What kind of interaction must the automata have in order to move from base syntactic abilities to fully fledged structuring semantic abilities? To answer this question, however, we need to answer a more general question: What is interaction as a distinctly computational notion?

ENTER THE COPYCAT: INTERACTION AS GAME AS COMPUTATION

According to the classical Church-Turing computability thesis, computation can be modelled on a mathematical function. The machine receives input; it undergoes a sequence of state transitions (moves or legal runs) and yields an output. During the computation, the machine shuts out the environment and accepts no further input until the initial input is processed. In this form, computation can be represented as a logical deduction where *to compute* means the same as *to deduce steps from an initial configuration*. Like deduction, the process of computation can then be characterized in terms of how the output (conclusion) is logically contained in the input (the premise) or how the output/conclusion is implied by the input/premise. But if this is all that computation is—algorithmic deduction—then what exactly is gained by it? Computation runs into the same problem that deduction leads to, namely the riddle of epistemic omniscience according to which the total knowledge of an agent can be said to be deductively closed:

$$(K_a p \wedge (p \rightarrow q)) \rightarrow K_a q$$

which says that if the agent a knows (K) the proposition p then it knows all its logical consequences q.

While such a verdict is absolutely sound and valid in some classical logic heaven, it has no ground in reality. A corresponding computational equivalent of deduction's problem of epistemic omniscience is the classical denotational semantics of programming languages, where all that is required to provide meaning to the expression of a language (i.e., the meaning of the program) or to know what the program does, is to know the given structured sets of data types (domains) and the interpretation function that maps those domains to one another. Here, the puzzling situation is that computation is supposed to yield new information, whereas the classical paradigm suggests a closed logical system wherein the output information is nothing but the input plus a machine that deduces steps from it. From an information-theoretic point of view, no new information is added by computation, and from an epistemic point of view no knowledge is gained by computation.

What the *classical* account of computation presupposes but does not incorporate as its intrinsic dimension is that which provides the input and consumes the output—namely, the *environment*. The system or machine becomes the model of what computes, at the expense of keeping the environment in the background. The environment is merely represented as an average behaviour, rather than something that dynamically and actively interacts with the system. But it is only by highlighting the role of the environment that the system's or machine's function—input-output mapping—can coherently make sense. Moreover, it is only in the presence of an environment (another system, machine, or agent) that computation can be understood as an increase—rather than mere preservation—of information, hence avoiding the riddle of epistemic omniscience. From this perspective, the environment is no longer a passive ambient space but an active-reactive element to which the system responds by accepting input from it and yielding output which the environment then consumes. In view of the role of the environment in the input-output mapping, the question of computation, rather than being couched in terms of what is a computable function, shifts to the question of how the computation is executed. By bringing the *howness* of computation into the foreground, the question of computability turns into the question *What is computation?*—that is, it now concerns the interaction or confrontation of actions between machine and environment.

With the understanding of interaction itself as computation, the system-environment (or player-opponent, machine-network) dynamics can be modelled as a so-called interaction or simply general game. The interaction game differs from game-theoretic account of games in a number of significant ways. It does not require procedural rules or rules that dictate how the game should proceed or what moves/actions should be performed in a given situation. Rules naturally emerge in the context of the interaction itself. Furthermore, unlike a game-theoretic conception of the game, an interaction game is devoid of both predetermined winning strategies and payoff functions that map the strategy profile of each player to its payoffs or rewards. In this respect, the generality of interaction makes games unifying themes in the study of structures, (information) contents, and behaviours.[253] From a technical perspective, interaction games capture the fundamental correspondences between computational behaviours, logical contents, and mathematical structures—that is to say, the so-called computational trinity.[254] In this sense, interaction games are *generalizations* of the Brouwer–Heyting–Kolmogorov interpretation, where the notion of construction can be interpreted in terms of computing programs, logical proofs, and composition of categories as mathematical objects. In Brouwer's terms, the notion of construction can be broadly understood as a mental construct capable of verifying the existence (denoted by an existential quantifier $\exists$) of an object that falls under it such that the construction of a (mental) object (*gegenstand* rather than *objekt*) can be said to be the proof of its existence.[255]

253 Treating games as unifying themes can perhaps be traced back to the work of Stanislaw Ulam on using the logico-computational notion of the game to study all mathematical situations, as expressed in his slogan 'Gamify (*paizise* from the Greek παιςιη, to play) everything'. See S. Ulam, *A Collection of Mathematical Problems* (New York: Interscience, 1960).

254 'The doctrine of computational trinitarianism holds that computation manifests itself in three forms: proofs of propositions, programs of a type, and mappings between structures. These three aspects give rise to three sects of worship: Logic, which gives primacy to proofs and propositions; Languages, which gives primacy to programs and types; Categories, which gives primacy to mappings and structures. The central dogma of computational trinitarianism holds that Logic, Languages, and Categories are but three manifestations of one divine notion of computation. There is no preferred route to enlightenment: each aspect provides insights that comprise the experience of computation in our lives.' Harper, *The Holy Trinity*.

