MASARYK UNIVERSITY
FACULTY OF ARTS

Department of Philosophy

PhD Thesis

2016 **Ivo Pezlar**

MASARYK UNIVERSITY
FACULTY OF ARTS

Department of Philosophy

Philosophy

Ivo Pezlar

Investigations into Transparent Intensional Logic: A Rule-based Approach

PhD Thesis

Supervisor: doc. PhDr. BcA. Jiří Raclavský, Ph.D.

Brno 2016

I declare this thesis contains my original work. Wherever contributions of others are involved, I make every effort to indicate this clearly with due reference to the literature.

.....

Preface

Since the inception of the canonical version of Transparent Intensional Logic (TIL for short) with the publication of Tichý's seminal book *The Foundations of Frege's Logic* (*FFL* for short) almost 30 years ago, there has not yet been any successful attempt at providing full-fledged deduction system for it. I struggled with it as well, tried many different approaches and hit a lot of dead ends along the way.[1] All that time I was failing to ask the most basic question of every inquiry – why?

With this newly acquired perspective I took a step back and started rereading *FFL* again. Slowly it became evident that TIL as presented there relies on a lot of implicit assumptions and that much human insight is needed in order to make it work due to the absence of any rules.[2]

Hence, my new recourse was clear: try to unearth these implicit assumptions via use of explicit rules. The initial aim of developing deductive calculus for TIL thus slowly transformed into developing calculus that would enable us to work explicitly with information previously directly inaccessible in TIL.

That being said, I do not see my investigations as invalidating the current body TIL works. It is rather alternative approach, an extension to TIL. Ideally, my intention is to preserve the past TIL research as it is and just provide explicit grounds for it that correspond much more closely to the way TIL is actually used in practice. As it usually is, however, this ideal was not achievable in all cases and I was forced to make various concessions in the process. Eventually, it became

[1] In hindsight, each failure brought me closer to the current system. To name the most prominent ones: Gabbay's Labelled Deductive Systems (LDS) [11], Shroeder-Heister's Definitional Reasoning (DR) [34], Martin-Löf's Constructive Type Theory (CTT [19]), and Dalrymple's Glue Semantics (GS) [4]. The system presented here draws inspiration from all of these systems.

[2] Parallels can be drawn to Peano's *Arithmetices principia: nova methodo exposita* (1889) [23], no doubt ingenious work, yet ultimately informal. See e.g., [41], p. 84.

a balancing act. Trying to be as faithful as possible to the original underlying ideas contained in *FFL*, while simultanously importing new ideas when necessary.

My project can be perhaps liken to those of Gentzen or Martin-Löf, i.e., an attempt of formalizing (via offering explicit set of rules) previously rather informal areas of reasoning:

> *Gentzen*:[3]
>
> Die Formalisierung des logischen Schließens, wie sie insbesondere durch Frege, Russell und Hilbert entwickelt worden ist, entfernt sich ziemlich weit von der Art des Schließens, wie sie in Wirklichkeit bei mathematischen Beweisen geübt wird. Dafür werden beträchtliche formale Vorteile erzielt. Ich wollte nun zunächst einmal einen Formalismus aufstellen, der dem wirklichen Schließen möglichst nahe kommt. So ergab sich ein "Kalkül des natürlichen Schließens".
>
> ([12], p. 176)

> *Martin-Löf*:
>
> What we do here is meant to be closer to ordinary mathematical practice. We will avoid keeping form and meaning (content) apart. [...] Thus we make explicit what is usually implicitly taken for granted. When one treats logic as any other branch of mathematics, as in the metamathematical tradition originated by Hilbert, such judgements and inferences are only partially and formally represented in the so-called object language, while they are implicitly used, as in any other branch of mathematics, in the so-called metalanguage.
>
> [...]
>
> Our main aim is to build up a system of formal rules representing in the best possible way informal (mathematical) reasoning.
>
> (Martin-Löf, [19], pp. 3–4)

[3]English translation: "The formalization of logical deduction, especially as it has been developed by Frege, Russell, and Hilbert, is rather far removed from the forms of deduction used in practice in mathematical proofs. Considerable formal advantages are achieved in return.

In contrast, I intended first to set up a formal system which comes as close as possible to actual reasoning. The result was a '*calculus of natural deduction*'..." (Gentzen, [13], p. 68).

Analogously, we also try to set up a system of formal rules that comes as close as possible to actual reasoning done in TIL. In other words, we attempt to put as much as possible from TIL metatheory into the object language.[4] What we get in return is eTIL (for explicit TIL) calculus.

However, it is important to note that from Tichý's point of view, our approach would be rather retrogressive in many respects: he used TIL to deal with formal languages, while we here use formal language to deal with TIL.

To whom is this text intended

We will not be explaining here the fundamentals of TIL-based semantic analysis. There are already plenty of quality resources that thoroughly deal with this topic (and much more): starting with *FFL* itself [38] and continuing to e.g., [20], [21], [9], [30], [32] and we see no point in adding just another one.

Instead we will focus on the core principles that fuel this semantic analysis. Thus, my target reader is either a TIL newcomer, someone who is completely new to the system and just wants to learn about the mechanisms of TIL without actually diving into natural language analysis itself, or a TIL veteran, someone who is already well acquainted with TIL, but is interested in new approaches to it. To put it briefly, the topic of this thesis is TIL itself, not what we can do with it, once we accept it.

That said, there are topics discussed here that, we believe, could be of interest to anybody who is interested in formal semantics, logical analysis of natural language, and in proof-theoretic semantics with focus on natural languages. In even broader terms, the intended audience for this thesis should lay somewhere at the intersection of philosophy of language, logic, linguistics, and computer science.

[4] In a way we are approaching Tichý's logic in similar way he approached Frege's logic, i.e., trying to reveal implicit principles.

Acknowledgements

I built on the works others. The two most crucial sources of inspiration (aside from Pavel Tichý, of course) are works of Gerhard Gentzen and Per Martin-Löf.

Furthermore, this work was not conceived of in a vacuum and there are many to whom I am very obliged. Especially, I would like to thank prof. Peter Schroeder-Heister, prof. Pavel Materna, prof. Grigory K. Olkhovikov, prof. Göran Sundholm, prof. Nissim Francez, Ansten Mørch Klev, Michael Arndt, Luca Tranchini, and Petr Kuchyňka for all the discussions and help they gave me during my studies.

And most importantly I would like to thank my supervisor Jiří Raclavský for all his guidance and support.

Contents

Chapter 1

Introduction

Summary Our initial goal was to develop deductive system (i.e., set of inference rules) for Transparent Intensional Logic (TIL) as presented in Tichý's seminal work *The Foundations of Frege's Logic* (*FFL* for short). In doing so, however, we hit upon several obstacles stemming from the very core of TIL itself. Most importantly, the formalization of TIL is rather removed from the way that it is used in practice and lot of things are left implicit. Considerable advantages in regard to analysis of natural language are achieved in return, however, at the cost of its deductive capabilities.

