Wolfgang Sohst / Berlin, Germany

Genuine Process Logic

Abstract:

The Genuine Process Logic described here (abbreviation: GPL) places the object-bound process itself at the center of formalism. It should be suitable for everyday use, i.e. it is not primarily intended for the formalization of computer programs, but instead, as a counter-conception to the classical state logics. The new and central operator of the GPL is an action symbol replacing the classical state symbols, e.g. of equivalence or identity. The complete renunciation of object-language state expressions also results in a completely new metalinguistic framework, both regarding the axioms and the expressive possibilities of this system. A mixture with state logical terms is readily possible.

Contents

* * *

1. Introduction

1.1 The starting situation

Among the many logical systems, the best known of which are propositional logic, syllogistics, predicate and quantifier logic, and modal logic, there are also a few for the logical mapping of process sequences.[1] The approaches developed in this area include, for example, the so called 'Propositional Dynamic Logic' (PDL), 'Second Order Process Logic' (SOAPL), 'Algorithmic Logic' or simply 'Process Logic'. In addition, a formalized process theory was developed somewhat later[2], which does not, however, understand itself as strictly logical formalism. All these approaches have in common the fact that they build on the basic model of classical propositional logic or even a form of modal logic and understand themselves as their extension.[3] In this sense, such statement systems have been designed as sequences of state symbols[4] linked by so-called 'paths', i.e. connectives between the designated states. The process theory of Gerhard Wunsch also builds on this scheme, but derives from the concept of physical state space rather than that of a logical system. Further, the purpose of all these theories or logical systems is practical in a more specific sense. The said process logics are about the modeling of the behavior of computers, not about everyday human thinking. Therein, the process logics developed so far differ significantly not only from the traditional syllogistic and the quasi-mathematical systems of modern propositional and predicate logic, as originally developed by Peano, Frege, Russell / Whitehead and various later authors. By modeling logical statements, these founders of modern logic basically wanted to clarify the cognitive orientation of man in everyday life, especially by attempting to eliminate ambiguities in natural languages.

1.2 The specific application field of the GPL

The approach described below with the abbreviated designation 'GPL' tries to restore this connection of formal process logic to the thinking of people in everyday life by means of explicit formalization of the procedure itself. The chosen designation as 'genuine process logic' merely serves to distance it from the

---

[1] See the summarizing account of Harel et al. [1982] and Knijnenburg et al. [1991] with numerous further references especially to the predecessors of the theoretical approach developed there. An up-to-date summary of the state of research in this field is offered by Troquard / Balbiani [2015].

[2] See Wunsch [2000] as a standard reference in this field.

[3] Knijnenburg explicitly says: „The meaning of the propositional connectives is exactly like in ordinary, classical propositional logic […]" (Knijnenburg et al. [1991], S. 183). Harel et al. in turn understand their approach as an extension of classical modal logic, see Harel et al., p. 144 (abstract and beginning of the essay).

[4] The text follows in the naming of symbol categories the usual convention, extended by the newly introduced category of the action operators, the availability, merger and splitting symbols (more details in the following text):

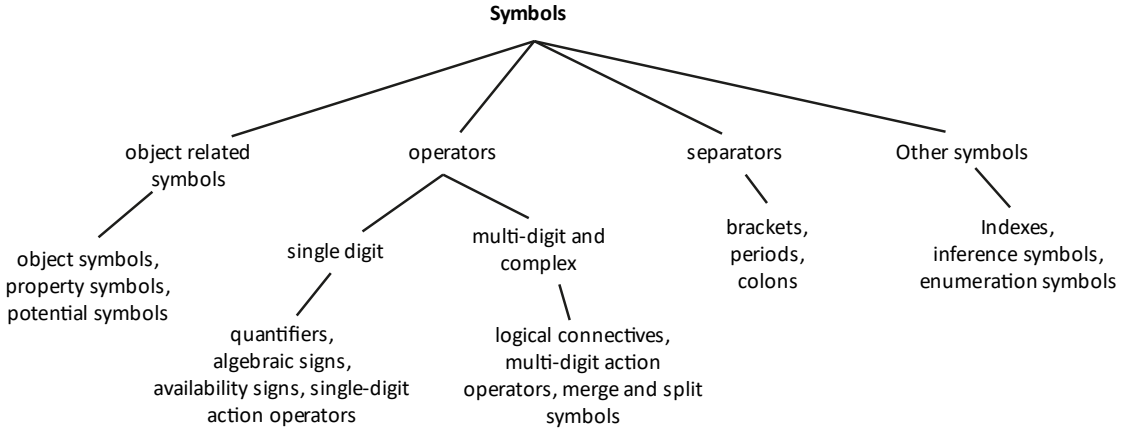

Fig. 1: A taxonomy of logical signs

above-mentioned, more technically oriented process logics of the 1980s.[5] Thereby the GPL wants to recover what was lost by their almost exclusively technical orientation, i.e. by their effort to optimize computer programs.[6] The motivation for such a project results from the conviction that the individual in everyday life primarily thinks and acts in a procedural manner and must do so to cope with the often highly dynamic reality of life. [7] This should apply culturally invariant. Even animals which at least for the most part, if not generally, cannot formulate concepts of their reality are obviously orientated in their habitats with astonishing precision and probability of success. In the flow of everyday life animals as well as humans think only secondarily in states, primarily in processes.[8] In doing so, we first analytically reflect on the current events, trying to apprehend their elementary process units. From the relevance of these elementary processes within the overall event, we infer to follow-up processes and events.

Of course, defining states also plays an important role in our cognition, but only a secondary, culturally mediated one. These states become socially important, especially in the normative sphere and communication of promises, obligations, contracts, laws, etc. Statements of state serve, in the context of social order, above all the fulfillment and control of behavioral expectations.[9] The individual being, animals as well as humans, usually needs explicit state determinations only as a complementary means. They may serve as clues for orientation, for example when one keeps in mind where one puts an object to find it later. However, the flowing everyday life takes place before and with us primarily in a procedural way. The question is, therefore, how such a deal with reality can be formally modeled.

The GPL differs insofar significantly from both the initially named process logics as the basis of computer algorithms (in the following: program logics) as well as from the well-known classical state logics, as it is intended to represent reality directly as a chain of events or processes. Afterwards, these can be connected again to each other by object states. In order to distinguish the GPL typologically from the classical state logics and the former process logics, which are also ultimately state logic, it must be understood as a pure *logic of action* (synonymously for 'action': of *processes* or *events*).

Nevertheless, the GPL is state-logically connectable at every point of a sequence of expressions formed with it: an inference of the GPL can readily serve as a starting point for further state-logical transformations, and conversely any state-logical result can be used without difficulty as a starting point for further developments in the GPL (see below section 8). This does not mean, however, that the GPL is basically just an extension of the known state logics with some additional operators. As will be shown

[5] The referred process logics of the 1980s do not describe processes in themselves as changes, but as sequences of machine states. In their depiction, the computer "jerks" from state to state according to its program until a goal is reached. In this context, it should not be a coincidence that the peak of interest in algorithmic process logic was reached approximately between 1975 and 1990. During this period, the founding of today's so-called 'digital revolution' took place, ie. the technical and thus necessarily formal development of strictly state-logically operating computer programs.

[6] All the programming languages used today are based, as far as their logical foundations are concerned, exclusively on state logic functions and structures. This is particularly evident in the control structures of the OPL discussed in Section 7 compared to those of the usual programming languages.

[7] See Cauley [1986] and Stadler [1989]. Although Kathleen Cauley, from her state of knowledge at that time, argues that logical knowledge differs from procedural and conceptual knowledge. She says: „Procedural knowledge refers to the task specific rules, skills, actions and sequences of action employed to reach goals. It shares no features with logical knowledge except occasional qualitative change." (ibid., p. 4). However, this difference should be abolished precisely by the present draft by bringing purely procedural knowledge into a formal logical form.

[8] I claim this without being able to refer to a study devoted to this question. Surprisingly, I could find none that explicitly addresses this issue.

[9] Humans are the only living entities able to distinguish between the normatively expected and the actual run of events by means of symbolic representation. However, they need state determinations to be able to match such snapshots of reality with their normative conceptions. Animals do this, if at all, not based on the comparison of concrete mental images of what they wish to realize with the real tenor, but based on preconscious and thus by themselves uncontrollable cognitive functions. – In the legal sphere, the focus is on states, especially in the so called subsumption technique, which every student of continental European law eagerly learns.

below, the GPL is fundamentally neither of situational states nor of object properties. It deals exclusively with effects. Consequently, a connection from the GPL to any state logic and vice versa always requires a 'translation' of states or object properties into action parameters. Examples are given later in Section 8: *Coupling of process-logical and state-logical expressions*.

The cognitive and consequently also formally great importance of state-logical systems should by no means be diminished. The process-logical formalism developed here does not see itself as a competitor but as an addition to the well-known state-logical systems.[10] Both are abstractions of reality, each with its own kind of validity claim. Thus, results of state-logical transformations and inferences can serve as a starting point for process-logical continuations of the modeling of reality and vice versa.

1.3 Differences of the GPL to the classical state logics

The most obvious difference between the GPL and all other so-called state logics, apart from the different character set, is that the GPL does not operate on either two-valued or multivalued truth values. State statements and truth values are inseparable. Truth values unfold their effect in a logical formalism only in the context of state determinations. This does not apply to process statements, although one could also qualify process statements in this regard. However, this would compromise the essence of process-logical structures. The logical content of procedural statements is not in their evaluation, e.g. as true or false, but in the specific procedural consistent relationship of these statements to each other. Thus, we are dealing here with a slightly different meaning of the term 'logical consistency'.[11]

The corresponding reservation also applies to the known modal and temporal logics.[12] Certainly, process or event expressions can only be understood as something that takes place in time. However, this does not mean that time-indexed statements are already genuine event statements, nor that a strict process logic in the sense of the GPL must provide their statements about times with truth values.

In addition, the elementary, classical state logics, following their premises, regularly lead to an equally state-bound inferential conclusion. Consequently, state logical sequences of expression - after the initial establishment of the state-bound premises - consist of their combination and transformation to infer a logically final state. In the GPL, by contrast, this looks quite different, in this respect like the program logics mentioned above. Process logics in general, including the GPL, do not rely on any static inference, but use their linguistic means for the general and purely formal modeling of dynamic structures. As a result, any valid expression following a preceding expression may theoretically also be considered as a conclusion. In the stronger sense of the word, a process-logical conclusion would be only one envisaged from the outset as the aim of a process chain and is then proven valid by its preceding and equally validated expression sequence.

---

[10] These include both the classical propositional, predicate, quantifier and modal logics, as well as the process logics such as the PDL designed to better control computers. The so-called predicate logic does not deserve its name, because grammatically the predicate is an activity word in all major languages of this world. In traditional logics, on the other hand, the (logical) predicate always appears only as an object property, bound to it by the grammatical copula. Classical predicate logic would therefore have to be better called 'status logic', 'coincidence logic' or something similar, while the term 'predicate logic' in the true sense of the word would belong to the OPL.