255 See L.E.J. Brouwer, 'Mathematics, Science and Language', in P. Mancosu (ed.),

In terms of a general game, the computing machine or system can be identified as a function box with input and output sides which represent the switching of roles between the system and the environment:

$$\begin{tikzcd} {} \arrow[r, "input\text{:}\,x"] & \boxed{\text{System: } f} \arrow[r, "output\text{:}\,f(x)"] & {} \end{tikzcd}$$

On the input side, the system is the consumer (querying the environment for an input which it can use) and the environment is a producer (providing the input); on the output side, the roles are reversed: the system is the producer (giving the output) and the environment is the consumer. In defining computational tasks as games played by the system and the environment alternatively switching their roles to react to one another (making a move, i.e., performing an action), what counts as a task for the environment is a computational resource for the machine, and what counts as a task for the machine is a resource for the environment. Within the framework of interaction games, computability can then be defined as the condition of winnability for the system against an environment—that is, the existence of an algorithmic winning strategy to solve a computational problem or perform a computational task in the presence of an active environment or opponent.

Access to (computational) resources and performance of permissible moves or actions are now explicitly stated as constraints put in place by the interchange of roles between the machine and the environment. The actions of the machine and its access to resources—i.e., the runtime or number of elementary operations performed by the machine and the space or memory required to solve a computational problem—are determined by its interaction with an environment that constrains its actions and its use of resources. In light of this, the interactive paradigm can naturally express computational complexity, with different levels of constraints imposed by the switching of roles expressing different classes of computational complexity. This switching of roles is formally defined by way of the logico-mathematical

From Brouwer to Hilbert: The Debate on the Foundations of Mathematics in the 1920s (Oxford: Oxford University Press, 1998), 45–53.

notion of duality of interaction—or simply, duality. Traditionally in logic, duality can be defined through De Morgan's laws, written in rule form as:

$$\frac{\neg(p \wedge q)}{\therefore \neg p \vee \neg q} \qquad \frac{\neg(p \vee q)}{\therefore \neg p \wedge \neg q}$$

and in the sequent or entailment form as:[256]

$$\neg(p \wedge q) \vdash (\neg p \vee \neg q) \qquad \neg(p \vee q) \vdash (\neg p \wedge \neg q)$$

Here, duality can be understood in terms of the operation of classical negation $\neg$ over conjunction and disjunction.

In mathematics, dualities can be generally defined as principles translating or mapping theorems or structures to other theorems and structures by means of an involutive function e.g., $f(f(x)) = x$ which is a function that, for every x in the domain of f, is its own inverse. A simple example of a mathematical duality can be given in terms of the algebraic connection between the numbers $\{-1, +1\}$. The involutive negation $(--x=x)$ connects these two numbers $(-(-1)=+1, -(+1)=-1)$. Alternatively, another elementary duality can be defined for sets where a set *Set* and its dual or opposite antiset Set^{op} are in complementary relations in terms of their given subsets:

256 Due to Gerhard Gentzen, sequent calculus is a style of formal argumentation in which proofs are written line by line through sequents consisting of premises and conclusions separated by a consequences relation symbolized by a turnstile ($\vdash$) which reads 'entails, yields or implies'. The premises (Γ) written on the left side of the turnstile are called antecedents, while conclusions (Δ) are written on the right side of the turnstile and are called consequents: $\Gamma \vdash \Delta$. Sequent calculus can be understood as the generalized form of a natural deduction judgement: A_1, ..., $A_n \vdash B_1, \dots, B_k$ where the commas on the left side can be thought of as 'and/conjunction' while commas on the right side (conclusions) can be understood as 'or/disjunction'. Accordingly, $A_1, A_2 \vdash B_1, B_2$ approximately reads as 'A_1 and A_2 yield or infer B_1 or B_2'. Since in sequent calculus each line of proof is a conditional tautology, meaning that it is inferred from other lines of proofs in accordance with rules and procedures of inference, it allows a representation of proofs as branching trees whose roots are the formulas that are to be proved.

For any given fixed set S, the subset $A \subseteq S$ has a complementary subset A^C such that A^C consists of elements of S not contained in A. Once again, applying the involution to A^C–$(A^C)^C$ yields A^C. In this fashion, $\mathcal{S}et$ includes $A \subseteq B$, while $\mathcal{S}et^{op}$ includes $B^C \subseteq A^C$.

