Published in Igor Sedlár and Martin Blicha (eds.). *The Logica Yearbook 2018*. London: College Publications, 2019, pp. 197-210. ISBN 978-1-84890-307-4.

Non-constructive procedural theory of propositional problems and the equivalence of solutions

IVO PEZLAR[1]

Abstract: We approach the topic of solution equivalence of propositional problems from the perspective of non-constructive procedural theory of problems based on Transparent Intensional Logic (TIL). The answer we put forward is that two solutions are equivalent if and only if they have equivalent solution concepts. Solution concepts can be understood as a generalization of the notion of proof objects from the Curry-Howard isomorphism.

Keywords: Transparent Intensional Logic, procedural semantics, logic of problems, procedural isomorphism

1 Introduction and motivation

There can be many different solutions to a single problem. For example, consider the problem whether $\sqrt{2}$ is an irrational number or not: we have geometric solutions, algebraic solutions, constructive solutions, etc.[2] But are all these solutions really different? In some cases, it seems easy to decide, but in other cases, it is not so obvious. Interestingly, these difficulties are not exclusive to relatively advanced mathematical problems and they appear at the level of the simplest logical problems as well.[3] Consider e.g., the following two solutions to the problem $A \to ((A \to B) \to B)$ carried out in a natural deduction system for propositional logic:

[1]Work on this paper was supported by Grant No. 17-18344Y from the Czech Science Foundation, GA ČR.

[2]See e.g., (Harris, 1971).

[3]In the case of propositional problems, solutions can be understood simply as proofs and problems as propositions. However, since this relation does not generally hold for all problems and solutions (e.g., it seems reasonable to say that 12 is a solution to the problem expressed by $5 + 7$, it makes less sense to say that 12 is a proof of $5 + 7$), we keep this terminology.

$$\textbf{Solution A}\quad \dfrac{\dfrac{\dfrac{A \to B \qquad A}{B}\,{\to}\mathrm{E}}{(A \to B) \to B}\,{\to}\mathrm{I}}{A \to ((A \to B) \to B)}\,{\to}\mathrm{I}$$

$$\textbf{Solution B}\quad \dfrac{\dfrac{\dfrac{\dfrac{A \to B \qquad A}{B}\,{\to}\mathrm{E} \qquad A}{B}\,{\to}\mathrm{I}}{(A \to B) \to B}\,{\to}\mathrm{I}}{A \to ((A \to B) \to B)}\,{\to}\mathrm{I}$$

Are these two solutions equivalent? At first glance we would probably say yes. After all, they start with the same premises and end with the same conclusion. But on the other hand, they are also clearly syntactically distinct: one has more steps than the other.

The received view would tell us that these two solutions are equivalent because they can be converted into each other. More specifically, the solution **B** can be reduced to **A** by removing all the 'unnecessary detours' (so-called *normalization* procedure). In this case, it is the first application of the implication introduction rule immediately followed by the application of the corresponding implication elimination rule. If we cut it away from **B**, we get **A**.[4] This approach is also corroborated by the propositions as types principle.[5] If we decorate the above derivations with the corresponding λ-terms (also called *proof objects*), we obtain:

$$\textbf{Solution A1}\quad \dfrac{\dfrac{\dfrac{x : A \to B \qquad y : A}{x(y) : B}\,{\to}\mathrm{E}}{\lambda x.x(y) : (A \to B) \to B}\,{\to}\mathrm{I}}{\lambda y.\lambda x.x(y) : A \to ((A \to B) \to B)}\,{\to}\mathrm{I}$$

$$\textbf{Solution B1}\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x : A \to B \qquad y : A}{x(y) : B}\,{\to}\mathrm{E}}{\lambda y.x(y) : A \to B}\,{\to}\mathrm{I} \qquad y : A}{(\lambda y.x(y))(y) : B}\,{\to}\mathrm{E}}{\lambda x.(\lambda y.x(y))(y) : (A \to B) \to B}\,{\to}\mathrm{I}}{\lambda y.\lambda x.(\lambda y.x(y))(y) : A \to ((A \to B) \to B)}\,{\to}\mathrm{I}$$

The concluding term $\lambda y.\lambda x.(\lambda y.x(y))(y)$ of the second solution can be rewritten into the concluding term $\lambda y.\lambda x.x(y)$ of the first one. More specifically, $\lambda y.\lambda x.(\lambda y.x(y))(y)$ is reducible to $\lambda y.\lambda x.x(y)$ via β-reduction. Hence, these two terms are considered to be equivalent and, consequently, so are the corresponding solutions.[6]

[4] See e.g., (Prawitz, 2006).