We tried to overcome these hindrances by providing alternative, more explicit formalization that would keep as close as possible to the actual reasoning done in TIL. This eventually led to a development of whole new calculus called eTIL (for *explicit* TIL).

1.1 Aim

The main aim of this thesis is to develop deduction system for Transparent Intensional Logic (TIL for short) as presented in *The Foundations of Frege's Logic* (*FFL* for short) [38] that would correspond to the way TIL is used in practice.[1] In return we hope to contribute to the development of TIL as a unifying frame-

[1] We take TIL presented in *FFL* [38] as the definitive (i.e., most developed) version of TIL as Tichý himself envisioned it. So, whenever we talk about TIL without any further specification, we will be referring to the system presented in this book. It follows that we will disregard to a large extent his work from earlier papers and deem them outdated and surpassed by TIL as presented in *FFL*. Therefore, *FFL* will be our primary source and also the ultimate judge in all disputes over possible interpretations of various aspects of TIL (should any arise).

work capable of semantic analysis of both non-empirical (i.e., formal/artificial languages) as well as empirical (natural languages) discourse.[2]

1.2 Opening problem

TIL provides very powerful and expressive language. High expressiveness, however, came at the price of virtually non-existent deductive system. More specifically, *FFL* presents no calculus (no inference rules) for TIL. And although it contains Chapter 13 titled *Inference*, it deals with deduction only on broader philosophical and conceptual level, no concrete rules for TIL are offered.

Earlier papers are much better in this respect (various rules for earlier versions of TIL are explicitly presented, see [22], [36], [37] or [39]). Unfortunately, we can only guess how much Tichý saw as compatible, and hence transferable to his later incarnation of TIL (with ramified type theory) as presented in *FFL*.[3] Thus, we will devise deductive system for TIL from the ground up based solely on the ideas and principles contained (either explicitly or implicitly) in *FFL*.

This, however, proved to be difficult due to insufficiently explicit character of TIL. The general method of solution to this problem, which will be developed here, may be described briefly as follows: so called constructions, which constitute the main subject matter of TIL, are assigned lower-level derivations and then rules are developed for manipulating these lower-level derivations.

As a result, we get an alternative approach to TIL that does not invalidate the current body of work, but offers a different perspective. To put it differently, we will explore and attempt to reconstruct the foundations of TIL in our own way.[4]

[2] Something other frameworks with similar goals are often struggling with, especially in connection with meaning of atomic sentences. See e.g., CTT [19], [33], [44].

[3] The most likely answer is: not much, since he chose not to mention any of it in *FFL*, not even among references. Hence, since none of it was actually included, it seems reasonable to assume that Tichý viewed it as either surpassed or outdated.

[4] Simply put, the main subject matter for us will be TIL itself and its metatheory rather than what can we do with TIL once we accept it. In other words, we want to provide deductive system based on rules *of* TIL, rather than on rules *for* TIL.

1.3 Context of the thesis

1.3.1 General context of TIL

TIL is one of the very few frameworks that pursue the ambitious (and nowadays almost "old-fashioned") goal of being the universal language of science by offering full-scale system that can deal with both natural and formal languages, thus embodying the Leibnizian ideal of being characteristica universalis as well as calculus ratiocinator (cf. [15], [16]).[5]

Consequently, it is difficult to pigeonhole TIL. Perhaps most accurately it can be described as general framework laying at the crossroads of philosophy of language, logic, and linguistics. This lack of rigid boundaries is, however, not an accident arising from insufficient focus. On the contrary, it is by design and it stems from its "all-encompassing" and not too humble goal mentioned above.

The motivation for undertaking such grand project is generally two-fold:

1. theoretical – we want as few basic principles as possible explaining as much as possible in regard to the meaning of both empirical and non-empirical language realms.

2. practical – we want to develop machines that can assist us in reasoning, which requires inference engine that can "reason" about both non-empirical as well as empirical problems.

In other words, we want to have everything under one roof, so to speak. Relying just on single "all-in-one" environment leads to easier implementation and overall ease of use.

1.3.2 General context of the present work

The calculus eTIL presented here is probably best seen as a result of applying some of the key tools and mechanisms of proof-theoretic semantics (PTS) [35] to TIL. More specifically, eTIL is in many important aspects inspired by the works

[5] Only few other contemporary systems attempted this bold endeavourer as well. To mention some of the two most prolific ones: Martin-Löf's CTT [19] and Gabbay's LDS [11], which I also discussed in [26].

of Gentzen [12] (meaning via rules, dual rules) and Martin-Löf [19] (judgements, explicit typing). Consequently, eTIL is rather detached from the initial philosophy that stood behind the development of TIL.

1.4 Related work

Due to the distinct nature of our investigation in comparison to the more traditional approaches to TIL (TIL is not our starting point, but rather a goal we want to arrive at), there are only few works on TIL that are in any way relevant to our current endeavour.[6]

1.5 Structure of thesis

In chapter 2 *Revisiting* FFL we return to *FFL* and examine the six foundational constructions of TIL one by one. Our main claim will be that TIL suffers from lack of explicitness (it provides no rules for dealing with constructions are given and lot of things are only implicitly assumed.).

In chapter 3 *Developing eTIL*$_{\mathbb{C}}$ we propose explicit rules for handling constructions that try to alleviate some of the shortcomings of the original specification of constructions discussed in the previous chapter. New notions of judgement and type stack are introduced.

Chapter 4 *Case studies: eTIL*$_{\mathbb{C}}$ will be devoted to concrete examples of application of our eTIL$_{\mathbb{C}}$ system, thus demonstrating it in practice. Specifically, we will be showing how various TIL analyses are to be translated into our system.

In chapter 5 *Generalizing eTIL*$_{\mathbb{C}}$ *into eTIL* we further generalize system eTIL$_{\mathbb{C}}$ The resulting system will be called eTIL. This generalization will permit us to introduce new notions and provide analyses previously unobtainable due to our restrictions to first-order constructions only.

Chapter 6 *Case studies: eTIL* will be dedicated to various examples demonstrating in practice new notions introduced in the previous chapter (higher-order constructions, improper constructions, higher-level rules,...).

[6] Possibly our closest ally is Aleš Horák. In his PhD thesis [17] he sketches rule scheme for TIL that is in principle similar to ours (example is offered in later section 2.4).

Chapter 2

Revisiting *FFL*

Summary In this chapter we revisit *FFL* and examine the six foundational constructions of TIL one by one. Our main claim is that TIL suffers from lack of explicitness – no rules for dealing with constructions are given and lot of things are just implicitly assumed. The main purpose of this chapter will be to demonstrate these aspects. No prior knowledge of TIL is assumed on the side of the reader. And this will hold for the rest of the thesis as well.

