Three approaches to congruence

Sacha Bourgeois-Gironde, Paul-André Melliès

Three approaches to congruence

Paul-André Melliès
University of Edinburgh

Sacha Bourgeois
Université de Nantes

The extension by Frege of *reference* from names to sentences is guided by an argument of substitutivity explicited in [1]:

> If we now replace one word of the sentence by another having the same reference, but a different sense, this can have no bearing upon the reference of the sentence.

The argument can be formalised as follows. Take a language L consisting of names $a, b, c, ...$ and sentences $p, q, ...$ It is always possible to remove a name a from a sentence p where it occurs, so that a *sentence with a hole* $C[-]$ is obtained. Filling $C[-]$ with the name a certainly constructs p again, and we write $p = C[a]$, but filling $C[-]$ with another name b constructs another sentence $C[b]$. Frege's argument asks that the sentences $C[a]$ and $C[b]$ have the same reference whenever the names a and b have the same reference. More formally, if $\approx_R$ is the binary relation relating all names and sentences having the same reference, then $\approx_R$ should verify the property that:

$$a \approx_R b \Rightarrow C[a] \approx_R C[b] \tag{1}$$

In Frege's mind, this property of $\approx_R$ is a sufficient justification that truth value is the reference of direct sentences, see again [1]:

> What else but the truth value could be found, that belongs quite generally to every sentence if the reference of its component is relevant, and remains unchanged by substitutions of the kind in question.

Frege's fruitful approach to reference, names and sentences can be abstracted one step further to the mathematical setting of a couple $(L, \mathcal{O})$ equipped with an equivalence relation $\approx$ on L, where:

1. L is a set of objects,
2. $\mathcal{O}$ is a set of functions from L to L,

We recall that an equivalence relation on L is a binary relation $\approx$ which is reflexive: $[\forall a \in L, a \approx a]$, symmetric: $[\forall (a, b) \in L^2, a \approx b \Rightarrow b \approx a]$, and transitive: $[\forall (a, b, c) \in L^3, (a \approx b \text{ and } b \approx c) \Rightarrow a \approx c]$. An equivalence relation $\approx$ on L is called a *congruence* on $(L, \mathcal{O})$ when $\approx$ is an equivalence relation and, for every a, b in L and ϕ in $\mathcal{O}$:

$$a \approx b \Rightarrow \phi(a) \approx \phi(b) \tag{2}$$

The relation $\approx$ should be considered in that case as a possible notion of *synonymy* on the language L with the constructors $\phi \in \mathcal{O}$. In Frege's construction, L is the set of names and sentences, $\mathcal{O}$ is the set of sentences with a hole, and $\approx$ is the binary relation

$\approx_R$ between names and sentences with the same reference. Observe that every sentence with a hole $C[-]$ is interpreted in $\mathcal{O}$ as the function from L to L which to the name a associates the sentence $C[a]$ [1].

As advocated by Frege, the *reference* relation $\approx_R$ is a congruence on $(L, \mathcal{O})$ since *having the same reference* is -1- an equivalence relation on L and -2- verifies $a \approx_R b \Rightarrow \phi(a) \approx_R \phi(b)$, at least if we limit $\mathcal{O}$ to the set of *direct* sentences, as explained in [1]. Frege also considers *sense* (of names) and *thoughts* (of sentences) to be substitutive: The relation $\approx_S$ which relates names and sentences with the same sense (thoughts) is also a congruence on $(L, \mathcal{O})$.

What appeals to us in Frege's approach to the problem of reference (and sense) is that he applies a formal argument: the congruency criterion on $\approx_R$ (and $\approx_S$), to solve his specific philosophical problem: How can I extract a notion of reference (and sense) on the language L itself from the knowledge of reference $\approx_R$ (and sense $\approx_S$) on names only.

This paper is therefore concerned with three particular methodologies to construct a congruence in a language $(L, \mathcal{O})$, among which Frege's specific solution stands. The first construction is derived from Leibniz principle that indiscernible objects should be identified. The second construction is apparented to Frege's solution by its use of a referential model. The third construction is our own contribution to this problem in the framework of rewriting systems, see [2] for further references.