[5] Also known as the Curry-Howard isomorphism or correspondence, see (Howard, 1980), (Curry & Feys, 1958).

[6] The equivalence of λ-terms has been thoroughly studied, most notably by Church himself. See e.g., (Church, 1954), (Church, 1993), (Anderson, 1998).

Our approach to solution equivalence based on Transparent Intensional Logic will also utilize λ-calculus, however, it will allow for partial functions and have a much more semantic flavor. For us, two solutions will be considered equivalent if and only if they have equivalent *solution concepts*. Intuitively, solution concepts can be understood as reified methods or procedures for solving problems. From a technical standpoint, they can be regarded as abstract generalizations of proof objects that need not be effective, i.e., constructive in the intuitionistic sense. This will enable us to analyze even incorrect solutions.[7]

2 TIL: the basics

Transparent Intensional Logic (TIL)[8] is a many-sorted type theory with hyperintensional semantics initially devised for natural language analysis. Similarly to Montague semantics, it utilizes λ-calculus but makes room for partial functions. The central notion of TIL is a construction, which can be understood as a reified abstract algorithm.[9] Constructions are assigned to linguistic expressions as meanings and encode procedures for determining their denotations. For example, propositions are understood as procedures for computing truth values (see also e.g., Jespersen, 2017, Muskens, 2005, Moschovakis, 2006).[10]

TIL typically relies on six fundamental kinds of constructions, however, we will use a generalized presentation requiring only four constructions:

$$\textit{constructions} \;:=\; x^{\alpha} \;\mid\; [C^{\alpha} C_1^{\beta_1} \dots C_m^{\beta_m}] \;\mid\; [\lambda x_1^{\beta_1} \dots x_m^{\beta_m} C^{\alpha}] \;\mid\; {}^{n}X^{\alpha}$$

where x is a variable, C_i is any construction, X is either a construction or a non-construction (i.e., an object that is not a construction, e.g., truth value, individual, number, etc.), and α, β_i are type metavariables.[11] The first three constructions are called *variable*, *composition*, and *closure* and they roughly

[7] We will leave the notion of incorrect solutions intentionally vague to not limit ourselves to some specific conception of solution correctness.

[8] See (Tichý, 1988), (Duží, Jespersen, & Materna, 2010), (Raclavský, Kuchyňka, & Pezlar, 2015).

[9] Not to confuse with intuitionistic constructions, which are essentially effective proofs.

[10] We will not go through the specifics of TIL-based analysis of natural language here, since it has been well covered in many other places already. See e.g., see (Duží et al., 2010), (Raclavský et al., 2015).

[11] The notation X^{α} means that X is an object of type α if X is a non-construction, otherwise it means that X is a construction typed to v-construct an object of type α. We will omit the type letters 'α', 'β', ... to simplify the notation. For a proper specification of constructions, see

correspond to variable, function application, and function abstraction from λ-calculus.[12] The last one, called n-*execution* (alternatively, n-stage execution or multiple execution), is a distinctive feature of TIL: given the procedural nature of constructions, this construction allows us to either execute them (possibly more than once) to find out what they construct, i.e., what is their output (if $n > 1$), or leave them as they are, i.e., don't execute them (if $n = 0$). What particular objects constructions construct may depend on a *valuation* v, i.e., an assignment of values to free variables. If that is the case, we say that they v-*construct* those objects.[13] For example, let us have some variable construction x and some valuation v that assigns to it some object X, then we can say that the construction x v-constructs X. If a construction C v-constructs nothing at all, we will say that it is a v-*improper* construction. Otherwise, we say that C is a v-*proper* construction. If we have two constructions C_1 and C_2 that v-construct the same object X, or they are both v-improper, we will say that C_1 and C_2 are v-*congruent* constructions, denoted as $C_1 \cong C_2$. If they are v-congruent for all valuations v, we will say that C_1 and C_2 are *equivalent*, denoted as $C_1 = C_2$.