2.1 Introduction

High aptitude of TIL in precise case-by-case analysis of various intricate aspects of natural language phenomena is generally known and well examined (recently, see e.g., [8], [6], [9] [31], [7], [21], [28]). However, the bedrock of TIL itself, namely the six basic constructions upon which the whole system stands (i.e., Variable, Trivialization, Execution, Double execution, Composition, and Closure), is considerably less explored.[1]

The canonic version of TIL was established by Tichý's book *The Foundations of Frege's Logic* [38]. There, at the beginning of *Chapter* 5 in *Section* 15 titled *Five modes of forming constructions* on pages 63–65, he presented five non-simple constructions (i.e., Trivialization, Execution, Double execution, Composition, and Closure) that establish together with Variable (the only simple construction introduced in previous *Chapter* 4) the basis of "language of TIL constructions".

[1] Recent exception to this is [32].

We start by introducing and examining these six constructions one by one.[2]

2.2 TIL constructions

2.2.1 Variable

In *FFL* Tichý introduces novel notion of Variables that do not take values, but rather retrieve them.[3] Tichý defines them as follows:

> **Definition 1 (Variable).**
>
> Let R be an arbitrary non-empty collection. By an *R-sequence* we shall understand any infinite sequence
>
> (s) $X_1, X_2, X_3, X_4, ...$
>
> (with or without repetitions) of members of R.
>
> For any natural number n let $|R|_n$ be the (incomplete) [i.e., depending on external sequence – author] construction which consists in retrieving the n-th member of an R-sequence. Constructions of this form will be called *variables*.
>
> ([38], p. 60)

> **Definition 2 (Valuation).**
>
> In this more general case [where various logical types are needed – author] we shall need whole arrays of sequences containing an R^i-sequence for each type R^i. We shall call such arrays *valuations*. Thus, where $R^1, R^2, R^3, R^4, ...$ is an enumeration (without repetitions) of all the types, a valuation is an array of the form

[2] Some of the criticism that will be presented here stems not really from properties of TIL itself, but rather from the fact that we approach it with different set of expectations and demands of what logical system should do and provide than did Tichý himself. For example, it is clear that for Tichý the matter of deduction was not as important topic as for the present author. After all, he omitted it almost completely from the *FFL* book. From this point of view, aspects of our criticism can be viewed as unfair – criticizing TIL for something it was not meant to be doing.

[3] We write 'Variable' with capitalized first letter in order to better distinguish Tichý's notion of variable from the more traditional ones.

$$(v)\quad \begin{array}{l} X^1_1, X^1_2, X^1_3, X^1_4, \ldots \\ X^2_1, X^2_2, X^2_3, X^2_4, \ldots \\ X^3_1, X^3_2, X^3_3, X^3_4, \ldots \\ X^4_1, X^4_2, X^4_3, X^4_4, \ldots \\ \vdots \end{array}$$

where $X^i_1, X^i_2, X^i_3, X^i_4, \ldots$ is an R^i-sequence. Let v be this valuation. Relative to v, variable $|R^i|_n$ constructs X^i_n, i.e., the n-th term of the R^i-sequence occurring in v.

([38], p. 61)

For example, let's have the following valuation-array (or valuation for short) v_1:

$$v_1 = \begin{pmatrix} true^1_1 & false^1_2 & \cdots & & \\ 1^2_1 & 2^2_2 & 3^2_3 & \cdots & \\ Alice^3_1 & Bob^3_2 & Cecil^3_3 & Dana^3_4 & \cdots \end{pmatrix}$$

Then, e.g., Variable $|R^1|_2$ receives through v_1 as value $false$, $|R^3|_1$ retrieves $Alice$, etc. So what is Variable on Tichý's account? It is essentially search and retrieve mechanism that takes as input the coordinates $\langle i, n\rangle$ and some valuation-array v_m and returns object at that position.

All TIL objects including constructions themselves receive type. From type-theoretical perspective, the valuation-array v_1 would look something like this:

$$(v_{1t}) = \begin{pmatrix} o & o & \cdots & & \\ \nu & \nu & \nu & \cdots & \\ \iota & \iota & \iota & \iota & \cdots \end{pmatrix}$$

where o, α, ι represent types of truth values, natural numbers and individuals (so called first-order objects in Tichý's terminology), respectively. But what about Variables $|R^i|_n$ themselves? What type do they have? Tichý offers the following answer:

(c_ni) Let τ be any *type of order n over* B. Every variable ranging over τ is a *construction of order n over* B. If X is of (i.e., belongs to) type τ then ^{0}X, ^{1}X, and ^{2}X are *constructions of order n over* B. Every variable ranging over τ is a *construction of order n over* B.

Let $*_n$ be the collection of *constructions of order n over* B.

([38], p. 61)

Let types o, α, ι from our example above constitute type base B_1. Then, by definition 16.1.– 1.(t_1i),[4] o, α, ι are types of order 1. Further, let's have Variables $|R^1|_1$, $|R^2|_1$, $|R^3|_1$ that range over objects of types o, α, ι, respectively. By definition 16.1.–2.(c_ni) above, they are constructions of order 1 and hence all receive the same type $*_1$. Thus, we end up with Variables whose types do not match the types of their values. That might sound odd, but remember that in TIL Variables do not really take values, they rather fetch them.

This complicates things, however, because from the type of Variable alone we cannot tell the type of its value. E.g., the only information we can gather from Variables $|R^1|_1$, $|R^2|_1$, $|R^3|_1$ of type $*_1$ (assuming base B_1) is that these three Variables take as values (= retrieve) either truth values, natural numbers or individuals. It is only after we supply some valuation-array v we can learn what types of values they will actually take.

This dependency on valuation-arrays is exactly the reason why Tichý calls Variables *incomplete* (or heteronomous) constructions ([38], p. 60). In other words, Variables need some "external" building material, in this case valuation-arrays, to be able to construct anything.

2.2.1.1 Summary

When stating that "variable x has type α" what is usually meant by it is that the variable x in question takes as values only objects of certain type α. E.g., if x is of boolean type, then x can take as values either $true$ or $false$; if x has type of natural numbers, then x can accommodate values such as $1, 2, 3, \ldots$, etc. In TIL, however, this is not the case. Types of TIL Variables are different from types of their values. Hence, from the type of Variable alone we cannot tell what values should it retrieve (or construct in Tichý's nomenclature).

2.2.2 Trivialization

Trivilization construction is the first non-simple construction introduced in *FFL*. Informally, it is possibly best understood as a primitive computation.

[4] See [38], p. 66, or appendix A.

Definition 3 (Trivialization).

Where X is any entity whatsoever, we can consider the trivial construction whose starting point, as well as outcome, is X itself. Let us call this rudimentary construction the *trivialization* of X and symbolize it as ^{0}X. To carry ^{0}X out, one starts with X and leaves it, so to speak, as it is.