1 The Leibniz principle

The simplest methodology to build a congruence on a language $(L, \mathcal{O})$ is a variation on Leibnitz principle: *Eadem sunt, quae sibi mutuo substitui possunt, salva veritate.*

In that prospect, the relation $\approx$ is defined as the least binary relation on L such that $a \approx b$ when [for every ϕ in $\mathcal{O}$, $\phi(a) = \phi(b)$]. In fact, every ϕ in $\mathcal{O}$ can be considered as an *observation* on elements of L, and thence $a \approx b$ means that the two elements a and b are *indiscernible* by the observations ϕ. To prove that $\approx$ is a congruence proceeds in two steps: First, show that $\approx$ is an equivalence relation because equality $=$ itself is an equivalence relation on L ; Then, observe that $a \approx b$ implies $\phi(a) = \phi(b)$ whenever $a, b \in L$ and $\phi \in \mathcal{O}$ and therefore $\phi(a) \approx \phi(b)$ (because $\approx$ is reflexive).

The same methodology applies if an equivalence relation $\sim$ replaces $=$. The relation $\approx$ is then defined as the least relation such that $a \approx b$ whenever $\phi(a) \sim \phi(b)$ for every ϕ in $\mathcal{O}$, and $\approx$ is shown to be a congruence. This construction is called the Morris style construction in the λ-calculus and Process Algebra community.

Observe that in many languages $(L, \mathcal{O})$ the object $\phi(a)$ is of higher complexity than a. For instance, in Frege's system, $\phi(a)$ is a sentence when a is a name. From that complexity prospect, the Leibnitz principle derives the meaning of simple objects a (how $\approx$ relates them) from the meaning of complex objects $\phi(a)$ (how $\sim$ relates them). Consequently,

[1] For simplicity, we prefer to neglect types in the definition of $(L, \mathcal{O})$ and replace them by adapted conventions. In Frege's case, we enforce that every sentence with a hole $C[-]$ associates to any sentence p the (conventional) sentence "this construction is not valid". Hence, every function in $\mathcal{O}$ is total.

Frege will apply another methodology to induce the reference $\approx_R$ on sentences (complex objects) from the reference $\approx_R$ on names (simple objects).

2 Frege's referential methodology

A model (W, Ψ) for a language $(L, \mathcal{O})$ is a set W and a function Ψ from L to W. So, every object a in L is associated to a *reference* $\Psi(a)$ in the model W. In that prospect, the *referential* choice for $\approx$ is to relate objects a and b with the same reference, hence:

$$\forall (a, b) \in L^2, \quad a \approx b \Leftrightarrow \Psi(a) = \Psi(b)$$

However, it is not always true that $\approx$ is a congruence. It depends on $\mathcal{O}$. In fact, suppose that there is a function ϕ in $\mathcal{O}$ such that two objects a and b with same reference, $\Psi(a) = \Psi(b)$, have images $\phi(a)$ and $\phi(b)$ with different references: $\Psi(\phi(a)) \neq \Psi(\phi(b))$. In that case, the operator ϕ can distinguish two objects a and b with the same reference: $a \approx b$, so that $\approx$ is not a congruence. Congruence of $\approx$ is a consequence of the following property:

$$\forall (a, b) \in L^2, \forall \phi \in \mathcal{O}, \quad \Psi(a) = \Psi(b) \Rightarrow \Psi(\phi(a)) = \Psi(\phi(b)) \tag{3}$$

In other words, the constructors in $\mathcal{O}$ should always respect the reference function $\Psi : L \to W$.

Let us turn back to Frege and explain how he attributes a reference to sentences. At the first construction step, only names are assigned a reference: if N is the subset of names in L and W is a (big enough) set consisting of *world* objects, the reference function Ψ goes from N to W. Because it is limited to names, Ψ is only a *partial* function from L to W. To improve the situation, Frege decides to extend his ontology W to a set W^* consisting of W and the truth values **true** and **false**. Then, the function $\Psi : N \to W$ can be extended to a *total* function $\Psi^* : L \to W^*$ in such a way that (W^*, Ψ^*) is a model of $(L, \mathcal{O})$. It is remarkable that Frege uses equation (3) to justify his choice of **true** and **false** as sentence references. We repeat the quotation here:

> What else but the truth value could be found, that belongs quite generally to every sentence if the reference of its component is relevant, and remains unchanged by substitutions of the kind in question.

In fact, as is explained hereabove, the equation (3) implies that reference, or $\approx_R$, is a congruence on $(L, \mathcal{O})$.