For example, the 0-execution construction ${}^0 12$ yields the number 12. Intuitively, '${}^0 12$' can be read as 'do not execute 12, just refer to it'. And we don't want to execute 12, because it is not a construction but a number, i.e., a non-construction.[14] By definition, the 1-execution construction ${}^1 12$ constructs nothing, i.e., it is an improper construction (we cannot execute non-constructions, only constructions). Analogously, the composition construction $[{}^0+ \ {}^0 5 \ {}^0 7]$ constructs 12 and so does e.g., the 1-execution ${}^1[{}^0+ \ {}^0 5 \ {}^0 7]$. Hence, we can say that they are congruent, and even equivalent constructions. On the other hand, the 0-execution ${}^0[{}^0+ \ {}^0 5 \ {}^0 7]$ constructs $[{}^0+ \ {}^0 5 \ {}^0 7]$, but e.g., 2-execution ${}^2[{}^0+ \ {}^0 5 \ {}^0 7]$ is again an improper construction (2-execution is essentially a two stage execution: the first stage gives us 12 and the second stage consists of its 1-execution ${}^1 12$. However,

(Tichý, 1988), (Duží et al., 2010) or (Raclavský et al., 2015). For a specification of n-execution, see (Pezlar, 2018).

[12] Although in many situations λ-terms and constructions can behave in similar fashion (e.g., both are open to α-, β-, η-conversions – this will be important later in Section 3.1), there are non-trivial conceptual as well as technical distinctions. Most importantly, constructions are not terms but abstract objects that can be executed to yield some other objects.

[13] To be more precise, constructions always construct with respect to some valuation v. In some cases, however, valuation does not affect on the overall result of the construction. If that is the case, we will simply speak of *constructing* instead of v-*constructing*.

[14] Note that 0-execution can play roughly the same role as do constants in impure/applied λ-calculus.

as we already know, this is an improper construction, hence the whole construction is improper).

To simplify the notation, we will denote 0-execution by **boldface** font, with the exception of standard connectives and operators such as $+$, $=$, $\forall$, $\rightarrow$, etc. that will be kept in normal font with 0-execution implicitly assumed. Also we will use infix notation whenever anticipated. For example, we will write $[\mathbf{5} + \mathbf{7}]$ instead of $[^0+\ ^0 5\ ^0 7]$ and $[A \supset B]$ instead of $[^0\supset\ A\ B]$.

All objects including constructions receive a type. If α and $\beta_1, \ldots, \beta_m$ are types, then $(\alpha\beta_1 \ldots \beta_m)$ is also a type. Specifically, a type of function from the elements of type $\beta_1, \ldots, \beta_m$ to the elements of type α. For example, the construction $[^0+\ ^0 5\ ^0 7]$ has type $*_1$ (so-called 1st-order construction),[15] while $^0[^0+\ ^0 5\ ^0 7]$ has type $*_2$ (2nd-order construction). On the other hand, non-constructions $+$, 5, and 7 have types $(\nu\nu\nu)$, ν and ν, respectively, where ν is the type of natural numbers.

2.1 Solution assignment

In (Pezlar, 2017) it was shown that the previous TIL-based non-constructive procedural approach to analysis of logical problems (see e.g., Materna, 2004, Materna, 2008) is too coarse-grained because it renders every solution to every logical problem as equivalent. The same paper tries to alleviate this issue by introducing a new notion called *solution constructor* that helps us to track the process of solution construction in a similar way as we would do with the Curry-Howard correspondence (free variables corresponding to assumptions, etc.).[16] The solution constructor can be used to analyze correct solutions, however, here we generalize it further into so-called *solution assignment* that can analyze any solution, both correct and incorrect. Our rationale is the following: dealing with incorrect solutions—or more precisely, with solutions that are believed to be correct, but later are shown to be incorrect—is a natural aspect of problem-solving (similarly as is e.g., debugging in programming) and by limiting our framework to correct solutions only we are unnecessarily restricting its expressive power.

[15] More specifically, the construction $[^0+\ ^0 5\ ^0 7]$ has type $*_1$ as its lowest possible order of type. Due to TIL's cumulative hierarchy of types, it is also of type $*_2$, $*_3$, ..., etc. To put it simply, any construction of type $*_n$ is also a construction of type $*_{n+1}$. For more about the ramified hierarchy of types in TIL, see (Tichý, 1988).

[16] Conceptually, the main change is that instead of λ-terms and propositions (as their types) we work with higher- and lower-order constructions, hence proof objects are considered as higher-order objects than the things they are proving. Technically, there are some slight differences induced by TIL idiosyncrasies. More on this below.