([38], p. 63)

Thus, going by the specification above, Trivilization takes object of certain lower type $(o, \alpha, \iota, ..., *_1, ...)$ and "packs" it into another higher type $(*_1, ..., *_2, ...)$. If we form Trivialization from, let's say, number 3 of type α, we get construction ${}^0 3$ with the nearest higher type, which is in this case $*_1$ (type of first-order constructions, i.e., constructions of first-order objects). Analogously, if we form Trivialization from construction ${}^0 3$ of type $*_1$, we get (higher-order) construction ${}^0({}^0 3)$ of type $*_2$ and so on. The idea is then that construction ${}^0 3$ constructs the number 3.

Note the slight peculiarity that sneaks into the definition. Tichý starts by stating that Trivialization can be formed from any entity whatsoever, i.e., either from non-constructions (objects of type o, α, ι) or other constructions (objects of type $*_1, *_2, ...$). Let's assume it was formed from non-construction, specifically from number 3, thus we get ${}^0 3$. But at the same time, it is this Trivialization construction ${}^0 3$ that is used to *construct* number 3 (Tichý explicitly writes "...${}^0 3$ *v*-constructs 3..."). Thus, we form ${}^0 3$ from 3 and at the same time we use ${}^0 3$ to construct 3.

Remark Although Tichý always talks about "*v*-constructing" (i.e., constructing relative to some valuation *v*), we will talk explicitly about *v*-constructing only in those cases where it actually matters. E.g., it seems redundant to say that construction ${}^0 3$ *v*-constructs 3 (i.e., that it constructs 3 relative to some valuation *v*) when the result does not actually depend on any specific valuation at all. In other words, although all constructions *v*-construct, we will mention the valuation *v* explicitly (i.e., "*v*-constructing") only when it affects the outcomes of constructions.

Thus, even though ${}^0 3$ is suppose to construct number 3, it in itself is not number 3. In other words, meaning of ${}^0 3$ is not 3 but rather "take 3 and return 3". It follows that ${}^0 3$ and 3 are, of course, objects of different types, i.e., $*_1$, α, respectively. Consequently, since TIL deals explicitly only with constructions, we

can never directly work with first-order objects. It always has to done via certain proxies (i.e., constructions).[5] To put it differently, in language of TIL there are no numbers, functions, truth values, etc., only constructions of thereof.[6]

Remark One might want to argue that "trivialized first-order objects" are *essentially* just first-order objects, that there is no substantial difference. We disagree for the simple fact that e.g., 3 and ^{0}3 are objects of different types, hence eligible to different operations. E.g., we cannot add (subtract, multiply,...) constructions and simultaneously we cannot have first-order objects (i.e., non-constructions) constructing anything upon execution, etc.

2.2.2.1 Summary

The idea of Trivialization is in itself quite unproblematic. Concepts of similar kind can be also found e.g., in Haskell programming language [24] under the moniker of *monads* (Trivilization roughly corresponds to `return` operation and "Trivialized" objects to "monadic values"). Unfortunately, the nature of Trivialization effectively prevents us from working directly with first-order objects. E.g., even though Trivialization construction ^{0}3 is suppose to be formed from 3, the 3 itself can never explicitly appear in TIL language.

2.2.3 Execution

Execution (or alternatively Single execution) construction seems to be the most problematic construction of all.[7]

> **Definition 4 (Execution).**
>
> For any entity X we shall also speak of the *execution* of X and symbolize it as ^{1}X. If X is a construction, ^{1}X is X. (The construction consisting in executing, or carrying out, construction X is clearly none other than X itself.)
>
> ([38], p. 63)

[5] We can always say things like "^{0}3 (*v*-)constructs 3", but then we are leaving the language of TIL constructions and moving into metatheory.

[6] This, of course, follows from the fact that Tichý strictly upholds Frege's distinction between sense and denotation [14].

[7] With the obvious exception of Double execution, which presupposes this one.

This notion of execution presented in the above definition seems particularly puzzling. We would say that it is far from being clear that execution of certain construction (which are often likened to procedures or calculations) is the same as the construction itself. Shouldn't execution of construction yield rather its result? Stating that Execution of construction is the same as the construction itself is hard to reconcile with most intuitions that are hold about what does it mean to execute some procedure or calculation.[8]

E.g., let's have the following construction:[9]

$$[^0+\ ^0 5\ ^0 7]$$

and its corresponding Execution

$$^1[^0+\ ^0 5\ ^0 7]$$

By the above definition, this should result back into $[^0+\ ^0 5\ ^0 7]$. But how is that possible? Shouldn't $^1[^0+\ ^0 5\ ^0 7]$ (i.e., execution of $[^0+\ ^0 5\ ^0 7]$) construct the number 12 rather than return the original construction? This certainly seems as the more intuitive option of the two. However, this conception is in clear discord with the above definition ("...[E]xecuting [...] construction X is [...] X itself."). In other words, it seems to me that X ("a procedure") and ^{1}X ("an execution of a procedure"), inherently, carry different sort of information. It is perplexing to say that Execution of certain construction yields that very construction back.

And possibly most importantly, Tichý specifies the behaviour of other constructions (and hence their "procedural" meaning) with explicit recourse to execution.[10] Hence, we have constructions that were specified using execution, but at the same time Execution is also introduced as a stand-alone construction in its own right. But, surely, if constructions are specified via the notion of execution, then Execution construction cannot be taken just as another "regular" construction. (We will return to this topic in later section 3.3.2.2.)

[8] We will write 'Execution' when referring to the construction specified in the above quotation, and 'execution' when referring to the general concept and its intuitive understanding.

[9] It is a construction called Composition, which will be discussed in more detail in the following section 2.2.5.

[10] The only exception being Closure constructions, where it is, arguably, implicitly presupposed. For more, see section 2.2.6.

It is also not without its own interest that Tichý never actually used Execution construction in the whole *FFL* book. He just introduced it and never did anything with it. Its specification is the first and the last time we see it.

Of course, if we are to side with Tichý and accept that ^{1}X is indeed just X, then his omission of Execution from the whole book starts to make sense. There is simply no need. So every time we see 'X' it is just a syntactic shorthand for '^{1}X'. With every construction X we can imagine the little '^{1}' in the upper left corner and nothing changes.

In other words, the lack of use of Execution construction directly follows from Tichý's specification of it (^{1}X is X). However, although this answers the question why he never used it, it hardly answers the much more important question what is actually Execution in the first place. Does it mean that all constructions are in some sort of state of endless execution? Or that every construction is its own Execution? Or maybe that the notion of executing X is somehow already implicitly assumed with X?

Whatever the case maybe, both the introduction of Execution constructions and then the subsequent identification of ^{1}X with X is very puzzling. (As already mentioned, we will return to this topic in section 3.3.2.2.)