[11] It is nonetheless plausible to call this form of being free from symbolic contradictions a form of logical consistency. Logical consistency is nothing more than the formal compatibility of several statements with each other. However, this only applies to the object language level. On the other hand, on a metalinguistic level, process logic statements would be exclusively two-valued in terms of their consistency, i.e. to be qualified either as consistent or not. *Tertium non datur*.

[12] Temporal logics are known to belong to the group of modal logics. Although it is possible in the temporal logics by introducing modal operators to change the truth value of a statement over different points in time and thus, for example, to modify the rigid statement "It is raining" to "It has rained" and "It will rain" etc. However, this does not change the fact that such colloquial verb phrases are logically treated here only as states. („It is the case that <*status statement*>).

Regarding the extensions of the predicate logic by quantifiers the same applies to the GPL as already mentioned above for the truth values: A quantification of process-logical statements does not make sense. However, where this is required, the formalism below provides an interface to all known state logics (see Section 8 below) so that, if necessary, it is also possible to work on a sequence of statements using these means.

Further innovations of the GPL compared to the designated state logical systems (including the so-called program logics) relate to the necessity of being able to formally create objects and let them vanish as well as to merge or split existing objects. State logic thinking is exclusively combinatorial thinking of objects normally given already at the beginning of an expression sequence. Admittedly it is possible to deduce new elementary objects from those given in the initial expression sequences. Thus, the propositional implication allows the logically valid conclusion of p on $p \rightarrow p \vee q$. However, the new object q appears here out of nowhere; it is not generated by any of the previously available objects. Furthermore there is definitely no state-logical expression and, in particular, no form of inference that leads to the merging of two previously separate objects in a single result object. In its conclusion, the expression $p \wedge q = r$ is logically not acceptable and thus invalid. Although it is possible to introduce an identity relationship in terms of predicate logic, it does not have a synthesizing effect[13], but merely produces a kind of state symmetry between two objects or object groups. The same applies vice versa for the splitting of an object.

The merging of several formal objects into new objects or, conversely, the splitting of symbolic units into a plurality of subordinate (individual) units is a basic cognitive function of human thought and should therefore be reproducible in a dynamic logical formalism.[14] In some way this is possible in set theory, but not in the state logical systems. Yet set theory, all in all, is a state logical formalism too. In contrast, the GPL allows generically the logical formalization of merging and splitting.[15]

1.4 An anthropological explanation of the present approach

If the above assertion is correct that the general cognition of living beings is primarily process- and only secondarily state-logically organized, then this argument becomes even stronger in relation to humans in that they, while growing up, become state logically intensely trained through linguistically coded norms. First and foremost, these abilities enable normative coded social orders. Legal norms are based on the possibility of the objective determination of social conditions, because otherwise it was neither possible e.g. to determine taxes nor whether a certain behavior falls under a penalty provision or not. This may well be a not entirely implausible explanation hypothesis that so far only formalisms have been developed in logic, which are state logically organized.

Social order and logic are closely connected, especially via the respective legal and economic order of a society. Social order is based on objective, i.e. generally applicable rules to deal with each other. However, rules of social interaction can only be applied to hypothetical or established facts. For this, Aristotle found the first formalization already in Greek antiquity.[16] That his logical considerations from the beginning favor a state logical form may have something to do with his skeptical relation to the older Greek natural philosophy, in particular, Heraclitus' intuition of the world as pure process and constant change. This was suspicious to him. Both in the fourth book of his *Physics* and in the fourth book of his

[13] See e.g. Kutschera/Breitkopf [1971], p. 129.
[14] See the recent contribution by Robert C. Berwick and Noam Chomsky in Berwick / Chomsky [2016]. They characterize this central ability of human cognition as the MERGE function.
[15] See Section 6: *Object Merging and Splitting* below.
[16] See the Aristotelian *Organon* (Aristotle [1995], Vol. 1), there the first analysis (doctrine of the conclusion) and the second analysis (doctrine of the proof).

Metaphysics[17] he deals with change by way of physical or developmental movement. Finally, he says about nature that it is the *principle* of movement, and consequently not the movement itself. That is, the ground state of the world is the realm of rules; it precedes every change by inner worldly movement.[18] Thereby, Aristotle not least set the course also for logic to this day. Only modernity recognizes that thinking in state sequences is a distinct and necessary feature of our *normative* orientation in social relations. In the language of Michel Foucault, all forms of state logic are a 'dispositive of power'[19], or less polemically: institutions to stabilize social order.

This is particularly evident in the intimate connection of all social order with a very specific concept of truth. Within the framework of modern social order, a statement can only be qualified as true or false by way of its review by at least one other authorized person. By contrast, an intuitive concept of truth in the sense of unquestionable givenness only for one and the same person is useless for this purpose. However, such an intersubjective check is not possible without reference to a 'snapshot' of the respective flowing situation: The flow of events must be frozen in its crucial aspects to be able to rate it socially. And because this evaluation is done primarily with linguistic means, it is the first task of those involved in such procedures to determine the truth of the statement in question, because false claims are disputed discursively from the outset in terms of their content. Hence our not only logical obsession with truth.

By contrast, process-logical thinking in the sense pursued here is closer to individual, primary world perception. It is less apodictic in its claim to validity and therefore probably less suitable as a 'dispositive of power' to use, too. If it serves more to understand the primary capabilities to orient oneself, the better.

That said, we come to the formal part of the GPL.

2. The notation of the pure process; the positive and the negative action operator

Despite all the differences mentioned above, the GPL makes use of other systems, here referred to as state logical, i.e. of certain conventions in the notation of their terms, which superficially resemble those of classical predicate logic. To be sure, they mean something fundamentally different here. Thus, 'A(p)' expresses a state in classical predicate logic, namely, that the object p has the property A. On the other hand, the term here is intended to mean that p can trigger or cause the event A. A has no defined property in the object language of the GPL, but can be the trigger of an elementary process.[20] This may sound harmless at first, especially as one can also map processes in classical propositional logic - but only as an overall statement, e.g. in the statement "It is raining". The predicate logic does not permit this anymore, because it must be understood, following its inventors, as a sentence-analytic construct. The classical predicate logic knows only state-descriptive sentences, for example: 'For the object a, if a is an object of type x and all x have the property F, then a also has this property', in short: $((x \in X) \wedge \forall x : \mathrm{F}(x) \wedge (a \in X)) \rightarrow \forall \mathrm{a} : \mathrm{F}(a)$. Now, the classic logical systems deal with statements such as 'Robert puts on

[17] Ibid., 1005bff. In Section 4: *Problems around the Law of Contradiction*, it is already quite clear how truth values of propositions necessarily combine with a state logical reasoning.
[18] Aristotle [1991], Vol. 1, 1015a12.
[19] To my knowledge, Foucault did not contribute explicitly to formal logic. However, he occasionally dealt quite well with the role of logic in society; see Schneider [2001], p. 299ff., available on the internet at http://ul.qucosa.de/fileadmin/data/qucosa/documents/15195/A45-Foucault-Wahrheitsproduktion.OCR.pdf.
[20] The expression 'p *can* be the trigger of A' indicates a modal root of the OPL. However, this does not mean that it borrows from modal *logic* formalisms. Rather, the modal aspect of OPL is directly metaphysical, i.e. here a formal reflex expresses itself with reference to a corresponding model of the world, imagined as a totality of the given out of reality and possibility. This is explicitly directed against the early Wittgenstein, whose sentence 1 of his *Tractatus logico-philosophicus* reads: "The world is everything that is the case" (Wittgenstein [1993], Vol. 1, p. 11). Apparently, the world is more than that, namely everything that *may be* the case, too. This is mirrored by the formal design of the OPL.

his jacket', 'The fans storm the stage' or the famous 'Romeo loves Julia' at best by taking them as a relation. Usually however they take it as a statement of an integral state in the grammatical form '*x* is *y*', whereby the actual event is being reduced to a state by an auxiliary verb ('to be' or 'to have') as the copula. However, that is missing the point. The actual statement tells us something about the particular event itself. Therefore, to go beyond this limitation, we need another form of predication and different operators.

2.1 The action object: latent potency and actual effect

The idea is to depict a possible or real event in a very general and elementary way, and to formally differentiate both from each other. Processes, as soon as they occur, always take place within one and the same or between several objects and thus present themselves (and only thereby!) as a single process between individual objects or in themselves.[21] For this, we have to first make a formal provision to be able to differentiate between the respective object that operates and its possible effect. Note: this is a distinction between the *possibility* of action and the objects from which they originate, or which are their goal. Why this distinction? Well, objects (in the widest sense of the word, including the living beings and especially humans) do not only act at present. Additionally, they are always situated in a possibility horizon that depends on their situation in the interplay with their environment. A bottle standing on a table can break if someone pushes it down, but it does not, unless the real conditions are met. This is especially relevant when dealing with people. People may reasonably assert that they could do this or that, provided the circumstances are met, or we attribute this capability to them, e.g. to justify their legal responsibility. A procedural formalism must therefore consider the important fact that objects can be attributed to *potential* effects in a very general sense, without this effect having to occur immediately. We refer to this potential form of effect as *latent*, whereas the actual effect occurs as *actual*.[22]

To formally fulfill this requirement, instead of the traditional state identifier (i.e. the state logic predicate), we need three distinct signs:

1. Object symbols

---

[21] This is the only extralogical, i.e. metaphysical axiom that is claimed by the OPL. For justification, see Sohst [2016], p. 123ff.

[22] Naturally, the OPL also notes process flows line by line using discrete expressions. However, an action expression does not stand for a fixed state, but rather for a specific, holistic *space of possibility*, which changes with each further expression. This space of possibility is configured through objective potentials of action, which, for living beings, we might better call capacities. The present approach, however, assumes that a space of possibility designated by us is only an abstraction within a fundamentally dynamic structure. Only in the language of logical abstraction does the term 'space of possibility' designate discrete, single states among others. In fact, i.e. from a prelogical or ontological point of view, a space of possibility is necessarily stretched over time. It is semantically meaningful only over its whole temporal extension. Therefore, even the conceptual singular of 'possibility space' is easily misunderstood. For the meaning behind this word to become comprehensible to us, we cognitively pick out individual moments from the flow of its own constant change. In their pre-linguistic existence, probably all living beings do this. In this broader sense, the GPL is also a kind of second-order state logic, namely a logic of the sequence of configurations of the possible as a result of realized dynamics. However, the action latencies given only for the logical representation and their updates in the form of discrete actions and their consequent changed states of possibility must not be confused with the static attributions of states to objects. The differences that arise from this changed perspective are the subject of this text.