Definition 1 (**Solution assignment**) *A solution assignment is a couple* $c :: C$ *where* C *is a propositional construction (a problem to be solved) and* c *is any construction* v*-constructing a construction of the same type as* C *(if any). A solution assignment* $c :: C$ *is said to be* satisfied *if* c v*-constructs* C*. Otherwise, we say that it is* unsatisfied*.*

Note that if c in $c :: C$ is an improper construction, then the whole solution assignment is unsatisfied (follows from Def. 1 and the definition of improper constructions). Also note that a solution assignment $c :: C$ can be satisfied even though C itself is an improper construction. Conveniently, this gives us a way to analyze situations when an agent tries to solve nonsensical, or more precisely, truth-valueless problems such as e.g., 'the largest prime number ends with the number 7', '3 divided by 0 equals 2', etc. Furthermore, solution assignments allow even 'bad' solutions such as solving C by considering some trivial construction of C (e.g., the 0-execution of C). Informally, this sort of degenerate solutions can be liken to the colloquial 'it is solved because I said so' method. Shortly put, with solution assignments we just check if c yields C, not whether c is also a good solution of C. Hence, $c :: C$ should not be read as 'c solves C' or 'c is a solution to C' but rather as 'c is an assumed solution of C', 'c is considered as a solution to C' or 'c solves C, allegedly' and similarly with the option that it might be false.[17] The separation of the good solutions from the bad ones will be dealt with at the level of solutions concepts.[18]

Definition 2 (**Solution concept**) *Let us have a solution assignment* $c :: C$*, then we will say that* c *is a* solution concept *of the problem* C*. If* c *solves* C*,*[19] *then we will say that* c *is a suitable solution concept. Otherwise, we will say that* c *is an unsuitable solution concept.*

At first glance, the notion of solution concept might seem enigmatic. There is, however, a simple idea behind it: it is a procedure, not necessarily

[17] We could restrict this condition and require that for every solution assignment $c :: C$ holds that c solves C. For example, Constructive Type Theory (see Martin-Löf, 1984) can be seen as a system where this restriction holds, i.e., every solution must be effective.

[18] At the LOGICA 2018 conference, prof. Duží posed the following question (my paraphrase): Cannot we avoid the necessity for the solution assignment, and thus for the use of higher-order constructions, simply by invoking the notion of refinement of constructions as defined in Duží et al. (2010), Chapter 5? Unfortunately, the answer is no. Although we could refine the logical connectives used, it would not help us to track the actual solution construction, which is our main objective.

[19] Or maybe more precisely, if c can be recognized as a solution of C. See e.g., (Wittgenstein, 1922), 6.2321 or (Wittgenstein, 1978), II-42.

effective, that represents solution construction – recall the dictum proofs-as-programs from the Curry-Howard isomorphism. To put it differently, we can think of solution concepts as abstract generalizations of proof objects from general proof theory. In contrast to them, solution concepts can be applied even to non-logical problems (although in this paper, we focus only on the logical ones) and can be defective. From this point of view, proof objects can be understood as a special case of solution concepts that are always effective.[20] Compare this with e.g., Constructive Type Theory (CTT), where every proof object must be effective, i.e., terminate in a finite amount of steps, which is not the case for solution concepts. On the other hand, similarly to proof objects in CTT, solution concepts are proper objects of TIL: they receive types, we can use them as arguments for higher-order functions, etc. (this will be important later in Section 3.1, where we discuss equivalence of solution concepts).

Note that the adoption of solution concepts helps us to explain why can we 'comprehend' even incorrect solutions, i.e., follow them, find mistakes in them, etc. From our perspective, they are not meaningless or nonsensical. They have an idea—some solution concept—behind them, it just happens to be an ineffective one that does not do the required job, i.e., a solution concept that does not solve the problem at hand.

As we already mentioned, solution assignments are used to emulate in TIL the solution tracking behaviour of the Curry-Howard isomorphism. We demonstrate this by formulating rules for implication, conjunction, disjunction, and negation capable of recording the solution forming process. We start with the implication introduction rule. Suppose that A and B represent any propositional constructions (i.e., constructions v-constructing the truth values $true$ or $false$ of type o), x and y are variables ranging over the objects of type $*_n$ (i.e., variables for constructions), and let $\supset$ be implication of type (ooo). Next, we establish premises for our implication introduction rule. Suppose that we have some solution assignment $x :: A$ such that x is of type $*_{n+1}$ and that it v-constructs A of type $*_n$. Further, assume we derive from it another solution assignment $y :: B$ such that y is of type $*_{n+1}$ and that it v-constructs B of type $*_n$ (thus, both are satisfied solution assignments). So we have:

[20] Historically, solution concepts can be loosely regarded as an explication of Wittgenstein's approach: 'The idea that proof creates a new concept might be also roughly put as follows: a proof is not its foundations plus the rules of inference, but a *new* building [= our solution concepts]...' (Wittgenstein, 1978), II-41.

$$\begin{array}{c}(x :: A)\\ \vdots\\ y :: B\end{array}$$

Now, we introduce the function ii of type $(*_n(*_n*_n))$ that will imitate the behaviour of the implication introduction rule. More concretely, ii is a higher-order function that takes a function—representing an inference from A to B—v-constructed by $[\lambda x\, y]$ and outputs a construction representing a conclusion of this derivation, otherwise it is undefined. If we put it together, we get the construction $[\mathbf{ii}\, [\lambda x\, y]]$ that v-constructs $[A \supset B]$. The corresponding rule $\supset\mathbf{I}$ then looks as follows:[21]

$$\frac{\begin{array}{c}(x :: A)\\ \vdots\\ y :: B\end{array}}{[\mathbf{ii}\ \lambda x\, y] :: [A \supset B]}\ \supset\mathbf{I}$$

Now, we move to the implication elimination rule. We start again by establishing the premises. Suppose we have two solution assignments $x :: [A \supset B]$ and $y :: A$ such that x and y are of type $*_{n+1}$ and that they v-construct $[A \supset B]$ and A, both of types $*_n$, respectively (i.e., they are satisfied solution assignments). Thus we have:

$$x :: [A \supset B] \qquad y :: A$$

Next, we introduce a higher-order function that will mimic the behaviour the implication elimination rule. Concretely, we introduce the function ie of type $(*_n *_n *_n)$ that takes two constructions of the form $[A \supset B]$ and A as arguments and outputs another construction of the form B, otherwise it is undefined. The resulting construction $[\mathbf{ie}\ x\ y]$ then v-constructs B. As the corresponding rule $\supset\mathbf{E}$ we get:[22]

$$\frac{x :: [A \supset B] \qquad y :: A}{[\mathbf{ie}\ x\ y] :: B}\ \supset\mathbf{E}$$

[21] We omit the outer brackets of the closure construction when in composition with **ii** or other constant to get a cleaner notation.

[22] Why do we need to work with constructions of the form $[\mathbf{ie}\ x\ y]$ instead of the more traditional form $[x\ y]$ as known from the Curry-Howard correspondence? From a TIL perspective, $[x\ y]$ is a composition construction, which means that its first component (x in this case) has to v-construct a function, otherwise it would be improper. Since in our rule $\supset\mathbf{E}$ the variable x v-constructs $[A \supset B]$, which is not a function, we have to introduce the auxiliary function ie to circumvent this issue. Similarly in the case of **ii**.

To recapitulate, the ii and ie are higher-order functions that capture the behaviour of the corresponding rules. More specifically, ii is a function of type $(*_n(*_n*_n))$ that takes a function (representing the inference of B from A) as an argument and returns a new construction of the form $[A \supset B]$. Analogously, ie is a function of type $(*_n *_n *_n)$ that takes two arguments of the form $[A \supset B]$ and A and returns a new construction of the form B.

The rules for conjunction, disjunction and negation will be presented more succinctly since they follow the same ideas as the rules above and share the general form with their counterparts from the Curry-Howard isomorphism. We start with conjunction:

$$\frac{x :: A \qquad y :: B}{[\mathbf{ci}\ x\ y] :: [A \wedge B]}\ \wedge\mathbf{I} \qquad \frac{z :: [A \wedge B]}{[\mathbf{pr}_l\ z] :: A}\ \wedge\mathbf{E}_l \qquad \frac{z :: [A \wedge B]}{[\mathbf{pr}_r\ z] :: B}\ \wedge\mathbf{E}_r$$