Remark In [32] is explored different explanation of Execution construction. More specifically, Tichý's original dictum "^{1}X is X" is interpreted as "^{1}X is *v*-congruent with X" (i.e., X and ^{1}X are considered to be different constructions constructing the same result). However, we do not adopt it here because it seems to be clashing with Tichý's specification of constructions that relies on the assumption that ^{1}X is X (see e.g., the specificaiton of Composition in section 2.2.5).

To recapitulate, there are two main argument lines against Execution construction:

- empirical: Tichý never actually uses Execution – he just specifies it and then never mentions it nor uses it again. And even if he would use it, there would be very little to gain from it. Of course, this is hardly surprising, since the very specification of Execution itself renders it superfluous. Why then adopt it as one of the six main building blocks of TIL?

- technical: various problems stemming from the way how Execution should work – when specifying constructions Tichý makes use of the notion of execution, yet simultaneously considers execution as one of the constructions as well. This fact is hard to reconcile.

Thus, the case for Execution as a construction is indeed a very difficult one. We have both empirical (no occurrence in whole book or subsequent works) as well as conceptual (it is used to specify the behaviour of other constructions, difficulties of coming up with useful examples despite it being one out of six foundational constructions of the whole TIL) arguments against it. And the fact that many TIL authors (see next section) often completely neglect it also doesn't help the case much.[11]

2.2.3.1 Others on Execution

In [6] Duží treats execution as "mode of construction" (p. 484). This piece of key information is, however, mentioned only in a footnote without any further clarification. Trend that is often repeated. E.g., in [20] executions are not mentioned at all as constructions. In [21] it is at least acknowledged that they exist, but not much more is explained:

> We can do without Tichý's constructions called *execution* and *double execution*.
>
> ([21], p. 36)

Argument or demonstration why we can disregard whole one third of the original fundamental constructions of TIL is, unfortunately, never presented.

Or compare e.g., with [9], where they write (again, only in a footnote):

> It turns out, however, that we occasionally also need a fifth and a sixth construction, called *Execution* and *Double Execution*.
>
> ([9], p. 6)

In their specifications of other constructions (i.e., Variable, Trivialization, Composition, and Closure) no appeal to execution is made at all contrary to Tichý. But

[11] Noteworthy exception to this is [32].

this, however, does not stop them from talking about "executing constructions". E.g., on p. 45 they write:

> *Remark.* ${}^1X$ is the procedure of executing X. Thus, if X is a construction then the execution of ${}^1X$ consists in executing X.
>
> ([9], p. 45)

So they are essentially stating that execution of executing X consists in executing X.

2.2.3.2 Summary

It is not clear what exactly is the role of the Execution construction. Stating that execution of a construction is the same as the construction itself seems to go against deeply rooted intuitions about what execution in general is suppose to represent.

Moreover, if ^{1}X is the same as X, why even introduce it as a new construction, since it adds no new information? And most importantly, the notion of execution is employed when specifying the behaviour of constructions, hence it seems strange to simultaneously regard it as a standalone construction as well.

These reasons lead us to the conclusion that Execution is not a construction at all (at least not in the same sense as the rest).

2.2.4 Double execution

Double execution is essentially just two stage Execution. Tichý specifies it as follows:

> **Definition 5 (Double execution).**
>
> If what is constructed by X is itself a constructions, one can execute X and go on and execute the result. We shall speak of this two-stage construction as the *double execution* of X and symbolize it as ^{2}X. For any entity X, the construction ^{2}X is *v*-improper (i.e., yields, relative to *v*, nothing at all) if X is not itself a construction, or if it does not *v*-construct a construction, or if it *v*-constructs a *v*-improper construction. Otherwise ^{2}X *v*-constructs what is *v*-constructed by what is *v*-constructed by X.
>
> ([38], p. 64)

The notion of Double execution construction clearly presupposes Single execution construction. And since we have excluded the latter from constructions, we have to exclude even the former. For this reason, we will not discuss it separately here.[12]

Remark It is unexpected that Double execution ${}^2({}^0[{}^0+\ {}^0 5\ {}^0 7])$ *constructs* 12, but ${}^1[{}^0+\ {}^0 5\ {}^0 7]$ *is* $[{}^0+\ {}^0 5\ {}^0 7]$. Shouldn't Single execution of $[{}^0+\ {}^0 5\ {}^0 7]$ yield rather the number 12? After all it does precisely this in the case of Double execution: its second stage is the Single execution of $[{}^0+\ {}^0 5\ {}^0 7]$ which yields 12. So why does it behave differently in the case of Single execution alone?

2.2.5 Composition

Composition construction is essentially TIL variant of function application but with couple of important differences.

> **Definition 5 (Composition).**
>
> Let $X_0, X_1, ..., X_m$ be arbitrary constructions. By the *composition* $[X_0 X_1 ... X_m]$ *of* $X_0, X_1, ..., X_m$ (in this order) we shall understand the construction which consists in: executing X_0 to obtain *m*-ary mapping, executing $X_1, ..., X_m$ to obtain an *m*-tuple of entities, and then applying that mapping to the *m*-tuple. Thus for any *v*, $[X_0 X_1 ... X_m]$ is *v*-improper if one of $X_1, ..., X_m$ is *v*-improper, or if X does not *v*-construct a mapping which is defined at the *m*-tuple of entities *v*-constructed by $X_1, ..., X_m$. If X_0 does construct such a mapping then $[X_0 X_1 ... X_m]$ *v*-constructs the value the mapping takes at the *m*-tuple.
>
> ([38], p. 64)

So, in case of the simplest Composition $[X_0 X_1]$ (just one function and one argument) the square brackets '[ ]' represent the following 3-step procedure:

1. execute X_0 to get its result (i.e., a function);
2. execute X_1 to get its result (i.e., an argument);
3. apply the result of X to the result of X_1.

[12] We will return to Double execution in later section 6.5.

For example, to execute Composition $[^0+\ ^05\ ^07]$ from earlier we have to, subsequently (recall that α is type of natural numbers):

1'. execute construction $^0+$ and get its result, i.e., addition function $+$ of type $(\alpha\alpha\alpha)$;[13]

2'. execute construction ^{05} and get its result, i.e., number 5 of type α; the same goes for ^{07}.

3'. apply function $+$ to numbers 5 and 7, which results in the number 12 of type α.

As you can see, this is quite a departure from the run-of-the-mill application as known from lambda calculus, i.e., apply some function f to an argument a.

Firstly, notice the change in the notion of Execution construction. By previous definition, if X is a construction, then its execution should yield back this very construction X. But now, executing X (let's say, construction of a function) results in the object constructed by X (i.e., some function) and not X itself as the definition of Execution suggests.

Secondly, and this follows directly from our earlier point that in TIL we cannot talk directly about non-constructions (i.e., results of first-order constructions), Composition tries to do too much at the same time. And, consequently, it leaves too much information hidden away from the reader. All the executions, returning results and their consequent applications as well as results of those are "running in the background", and hence are not explicitly accessible to us. It is all happening under the hood of TIL, so to speak.