This possibility space, which is modified in each case by an action expression, is defined by the objects given at the respective point of the overall process with their possible connecting actions. It behaves as in our everyday lives: When we pursue a particular goal, we often come up against alternatives to the achievement of our goals and therefore have to decide how to go on. We can consider such branching as process nodes. Each such process node, if one traces it with all its execution options in the OPL, would correspond to a single line of expression with certain action objects. At each point that requires a decision from us, we reflect newly the possibilities that arise from the various path alternatives, and not the current state of the objects around us. Each process node consequently confronts us with a changed possibility space because of the already implemented actions. The OPL is a formalism for describing the dynamic development of possibility spaces.

2. Latency symbols (for the potential effect of an object)
3. Action symbols (for the actual effect)

A complete action expression must therefore always be composed of signs of these three basic types. With reference to the state logical notation, we first use the square bracket instead of the round to attribute an effect to an object. After all, the expression for a state and a potential for action have in common that they are necessarily assigned to the core of an object whose predicate they are. The difference in parentheses, however, allows the later mixture of both systems of statement without the risk of confusion, which expression belongs to which system. The structural similarity of the spelling thus facilitates a mixture of both systems. For the assignment of a possible action to an object, i.e. for the formalization of pure latency, we first only write

F[*a*].

The Latin capital letter in front of the parenthesis stands for the latent effect, the italic lowercase letter in parenthesis for the affected object. We call such an expression an *action object*. The object expression in the bracket can also be multi-digit:

F[*a, b, c*].

This means that a single potency is assigned to an entire group of objects. We call this an *action item group*. This natural possibility was obviously overlooked in classical predicate logic. In any case it is not anywhere formally provided, although it is applicable to state logic systems too.

However, expressions for action objects or action object groups do not yet describe an actual process. They only assign a latent effect to an object. A complete action expression only comes about when expanded by the action operator described in 2.2 below. There, too, the question of how to solve elementary processes, e.g. 'Peter is writing', 'The cat runs', 'It may rain', etc., which are single-digit, while many other processes are two- or even many-digit.

The negation of such a latent potential is expressed by the usual negation sign '¬', which we call the 'negation of potentiality'. From the position of its emergence in a sequence of expressions, it negates the further possibility of the denoted effect, i.e. determines its impossibility. The formal meaning of such a negation is that the correspondingly denoted object cannot, as a source object, stand in any expression of effective action with its previously negated power. However, this negation can be reversed by another expression. This is also implicitly possible by positing the object in question on the right side of an action operator with its corresponding potential, now positively denoted. Note: The negation of possibility thus indicated makes use of a different concept of modality than is used in modal logic. In modal logic, the term 'possibility' stands on a scale between the extremes of impossibility and necessity. Here, on the other hand, the impossibility of an effect removes the denoted latent potency of action from the referred object. This form of negation thus refers to the relation of possibility and reality, with which modal logic has explicitly nothing to do.

Action object groups must have a functionally clear relationship to simple action objects to avoid confusion. We therefore define their relationship to each other and their internal structure as follows:

DEF 1: *Action object and action object group*

1. The action object is represented by expressions of the form F [*a*], where 'F' stands for any object-specific potency and '[*a*]' stands for any object. By prefixing the expression for the potency, the assignment of this potency to the content of the following parenthesis is displayed.
2. F (the sign of an impact potency) and *a* (the object sign) are object and action constants. A set-theoretic generalization in the sense that they are elements of specific sets of effects

(synonymously: impacts) or objects, that is, a second-order effect and object designation, is not part of the elementary GPL, which is elementary in this respect.

3. The square brackets around an object identifier express the area of validity of the effect assigned to one or multiple objects. Thus, a pair of square brackets may contain any number of object constants separated by commas: F[*a*, *b*, *c*, … *n*]. Such a construct is called an *action object group*.
4. Effect constants are defined within an expression sequence for a specific area of validity. Any use deviating from this definition within a sequence of expressions is invalid.
5. As an exception from the above no. 4, the scope of an action object group only changes in the cases that a) an object of the action object group disappears in the further course of an expression sequence or another object is explicitly included in the group. In the case of the disappearance of an object that previously belonged to an action object group, the referred action object group is implicitly reduced by this object. In the case of a new inclusion of an object in an action object group, this must be explicitly indicated in a formally correct manner. The formal notation for this is: <old action object group> <action operator[23]> <new action object group>.

In any case, every expression of the GPL that stands for an action object (group) is, from a colloquial perspective, a subject and its predicate, whereby the predicate corresponds indeed to the verb of the statement (as in everyday language) and not, as in the state logic, only to the property of a substance. However, the expression F [a] in the GPL does not mean that the object kernel *a* actually has some object property in the form of a static action potential, but only that this object kernel *a* may have the effect F within the given logical scope. Action objects and action object groups are thus modal expressions in the sense of: 'It is possible that F can unfold the effect [*a*, b, *c*, ... *n*].'

2.2 The action operator

As mentioned above, we also need a new operator, which denotes the actualization, i.e. the actual occurrence of an otherwise only latent effect. Since such an impact operator has two digits, we refer to the action object or the action object group on the left as the source object, and the action object or action object group on the right as the target object. The action operator thus changes the latent structure of action between the involved objects. I use the character, '⇒' (English: 'CAUSES') for it. The action operator in its standard form will be placed between two objects with their assigned latent potentials.[24]

A negation of the action operator is usually dispensable, because the non-occurrence of a process usually needs no mention. For the sake of clarity, however, it may be useful in certain cases to explicitly indicate the non-occurrence of a process. For this purpose, we use the character ⇏. Categorically, it means 'does not cause' and is therefore a shorthand notation for '¬ ⇒ '. The inhibition of the effect, formally: the negative action operator, has no hidden modal semantics. The simple negation of the actual effect is modally indifferent.

Finally, a variant of the action operator is missing, which allows to reproduce object-internal processes, as expressed in sentences such as: 'I run', 'It rains', etc. Formally, such processes are single-digit. Nevertheless, the formal type is an action operation. We introduce a positive and negative variant of the standard action operator whose symbol is simply the inverse of the parent form: ⇐ for the positive single-digit and ⇍ for the negative single-digit action operator.

[23] For the introduction of the action operator, see the following section 2.2.

[24] It is true that there can be any number of action objects on both the right and the left side of an action operator. However, these must be linked by the classical ∧, ∨ or ¬ connectives, otherwise the corresponding expression is not well-formed. The classical implication is also not allowed here because it is not a simple connector, but already a fully truth-value qualified sentencial connector. However, since the OPL does not work with truth values, the implication is basically not applicable.

So much to the introduction of the elementary action operator, the formal reflexive action and its negation.

For the characters '⇒' and '⇐ ' as well as for their negative correspondences, '⇏' and '⇍', applies:

DEF 2: *The action operator*

1. The positive double-digit ⇒ operator[25] as well as its correlative single-digit ⇐ operator and their negations ⇏ and ⇍ are temporally indifferent, i.e. they do not refer to a particular moment of effect in time, nor to their duration. They share this metalinguistic property with the terms of propositional logic and the predicate variables of predicate logic.
2. The places of an action operator to be filled can be composed of a plurality of source or target objects. In this case, these form a common efficient cause and are to be connected by the ∧, ∨ or ¬ connectives.[26]
3. The effect operator, of whatever kind, does not say whether the acting object has exhausted its potency, nor which subsequent processes it sets in motion in the target object area.
4. The action operator is modally qualified.[27] By its use, it only says that the designated effect occurs. Without its use, only a latent effect is given.
5. Metalinguistically, the action operator is structured in a binary way: either an effect occurs, or it does not occur. Again, *tertium non datur*. However, this does not relate to any object-language truth value of the expression concerned.
6. The action operator updates logical possibilities: Updating a latent effect creates a new latent effect. The process denoted by the action operator is therefore that of a change in the scope of possible action. This does not require any denotation of states.

The two places to the left and right of the operator, as we have said, are called the 'source object' and the 'target object' of an effect. This also applies to action object groups. Both sides of the operator are formally represented in the same way.

Now it is another and very fundamental feature of process thinking, in contrast to the state-fixed understanding of the world, that individual processes per se are always snippets from a virtually infinite process universe. This requires us formally to use source and target objects in the GPL only with procedural connector characters, to indicate that our expressions are always just snippets of a procedural *holon*, i.e. a vast process whole. For example, if a final statement from the GPL is (reading direction always from left to right):

$$\ldots$$
$$F[a] \Rightarrow G[b],$$

in principle, the chain of effects described here is not complete. This should be illustrated by three leading and subsequent points, i.e.

---

[25] If, in the following, the term 'action operator' is used without any further specification, this always means both its positive and negative form.
With regard to the Aristotelian theory of causes, which is known to name the four causes: 1. efficient cause (*causa efficiens*), 2. material cause (*causa materialis*), 3. formal cause (*causa formalis*) und 4. final cause (*causa finalis*), initially, only the efficient cause is formally treated. But below, I will also go into the formal treatment of the final cause.

[27] It is however not qualified in terms of modal logic. The difference lies in what it means that something is not possible. In modal logic, possibility lies in the middle between the extremes of necessity and impossibility; on top of that, in modal logic the necessary (in this case quite contrary to the common sense) is also qualified as possible. Within the OPL, on the other hand, possibility is the mode that opposes the other mode of the real. The real is no longer possible here because it has already been realized. There is a common generic concept of possibility and reality: This is the given. The classification of reality and possibility as species of the genre of the given opens up in the first place the formal connection of a transition of possibility to reality.

$$\ldots F[a] \Rightarrow G[b] \ldots .$$

Single-digit effects are expressed as follows:

$$\ldots F[a] \Leftarrow \ldots \text{ , i.e. the right side of the operator remains empty.}$$

The inclusion of another (here: previously created[28]) object in an already existing action object group is indicated by:

$$\ldots F[a] \Rightarrow F[a, b] \ldots .$$

However, for the sake of avoiding unnecessary symbols, the leading or following three points may be omitted. However, this does not mean anywhere in the GPL that an object remains in a determined state after a completed effect. A standalone expression of the form F[*a*] only indicates a latent and, together with an action operator, a current process.

Now it will usually make sense to indicate upon which object(s) an effect subsequently may operate. For example, we use the expression F[*a*] ⇒ G[*b*], meaning: Bank *a* finances company *b*, which builds a gas turbine G with the money. It will often be necessary in such semantic contexts to indicate to whom this effect G, i.e. the construction of the gas turbine, refers. In our example, e.g. one would ask for whom *b* is the gas turbine being built. If we wanted to express this by means of another action operator, which points to the following affected object, then we would come into an infinite chain of follow-up references, because each affected target instantly triggers new effects.