In the framework of interaction, duality does not need to be predefined by means of the classical negation operator. For negation we substitute the reversal of symmetries or swapping of the roles in the game.[257]

In the context of interaction, there can be many configurations of how the game can evolve via the interchange of roles. For example, depending on whether the confrontation of actions between system and environment is synchronous or asynchronous, whether the game can branch to subgames, or whether the history of past interactions can be preserved and accessed or not, the game may exhibit new behaviours with higher computational complexity. The classical version of the Church-Turing paradigm of computation represents an elementary or restricted form of interaction games where computability can be understood as the winnability condition within a two-step game (i.e., a game with two moves only) of input and output. In this game, there is one and only one right output for a given input. Similarly, in classical logic, the notion of truth can be thought of as the winnability condition restricted to propositions in a zero-step game (i.e., games with no moves) which is automatically lost or won depending on whether the propositions are false or true.

257 This way of understanding duality can reveal many different forms of dualities other than those available in classical logic and algebraic structures. Dualities can be topological and geometrical in such a way that they can be applied to geometric-topological objects (e.g., by reversing the dimensions of the features of a cube, one can obtain an octahedron, and vice versa). In the same vein, the interactive duality between +1 and −1 is an interval [−1,+1] of reals that topologically connects them. Or in category theory, where duality can be roughly understood as turning around morphisms (i.e., swapping the source and target of the arrows) as well as reversing the composition of morphisms so that, for example, by inverting morphisms in the category C its dual C^{op} is obtained, and vice versa.

To understand the basic structure of interaction as computation, let us briefly examine the basic formal definition of an interaction game:

The game G for the player $\top$ (the system, the prover) and the opponent $\bot$ (the environment, the refuter) is a 4-tuple $\langle R_G, m_G, P_G, \vdash_G \rangle$, where:

- M_{G} is a set of moves (actions performed by $\top$ and $\bot$)
- λ_{G} is a function that labels the moves of $\top$ and $\bot$ and whether the moves are defences/answers (D) or attacks/queries (A): $\lambda_G\colon M_G \rightarrow (\bot,\top) \times (A,D)$

  So that $(\bot,\top) \times (A,D) = \{\bot\ A, \bot\ D, \bot\ A, \bot\ D\}$ and $\lambda_G = \langle \lambda_G^{\bot\top}, \lambda_G^{AD} \rangle$,

And where runs are sequences of moves, and positions in the game are finite runs.

- The play $P_G \subseteq M_G^{alt}$ is a non-empty subset of the set of alternating sequences of moves between $\top$ and $\bot$, M_G^{alt}. The set M_G^{alt} can be represented as the play or set of switching moves s:

  $$s = \lambda_G\, a_{1_\bot}\, a_{2_\top} \ldots a_{2k+1_\bot}\, a_{2k+2_\top} \cdots$$

  Additionally, since the alternating sequences of moves implies the switching of roles, the labelling function λ_{G} also finds its dual, the reverse function $\overline{\lambda_G}$ such that for a move m we have: $\overline{\lambda_G}(m) = \bot A \Leftrightarrow \lambda_G(m) = \top A$.

- $\vdash_G$ is a satisfying or justificatory relation that the represents permission to perform actions or to make moves:

  If the move m is the initial move, it either needs 'no justification', $\star \vdash_G m$; or if it requires to be justified by another move n if n has been antecedently justified/permitted, $n \vdash_G m$. All subsequent moves bear justificatory information or a pointer to an earlier move n played. Similarly, the switching of roles also ranges over permissions on the moves:

$$m \vdash_G n \wedge \lambda_G^{AD}(n) = D \Rightarrow \lambda_G^{AD}(m) = A$$

Within this game, we can then have subgames of the general form A and B and their compositions denoted by the tensor product $\otimes$ which are constructions over G. Subgames can be thought of as threads and subthreads opened up within a dialogue. Thus, for the given subgames A and B, we have basic compositions of $A \otimes B$ which, following Andreas Blass's semantics of interaction for the linear logic operator $\otimes$, can be interpreted as saying that the play can continue in *both* A and B alternately:[258]

$$M_{A\otimes B} = M_A + M_B$$

$$\lambda_{A\otimes B} = \lambda_A + \lambda_B$$

$$P_{A\otimes B} = \{s \in M_G^{alt}\}$$

$$\vdash_{A\otimes B} = \vdash_A + \vdash_B$$

The game always starts with an action or move made by $\perp$ since it is the environment that constrains the system and keeps it going. Accordingly, the opening move (played by $\top$) is always a request for data or a query from $\perp$. At the level of a subgame, permissible moves and the switching of roles become more complicated. Only $\perp$ can switch between subgames, while $\top$ can only react to the latest subgame in which $\perp$ has played. This constraint can be formalized as follows:

For any play $s \in P_{\mathrm{A}\otimes\mathrm{B}}$, if s_i is in A and s_{i+1} in A then $\lambda_{\mathrm{A}\otimes\mathrm{B}}(s_i) = \top$ and $\lambda_{\mathrm{A}\otimes\mathrm{B}}(s_{i+1}) = \perp$.