Thirdly, there is a puzzling aspect about the type signature of Composition. Tichý defines the behaviour of Composition from type-theoretical perspective in the following manner (for full definition, see appendix A):

> ($c_n$ii) If $0 < m$ and $X_0, X_1, ..., X_m$ are *constructions of order n* then $[X_0 X_1 ... X_m]$ is a *construction of order n over* B. ...
>
> ([38], p. 66)

Let's examine the Composition $[^0+\ ^05\ ^07]$ again. All constructions $^0+$, ^{05}, ^{07} construct first-order objects, therefore they are constructions of type $*_1$. Now, if

[13] The notation '$(\alpha_3\alpha_2\alpha_1)$' can be alternatively written as '$(\alpha_1 \to \alpha_2 \to \alpha_3)$', i.e., type of function that takes two arguments of types α_1 and α_2 and returns value of the same type α_3.

we put them into a Composition, then, by ($c_n$ii), we also have a construction of type $*_1$. So $[^0+\ ^05\ ^07]$ is also of type $*_1$. I.e.,:

$$[^0+\ ^05\ ^07] \text{ —} *_1$$

$$| \quad | \quad |$$

$$*_1 \quad *_1 \quad *_1$$

Thus, somewhat strangely, we have an object of type $*_1$ that contains three other objects of the very same type $*_1$.

What it effectively means is that from the type of the Composition $[^0+\ ^05\ ^07]$, i.e., $*_1$, we learn nothing about how it operates, e.g., how many arguments are being constructed and so on. Or to put it differently, first-order Composition seems to take an arbitrary number of constructions of type $*_1$ and then flatten them together into type $*_1$. E.g., different Composition $[^0+[^0+\ ^05\ ^07]\ ^012]$ would receive the same type as $[^0+\ ^05\ ^07]$, i.e., $*_1$.[14]

Remark Sometimes TIL constructions are likened to programs. If this metaphor is expected to uphold, then type of constructions should correspond to the program specification. This is, however not the case in TIL. E.g., $*_1$ cannot be taken as a specification of the "program" $\lambda x\ [^0\text{Succ}\ x]$[15] for the simple fact that it does not actually specify anything. It tells us nothing about the behaviour of the construction itself.[16]

2.2.5.1 Summary

Even though the notation '$[X_0\,X_1]$' of Composition suggests that it is simple, straightforward operation, in reality it is quite complex process (paragraph with nine lines of informal explanation was needed in the original Tichý's book to explain how does it suppose to work). Additionally, Composition behaves oddly

[14] Of course, we could say e.g., "Composition $[^0+\ ^05\ ^07]$ of type $*_1$ contains three Trivializations of type $*_1$ that construct objects of type $(\alpha\alpha\alpha)$, α and α, respectively", but that would again pull us straight out of the object language and into metatheory.

[15] This is a construction known as Closure, which will be the topic of the next section 2.2.6.

[16] There is a sharp contrast with lambda calculus with so called Curry-Howard-de Bruijn correspondence, see [3], [18], [5]. E.g., lambda term/program $\lambda x.t$ would receive type $\alpha \to \beta$, but in TIL type of the corresponding construction (i.e., Closure) would be simply $*_1$.

from type-theoretical point of view, since it receives the very same type as the constructions it contains.

2.2.6 Closure

The last construction we will briefly inspect is the Closure construction.

> **Definition 6 (Closure).**
>
> ...[L]et τ be a collection, $x_1, ..., x_m$ distinct variables ranging over the respective collections $\xi_1, ..., \xi_m$, and v a valuation. Any construction Y can be used in constructing a mapping from $\xi_1, ..., \xi_m$ into τ; we shall call this latter construction the τ-*closure of* Y *on* $x_1, ..., x_m$, or briefly $[\lambda_\tau x_1 ... x_m Y]$. For any v, $[\lambda_\tau x_1 ... x_m Y]$ v-constructs the mapping which takes any $X_1, ..., X_m$ of the respective types $\xi_1, ..., \xi_m$ into that member (if any) of τ which is $v(X_1/x_1, ..., X_m/x_m)$-constructed by Y, where $v(X_1/x_1, ..., X_m/x_m)$ is like v except for assigning X_1 to $x_1, ...$, and X_m to x_m.
>
> ([38], p. 65)

Due to its close resemblance to ordinary λ-abstraction (note that throughout its definition there are no explicit mentions of execution or "carrying out"), Closure construction seems to be least problematic construction.[17]

2.2.6.1 Summary

Closure construction is reminiscent of typical function abstraction as known from lambda calculus. Consequently, it seems largely unproblematic. Nevertheless, there are some ambiguities regarding its formation, especially its relation to the other construction that is capable of constructing functions, i.e., Trivialization (we will return to this topic later in later section 5.6).

2.3 Further examination of TIL

Aside from constructions themselves, there are at least three other more general aspects of TIL that also deserve a closer look. The first of them, our inability

[17] This, however, changes once we adopt β- and η- conversions, which are not present in *FFL*. More on this in later section 5.6.

to talk directly about non-constructions, has been already discussed. The second of them is type declaration, which we only briefly touched upon throughout our examination of constructions. The third one is lack of assertoric force in constructions.[18] In the following two sections we will examine the last two in more detail.

2.3.1 On type declaration

By type declaration is usually understood explicit assignment of types to certain objects so that the corresponding logical system (whatever it may be) can work with them correctly.[19]

For example, if we declare that objects $0, 1, a$ and a function $Addition$ have the following types

0 has type *number*

1 has type *number*

a has type $letter$

$Addition$ has function type $(number \to number \to number)$

then the corresponding system will "know" e.g., that $Addition$ function cannot be applied to arguments of type $letter$. In other words, it will "learn" that it makes no sense to add together numbers and letters.

Tichý does this implicitly via tables (see e.g, [38], p. 67):

entity	*type*	*description:*
0	α	the number nought
1	α	the number one
Suc	$(\alpha\alpha)$	the successor mapping, i.e., the mapping which takes 0 to 1, 1 to 2, etc.
⋮	⋮	⋮

or e.g, ([38], p. 202):

[18] There is one more general problematic aspect of TIL, specifically the notion of constructing itself, to which we will return later in section 5.5.

[19] For the origins of explicit typing, see e.g., [1].

office	*type*	*description:*
A	$\iota_{\tau\omega}$	the office of the author of *Waverly*
Ai	$\iota_{\tau\omega}$	the office of the author of *Ivanhoe*
Sf	$(\iota_{\tau\omega})_{\tau\omega}$	the office of Scott's favourite ι-office
P	$o_{\tau\omega}$	the proposition that the author of *Waverly* is a poet
⋮	⋮	⋮

And repeatedly on many other places in *FFL*.