To avoid such chain reactions - which does not preclude reference to them in a later step - we simply put behind the target object, if necessary, just a reference pointing to its secondary target object, i.e.:

$$\ldots F[a] \Rightarrow G[b]c \ldots$$

If *a* is itself the secondary target, the expression is just as simple:

$$\ldots F[a] \Rightarrow G[b]a \ldots,$$

and in the case of an affected action object group:

$$\ldots F[a, b, c] \Rightarrow G[d]a \ldots$$

The above expression is a very compact notation for a colloquially rather complex event. The labeling of the secondary target object used here is not mandatory. However, putting them into action triggers the formal necessity to have the secondary target object once again act as a source object in the subsequent expression sequence. Consequently, an expression having such a secondary effect identifier cannot be in the last position of an expression sequence. Something must happen to the respective secondary target object. However, this need for subsequent mention is omitted if the secondary target object is behind a negative impact operator. A practical example of this follows below in Section 5.

For the sake of transparency, we only use the labeling of secondary target objects in the following, if this is advisable in a specific context.

3. Admissibility of the classic ∧- and ∨-Connectives and the classic negation (¬)

We can now construct more complex expressions from the already explicated form set. This concerns first the possibility of a multiplicity of both causes and effects. The following expressions are valid in the GPL. However, the classical operators '∧' and '∨' retain only part of their classical meaning here:

$$\ldots F[a] \Rightarrow G[c] \vee H[d] \ldots ,$$

[28] For the introduction of the availability operators, see section 4 below.

$$\ldots\ F[a] \vee F[b] \Rightarrow G[c] \wedge H[d] \ldots .$$

or also

$$\ldots\ F[a] \Rightarrow G[b] \wedge H[b] \Rightarrow I[c] \ldots .$$

The connectives '$\wedge$' and '$\vee$' are here, quite traditionally, always double digits. However, with the following proviso: they are (logically speaking) not sentential connectives with a truth value that can be represented in a truth table. This confirms the already mentioned fundamental paradigm of GPL: The formal validity of their expressions does not imply their suitability for their evaluation as true or false or an even more extensive range of values. The GPL knows only a metalinguistic admissibility or inadmissibility of their expressions, i.e. only the difference between well-formed and non-well-formed expressions. This is in line with the reality that should be formalized with GPL: Events either happen or not; Statements that refer to the pure event without containing a metalinguistic assessment of the statement can therefore give no metalinguistic information whether the respective statement is true or false. This in turn means that expressions of the GPL cannot be represented in the form of truth tables. In a sense, it is formally purer than many of the classical logics, which basically operate on two different levels, the object-language statement level and the metalinguistic level of evaluation.[29]

First, we turn to the question of the applicability of the, $\wedge$ 'and, $\vee$' connectives.[30] Now, can the '$\vee$' connective (i.e. the disjunction) also be on the right side of the action operator? This question again presents us with a decision that can only be made in terms of the application of the formalism to reality. If we take formal-logical expressions directly as ontological, albeit highly generalized, images of empirical reality, the possibility of the implied real disjunction seems quite unclear. Only in the field of quantum mechanics and below would a physicist normally speak of real indecision. Philosophically, on the other hand, the situation is by no means clear even for the higher levels of existence. If anything, it is rather controversial.[31] Fortunately, we do not have to decide this question either here. For the GPL, I claim only

[29] And even this labeling of the evaluation of classical logical statements is not yet complete. In the evaluation e.g. of statements of classical propositional logic, it is not clear whether their assessment as true or false also refers to or not refers to the empirical basic statement in the case of their application. For example, consider the two statements 'The apple is ripe' (= *p*) and 'I like to eat apples' (= *q*). These statements can be logically modeled very simply as, $p \wedge q$ '. If both subexpressions are logically true, then according to the classical two-valued propositional logic the total expression is at least formally true. But how can we justify such a purely formal truth on a more general level? Logically, we judge the premises p and q in an abstract, i.e. hypothetical-empirical sense as true or false. But if this empirical connection of the premises is also to apply to the conclusion, then it is no longer a purely formal truth. From this impurity, however, then, *mutatis mutandis*, the entire formal concept of truth would ultimately be infested.

I do not want to talk about pragmatism parroting John Dewey (see Dewey [2002]). The problem of the connection between sign and signified is ultimately metaphysical and cannot be dealt with here. The said semantic ambivalence of the so-called truth values has therefore ensured continued discussions since its beginnings. It led Tarski to declare that the phrase 'snow is white' is true exactly when snow is white (Tarski [1944], section 1.4). It is alarming when, in a discipline like logic, even such a modest insight requires special expertise. However, the relationship between formal and empirical logic need not be discussed further here, as the GPL does not operate on truth values.

[30] In the textbook literature on logic, the terms 'connective' and 'sentential connector' are usually only very vaguely distinguished, although both are quite different. A connective does not necessarily have truth-relevant properties, whereas a sentential connector always has. Connectives are functionally much more primitive than sentential connectors. Thus, if one uses the signs '$\wedge$' or '$\vee$' as the connective, they only say that two terms in the respective context have a functional connection. On the other hand, if they are used as sentential connectors in the context of propositional logic, they are semantically defined by the known truth tables. Since the GPL does not operate with truth values, these signs must be understood therefore strictly as a connective.

[31] In Sohst [2016], p. 80ff. I have dealt in detail with the question whether, in view of our current scientific knowledge of the world, it is even possible to say whether the course of the world is strictly determinate. I think there to have shown that this cannot be deduced from our previous knowledge, and that such determination is even impossible for empirical reasons that seem virtually incontestable.

that it should be useful as an epistemological or communicative tool. Those who agree with this opinion, because they take it for granted or even trivial, but it should be remembered that the underlying question of the empirical validity of logical statements is not yet answered. From the said conception of logic in general, and thus also the GPL as a pure cognitive or orientation tool, it follows, however, that we can allow the disjunction of a process without further reservations also on the impact side. This greatly enhances the operational freedom of the GPL.

The admissibility of the classically two-valued negation (¬) has already been mentioned above in connection with the negative operative operator.

4. The positive and negative availability operator

However, this raises the much more general question of how to connect to an atomic process statement of the kind shown above. Therefore, before we can deal with branching due to a process-logical disjunction, we must look at another fundamental and quite new aspect of the GPL. As stated, the GPL does not describe object states, but active objects, i.e. objects as units of effect. This results in the requirement to formally model the formation and the disappearance of active objects. Thus, we establish that well-formed expressions may contain only those active objects that have been formally generated before and have not yet been eliminated.[32] We define the character '↗' ("UP") for the introduction of a new active object and the character '↘' ("DOWN") for its elimination. For these signs holds:

DEF 3: *Availability operations*

1. By the ↗-operation, a real object is generated in its entirety, i.e. as an object together with its specific effect. In its further use, this object may also adopt any other effects which need not be indicated by an ↗-operation to be valid. On the other hand, a simple introduction of a new action for a previously created object requires no explicit mention.
2. The ↘-operation removes the entire affected active object, i.e. it is thus no longer available for any further operation. Therefore, the d operator is applied only to the object kernel, that is, for example, ↘ *p* instead of ↘ A[*p*].
3. Before an ↗ and after a ↘-operation, there is no need for a '... ' mark, i.e. no inclusion mark, if they are the first, last, or sole operation within an expression on their side of the expression.
4. ↗- and ↘-operations can only be used within a complete action operation, i.e. in an operation containing the action operator ⇒.
5. Before the introduction of an object by ↗ and after the removal of an object by means of ↘, an expression containing such an object not yet introduced or already eliminated is invalid.
6. The ↘ operation immediately stops the entire effectiveness of an object. That is, such an operation cannot trigger any further effects, but will be only the effect of preceding events. The ↘ operation can therefore only take place on the right side of an action expression.
7. The ↗ operation may appear on both sides of an effect expression. However, it must always refer to a specific effect of the object created, i.e. it cannot create the object kernel alone.
8. Both the ↗ and the ↘ operation can be elementary parts of a complex statement.

From this follows another important distinction to all state logics: The object-language 'horizon' of an expression sequence, as that which an expression sequence deals with, is not fixed from the beginning,

[32] This procedure will remind logicians who are familiar with modern programming languages of the need to instantiate program objects and of their possible release. However, as already explained at the beginning, this is not about computer worlds but about our immediate cognitive reality. Now, for most of the objects that surround us, we do not have to worry about their explicit generation. However, in a formal system like the GPL, it is sensible to indicate, for reasons of transparency, which objects are procedurally already or still available. In the real world, this is more akin to the practical availability of objects, not their fundamental existence.

but can change considerably in the course of the expression sequence. The GPL is thus also formally dynamic. It not only describes processes but is also consistent in considering changes in the object inventory in the specific application.

This brings us back to the question of what a connection to complex expressions, such as disjunctions, might look like. Basically, any object on the right side of an operation that has not been eliminated because of the effect can be used to represent consequential effects, however not necessarily. For example, let's look at this expression sequence:

1. … ↗V[*a*] ⇒ ↗ L[*c*]*a* ∧ ↗E[*b*]*a*
2. E[*b*] ⇒ ↘*a* ∨ K[*a*] …

This sequence says in colloquial language:

1. An object *a* appears whose effect V is to produce the active objects L[*c*] and E[*b*].
2. Of these two new action objects, object *b* either causes *a* to no longer have the effect V or *a* to have the new effect K.

To illustrate the example in an everyday situation, one could imagine it (somewhat decorated): „ […] One morning, Albert appears in a student flat-sharing community and somehow manages to make sure that the cohabitee Claudia has a crush on him, while her roommate Bernd becomes seriously jealous. In his resentment Bernd either ushers Albert out of the apartment, or he manages to persuade the competitor to bake a delicious cake for all residents. […]“[33] For this somewhat more complicated sequence of events, the above formal presentation is quite compact. Furthermore, I would not know how to formulate them state-logically while fully preserving the dynamic core statement.

Now, however, it will be necessary to clarify which binding effect a once introduced action operation has and how an object kernel can be attributed new effects. To be specific: Does Claudia must have a crush on Albert from now on forever? Hardly likely; reality also requires formal adaptability here. Basically, action-like relationships are shown in binary form in the GPL. They are either 'on' or 'off'. We thus assume that a once established relationship does not change automatically. For example, Albert himself is unable to eliminate Claudia's affects to him, unless his own latent action profile changes accordingly. Once an effect relationship has occurred, it can only be canceled in the GPL by another, further action operation. This operation is exactly that form of negation that we have already encountered above with the action sign '⇏'. This negation is now used. We continue our above expression in the sense of a fictional event, according to which another person named Franz appears and Claudia makes it clear that there is no point in being in love with Albert. To formalize this process, we repeat the initial expression sequence and add another line that does not yet have to produce the decision regarding the preceding disjunction:

1. … ↗V[*a*] ⇒ ↗L[*c*]*a* ∧ ↗E[*b*]*a*
2. E[*b*] ⇒ ↘*a* ∨ K[*a*]
3. ↗U[*f*] ⇏ L[*c*]*a* …

The original relationship ↗ V[*a*] ⇒ ↗ L[*c*] is thus no longer available. The secondary target at the end of line 3 follows a negative action operator. Further reference to the core object *a* is therefore no longer necessary (see above explanation at the end of section 2).