The continuation of the game at the level of subgames is denoted as $A \multimap B$—that is, subgame B consumes or uses as a computational resource

258 A. Blass, 'Is Game Semantics Necessary?', in *Computer Science Logic: International Workshop on Computer Science Logic* (Dordrecht: Springer, 1993), 66–77.

the subgame A. The symbol $\multimap$ stands for linear implication, where the implication is resource-sensitive: the environment is a resource for the task of the system (a computational problem) and by consuming this resource the system performs a task to yield an output which is now a resource for the environment. The game $A \multimap B$ can be intuitively thought of in terms of access-protocols between a server and a client (or machines on a network). The server $A \multimap B$ acts as a server of type B that provides data of type B if it has access—exactly once as a client—to server A which provides data of type A. The client of $A \multimap B$ will therefore act not only as a client of type B but also as a server of type A which provides the input or resource from A.[259] The game $A \multimap B$ is characterized in terms of the game's 4-tuple:

$M_{A \multimap B} = M_A + M_B$

$\lambda_{A \multimap B} = [\overline{\lambda}_A, \overline{\lambda}_B]$ such that for move m, $\overline{\lambda}_A(m) = \top$ when $\lambda_A(m) = \bot$, and $\bot$ when $\lambda_A(m) = \bot$.

$\top_{A \multimap B} = \{s \in M^{alt}_{A \multimap B}\}$ where the first move in $P_{A \multimap B}$ must be performed by $\bot$ in B, and the opening moves in A are labelled as $<$ by the function $\overline{\lambda}_A$.

$\vdash_{A \multimap B} = \star \vdash_{A \multimap B} m$ if the initial move requires no justification/permission, or else $\vdash_{A \multimap B} = m \vdash_{A \multimap B} n$.

In this setting, if we were to represent a computational problem (e.g., computing n+1 in the classical Church-Turing paradigm), this game could be diagrammed as a tree consisting of nodes and branches. At the top of the tree, we have a game played by the machine $\top$ represented as the first node that ramifies to the first-level branches which are $\bot$-labelled (corresponding to inputs provided by the environment) and leading to the second-level nodes of the tree which are games played by the environment $\bot$ $(\bot_1, \bot_2, \bot_3, \ldots)$. In the same vein, the second-level nodes lead to second-level branches, which

259 Ibid., 71.

are $\top$-labelled ($\top_1$, $\top_2$, $\top_3$, ...) and correspond to the output. The second-level branches lead to the third-level nodes which represent the winnability of games played alternatively by the environment and the machine ($\bot,\top,\bot,\top\ldots$).

Such a two-step game would count as computation in terms of input-output mapping functions. More general interaction games (i.e., games with more than two steps or ordered two-level branching) represent computational *behaviours* that are not exactly functions in the classical mathematical sense. This is in fact the single most significant underlying claim of the interactionist approach to computation: that not all computational behaviours or tasks can be modelled on functions in the mathematical sense. Rather than input-output mapping functions performed by a machine, behaviours are evolving interactions between machine/system and environment. Therefore, realizing behaviours is not simply a matter of simulating an observable behaviour by means of a system's function (a simulator)—as a behaviourist or a traditional functionalist would claim—but of reenacting the interaction between system and constraining environment. Such an interaction is modelled on a game that is not restricted to two steps, i.e., a game that is played on subgames and their branches.

In view of the minimal interactional or game constraints introduced above, it would not be difficult to imagine how the relaxation of existing constraints or the addition of new constraints to the game could result in more complex (computational) behaviours. Additional playing constraints can be generated as the game or interaction progresses, thus allowing the adoption of new strategies by players, i.e., *rules* by which the game can be played (cf. the notion of open harness introduced in chapter 1). Alternatively, some existing constraints can be relaxed or additional interaction operations can be added to enrich the semantics of interaction, hence generating more complex computational behaviours expressed by different compositions of strategies.

For example, the constraint of responding to the latest move of the opponent in the subgame can be suspended and an interaction feature (operation) added so that both players can be free to play their opening moves in subgames. An oversimplified but nevertheless helpful example of such a modification can be expressed in terms of the everyday use of personal computers. Traditionally, strategies in games are understood as functions from positions (histories of past interactions) to moves or actions. However, if the strategy of the computer was simply a function from positions to moves, it would have then been required to check the entire ever-increasing history of interactions—of moves and counter-moves. The task of the computer would then over time run into complications such as the exponential slowing of speed (time to process) and the increase in space or memory needed to respond to the next countermove. But neither is it the case that the computer just responds to our last keypunch, since it is capable of storing, accessing, and scanning histories of past interactions.[260]