The **ci** constructs the 'conjunction introduction rule' function in a similar way as did **ii** previously for implication introduction rule. The components $\mathbf{pr}_l$ and $\mathbf{pr}_r$ construct the familiar left and right projection functions. Types of all these functions can be easily inferred: **ci** has type $*_{n+1}$ and constructs a higher-order function of type $(*_n *_n *_n)$ and $\mathbf{pr}_l$ and $\mathbf{pr}_r$ are also of type $*_{n+1}$ and construct higher-order functions of type $(*_n*_n)$. Next, we introduce the rules for disjunction:

$$\frac{x :: A}{[\mathbf{j}\ x] :: [A \vee B]}\ \vee\mathbf{I}_l \qquad \frac{y :: B}{[\mathbf{k}\ y] :: [A \vee B]}\ \vee\mathbf{I}_r$$

$$\frac{\begin{array}{ccc} & (x :: A) & (y :: B) \\ & \vdots & \vdots \\ c :: [A \vee B] & d :: C & e :: C \end{array}}{[\mathbf{de}\ c\ \lambda x\ d\ \lambda y\ e] :: C}\ \vee\mathbf{E}$$

The **j** and **k** are constants that tell us from which higher-order construction was the disjunction constructed, the **de** then constructs the 'generalized disjunction elimination rule' function of type $(*_n *_n (*_n*_n)(*_n*_n))$. Finally, the rules for negation are analogous to the rules for implication:

$$\frac{\begin{array}{c}(x :: A) \\ \vdots \\ y :: \bot\end{array}}{[\mathbf{ii}\ \lambda x\ y] :: [\neg A]}\ \neg\mathbf{I} \qquad \frac{y :: [\neg A] \qquad x :: A}{[\mathbf{ie}\ y\ x] :: B}\ \neg\mathbf{E}$$

The construction $[\neg A]$ is equivalent to $[A \supset \bot]$ and $\bot$ is to be understood as a construction of a proposition that is always false.

With inference rules ready, we can now define solutions in a more precise way:

Definition 3 (**Solution**) *A solution (or a solution tree) is a finite sequence of solution assignments $\langle c_1 :: C_1, \ldots, c_n :: C_n, c :: C\rangle$ each of which solution assignment is either an axiom, or an assumption, or follows from the preceding solution assignments in the sequence by some inference rule of the system. The last solution assignment will be also called conclusion.*

A solution tree can be also written vertically as $\dfrac{c_1 :: C_1 \quad \ldots \quad c_n :: C_n}{c :: C}$.

For example, the solution **A** can be analyzed in TIL in the following way:

$$\textbf{Solution A'}\ \dfrac{\dfrac{\dfrac{x :: [A \supset B] \qquad y :: A}{[\mathbf{ie}\ x\ y] :: B}\ \supset\mathbf{E}}{[\mathbf{ii}\ \lambda x\,[\mathbf{ie}\ x\ y]] :: [[A \supset B] \supset B]}\ \supset\mathbf{I}}{[\mathbf{ii}\ \lambda y\,[\mathbf{ii}\ \lambda x\,[\mathbf{ie}\ x\ y]]] :: [A \supset [[A \supset B] \supset B]]}\ \supset\mathbf{I}$$

Note that each solution assignment is satisfied and every corresponding solution concept is suitable.

To incorporate even incorrect solutions, we have to appropriately modify the definition of solution trees. Specifically, by allowing to add to the sequence solution assignments that are neither axioms, assumptions, or follow from the inference rules. As an example of an incorrect solution, or more specifically, of an incorrect solution step, consider the following:

$$\dfrac{\begin{array}{c}A\\ \vdots\\ B\end{array}}{[A \wedge B]}$$

where the conjunction $[A \wedge B]$ is incorrectly inferred instead of the implication $[A \supset B]$. In TIL, we can analyze it as follows:

$$\dfrac{\begin{array}{c}x :: A\\ \vdots\\ y :: B\end{array}}{[\mathbf{ci}\ \lambda x\, y] :: [A \wedge B]}$$

Notice that the concluding solution concept is unsuitable and the solution assignment is unsatisfied: the function constructed by **ci** expects two arguments but receives only one. In other words, the solution concept [**ci** $\lambda x\, y$] is an improper construction. However, it does not mean that [**ci** $\lambda x\, y$] as such becomes nonsensical in TIL. It is still a construction in its own right, even though improper, i.e., we can use it as an object of predication. For example, we can introduce a higher-order unary function Improper of type $(o*_n)$ for checking properness of constructions. If we feed it the construction [**ci** $\lambda x\, y$], i.e., form the composition [**Improper** ^{0}[**ci** $\lambda x\, y$]], we get the value $true$.