But now let us also bring about the pending decision of the disjunction… ↘ *a* ∨ K[*a*]. We therefore write (including the preceding lines):

[33] This example contains relational specifications that the previous formal expression does not yet depict, e.g. that said Claudia only loves Michael and no-one else. Diese fehlenden relationalen Merkmale werden wir erst weiter unten einführen. We will introduce these missing relational features below

1. … $V[a] \Rightarrow \nearrow L[c]a \wedge \nearrow E[b]a$
2. $E[b] \Rightarrow \searrow a \vee K[a]$
3. $\nearrow U[f] \nRightarrow L[c]a$
4. $E[b] \Rightarrow K[a]$ …

Now, logic is generally understood to be the art of reasoning. Where is the logical inference here? In fact, several conclusions are possible from what has happened, for example:

…
$\therefore$ $E[b] \nRightarrow \searrow a$

or

…
$\therefore$ $V[a] \nRightarrow L[c]$

Both conclusions are formally valid following the previous determinations. They also correspond to the expected result from our fictional example: After convincing his opponent to bake a cake, Bernd is no longer able to throw him out, and Albert in turn cannot induce Claudia any longer to love him. In principle, however, it should be noted here that conclusions in all process logic, not just the GPL, only play a provisional role. The habitat of the real logical conclusion is the ancient syllogism. Already in classical-modern logics, recast is more important than the end. Finally, in the known process logics as well as in the GPL, the conclusion is nothing more than the somewhat random end of a sequence of expressions that could be continued if necessary.

5. Process-logical action relations

We have already mentioned that realized effects often need to be supplemented with a relational feature to better reflect the intended reality. Particularly in hierarchical social relationships, the roles of the parties involved everywhere also define their interaction, e.g. between doctor and patient, teacher and student, civil servant (representative of the state?) and citizen, employer and employee etc. Such asymmetries of the relationship of actions and effects, which are frequently encountered not only in the social, but also in the purely physically or chemically defined conditions, must necessarily be reproducible within the framework of a formal process logic. This shows a strong practical difference to the state-based process logic of the type mentioned at the beginning.

Such additional features of action relationships must, to be realistically applicable, always focus on the overall action relationship, not just on the involved object capabilities. For example, an employer may give instructions to the employee, but not vice versa. However, this does not apply to the basic right to terminate the employment relationship; this right is mutual. In such cases, the same object cores stand parallel to each other in different interaction with one another. Consequently, the said qualification of the action relationship must always refer to its concrete totality of all effects involved.

However, such a concrete distinction should now be preceded by a general categorization, which relations are permissible at all, that is, a form catalog of the permissible relations of action between objects. This can be taken from the classical relation logic. We define:

DEF 4: Permitted action relationships

Effects of actions can be formally valid in the following way:

1. Reflexivity: This concerns the action species:
   $F[a] \Rightarrow G[a]$
   $F[a] \nRightarrow G[a]$
   Note that the reflexivity (self-efficacy) here refers to the object nucleus, while the concrete effect must differ. If an expression differs neither in its core expression nor in its concrete

effect, there would be no effect whatsoever. Such reflective relationships are not self-evident. If they are to be admitted, they must be explicitly awarded for this; see below. If the reflexive relationship is allowed, then

$$F[a] \Rightarrow G[a] \text{ equally } G[a] \Rightarrow F[a], \text{ and for}$$
$$F[a] \nRightarrow G[a] \text{ equally } G[a] \nRightarrow F[a].$$

The action relationship $F[a] \Rightarrow \searrow a$ can never be reversed, because a removed object cannot exert any further effect, see above Def. 3.6.

2. *Transitivity*:

$$\begin{array}{l} F[a] \Rightarrow G[b] \\ G[b] \Rightarrow H[c] \\ \hline \therefore F[a] \Rightarrow H[c] \end{array}$$

Transitivity is available for both the positive and negative standard action operator. It is presented above as a conclusion to illustrate the formal potential of the action operator. It's basically just an abbreviated illustration for a chain inference between F[*a*] and H[*c*]. Empirically, chains of effects are never overall transitive, because they always require the causal intermediate step to achieve the result. However, the transitive relation of action has a high practical relevance insofar as we often sum up many sub-events in our images of real events. For example, we say: 'The dog was injured by that driver', although the driver did not do it himself, but the car he steered. With such sentences we express a responsibility rather than a causal relationship, whereby causality still plays an important role. Often, we shorten causal relationships simply because the respective description level suggests this. For example, if I say, "Because of the cold, I have a stuffy nose," this sentence expresses a much more complex interdependence. Nevertheless, the possibility of such sentences is not only sensible but even indispensable for everyday orientation.[34] But because the possibility of analyzing and synthesizing such detail and global action chains is constantly needed, we must also provide the formal-logical possibility for their representation in an action logic.

3. Commutativity:

$$\begin{array}{l} F[a] \Rightarrow G[b] \\ \hline \therefore G[b] \Rightarrow F[a] \end{array}$$

Here, too, the non-compulsory formal alternative of action reversal is illustrated as a conclusion in propositional logic, this type of reversibility is also referred to as symmetry based on physical terminology, which is particularly useful in process logic.[35] Again, the commutative or symmetrical effect by no means self-evident. A reversal of effects is empirically not the rule. A subspecies of the symmetric relationship is its negative reversibility:

$$\begin{array}{l} F[a] \Rightarrow G[b] \\ \therefore G[b] \nRightarrow F[a] \end{array}$$

This again clearly shows the difference between the propositional negation and the negative effect operator. It is something completely different to explicitly allow the negative reversal of effects, as stated above, or simply not to say anything about the commutativity of a

[34] A closer look reveals further questions: Under what circumstances is it right to speak of a set of parts as the actual cause of an effect? In a car accident, it should not be difficult to consider the car in question as an action unit. But what about complex natural events, e.g. rain, landslides or tidal waves? Are mass social phenomena such as collective consumer behavior, political elections or even riots and war events sensibly to treat as causally unified events? Fortunately, we do not have to go back to this question here. In Sohst [2016], p. 143 ff., I discuss this fundamental ontological question in detail.

[35] See, for example, Kutschera / Breitkopf [1971], p. 130.

concrete relationship of action. In this case, from the non-distinction of commutativity merely follows the simple negation of the inversion, i.e. $\neg(G[b] \Rightarrow F[a])$ or, which means the same thing, $G[b] \neg \Rightarrow F[a]$, which does not apply categorically, but only for the respective expression, so long as no further expression removes this restriction.

It is clear, therefore, that the presence of the simple or negative symmetry of action must in any case be explicitly indicated formally, if only to distinguish it from the positive symmetry, since these two action alternatives are mutually exclusive.

4. *Distributivity*:

   The following applies:

$$\frac{F[a] \Rightarrow (G[b] \vee H[c]) \wedge [I[b] \vee K[c])}{\therefore F[a] \Rightarrow ([G[b] \wedge (I[b] \vee K[c])) \vee (H[c] \wedge (I[b] \vee K[c]))}$$

   This in turn yields the validity of all the following conclusions:

$\therefore F[a] \Rightarrow G[b] \wedge (I[b] \vee K[c])$ (Continuation with the 1st main alternative)
$\therefore F[a] \Rightarrow H[c] \wedge (I[b] \vee K[c])$ (Continuation with the 2nd main alternative)
$\therefore F[a] \Rightarrow H[c] \wedge I(b]$ (Continuation with the 1st subalternative)
$\therefore F[a] \Rightarrow H[c] \wedge K[c]$ (Continuation with the 2nd subalternative)

   Distributivity, like commutativity or symmetry of action, requires explicit formal labeling.

5. *Labeling Rules:*
   a) The relational characterization of interaction takes place at the impact operator. It applies to the entire qualified expression consisting of source and target objects.
   b) Once a characterization (i.e. the label for an expression) has been introduced, it is only binding for the source and target objects involved in the selected relationship. Other mutual effects of the same object cores are not affected.
   c) If several objects connected to each other by the $\wedge$-connective are involved on the source or destination side, then the relational characterization of the action relationship always applies to the entirety of the objects connected in this way. This means that the same effect cannot take place between individual elements of the participating object group independently of their relational identification.
   d) If several objects connected to each other by the $\vee$ connective are involved on the source or target side, then the relational characterization of the action relationship applies only to the individual members of the disjunction in relation to the source or target object, which is the same for all disjunctive alternatives.
   e) Relational identifications of action relationships can subsequently be reversed.
   f) Relational identifications of interaction may basically, i.e. under appropriate circumstances, be used cumulatively.

The explicit identification of an action operator as reflexive, commutative, transitive or distributive is required so that appropriate valid conclusions can be drawn from the relevant expressions. Without such labeling, all inferences that require it are invalid.

We characterize an action relationship as reflexive, transitive, commutative or distributive by means of a subscript behind the affected effect operator, i.e.

$\Rightarrow_{R+}$ for the positive and $\Rightarrow_{R-}$ for the negative reflexivity
$\Rightarrow_{K}$ for the commutativity
$\Rightarrow_{D}$ for the distributivity
$\Rightarrow_{-T}$ for the negation of transitivity
$\Rightarrow_{-(<label>)}$ for the cancellation of any relational labeling.

The concrete use of these additional labelings can be illustrated for example in this way:

$F[a] \Rightarrow_{R-,K} G[b]$ | the action relationship is negatively reflexive and commutative
$G[a] \Rightarrow_{R-,K} H[c]$ | and the subsequent action relationship as well.
$H[c] \Rightarrow \searrow a$

6. Object merging and splitting

6.1 Fundamentals

At the beginning, I pointed out that the GPL presented here has a possibility that is inherently not accessible to the state logical systems, because they cannot reflect changes in the respective object existence beyond the limits of the properties of these objects. However, our everyday life is full of examples of object fusion and splitting. Thus, in all production processes, where a larger totality is produced from individual parts, new objects are created that did not exist before. Conversely, each detachment of a piece from a preceding object unity is always a splitting, when, after having removed it, the number of independent, i.e. individual objects turns out to be bigger than before. The naturalness and frequency of these events in everyday life is so ubiquitous that it is astonishing that so far no logical formalism has been invented which enables their reproduction.

An expression of the fusion creates a new, unified target object from a multiplicity of source objects. We use the sign '⊕' for it. The reverse process, the splitting has a source object to the object and produces from it a multiplicity of independent targets. We use the sign ‚∅'. Both processes are formally realized by corresponding operators, which, because they are closely related to each other, are referred to below with the English initials of their identifiers as 'FS operators'.