Interaction-as-computation, in this sense, can be seen as pertaining to possible compositions of strategies which, depending on how the semantics of interaction is interpreted (i.e., the interpretation of constraints and operations involved in the interaction), would cover richer computational behaviours not limited to the computation of functions. Furthermore, in the interaction schema, computational criteria such as sequentiality and synchronicity—sequentiality of moves and synchronicity of interactions—are no longer dominant. In fact, sequential-synchronic games or computations are special cases of true concurrent asynchronous games as the most general category of interaction games. Such a general category of games *G* can be represented by the so-called asynchronous copycat strategy.[261] *G*—as represented by a copycat—is nothing but the game or interaction itself independent of the order of the actions performed by agents, processes, or players involved in the interaction or playing of the game. In other words, in *G*, games between players can be played out-of-order or in partial order and the result would still remain the same. From an information theoretic standpoint, the notion of copycat stands for the conservation of the flow of information.

260 For a survey of interaction operations and relaxation of constraints in the context of semantics of interaction and computability see G. Japaridze, 'Introduction to Computability Logic', *Annals of Pure and Applied Logic* 123:1–3 (2003), 1–99.

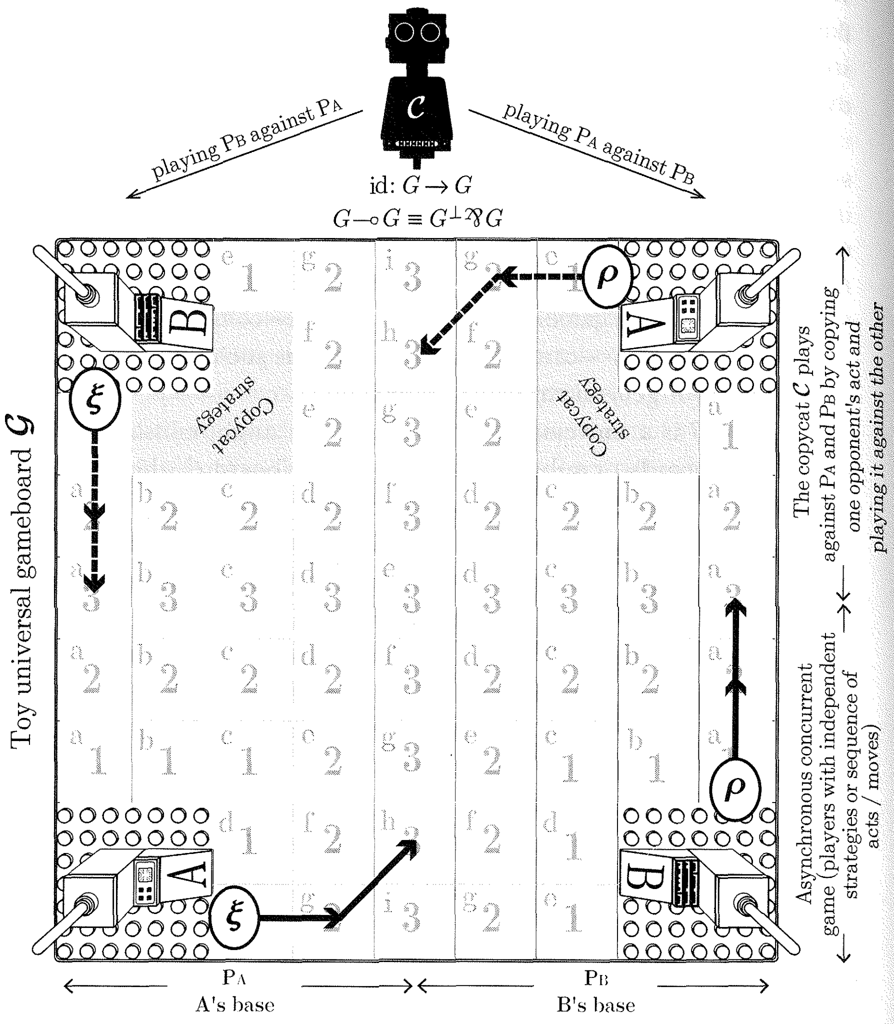
261 See S. Abramsky, 'Concurrent Interaction Games', in J. Davies, B. Roscoe et al. (eds.), *Millennial Perspectives in Computer Science* (Basingstoke: Palgrave Macmillan, 2000), 1–12.

Whereas information is conserved for the total system, there is information flow and information increase relative to interacting subsystems. Copycat strategies and their compositions enable the development of an explicitly dynamic theory of information processing.

Let us clarify, by means of intuitive diagrams, the copycat as the encapsulation of the general category of games, or interaction itself: G can be defined as the identity map or morphism of the game itself G: $G \rightarrow G$. In Church-Turing computability terms, this can be interpreted as the identity function that maps natural numbers to natural numbers N: $\mathbb{N} \Rightarrow \mathbb{N}$. Then N would be a general game on which many games—computable functions over natural numbers—can be played. Defined as such, G is represented by a copycat (or general strategy of interaction).