3 Solution equivalence

When do we generally consider two solutions as equivalent? Arguably, we would say that two solutions are equivalent when they follow the same general method. As mentioned above, we code this method via higher-order TIL constructions called solution concepts. Thus the issue of solution equivalence is reduced to the question of solution concept equivalence.

Definition 4 (**Solution equivalence**) *Solutions S_1 and S_2 are equivalent if and only if the solution concepts appearing in their conclusions are equivalent.*

Observe that the initial question 'When are two solutions equivalent?' is thus transformed into 'When are two solution concepts used by two different solution trees equivalent?' which will be the focus of the following section.

3.1 Fine-tuning the solution equivalence threshold

Let us return to the solutions **A** and **B**. Applying the approach discussed above, we learn that their respective solution concepts are:

[**ii** λy [**ii** λx [**ie** [**ii** λy [**ie** $x\ y$]] y]]] and [**ii** λy [**ii** λx [**ie** $x\ y$]]]

Solution concepts were designed to emulate proof objects based on λ-terms. Naturally, we can reuse the criteria for equivalence of λ-terms as well with appropriate changes where necessary. The standard conception is to regard λ-terms as equivalent if they are λ-convertible, i.e., α-, η and β-convertible. We can carry over this general approach to TIL as well. However, increased caution is necessary due to the fact that β-conversions and η-conversions

are generally not equivalent transformations in TIL.[23] For these reasons we exploit here the variant of construction equivalence put forward by Duží and Jespersen (2012) called procedural isomorphism alternative ($\frac{3}{4}$) with the difference that we drop η-reduction.[24]

Definition 5 (**Procedural isomorphism**) *Let C and D be constructions. Then C and D are procedurally isomorphic if and only if either C and D are identical (i.e., the same construction) or there are constructions $C_1, \ldots, C_n$ $(n > 1)$ such that ${}^0C = {}^0C_1$, ${}^0D = {}^0C_n$, and for each C_i, $C_{i+1}(1 \leq i < n)$ it holds that C_i, C_{i+1} are either α-equivalent (i.e., they differ only by having different λ-bound variables) or β_r-equivalent (i.e., equivalent via restricted β-conversion by name).*

We prefer this variant of procedural isomorphism to the later one proposed in (Duží & Jespersen, 2015) and dubbed alternative (A1"). Even though it also omits η-conversion, it relies on β-conversion by value which is not a best fit for us – we are not really interested in what the solution concepts construct (i.e., what are their 'values'), but rather in the solution concepts themselves (i.e., in their 'names'). For that reason we choose β_r-conversion, which is essentially just a tool for a formal simplification of constructions (see Duží & Kosterec, 2017) guaranteeing that all the resulting transformations of constructions will be equivalent even in the presence of partial functions and potentially improper constructions and that is all we need for analyzing propositional logical proofs.

We write the fact that two constructions C and D are α-, β_r-equivalent as $C =_\alpha D$ and $C =_{\beta_r} D$, respectively.

Definition 6 (**Equivalence of solution concepts**) *Solution concepts c_1 and c_2 are equivalent if and only if they are procedurally isomorphic. We denote this as $c_1 \equiv c_2$.*

It follows that when we say that two solutions are equivalent it means that their respective solution concepts are procedurally isomorphic. Note that if c is a suitable solution concept for C and $c_1 \equiv c_2$, then c_2 is also a suitable solution concept for C. Also note that if c_1 and c_2 are procedurally isomorphic ($c_1 \equiv c_2$), then c_1 and c_2 are congruent ($c_1 \cong c_2$), but not vice versa.

[23] Or any other logic of partial functions, see e.g., Moggi (1988).

[24] In theory, other alternatives could be chosen as well. For a great survey of possible variants, see (Duží, 2017). Our approach essentially corresponds to Duží's variant **C6**, modulo meaning postulates.

Let us now return to our initial question: are the solutions **A** and **B** equivalent? From the perspective of TIL, they are equivalent because their corresponding solution concepts are procedurally isomorphic.

4 Conclusion

In this paper we expanded upon the non-constructive procedural approach to analysis of problems proposed in Pezlar (2017) by addressing the issue of solution equivalence. In short, we take two solutions as equivalent if and only if their solution concepts are equivalent, i.e., procedurally isomorphic. Solution concepts can be understood as a generalization of the notion of proof objects from the Curry-Howard isomorphism. The main difference is that solution concepts need not be effective, which allows us to analyze even cases involving incorrect solutions. Future work lies mainly in extending our framework towards predicate logic, incorporating mathematical problems, and finally investigating the possibility of expanding our approach towards empirical problems as well.