3 The pragmatist approach

Natural languages are so interwoven with concepts that it is often wiser to experiment an idea on a specified and well-bracketed region of the language and subsequently, if everything works properly there, to extend the experimentation to wider and less understood fragments of the language. This methodology is recommended if we want to understand the relationship between meaning on one hand and use on the other hand.

For this reason, we reorient our investigations and consider a particular *formal* language called the Calculus of Communicating System (CCS) and introduced by Milner in 1980 to describe parallel computations (processes) interacting in a distributed network. We choose this language for simplicity but the methodology we develop is generic and applies on many other languages — see for instance the treatment of multiplicative linear logic [5] operated by the first author in [2].

CCS (statics). We suppose a set A of *tokens* $a, b, c, ...$ and $\bar{a}, \bar{b}, \bar{c}, ...$ among which the token τ plays the special role of a silent action. A *process* P in CCS is recursively defined as either -1- the empty process **nil**, -2- the sequential composition $a.P$ or $\bar{a}.P$ or $\tau.P$ of a token and a process, -3- the composition $P|Q$ of two processes, -4- the summation $P+Q$ of two processes. The process $a.P$ can be interpreted informally as "emit a and perform P", $\bar{a}.P$ as "receive a and perform P", $P|Q$ as "perform P and Q in parallel", and $P+Q$ as "perform one process between P and Q".

CCS (dynamics). A process P evolves in time by emitting a token: $P \stackrel{a}{\to} Q$, receiving a token: $P \stackrel{\bar{a}}{\to} Q$, or performing an *internal* transition: $P \stackrel{\tau}{\to} Q$. The formal definition of $\stackrel{x}{\to}$ for $x = a$, $x = \bar{a}$ or $x = \tau$ is by structural induction (induction on the structure of the processes): First of all -1- $x.P \stackrel{x}{\to} P$ indistinctly for $x = a$, $x = \bar{a}$ and $x = \tau$, -2- if $P \stackrel{a}{\to} P'$ and $Q \stackrel{\bar{a}}{\to} Q'$, then $P|Q \stackrel{\tau}{\to} P'|Q'$, -3- if $P \stackrel{a}{\to} P'$, then $P + Q \stackrel{a}{\to} P'$. The transition $P \stackrel{x}{\to} Q$ is called *external* when x is not the silent token τ.
We clarify the definitions with four examples. The process $P_1 = a.\mathbf{nil}$ emits a and then stops:

$$P_1 \stackrel{a}{\to} \mathbf{nil}$$

The process $P_2 = (\bar{b}.\mathbf{nil} + c.\mathbf{nil})$ either receives b or emits c, and then stops:

$$P_2 \stackrel{\bar{b}}{\to} \mathbf{nil}$$

or

$$P_2 \stackrel{c}{\to} \mathbf{nil}$$

The process $P_3 = a.(\bar{b}.\mathbf{nil} + c.\mathbf{nil})$ emits a, then either receives b or emits c, then stops:

$$P_3 \stackrel{a}{\to} P_2 \stackrel{\bar{b}}{\to} \mathbf{nil}$$

or

$$P_3 \stackrel{a}{\to} P_2 \stackrel{c}{\to} \mathbf{nil}$$

If $Q = b.\mathbf{nil}$, then Q interacts with P_3 during the following computation:

$$(P_3|Q) \stackrel{a}{\to} (P_2|Q) \stackrel{\tau}{\to} \mathbf{nil}|\mathbf{nil}$$

where Q emits and P_2 receives simultaneously the token b.

Bisimulation. One fundamental motivation of Milner is to define the meaning of a process P from the way it interacts with its environment. Milner delivers his basic intuition in [4]:

> "The meaning of a program [= process] should express its history of access to resources which are not local to it"

By access to non local resources, Milner means the *external* transitions $P \xrightarrow{a} Q$ or $P \xrightarrow{\bar{a}} Q$ because these transitions make a process P interact with independent processes. Milner's genius is to introduce an elegant treatment of history of access with the notion of *bisimulation* we define now. Let us write $P \stackrel{\tau}{\Rightarrow} Q$ when P performs a sequence of internal transitions to Q, $P \xrightarrow{\tau} \cdots \xrightarrow{\tau} Q$, and $P \stackrel{x}{\Rightarrow} Q$ when $P \stackrel{\tau}{\Rightarrow} P' \xrightarrow{x} Q' \stackrel{\tau}{\Rightarrow} Q$ for some processes P' and Q'.