DEF 5: The FS operators

We define the FS operators as follows:

1. 1. For the fusion, at least two different source objects must be named on the left side of the ⊕-operator, separated by commas for better distinction.
2. On the right side of the ⊕-operator exactly one new target object appears, i.e. an object that has not yet been formally created. Its production, in the course of a fusion therefore requires no prior positive availability operation.
3. 3. In subsequent expressions of a fusion, the fused source objects remain effectively available to each other only internally. They cannot exert an external effect as long as they have not been removed from the fusion by subsequent separation. We call this logical feature 'endogenous constraint'.
4. The source object of split objects (operator: ∅) only perishes and is therefore no longer available when it completely disassembles into its individual parts. Only in this case, a corresponding object must be newly created to become available again. The demise of an object due to total splitting must be displayed separately. If this is not done, it will still be available in the further course of events.

Let's look at an example expression sequence using the FS operators. We imagine that Albert, here: *a*, as an avid cyclist buys three new items of a racing bike and combines them into a new racing bike:

1. … $\nearrow P[a] \Rightarrow \nearrow b \wedge \nearrow c \wedge \nearrow d$

2. $A[a] \Rightarrow F[b \oplus c \oplus d] \Rightarrow \nearrow f$

The unifying fusion of the parts *b*, *c*, and d into the new object *f* is done formally simply by bracketing the parts to be united and an action operator pointing to the creation of the new object *f*. If the target object of the fusion already exists, because (in our example) Albert buys new parts, but builds these into an already existing bicycle, we express this as follows:

1. … $\nearrow f$
2. $\nearrow P[a] \Rightarrow \nearrow b \wedge \nearrow c \wedge \nearrow d$
3. $A[a] \Rightarrow F[b \oplus c \oplus d]f$ …

In this case, the bicycle f is first generated at the beginning of the process sequence to then be used as a target object. This gives us the consistency of our respective object-effect universe.

Similarly, the dissolution of an object into its constituent parts and the detachment of individual parts from a persisting object will be represented, whereby the named parts are generated here simultaneously:

1. … $\nearrow a$
2. $D[a] \Rightarrow \nearrow b \varnothing \nearrow c \varnothing \nearrow$ …

6.2 Σ-Objects

However, the formalization of an object fusion opens further possibilities of representation if we distinguish between two fundamentally different forms of such a fusion, namely:

a) Demise of the elementary objects by fusing into a new simple object
b) Continuation of elementary objects after having been integrated into a new, complex object.

If we make an object fuse in the sense of (b) above, the result is a new, complex object, i.e. while preserving the demise of its elements from fusion, this new object can be used in the following as an independent entity like any other simple object. In addition, we can now use this new object to manage not only "external" effects from and to other objects other than itself, but also to deal with its own components, i.e. for internal control. The basic idea of such constructs is that such a fusion creates a superordinate unit of action that organizes its elements in such a way that the leading unit of action, in interaction with changed environmental conditions, is adaptable in terms of its internal structure. From now on I call such complex objects Σ objects. However, the term 'Σ-object' is a class term. When creating a concrete object of this type, the type designation Σ is therefore to be added to the actual object designation as a superscript, to maintain the distinction of its elementary objects, e.g. in the form $F[p^{\Sigma}]$.

Suppose we create such a Σ-object named *p*, which results from the integration of the persisting elementary objects *a*, *b* and *c*. The elementary objects experience, as a result of the fusion into a Σ-object, a change of their potentials for action precisely to the extent necessary for the creation and maintenance of this new object. In addition, their action profiles remain unaffected. To create such a Σ-object, it is therefore necessary to specify the changes in the action profiles of its constitutive elementary objects that occur through the fusion. The GPL does not have an operator that indicates the change in potency. However, there is the formal possibility of a positive, i.e. new assignment of action powers by means of

a according assignment expression to an existing object, as well as the reverse formal possibility of a negative assignment, i.e. a deletion, of potential effects by means of a potential negation (symbol: $\neg$).[36]

With the help of these two formal operations, it is possible to specify for each elementary object exactly what changes it undergoes through its integration into a Σ-object. But such representations of change are not compulsory. Taken purely formally, every elementary object of an Σ-object is curtailed in its horizon of action by the fact that as an element of the Σ-object it can no longer develop its own external effect. However, this is already a consequence of the fusion anyway and therefore does not need to be displayed separately. Expressions for the creation of Σ-objects must, however, at the moment of their formation by fusion, explicitly indicate all affected elementary objects and the modification operations taking place on them. Otherwise, such an expression is incomplete and therefore no longer well-formed.

As a form of the notation of the processes involved, we determine:

1. To introduce the new Σ-object following the convention already mentioned above, wherein the initial object example V [a] is freely chosen:

   $$V[a] \Rightarrow F[a \oplus b \oplus c] \Rightarrow \nearrow p^{\Sigma} \ldots$$

2. However, the above expression does not say anything about the mentioned changes to the potential effects of the affected elementary objects. Since these take place logically simultaneously with the generation of the Σ-object, but their mention in the same line can become very confusing, we write three continuation points at the end of the line generating the Σ-object and complete the said changes of the elementary objects in the following line, although there is no need for any further action operator, but rather a formal indication of the changed action profiles of the integrated elementary objects. As usual, we write these changes in square brackets after the Σ sign, otherwise multiple changes to the same elementary object in simple parentheses; as a delimiter we use the semicolon at all levels, for example:

   $$p^{\Sigma}\,[(\neg K[a]; L[a]; M[a]); (L[a]; N[a]); (\neg O[c])].$$

   This will reveal the nested structure of the new Σ object. If the affected elementary objects still have other potential effects before their integration, which are not mentioned here, these remain untouched.

3. Furthermore, in the further course of the expression sequence, it will often also be necessary to add further internal elementary objects to the newly-created Σ-object or to delete existing elementary objects. This, too, can be formally expressed without further ado:

   $$A[p^{\Sigma}] \Rightarrow [\ldots \nearrow (L[d]; M[d]; N[d])], \text{ i.e.}$$

   $$A[p^{\Sigma}] \Rightarrow [\ldots \searrow (a; b)].$$

In addition, a Σ-object can formally act without restrictions and completely independently like an elementary object. In particular, it can also have potential effects, e.g. be assigned or denied to it in the form $F[p^{\Sigma}]$.

7. Control structures

The previous section already mentioned the possibility of a process-logical presentation of internal control structures. Such controls (the term will be discussed in more detail below) may refer both to the possibility of altering the internal states of a Σ-object as set forth above, as well as to the

[36] See above section 2.1

interdependencies of updating an effect between free, i.e. objects not bound in Σ objects, including the interaction between simple and complex objects.

In all contemporary programming languages, control structures are an essential element of each language. However, the GPL is not a programming language, but a logical formalism.[37] It offers control structures, but only to the minimum extent necessary for the formal representation of process dependencies. Process logic cannot do without this, because formally an essential feature of process structures is the dependence of the updating of an initially only latent effect of certain environmental conditions.

First, it must be clarified what a process condition is. In fact, in the context of GPL, it simply means defining a limiting condition when a latent object power is updated. Thus, process conditions never force the actualization of a potency, but always only restrict the arbitrariness of the occurrence of an effect. On the other hand, when the condition occurs, they enforce the specified follow-up effect. In this regard, the process conditions formalized below are deterministic. Such conditions may be coupled in the GPL to a single consequence effect or a co-occurring multiplicity of such consequences. This also shows an essential difference to all control structures of the usual programming languages: Since these are all based on the well-known state logics, and indeed predominantly on a two-valued propositional and predicate logic, all control structures also work only through the control of states of other objects in the respective program horizon. That means: They do not control processes but only states. By contrast, the basically very simple control structure reacts directly to other processes, not just to their resulting state sequences. This, in turn, has implications for the validity of such structures, as will be seen below.

In the GPL we use only a single control operator to indicate the dependence of an action condition on the occurrence of other effects whose sign is the word 'IF' known from the programming languages. The syntax for his use is, for example:

$$\text{IF } \{<\text{action potency 1, 2, } \ldots n>) \Rightarrow (<\text{action potency 3, 4, } \ldots n >)\}$$

or in the case of the negative condition, i.e. not the positive triggering of a subsequent action, but the prevention of a consequential action

$$\text{IF } \{(<\text{ action potency 1, 2, } \ldots n>) \nRightarrow (<\text{action potency 3, 4, } \ldots n >)\}$$

the affected potencies on both sides of the dependent effect may or may not be the same. The above notation implies, within the inner brackets, a logical AND operation between the individual action powers on both sides of the effect operator. An OR operation is also possible without further ado, but must then be explicitly indicated by using the $\vee$ symbol.

The definition of such a control structure is therefore basically only a deterministic generalization of the otherwise arbitrarily occurring effects. 'IF' has the effect that a certain consequential effect always occurs when the specified conditions of effect occur. Thus, the IF operator not only enables control of causal relationships, but also their automation. We can now give a specific IF expression as well as entire complexes of nested IF operators each an individual name. Then we need only mention the name of this IF operator in an expression sequence at the point where the trigger already set in the IF expression occurs. Then we can jump right to the end of the previously defined sequence of effects.

For example, consider the following IF expressions as given, where we combine two elementary operators into one complex IF expression and give each of these operators a unique name following the expression:

[37] Control structures were adopted by computer science early on from cybernetics. However, informatics in the form developed today is still based on state-logical paradigms, where process dependencies can only be represented after a corresponding transformation into a series of object states into control structures. In contrast, the GPL provides these means from the outset, i.e. as an instant property of process-logical formalism itself.

$$\text{IF } \{F[a] \vee G[b] \Rightarrow F[b], H[b]\} \text{ IF\_1}$$
$$\text{IF } \{F[c] \nRightarrow L[b]\} \text{ IF\_2}$$
$$\text{IF } \{\text{IF\_1}, \text{IF\_2} \Rightarrow M[c]\} \text{ IF\_3}$$
$$\ldots$$
$$G[b] \wedge F[c]_{\text{IF\_2}} \nRightarrow L[b], \Rightarrow_{\text{IF\_3}} M[c].$$

This is a simple example of nesting. Even a simple chaining is possible:

$$\text{IF } \{F[a] \vee G[b] \Rightarrow F[b], H[b]\} \text{ IF\_4}$$
$$\text{IF } \{F[b] \nRightarrow L[b]\} \text{ IF\_5}$$
$$\ldots$$
$$G[b]_{\text{IF\_4, IF\_5}} \nRightarrow L[b].$$

By constructing even more complex conditional structures, it is possible to present very extensive effects with a low expression length.

Just as simple is the abrogation of once defined conditions. It takes place by simple negation of the relevant IF expression, designated by its unique name, for example:

$$\neg \text{ IF\_5}.$$

The possibility of negating an effect condition clearly shows the difference to the control structures of common programming languages. Since we are dealing with a conditional operator in the IF operator of the OPL that is 'incessant', i.e. not only in the case of a concrete inquiry of the status, but permanently in its entire horizon of effectiveness, we must also cancel it again if the condition no longer applies. In contrast, in the case of programming languages, control structures always occur at very specific program points at which certain states in the program horizon of known objects are queried. Programmatic control structures therefore have only a very local and punctual, i.e. unique effect.