References

Anderson, C. A. (1998). Alonzo Church's Contributions to Philosophy and Intensional Logic. *Bulletin of Symbolic Logic*, *4*(2), 129–171.

Church, A. (1954). Intensional Isomorphism and Identity of Belief. *Philosophical Studies*, *5*(5), 65–73.

Church, A. (1993). A Revised Formulation of the Logic of Sense and Denotation. Alternative (1). *Nous*, *27*(2), 141–157. doi: https://doi.org/10.2307/2215752

Curry, H. B., & Feys, R. (1958). *Combinatory Logic* (Vol. 1). North-Holland Publishing Company.

Duží, M. (2017). If structured propositions are logical procedures then how are procedures individuated? *Synthese*, *196*(4), 1249–1283. doi: https://doi.org/10.1007/s11229-017-1595-5

Duží, M., & Jespersen, B. (2012). Transparent quantification into hyperintensional contexts de re. *Logique et Analyse*, *55*(220), 513–554.

Duží, M., & Jespersen, B. (2015). Transparent quantification into hyperintensional objectual attitudes. *Synthese*, *192*(3), 635–677. doi: https://doi.org/10.1007/s11229-014-0578-z

Duží, M., Jespersen, B., & Materna, P. (2010). *Procedural Semantics for Hyperintensional Logic: Foundations and Applications of Transparent Intensional Logic*. Dordrecht: Springer. doi: https://doi.org/10.1007/978-90-481-8812-3

Duží, M., & Kosterec, M. (2017). A Valid Rule of β-conversion for the Logic of Partial Functions. *Organon F*, *24*(1), 10–36.

Harris, V. C. (1971). On Proofs of the Irrationality of the Square Root of 2. *The Mathematics Teacher*, *64*(1), 19–21. doi: https://doi.org/10.2307/27958508

Howard, W. A. (1980). The formulae-as-types notion of construction. In H.B., J. R. Hindley, & J. P. Seldin (Eds.), *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism.* Academic Press.

Jespersen, B. (2017). Anatomy of a Proposition. *Synthese*, *196*(4), 1285–1324. doi: https://doi.org/10.1007/s11229-017-1512-y

Martin-Löf, P. (1984). *Intuitionistic type theory*. Bibliopolis.

Materna, P. (2004). *Conceptual systems*. Logos.

Materna, P. (2008). The notion of problem, intuitionism and partiality. *Logic and Logical Philosophy*, *17*(4), 287–303. doi: https://doi.org/10.12775/LLP.2008.016

Moggi, E. (1988). *The Partial Lambda-Calculus* (Unpublished doctoral dissertation). Faculty of Mathematics and Informatics, University of Edinburgh.

Moschovakis, Y. N. (2006). A Logical Calculus of Meaning and Synonymy. *Linguistics and Philosophy*, *29*(1), 27–89. doi: https://doi.org/10.1007/s10988-005-6920-7

Muskens, R. (2005). Sense and the computation of reference. *Linguistics and Philosophy*, *28*(4), 473–504. doi: https://doi.org/10.1007/s10988-004-7684-1

Pezlar, I. (2017). Algorithmic Theories of Problems. A Constructive and a Non-Constructive Approach. *Logic and Logical Philosophy*, *26*(4), 473–508. doi: https://doi.org/10.12775/LLP.2017.010

Pezlar, I. (2018). On Two Notions of Computation in Transparent Intensional Logic. *Axiomathes*, *29*(2), 189–205. doi: https://doi.org/10.1007/s10516-018-9401-7

Prawitz, D. (2006). *Natural Deduction: A Proof-theoretical Study*. Dover Publications, Incorporated.

Raclavský, J., Kuchyňka, P., & Pezlar, I. (2015). *Transparentní intenzionální logika jako characteristica universalis a calculus ratiocina-*

tor. Brno: Masaryk University Press (Munipress).
Tichý, P. (1988). *The Foundations of Frege's Logic*. Berlin: de Gruyter.
Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus*. London: Kegan Paul, Trench, Trubner & Co., Ltd.
Wittgenstein, L. (1978). *Remarks on the Foundations of Mathematics* (Revised ed ed.). Oxford: Basil Blackwell.

Ivo Pezlar
Czech Academy of Sciences, Institute of Philosophy
The Czech Republic
E-mail: pezlar@flu.cas.cz