A *bisimulation* $\mathbf{B}$ is a binary relation on processes such that for every token x:

- whenever $P \stackrel{x}{\Rightarrow} P'$ and $P\mathbf{B}Q$, there exists a process Q' such that $Q \stackrel{x}{\Rightarrow} Q'$ and $P'\mathbf{B}Q'$,
- whenever $Q \stackrel{x}{\Rightarrow} Q'$ and $P\mathbf{B}Q$, there exists a process P' such that $P \stackrel{x}{\Rightarrow} P'$ and $P'\mathbf{B}Q'$.

Two processes P and Q are said *bisimilar* when there exists a bisimulation $\mathbf{B}$ such that $P\mathbf{B}Q$. We write $P \approx_B Q$.

The relation $\approx_B$ formalises Milner's intuition on *meanings*. Several important equalities are verified for every processes P, Q, R, in particular

$$P + Q \approx_B Q + P$$

and

$$(P + Q) + R \approx_B P + (Q + R)$$

and

$$P + \mathbf{nil} \approx_B P \approx_B P + P$$

But $\approx_B$ also establishes subtle distinctions between processes, for instance that the equivalence $a.(P + Q) \approx_B (a.P + a.Q)$ does not hold when $P = b.\mathbf{nil}$ and $Q = c.\mathbf{nil}$. Let us explain why. Formally, $a.(P+Q) \stackrel{a}{\Rightarrow} P+Q$ whereas $(a.P+a.Q) \stackrel{a}{\Rightarrow} P$ or $(a.P+a.Q) \stackrel{a}{\Rightarrow} Q$. If there were a bisimulation $\mathbf{B}$ to relate $a.(P + Q)$ and $a.P + a.Q$, this bisimulation $\mathbf{B}$ would also relate $P + Q$ to P or to Q (by definition of a bisimulation $\mathbf{B}$). But this is impossible because no bisimulation $\mathbf{B}$ relates $b.\mathbf{nil} + c.\mathbf{nil}$ to $b.\mathbf{nil}$, or relates $b.\mathbf{nil} + c.\mathbf{nil}$ to $c.\mathbf{nil}$ (easy to check). We conclude that $a.(P + Q)$ and $a.P + a.Q$ are *not* bisimilar: $a.(P + Q) \not\approx_B a.P + a.Q$. Semantically, the distinction is justified because the process $a.(P + Q)$ retains the ability to choose between P and Q after the emission of a, whereas $a.P + a.Q$ makes the choice when it emits a.

Non-congruence. One drawback of Milner's approach is that, surprisingly, $\approx_B$ is not a congruence. We explain why. Take $P = a.\mathbf{nil}$ and $Q = b.\mathbf{nil}$. The two processes P and $\tau.P$ are bisimilar [2] but not $P + Q$ and $(\tau.P) + Q$. In fact, it is possible that $(\tau.P) + Q$ transits *internally* to P: $(\tau.P) + Q \stackrel{\tau}{\Rightarrow} P$ without any corresponding internal transition from $P + Q$ to a process bisimilar to P. Thus, $P \approx_B \tau.P$ but $P + Q \not\approx_B (\tau.P) + Q$.

[2] Because $P \stackrel{\tau}{\Rightarrow} P$ and $\tau.P \stackrel{\tau}{\Rightarrow} P$.

Milner's bisimulation $\approx_B$ seems a perfectly correct notion of meaning, but it is not. So, where is the defect?

Canonical language. We need to be more precise when we say that $\approx_B$ is not a congruence. Here, we give the *canonical* language $(L, \mathcal{O})$ of CCS we consider:

1. L is the set of processes of CCS,
2. to every process P in L is associated a function $\phi_P : L \to L$ (resp. ψ_P) which to every process Q associates the process $P|Q$ (resp. $P + Q$),
3. $\mathcal{O}$ is the set of all ϕ_P's and ψ_P's for processes P in L.

The equivalence relation $\approx_B$ is not congruent on $(L, \mathcal{O})$ because $P \approx_B \tau.P$ holds but not $\psi_Q(P) \approx_B \psi_Q(\tau.P)$ when $P = a.\mathbf{nil}$ and $Q = b.\mathbf{nil}$.