And even though Tichý presents these tables only as examples, what he is effectively doing is introducing objects and declaring their types, which he then later uses to form various constructions. Without tables such as these we would not know that e.g., Suc is suppose to be treated in TIL as a successor function of type $(\alpha\alpha)$.[20]

Remark The symbol '/' is used by Tichý, Duží et al. [9] and Raclavský et al. [32] with different meanings. In all these works we can encounter the following three different forms of expressions:

i) A/ϕ

ii) ${}^0A/\psi$

iii) ${}^0A/\phi$

where ϕ is type of any non-construction and ψ is type of construction of any order. For example:

i)' $Alice/\iota$

ii)' ${}^0Alice/*_1$

iii)' ${}^0Alice/\iota$

[20]Note the order of business here: Tichý first prepares the building blocks by introducing and typing first-order objects/non-constructions, then he uses them to build constructions. This will become important in later section 3.3.

Expressions of the form i) and ii) are used e.g., in [9] and the symbol '$/$' is read as "has type", "is of type", etc.

Expressions of the form iii) are used e.g., in [32] and '$/$' is read as "constructs object of type".

Expressions of the form i) are used by Tichý in pre-*FFL* era. Its meaning is, however, difficult to discern. As already mentioned, in *FFL* he is not using this symbol at all and its role is effectively replaced by "typing tables" discussed above.[21] And in his earlier papers, where it is used, he didn't yet introduced Trivialization '^{0}' (or higher types $*_n$) and also considered non-constructions (first-order objects) as constructions constructing themselves. So, it can go either way and the '$/$' in $Alice/\iota$ could be read both ways.

2.3.2 On assertoric force of constructions

What information does construction

$$[{}^0\text{Odd}\ {}^0 3]$$

alone actually convey? It cannot be that number 3 is odd. If it did, then

$$[{}^0\text{Odd}\ {}^0 2]$$

would tells us that 2 is odd as well, since both formally and procedurally there is no difference between them.

The fact is that in this form it is nothing more than just a query about the "oddness" of 3. What we need to make it informative is to convey somehow that the former constructs $true$, while the latter $false$.[22]

Note that forming some auxiliary construction

$$[[{}^0\text{Odd}\ {}^0 3]\ {}^0 = {}^0\text{true}]$$

[21] Again, one could only speculate as to why. But whatever the reason was due to its omittance we will not use it here either.

[22] This is exactly the reason why we introduce higher-level judgements of the general form $A \Rightarrow a$, where A is construction and a constructed object, in later chapters. Thus we get $[{}^0\text{Odd}\ {}^0 3] \Rightarrow true$ and $[{}^0\text{Odd}\ {}^0 2] \Rightarrow false$, respectively. And while $[{}^0\text{Odd}\ {}^0 3]$ alone cannot be considered as a fact, $[{}^0\text{Odd}\ {}^0 3] \Rightarrow true$ can be. But more on this in the dedicated section 5.1.

won't solve anything. The same problem appears again, since there is nothing preventing us from forming also the construction

$$[[^0\text{Odd}\ ^0 2]\ ^0 = {}^0\text{true}]$$

And if the former tells us that 3 is odd, then the second must be doing the same but with number 2. And so on and so on (i.e., $[[[^0\text{Odd}\ ^0 3]\ ^0 = {}^0\text{true}]\ ^0 = {}^0\text{true}]$ and $[[[^0\text{Odd}\ ^0 2]\ ^0 = {}^0\text{true}]\ ^0 = {}^0\text{true}]$,...). So, clearly, something different is needed.

And, of course, it is the same case with mathematical calculations such as

$$[[^0+\ {}^0 5\ {}^0 7]\ ^0 = {}^0 12]$$

and

$$[[^0+\ {}^0 5\ {}^0 7]\ ^0 = {}^0 11]$$

and so on. If the first is to be taken as stating a fact, then so must be the second.

Remark Tichý writes:[23]

> It is arguable that there is little reason to distinguish between construction $[^0\text{Odd}\ ^0 3]$ and the fact that 3 is odd. $[^0\text{Odd}\ ^0 3]$ is a *decision procedure*, and a decision procedure which comes out positively (it can be argued) is naturally called a fact. ([38], p. 69)

Putting aside that he never actually argues for it, we don't think there is anything natural about calling positive decision procedures as facts. Aren't "facts" (both mathematical and empirical) the very things by which decision procedures are decided? Surely, result of the decision procedure $[^0\text{Odd}\ ^0 3]$ depends upon the "mathematical *fact*" whether 3 is odd or not (or more precisely, whether it is a multiple of 2 or not). These two things cannot be conflated together.

Tichý seems to be aware of this, so he tries to explain:

> But a decision procedure cannot depend for its integrity on being a fact. $[^0\text{Odd}\ ^0 2]$ is just as good a decision procedure as $[^0\text{Odd}\ ^0 3]$, because the concept Odd can be applied to 2 just as well as it can be applied to 3.
>
> ([38], p. 69)

[23] In quotations I retain the original notational styles of the source material, unless there is a conflict with our own notation (see section 3.1.1). E.g., in the quote above there is '$[^0\text{Odd}\ ^0 3]$', while in the *FFL* itself it is written down as '[**Odd3**]'.

In conclusion, decision procedures cannot be considered as replacements for facts. Saying that positive decision procedures are facts cannot be taken as anything more than just a figure of speech.

Remark Of course, the fact that $[^0\text{Odd}\ ^03]$ constructs *true* does not depend on reasoner's individual knowledge. Saying, however, that $[^0\text{Odd}\ ^03]$ carries the same information as "$[^0\text{Odd}\ ^03]$ constructs *true*" is simply untenable.

If it did, then we would learn nothing new if we would move from

$$(1) \qquad [^0\text{Odd}\ [^0-\ [^0\times\ ^01909\ ^02]\ [^0\div\ ^019075\ ^05]]$$

to

$$(2) \qquad [^0\text{Odd}\ [^0-\ [^0\times\ ^01909\ ^02]\ [^0\div\ ^019075\ ^05]] \text{ constructs } true$$

but the fact is that (2) saves us the calculation. Sure, we still do not know what number are we talking about exactly, but we know it is an odd number. And principally, there is no difference to just "$[^0\text{Odd}\ ^03]$ constructs *true*". However, in the case (1) a calculation has to be performed in order to conclude whether it constructs *true* or $false$. Different possible conclusions must mean that (1) and (2) carry different information.

To reiterate, we cannot conflate procedures (no matter what the outcome is) with facts. Not to mention, claiming that e.g., construction such as

$$[^0\text{Odd}\ [^0-\ [^0\times\ ^01909\ ^02]\ [^0\div\ ^019075\ ^05]]$$

expresses a fact seems counter intuitive, to say the least.