Imagine G is a universal game board which might consist of many other game boards or subgames. On the game board G, there are two teams of players, agents, or processes A and B, each with their respective base or domain of moves and distributed in an asynchronous fashion over the board. The actions or moves in each team can be synchronically and sequentially ordered. But with respect to the actions or moves of the other team, they are asynchronous, i.e., either out-of-order or in partial order. The cells of the game board are labelled in accordance with the players' base. As possible actions in the game, the moves (represented by arrows) performed in each team's base can be thought of as legal transits from one labelled cell to another. Such transits could be classically understood as functions or state transitions, where the previous cell traversed by a move is an input for the next cell in that move and, correspondingly, the next cell an output for the previous cell.

Now enters the copycat: an agent who beats both A and B by copying one team's plays and running them against the other team, or copying moves from one game board to another and vice versa. It copies A's move and plays it against B by changing its labelled cells from A's base to corresponding labelled cells in B's base, and vice versa for team B. That is, the copycat plays moves in the domain A in the codomain B, and plays moves in domain B in the codomain A. In this scenario, whichever team wins, the copycat wins. With the understanding that winnability in interaction games



Toy universal gameboard $\mathcal{G}$
$\mathcal{C}$
playing P_B against P_A
playing P_A against P_B
id: $G \to G$
$G \multimap G \equiv G^{\perp} \parr G$
Copycat strategy
Copycat strategy
A
B
A
B
ξ
ξ
ρ
ρ
P_A
A's base
P_B
B's base
Asynchronous concurrent game (players with independent strategies or sequence of acts / moves)
The copycat $\mathcal{C}$ plays against P_A and P_B by copying one opponent's act and playing it against the other

The hiding copycat

is tantamount to computability, winnability is not essential for the copycat since if both *A* and *B* lose—i.e., incomputability as a lost game—there would be no win for the copycat. The true significance of the copycat is that it represents the flow and increase of information under *any* particular interactive configuration (i.e., distributed in space and time) for a set of agents or processes: even if the teams involved do not synchronically react to one another, the copycat captures their moves as a general interaction. Moreover, from a logico-computational perspective, the copycat reveals the surprising power of copying information from one game board or place to another, through which we can arrive at an emergent view of logics and language as simple or composite interactions or copycat strategies.

As an agent that incarnates the universal category of the game or the interactive logic of computation, the copycat shows that sequential-synchronic computation is a subset of asynchronous-concurrent computation, just as the first-person game is a subset of the two-person game (I-thou game). In a nutshell, interaction in its most general form is the generalization of computation as such. There is no process that does not speak to another process, irrespective of how such processes react to one another. The logical equivalent of the copycat as the agent of interaction, game, or computation is a *dialogue*, and dialogue is the engine of semantics: there is no monologue or private thought without a dialogue, an interaction within and over language, and correspondingly there is no dialogue without an information gain or new knowledge made possible by an interaction between the system and its environment, an agent and its dual, a thesis and a counter-thesis.

FROM SEMANTICS OF INTERACTION TO INTERACTION AS THE GAME OF SEMANTICS

Refounding logic on generalized interaction—the deepest computational phenomenon—allows us to understand precisely what form of logico-computational interactions the automata must exhibit in order for them to cross the bridge between syntax and semantics. In other words, such an interactive view of logic—as formulated particularly in proof theory—provides the logico-computational framework through which the minimal generative syntax of the automata can be transformed into concrete syntax, where syntactic strings are comprised of structured units and relations similar to the various syntactic constituents with invariable roles that make up the structure of our natural language sentences. On another level, the interactive view of logic presents a logico-computational framework within which role-based syntactic constituents can be handled as logical expressions which subsequently, through dialogical interaction, find semantic values or meanings. In this sense, meanings can be characterized as invariants with regard to the interaction processes. In short, the interactive conception of logic can simultaneously specify the type of interactions necessary for the formation of concrete 'sentential' syntax, and the interactions necessary for meaning-conferring dialogues. The syntax-semantics interface is built upon interaction.

To provide even a rudimentary account of such frameworks for bridging syntax and semantics would entail an introduction to recent developments in logic, mathematics, and theoretical computer science—itself a gargantuan task that would surely distract us from the main themes of this book.[262] To this extent, at the risk of misrepresentation, only the most basic intuitive ideas behind the interactive logical framework of the syntax-semantics interface will be presented here.

Traditionally in proof theory, the meaning of a proposition has been understood as the proof of that proposition or the verification of it (provided that the process of verification is grounded on the notion of proof).[263] The meanings expressed by (linguistic) statements are then laid out not by means

262 For introductions to developments at the intersections of theoretical computer science, logic, mathematics and linguistics, see J. Ginzburg, *The Interactive Stance* (Oxford: Oxford University Press, 2012); A. Lecomte, *Meaning, Logic and Ludics* (London: Imperial College Press, 2011); and J. Trafford, *Meaning in Dialogue: An Interactive Approach to Logic and Reasoning* (Dordrecht: Springer, 2017).

