 2010. I mean it!: detecting user intentions to create believable behaviour for virtual agents in games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 83–90.

[5] Vladimir Lifschitz and Wanwan Ren. 2006. A modular action description language. In *AAAI*, Vol. 6. 853–859.

[6] Chris Martens. 2015. Ceptre: A language for modeling generative interactive systems. In *Eleventh Arti cial Intelligence and Interactive Digital Entertainment Conference*.

[7] Chris Martens, Adam Summerville, Michael Mateas, Joseph Osborn, Sarah Harmon, Noah Wardrip-Fruin, and Arnav Jhala. 2016. Proceduralist readings, procedurally. In *Twel h Arti cial Intelligence and Interactive Digital Entertainment Conference*.

[8] Michael Mateas and Noah Wardrip-Fruin. 2009. Defining operational logics. *Digital Games Research Association (DiGRA)* 4 (2009).

[9] Joshua McCoy, Mike Treanor, Ben Samuel, Aaron A Reed, Michael Mateas, and Noah Wardrip-Fruin. 2013. Prom Week: Designing past the game/story dilemma.. In *FDG*. 94–101.

[10] Joseph C. Osborn. 2017. Operational Logics Catalog. (2017). http://operational-logics.soe.ucsc.edu

[11] Joseph C Osborn, Dylan Lederle-Ensign, Noah Wardrip-Fruin, and Michael Mateas. 2015. Combat in Games. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*.

[12] Joseph C. Osborn, Adam Summerville, and Michael Mateas. 2017. Automatic Mapping of NES Games with Mappy. In *Proceedings of the 2017 Workshop on Procedural Content Generation*.

[13] Diego Pérez-Liébana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M Lucas. 2016. Analyzing the robustness of general video game playing agents. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 1–8.

[14] Adam M Smith, Eric Butler, and Zoran Popovic. 2013. Quantifying over play: Constraining undesirable solutions in puzzle design.. In *FDG*. 221–228.

[15] Adam M Smith, Mark J Nelson, and Michael Mateas. 2009. Computational Support for Play Testing Game Sketches.. In *AIIDE*.

[16] Adam Summerville, Joseph C. Osborn, Christoffer Holmgård, Daniel Zhang, and Michael Mateas. 2017. Mechanics Automatically Recognized via Interactive Observation: Jumping. In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (2017).

[17] Adam Summerville, Joseph C. Osborn, and Michael Mateas. 2017. CHARDA: Causal Hybrid Automata Recovery via Dynamic Analysis. In *Proceedings of the International Joint Conference on Arti cial Intelligence* (2017).

[18] Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. 2012. Game-O-Matic: Generating Videogames that Represent Ideas.. In *PCG@ FDG*. 11–1.

[19] Mike Treanor and Michael Mateas. 2011. BurgerTime: A Proceduralist Investigation. In *Conference of the Digital Games Research Association-DIGRA 2011*.

[20] Mike Treanor, Michael Mateas, and Noah Wardrip-Fruin. 2010. Kaboom! is a Many-Splendored Thing: An interpretation and design methodology for message-driven games using graphical logics. In *Proceedings of the Fi h International Conference on the Foundations of Digital Games*. ACM, 224–231.

[21] Mike Treanor, Bobby Schweizer, Ian Bogost, and Michael Mateas. 2011. Proceduralist Readings: How to Find Meaning in Games with Graphical Logics. In *Proceedings of the 6th International Conference on Foundations of Digital Games (FDG '11)*. ACM, 115–122. DOI: http://dx.doi.org/10.1145/2159365.2159381

[22] Mike Treanor, Bobby Schweizer, Ian Bogost, and Michael Mateas. 2012. The micro-rhetorics of Game-O-Matic. In *Proceedings of the International Conference on the Foundations of Digital Games*. ACM, 18–25.

[23] N. Wardrip-Fruin. 2005. Playable media and textual instruments. 34 (2005), 211–253.

[24] Noah Wardrip-Fruin. 2006. Expressive Processing: On Process-Intensive Literature and Digital Media. (2006).

[25] Noah Wardrip-Fruin, Michael Mateas, Steven Dow, and Serdar Sali. 2009. Agency reconsidered. *Breaking New Ground: Innovation in Games, Play, Practice and eory. Proceedings of DiGRA 2009* (2009).