Strong bisimulation. Milner's $\approx_B$ interprets P and $\tau.P$ as synonyms because the two processes have the same history of accesses to non local resources (see Milner's quotation). However, P and $\tau.P$ should be distinguished if we think of their symbolic behaviour inside a sentence: the sign τ in the sentence $\psi_Q(\tau.P)$ interacts with the sign $+$ so that $\psi_Q(\tau.P)$ transits (internally) to P: $\psi_Q(\tau.P) \stackrel{\tau}{\Rightarrow} P$, but this specific interaction is impossible from $\psi_Q(P)$. Thus, Milner violates CCS computations when he interprets the token τ as *silent*. The token τ cannot be silent and innocent as soon as it combines and interacts with other signs, for instance $(\tau.P) + Q \stackrel{\tau}{\to} P$.

The simplest way to repair $\approx_B$ and obtain a congruence is to trace the τ's in the semantics. A *strong* bisimulation $\mathbf{C}$ is a relation such that for every token x:

- whenever $P \stackrel{x}{\to} P'$ and $P\mathbf{C}Q$, there exists a process Q' such that $Q \stackrel{x}{\to} Q'$ and $P'\mathbf{C}Q'$,
- whenever $Q \stackrel{x}{\to} Q'$ and $P\mathbf{C}Q$, there exists a process P' such that $P \stackrel{x}{\to} P'$ and $P'\mathbf{C}Q'$.

Two processes P and Q are said *strongly bisimilar*, $P \approx_C Q$, when there exists a bisimulation $\mathbf{C}$ such that $P\mathbf{C}Q$.

The pragmatist placard. As could be anticipated from our development, $\approx_C$ is a congruence [3] in $(L, \mathcal{O})$. Weaponed by this positive result, [2] sharpens Milner's principle into the following placard:

> "The meaning of a program [= process] should express the history of its symbolic interactions within the sentence, in particular with the constructors of $\mathcal{O}$"

This definition of meanings is radical in that it does not project any extraneous (human) interpretation into the raw mechanic life of symbolic computation. Whenever the formal

[3] In fact, Milner himself introduces the strong bisimulation in [4] and proves the congruence result on $\approx_C$. However, his motivations are quite different.

language is based on symbolic terms [= processes in CCS] and rewriting transformations [= transitions], meaning becomes a consequence of Rewriting itself, and the synonymy relation $\approx_C$ which describes the symbolic interactions is always a congruence, see [2] for formal details.

This purely symbolic approach to meanings is so convincing that we could hardly be the first to discover it. Indeed, we find some anteriority in Peirce's description of *meanings* of thoughts as the *habits* these thoughts produce, see [5].

> "To develop its meaning, we have, therefore, simply to determine what habits it [= the thought] produces, for what a thing means is simply what habits it involves."

As a matter of fact, what else but a term is a symbolic thought? and what else but the interaction is the habit this symbolic term involves?

4 Conclusion

The article classifies three kinds of congruence constructions on a language $(L, \mathcal{O})$. Other constructions exist in the litterature, and this taxonomy should be carried on. In particular, we see our third construction as a radical interpretation of Peirce's description of meanings — an interpretation which we hope will revitalise this fascinating approach to language and meaning.

References

[1] Gottlob Frege, **On sense and reference**, in *Translations from the Philosophical Writings of Gottlob Frege*, ed. Peter Geach and Max Black, Oxford Blackwell, 1952.

[2] Paul-André Melliès, **A double category for multiplicative linear logic**, presented at the conference on Logic and Models of Computation in Marseille, September 1996. Submitted to publication. Available at **http://www.dcs.ed.ac.uk/home/paulm/**

[3] John Morris, **Lambda-calculus models of programming languages**, Ph.D Thesis, MIT, Cambridge, 1968.

[4] Robin Milner, **A Calculus of Communicating Systems**, Lecture Notes in Computer Science 92, Springer Verlag, 1980.

[5] Jean-Yves Girard, **Linear logic**, in *Theoretical Computer Science* 50, 1987.

[4] Robin Milner, **Processes: a mathematical model of computing agents**, In *Logic Colloquium '73*, pages 157-173. North Holland, 1975.

[5] Charles Sanders Peirce, **How to make our ideas clear**, in *Collected Papers*, Cambridge, Mass., Harvard University Press.