263 For an interactionist interpretation of the program of verificationism, its relation to computation and proof see, A. Naibo et al., 'Verificationism and Classical Realizability', in C. Başkent (ed.), *Perspectives on Interrogative Models of Inquiry* (Dordrecht: Springer, 2016), 163–97.

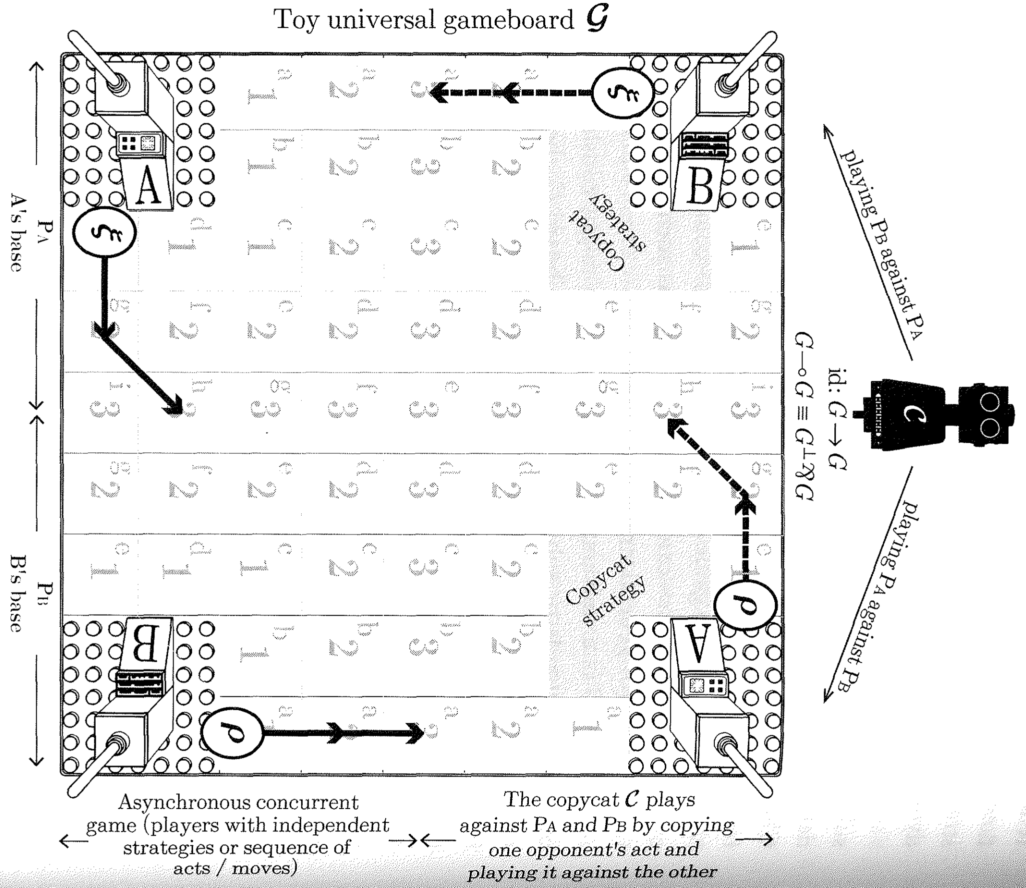
of truth-conditionals but by means of the construction of proofs that are *syntactical* in nature. In classical proof theory, the meaning of the sentence is the set of its proofs, or, more accurately, knowledge of the conditions of its assertion, which then counts as knowledge of what would count as a proof, and hence the meaning of the sentence. Thus, in determining the meaning of a sentence priority is given not to the notion of truth but to that of proof. Or, from the viewpoint of the pragmatic theory of meaning-as-use, the meaning of a sentence or formula is not explained by their truth, but by their use or consequences in a proof:

$$\frac{\vdash A \quad A \vdash \boldsymbol{B}}{\vdash \boldsymbol{B}}\,\text{cut} \rightsquigarrow \frac{\vdash A \quad A \vdash}{\vdash}\,\text{cut}$$

(Notice that A on the left and the right side of the consequence relation or turnstile can be seen as a duality: proofs of A versus proofs of $A^{\perp}$, where the linear negation $\perp$ signifies the switching of roles or, in this case, swapping the *place* of A with regard to the turnstile.)