On the other hand, the horizon of validity of a GPL control structure can certainly be restricted when applied to a Σ object.[38] If an IF operator is applied to the elementary objects of a Σ-object, it can by definition have no effect outside this object because the internal latent effects of a Σ-object cannot be directly updated from the outside. Σ objects must always be addressed from the outside as independent integral entities. However, indirectly, an external influence on the internal potencies of the Σ-object can be represented by the respective Σ-object itself appearing on the left side, i.e. the conditional side of an IF operator, whereby complementarily on the right side an Σ-internal order of action is determined. In this way, even a coupling of so-called sibling objects can be represented. For example, suppose that we want to represent the history of the action of an object that is an integral part of an Σ-object, but also, ontologically speaking, has an existence outside of that object. This is possible in the GPL as well.[39] Such relationships are mapped in the OPL as two different objects, because the respective action profiles differ. Ontologically, an object identity is present here. However, this does not come into its own in terms of formal logic because logic is generally not an instrument for mapping ontological conditions, but rather a means of plausibility of assertions in linguistic communication.[40] For this reason, it is formally better to logically represent such different roles of ontologically identical objects as different, but specifically coupled, objects. I refer to such coupled objects as sibling objects.

[38] This has a certain similarity with the so-called encapsulation in the object-oriented programming languages.
[39] Such conditions are the rule, especially in the social sphere. We usually refer to these different functions as the different roles of a person. For example, a person may have a role as an employee of a company and remains unaffected by it as a "free" person outside that company.
[40] In this regard, I join the group of representatives of the so-called inferential role semantics, in particular the inferentialism of Robert B. Brandom, see Brandom [1994].

The process-logical coupling of sibling objects can be formally represented by two successive IF operators, whereby in the middle between the 'free' outer and the bound Σ-internal object, the Σ-object always acts as an intermediary. For example, let a_1 be the 'free' outer object, G[b^{Σ}] a Σ-object named *b*, and a_2 an internal element of the Σ-object. Then we note:

$$\text{IF } \{F[a_1] \Rightarrow G[b^{\Sigma}]\} \text{ IF\_6}$$

$$\text{IF } \{G[b^{\Sigma}] \Rightarrow H[a_2]\} \text{ IF\_7}$$

If IF_6 occurs, IF_7 is also triggered immediately. As a result, an external event of the "free" sibling object is coupled to the effect of its Σ-internal sibling object. Of course, such a coupling is not limited to sibling objects. However, the possibility of creating sibling objects in particular allows the logical mapping of relationships that often play a decisive role in reality. For example, a person can develop very different effects in the different roles of their life context. However, if she physically dies in one of these roles, she is no longer available in any other role, too. Therefore, such relationships must also be logically representable.[41]

Further formal arrangements for the representation of control structures are not envisaged in the GPL, because all necessary process-logical connections can already be represented in this way.

8. Alternative process flows

The reality in which we have to orientate ourselves, confronts us with another peculiarity that we usually deal with in everyday life without difficulty, but which has never been mentioned explicitly in Western logic. This concerns the subjective interpretation of events, which we are aware of, but not sure, whether our own view corresponds to reality. For example, a known customer suddenly no longer pays the delivered goods. Is he up to something against us and wants to use the owed sum as a pledge? Is he possibly insolvent and cannot pay? In such situations we provisionally gain security by going through various alternative courses of action. Nevertheless, in order to make the unsafe variants again realizable, i.e. we often come up with a result that can also be achieved through several alternative routes because of the existing uncertainties.

The GPL can map such alternative process flows within an overall conclusion. In the course of their development, divisions initially arise as a result of the uncertain alternatives of an initially unambiguous process unit. These alternative pathways may or may not be merged later:

[41] Denis Noble in Noble [2017], p. 69ff., describes very impressive cases of such dependency on the molecular-biological level.

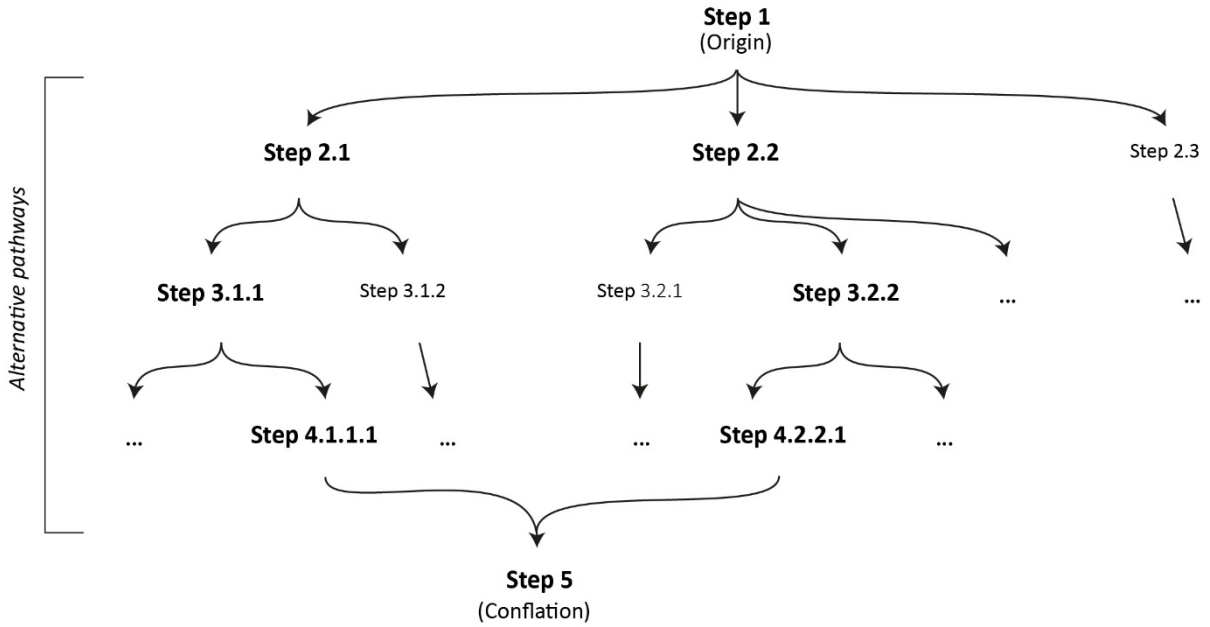

Fig. 2: Splitting and merging of alternative pathways

We see here several alternative pathways, some of which are not pursued, represented by, ... '. However, two strands converge at the end, the one over the steps {1, 2.1, 3.1.1, 4.1.1.1, 5} and the other over {1, 2.2, 3.2.2, 4.2.2.1, 5}. State logics can hardly represent such alternative splits and possible conflations, because states by definition cannot map developments.

9. Coupling of process-logical and state-logical expressions

Previously, we mentioned that process and state logic expressions can complement each other. From the practical perspective of the application of logical formalisms, this is even a necessity if we want to comprehensively model reality. Although the GPL claims to represent the primary access not only of humans, but also of all higher animals in the world. Nevertheless, state focused propositions are indispensable for every living being. Even a squirrel hiding acorns somewhere in an earth cave, not only has to remember the location of the cave, but also the simple fact that acorns are there at all. This is important status information. Expressions of the GPL and the traditional state logical systems can be easily mixed in this regard.

The question, however, is how one can even extract cognitive states out of genuine process flows and vice versa. Even a brief reflection on this shows in which different worlds process thinking and state thinking are involved. Although processes, to be able to be treated as individual (i.e. single) processes at all, always lead from a start to an end, these boundaries turn out to generate objects each with determined state properties.[42] However, the formalism of the GPL nowhere yields a determinate state, and conversely fixed predicate logical statements never yield a procedural cause-effect relationship. In the case of a formal language change, the respective missing element must therefore be introduced *ad hoc*. Accordingly, such an introduction should be expressed by a clear marking of the respective system change.

As an indication for the change of the logical system I propose the following: , $|^{GPL}$ ' when changing to the GPL and , $|^{P}$ ' when turning back to predicate logic. Specifically, this could look like this:

1. … $\nearrow \mathrm{K}[a] \Rightarrow \nearrow b$ | An initial GPL statement
2. $\mathrm{L}[b] \Rightarrow |^{P}\, \mathrm{F}(b) \in X$ | Switching to predicate logic directly after the action operator
3. $\forall(x) : \mathrm{G}(x)$ | Going on in predicate logic fashion after the change

[42] See in detail Sohst [2016], p. 123ff.

4. ∴ F(*b*) ∧ G(*x*) | Predicate logical conclusion from 3.
5. F(*b*) ∧ G(*x*) $|^{GPL}$ ⇒… | Newly switching to GPL.

Such a change certainly requires scrutiny to always be clear about which level of expression one is currently dealing with. The graphical distinction between square and round brackets of the respective objects somewhat intuitively eases this task.

10. Functional completeness and consistency of the GPL

Since the invention of mathematical logic, beginning in the late nineteenth century, formal-logic systems must prove the validity of their conclusions by passing various formal examinations. The two most important of these suitability tests are known under the terms 'functional completeness' and 'consistency' (meaning: without formal contradictions). Logicians, however, despite the rigidity of their validity claims, are altogether astonishingly tolerant people when it comes to letting deviants have their say. So there are now a great deal of logical approaches that do not meet one or both of the above requirements and still have their practical justification.[43] The question, therefore, is whether or to what extent the process logic presented here offers the possibility of proving its functional completeness and consistency.

10.1 Functional completeness

A set of connectives is called functionally complete if, and only if, a small subset or at best only one of these connectives suffices to replace all the other connectives of a given formal language. It does not matter if the total set of connectors contains single-digit or multi-digit connectives. The functional completeness is therefore the result of a kind of compression: An arbitrary set of logical connectives can be reduced to a smaller number in such a way that all other connectives can be derived from the functionality of the remaining elements of this subset.[44] In the case of the GPL, however, I do not see how the previously introduced set of connectives can be reduced. We have introduced it constructively, ie. successively, because of considerations of what further formal operations are necessary to represent a processual reality. I would not have introduced any of the subsequently introduced connectives if they could be represented using the already existing set of symbols.

The GPL is thus compared with e.g. to the elementary propositional logic an elaborate system, because it requires a larger number of connectives and syntactic symbols than propositional logic. In comparison with other, more complex classical logical languages, however, the comparison is more moderate. Extending the predicate logic by quantifiers and the usual modal-logical systems are not simpler in construction than the GPL.