The shortcoming of the classical meaning-as-proof paradigm is that the proof is conceived statically and monologically, in tandem with the mathematical interpretation of classical logic. For example, in this classical setup, given A (a formula, a logical expression, a piece of syntax, a proposition), if one has the proof—the meaning of A—then one also has the meaning of $\neg A$. That is to say, having the proof implies having the disproof by way of the classical negation that negates some unspecified or arbitrary iteration of A. But with the introduction into classical proof theory of interaction (i.e., dualities as the interchange of roles rather than as classical negation), the proof or determination of meaning takes on a different form. The meaning of A can only be determined through interaction or dialogue with its counter-proof or refuter, and vice versa. The determination of meaning or proof can only be achieved by stepping outside of the static-monological framework of proof into one where proof is the interaction between a prover (player) and a refuter (opponent), A and $\neg A$, or, more precisely, A and the model of $\neg A$.

Just like the environment in the classical Church-Turing paradigm of computation, where it was presupposed but never explicitly asserted, in


Toy universal gameboard $\mathcal{G}$
playing P_B against P_A
playing P_A against P_B
$\mathcal{C}$
id: $G \to G$
$G \multimap G \equiv G^{\perp} \parr G$
Copycat strategy
Copycat strategy
A
B
B
A
ξ
ξ
ρ
ρ
P_A
A's base
P_B
B's base
Asynchronous concurrent game (players with independent strategies or sequence of acts / moves)
The copycat $\mathcal{C}$ plays against P_A and P_B by copying one opponent's act and *playing it against the other*

The hiding copycat

is tantamount to computability, winnability is not essential for the copycat since if both A and B lose—i.e., incomputability as a lost game—there would be no win for the copycat. The true significance of the copycat is that it represents the flow and increase of information under *any* particular interactive configuration (i.e., distributed in space and time) for a set of agents or processes: even if the teams involved do not synchronically react to one another, the copycat captures their moves as a general interaction. Moreover, from a logico-computational perspective, the copycat reveals the surprising power of copying information from one game board or place to another, through which we can arrive at an emergent view of logics and language as simple or composite interactions or copycat strategies.

As an agent that incarnates the universal category of the game or the interactive logic of computation, the copycat shows that sequential-synchronic computation is a subset of asynchronous-concurrent computation, just as the first-person game is a subset of the two-person game (I-thou game). In a nutshell, interaction in its most general form is the generalization of computation as such. There is no process that does not speak to another process, irrespective of how such processes react to one another. The logical equivalent of the copycat as the agent of interaction, game, or computation is a *dialogue*, and dialogue is the engine of semantics: there is no monologue or private thought without a dialogue, an interaction within and over language, and correspondingly there is no dialogue without an information gain or new knowledge made possible by an interaction between the system and its environment, an agent and its dual, a thesis and a counter-thesis.

FROM SEMANTICS OF INTERACTION TO INTERACTION AS THE GAME OF SEMANTICS

Refounding logic on generalized interaction—the deepest computational phenomenon—allows us to understand precisely what form of logico-computational interactions the automata must exhibit in order for them to cross the bridge between syntax and semantics. In other words, such an interactive view of logic—as formulated particularly in proof theory—provides the logico-computational framework through which the minimal generative syntax of the automata can be transformed into concrete syntax, where syntactic strings are comprised of structured units and relations similar to the various syntactic constituents with invariable roles that make up the structure of our natural language sentences. On another level, the interactive view of logic presents a logico-computational framework within which role-based syntactic constituents can be handled as logical expressions which subsequently, through dialogical interaction, find semantic values or meanings. In this sense, meanings can be characterized as invariants with regard to the interaction processes. In short, the interactive conception of logic can simultaneously specify the type of interactions necessary for the formation of concrete 'sentential' syntax, and the interactions necessary for meaning-conferring dialogues. The syntax-semantics interface is built upon interaction.

To provide even a rudimentary account of such frameworks for bridging syntax and semantics would entail an introduction to recent developments in logic, mathematics, and theoretical computer science—itself a gargantuan task that would surely distract us from the main themes of this book.[262] To this extent, at the risk of misrepresentation, only the most basic intuitive ideas behind the interactive logical framework of the syntax-semantics interface will be presented here.

Traditionally in proof theory, the meaning of a proposition has been understood as the proof of that proposition or the verification of it (provided that the process of verification is grounded on the notion of proof).[263] The meanings expressed by (linguistic) statements are then laid out not by means

262 For introductions to developments at the intersections of theoretical computer science, logic, mathematics and linguistics, see J. Ginzburg, *The Interactive Stance* (Oxford: Oxford University Press, 2012); A. Lecomte, *Meaning, Logic and Ludics* (London: Imperial College Press, 2011); and J. Trafford, *Meaning in Dialogue: An Interactive Approach to Logic and Reasoning* (Dordrecht: Springer, 2017).

263 For an interactionist interpretation of the program of verificationism, its relation to computation and proof see, A. Naibo et al., 'Verificationism and Classical Realizability', in C. Başkent (ed.), *Perspectives on Interrogative Models of Inquiry* (Dordrecht: Springer, 2016), 163–97.