However, the notion of functional completeness has another meaning that goes beyond the simple reduction in the number of connectives. In multi-valued logical systems, it is questionable whether it is fundamentally possible to represent the *n* sentential connectives of such systems by a significant smaller amount or even by only one out of the set. In fact, Peter Markmann has recently been able to show that in every *n*-valued logical gate a central sentential connective can be determined, with the help of which, by means of numerous systematic transformations, all the other sentential connectives of such a gate can be represented.[45] For the GPL, however, this is initially of no importance, since it does not work at all with values in terms of object language. The question of the functional completeness of the GPL is therefore to be answered in the negative insofar as it is not possible to reduce the amount of the

[43] A very good overview of the amazing variety of such approaches is given by Priest [2001].
[44] For elementary propositional logic, such a reduction is possible even on a single connective. This is called the Sheffer stroke. Within the 16 sentential connectives of the two-valued two-digit propositional logic, there are even two such Sheffer functions, namely the NAND and the NOR-connective.
[45] Still unpublished work. For more than two years, however, I have studied with him the systematics of transforming *n*-valued logical gates and have shown him that each such gate can be represented by a single so-called key connective.

previously introduced set of connectives. On the other hand, the necessity of the functional completeness proof in its classical form does not apply, if only because no truth values are used in the GPL.

9.2 Consistency

A logical system is free of contradictions if one cannot derive a true expression and at the same time its negation from a set of given expressions. However, this concept of consistency is not applicable to the GPL because it does not work with truth values. But in a somewhat weaker sense, consistency must also apply to calculi such as the GPL, in much the same way as we assume empirical reality to be consistent physically, biologically, etc. Such a consistency axiom underlies the entire modern concept of science.

Not only does this term apply to the GPL, it also fulfills the corresponding requirement. Since all logic, and consequently also the GPL, are only epistemological tools, and consequently their expressions have no direct ontological claim to validity, it can only depend on the set of rules for the formation of well-formed expressions, whether a logic is consistent or not. However, since the GPL does not use the truth values and thus the state logic kind of conclusions, the possibilities of forming inconsistent expressions are even smaller here than in the traditional logics. In fact, all those specific contradictions are eliminated that can only occur in state logical systems based on the evaluation of their expressions. What is left are possible ambiguities in the syntax and rules of operation of the individual elements of the GPL. Although I have tried to be as unambiguous as possible, I cannot rule out the need for correction. However, this could always be done by appropriate amendments in the definition of the syntax and the elementary functions.

I therefore think that the GPL can be characterized as consistent as long as no contradictions can be found and it can in principle be asserted that even if such contradictions occur, they can be remedied by a correction of their formal rules.

11. Is the GPL a logic at all?

In view of the essential deviations of the GPL in their basic forms and principles from the traditional logics - above all: the complete omission of bivalent or multivalent truth values of statements as the basis of the logical transformation and final evaluation - the question arises whether the GPL is about logic at all. An answer to this depends on what one explains to be the essence or the indispensable (categorematic) attributes of formal logical languages. Those who insist that formal logical languages must have exactly those features that have been deliberately abandoned here in favor of other possibilities will certainly deny that the GPL is a formal logic. However, there are good reasons not to submit to such dogmatism.

The question of whether the GPL is a logic can only be answered by using a specific definition of the term 'formal logical language'. What at first looks like an unsolvable task, because it allows so many answers, turns out eventually to be quite manageable, if we approach such a definition attempt in the reverse manner: what can a formal logical statement system certainly not be, simply for the reason that it simply cannot meet the expectations associated with it? From this negative perspective it turns out that a commonly used criterion for defining logic is in fact not good: it is often said that logic is the basis of the correct thinking.[46] However, the attribute 'correct' in this context can mean two things: firstly, the purely formal inner consistency of a chain of linked statements, and, secondly, the validity of the evaluations and conclusions of traditional logic that correspond ontologically with extralinguistic reality. However, the traditional logics only do the former, and so does the GPL. Hence there is no reason to deny the GPL its status as logic. As for the correspondence with extralinguistic reality, neither the traditional logics nor the GPL are able to do so or even to guarantee it. Formal statement systems are basically indifferent to extra-linguistic reality. No logic, neither the traditional species nor the GPL, can say

[46] Here I use the term 'logic' as an abbreviation for 'formal logical language'.

something about extralinguistic reality. This was already the cardinal error of Anselm's "proof of God", and G.E. Moore's "assertion"[47] that a statement of fact never suggests its ethical quality applies in general to any logical statement: what is logically conclusive, error-free, or in any other way formally valid does not say much about the ontological status of such a statement. Again, this criterion is not suitable for clarifying the question of whether the GPL is even a logic.

But if the logic is not a direct empirical cognitive aid to knowledge, there is only one other explanation why we value it so much. This explanation refers to a trend of the theories of meaning of language, which is commonly referred to as the 'usage and meaning theory' of language.[48] According to this view of human language, one of its most influential representatives, Robert B. Brandom, is essentially concerned with understanding how language serves as a means of creating symbolic, social commitment. In conversation, we can only persuade other people with the statements of fact that we claim to accept these allegations and to act in accordance with them if our presentation of the facts is sufficiently plausible. This is where logic comes into play as one of the most important tools for generating communicative liability. Linguistic communication often involves a chain of innumerable concepts and implicit assertions that cannot possibly be empirically verified in a concrete conversation. However, to be able to check a minimum of plausibility as the recipient of corresponding messages, we often have to restrict ourselves to examine only the internal consistency of the asserted statements. Exactly this is also a good candidate for the definition of formal logical statement systems in general: A logic is therefore any such system of formal statements that is suitable for such a plausibility check.

Based on this criterion, I believe that the GPL, like the traditional logics, is a true formal language. The above criterion is also not too far: mathematics, for example, is by no means suitable in the comprehensive sense as formal-logical statement systems to prove the plausibility of common-language expressions. At best, it meets this requirement in arithmetic assertions. However, further mathematical constructs in everyday communication are no tools for validating general statements.

Another question is whether the GPL is more like the programming languages for digital processors. These are also formal languages, but not statement systems. Programming languages are rather command systems: they control the behavior of machines. To that end, they use the principles of formal logic, but only among other things. However, this is only a means for an altogether different, namely practical purpose. In this respect, it is the same with the GPL as it is with the traditional logic: it is not practically oriented, but it can easily be implemented in programming languages as well. In this sense, I am of the opinion, that the GPL is a fully-fledged formal-logical language.

[47] See Moore [1996], p. 77ff.
[48] One of its most influential contemporary representatives is Robert B. Brandom, whom I greatly appreciate in this regard, see especially Brandom [1994].

Bibliography

(*This essay was originally written in German. Thus, the footnotes refer to the German edition of the mentioned books and essays except those, where no translation into English existed at the time of the authoring of the essay.*)

Aristoteles [1991]: *Metaphysik*. Annotated Greek-German edition in two volumes. Felix Meiner Verlag, Hamburg 1991

Aristoteles [1995]: *Philosophische Schriften*. Six volumes. Felix Meiner Verlag, Hamburg 1995

Cauley, Kathleen M. [1986]: *Studying Knowledge Acquisition: Distinctions among Procedural, Conceptual and Logical Knowledge.* Paper presented at the 67th Annual Meeting of the American Educational Research Association (San Francisco, CA, 16.-20. April 1986), im Internet abrufbar unter: https://files.eric.ed.gov/fulltext/ED278682.pdf (last access: 24.11.2017)

Dewey, James [2002]: *Logik. Die Theorie der Forschung*. Aus dem Englischen von Martin Suhr. Suhrkamp Verlag, Frankfurt am Main 2002. Title of the English original edition: *Logic. The Theory of Inquiry.* Southern Illionois University Press, Carbondale 1986.

Harel, David / Kozen, Dexter [1982]: *Process Logic: Expressiveness, Decidability, Completeness*. In: Journal of Computer and System Sciences, Vol. 25 Nr. 2 (1982), S. 144-180.

Knijnenburg,, P.M.W. / Leeuwen, J. van / Parikh, Rohit [1991]: *On models for propositional dynamic logic*. In: Theoretical Computer Science 91 (1991), S. 181-203.

Kutschera, Franz v. / Breitkopf, Alfred [1971]: *Einführung in die moderne Logik*. Reihe Kolleg Philosophie, Alber Verlag, Freiburg ^{5}1985

Priest, Graham [2001]: *An Introduction to Non-Classical Logic. From If to Is.* Cambridge University Press, Cambridge (UK) ^{2}2008.

Schneider, Ulrich Johannes [2001]: *Foucaults Analyse der Wahrheitsproduktion.* In: Französische Nachkriegsphilosophie: Autoren und Positionen, Hg.: Günter Abel, Berlin Verlag Arno Spitz GmbH / Nomos Verlagsgesellschaft, Berlin 2001.

Sohst, Wolfgang [2016]: *Reale Möglichkeit. Eine allgemeine Theorie des Werdens.* Xenomoi Verlag, Berlin 2016.

Stadler, M.A. [1989]. *On Learning Complex Procedural Knowledge*. Journal of Experimental Psychology: Learning, Memory, and Cognition, Vol.15, No. 6, S. 1061-1069.

Tarski, Alfred [1944]: *The Semantic Conception of Truth and the Foundation of Semantics*. In: Philosophy and Phenomenological Research No. 4 (1944), online accessible at: http://www.ditext.com/tarski/tarski.html

Troquard, Nicolas / Balbiani, Philippe [2015]: *Propositional Dynamic Logic*, in: The Stanford Encyclopedia of Philosophy (Spring 2015 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/spr2015/entries/logic-dynamic/>.

Wittgenstein, Ludwig [1993]: *Werkausgabe.* In acht Bänden. Reihe Suhrkamp Taschenbuch Wissenschaft, Bd. 501ff., Suhrkamp Verlag, Frankfurt am Main ^{9}1993.

Wunsch, Gerhard [2000]: *Grundlagen der Prozesstheorie. Struktur und Verhalten dynamischer Systeme in Technik und Naturwissenschaft*. B.G. Teubner, Stuttgart 2000.

Appendix: List of symbols used in the GPL

General symbols:

P[*x*]	Object with latent action potency
⇒	positive action operator (double-digit)
⇏	negative action operator (double-digit)
⇐	positive action operator (one-digit)
⇍	negative action operator (one-digit)
…	Continuity symbol
↗	positive availability operator (one-digit, i.e. „creation“)
↘	negative availability operator (one-digit, i.e. „demise“)

Relationale Indikatoren:

$\Rightarrow_{R+/-}$	Subscript for the positive or negative reflexivity
$\Rightarrow_{K}$	Subscript for the Commutativity
$\Rightarrow_{D}$	Subscript for the Distributivity
$\Rightarrow_{-T}$	Subscript for the negation of Transitivity
$\Rightarrow_{-(<Kennzeichner>)}$	for the suspension of a relational labeling.

Fusion / Split-uo:

⊕	Fusion operator
∅	Split-up operator

Koppelungszeichen:

$\|^{P}$	Change of the formal language to predicate logic
$\|^{GPL}$	Change of the formal language to GPL