Remark Tichý writes on ([38], p. 72):

> To *assert* that the division of 3 by 0 is undefined is to say that the construction $[^0\div\ ^03\ ^00]$ does not belong to that class [i.e., class of proper construction Pr_1 of type $(o*_1)$]. In other words, it is to *assert* the second order construction
>
> $$[\neg[\mathsf{Pr}_1\ ^0[^0\div\ ^03\ ^00]]]$$
>
> [emphases author]
>
> ([38], p. 72)

So Tichý himself sees a difference between construction and its assertion (note the phrase "...to assert (...) construction...'). In other words, by simply writing down '$[\neg[\mathsf{Pr}_1\ {}^0[{}^0\div\ {}^0 3\ {}^0 0]]]$' we are not asserting anything, we are just formulating (decision) procedure. What does it even mean to assert construction in TIL? Arguably, it means to state that it constructs *true*, however, we have no tools for declaring this in TIL presented in *FFL*.[24]

2.4 Summary

In this chapter we have identified several issues with TIL constructions. Briefly, we can sum them up as follows ("proposed fixes" will be the topic of the next chapter):

1. *Variable* – type of Variable alone does not tell us what type of object it should construct.

   proposed fix: introduction of explicit formation rules and explicit typing including the type of constructed object.

2. *Trivialization* – analogous issue as above.

   proposed fix: same as above.

3. *Execution* – ambiguous nature of Execution as a construction (What is it exactly, how does it suppose to work and to what purpose?).

   proposed fix: excluding Execution from constructions and turning it into an explicit rule.

4. *Double execution* – presupposes Execution.

   proposed fix: same as above.

[24] It is true that Tichý discusses assertions in *Chapter Ten: Church's Logic of Sense and Denotation*, section *33. Assertion* ([38], pp. 162–167), however, he never actually carries it over to TIL system itself – he considers it only in connection with Church's system. In earlier papers (see e.g., [22], [36], recently it reappears also in [32]) Tichý utilizes the notion of match that can emulate the role of assertions to a certain degree, however, it is not carried over to *FFL*.

5. *Composition* – tries to do too many things at once and all of them are implicit; also similar issue as with Variable.

   proposed fix: introduction of explicit execution rules and explicit typing.

6. *Closure* – similar issues as with Variable. (Additional problems appear when we introduce β- and η-conversions).

   proposed fix: introduction of explicit execution rules and explicit typing.

Additionally, we have also discussed three other issues, which are linked with the general lack of explicit information in TIL. More specifically:

1. No possibility to talk directly about first-order objects (non-constructions).

   proposed fix: introduction of judgements.

2. Types of objects are assumed in metatheory, they are not explicitly declared in TIL language.

   proposed fix: same as above.

3. No assertoric force of constructions.

   proposed fix: introduction of higher-level judgements.

Thus, the main claim of this chapter can be summed up as: TIL from *FFL* is insufficiently explicit. Consequently, this lack of explicit formalization hinders attempts at providing suitable deduction system.

How can insufficient formalization negatively impact development of rules? Suppose, e.g., that I want to devise rule that would tell us how to "execute" (carry out, evaluate, compute,...) the Composition construction

$$[^0+\ ^0 5\ ^0 7]$$

and all others that share this general form (i.e., construction capturing the process of addition of two numbers). Such rule has to contain the addition operation and its operands (i.e., they have to be present in the rule) – recall that ${}^0 5$ and ${}^0 7$ are not numbers, but constructions of numbers, therefore we cannot add them together directly. What we can add together is only the results of these constructions. And it goes analogously for the ${}^0+$ construction. Yet with the tools of TIL it is not possible to formulate such rule, since first-order objects are not part of the language at all.

Remark To give some samples for better comparison, in Peano arithmetic the "rule" for addition looks as follows:

$$a+0=a$$

$$a+S(b)=S(a+b)$$

Note that all the symbols 'a', '$+$', '0' appearing above stand for first-order objects from TIL perspective.

It is the same case in Martin-Löf's CTT:

$$\frac{a \in \mathsf{N}}{a+0=0 \in \mathsf{N}}$$

$$\frac{a \in \mathsf{N} \qquad b \in \mathsf{N}}{a+succ(b)=succ(a+b) \in \mathsf{N}}$$

In other words, in these systems we deal directly with first-order objects, something that is not feasible in TIL.

Horák [17] seems to be aware of this issue as well. His example of addition rule for TIL looks as follows (p. 126):

$$\frac{\mathsf{a} :: A \qquad \mathsf{b} :: B \qquad \mathrm{add}(\mathsf{a},\mathsf{b}) = \mathsf{c}}{[A\ {}^0+\ B] \to {}^0\mathsf{c}}\ (+\mathrm{R})$$

Note specifically that the addition function add is applied to what must be first-order objects (otherwise it would not make sense that the result equals to "non-trivialized" c) and the result is then returned back to the "construction" level via Trivialization.[25] (We will return to this topic later in section 5.5.)

[25] Horák's general inference rule (or reduction rule as he calls it) schema looks as follows

$$\frac{P_1; \quad \ldots \quad ; P_n}{C_1 \to C_2} \text{ (rR)}$$

Note the distinct nature of these rules. They just tell us how to rewrite one construction (C_1) into another (C_2), the whole constructional realm ("...construction constructs...") is not involved at all.

Chapter 3

Developing eTIL$_{\mathbb{C}}$

Summary In this chapter we propose explicit rules for handling constructions that try to alleviate some of the shortcomings of the original specification of constructions discussed in the previous chapter. New notions of judgement and type stack are introduced.

3.1 Preliminary notes

In this chapter we focus only on a fragment of TIL containing $*_1$ as its highest order of constructions (more on this topic in section 5.4) and having no improper constructions (we will return to this topic in section 5.3).

3.1.1 Notation

In the upcoming chapters we adopt the following notation changes.

Instead of unwieldy superscript '^{0}' to mark Trivialization we will use boldface font.[1] Hence '^{0}A' becomes '**A**', '[^{0}Succ ^{0}0]' becomes '[**Succ 0**]', etc. Additionally, Trivializations of the most common arithmetic operators and logical connectives will be also written without the superscript. Hence '$^0+$' becomes '$+$', '$^0\supset$' becomes '$\supset$', etc. Same will hold also for equality, so '$^0=$' becomes '$=$'.

Similarly to Tichý, we omit the subscript '$_\tau$' at 'λ_τ' (see 2.2.6) and we will

[1] Tichý utilizes this convention later in *FFL* as well, however, we will use serif font, so Tichý's '**A**' becomes '**A**'.

write Closures of the general form

$$[\lambda w[\lambda t\ [[A\ w]\ t]]]$$

in simplified manner as '$\lambda w \lambda t\ [[A\ w]\ t]$'.[2]

In place of 'α', 'o', 'ι', and '$*_1$' we will write '$\mathbb{N}$', '$\mathbb{B}$', '$\mathbb{I}$', and '$\mathbb{C}$', respectively.

Further, we will use:

- $\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots$ as metavariables for any objects (either constructions or non-constructions),
- $A, B, C, \ldots$ as metavariables for constructions,
- $a, b, c, \ldots$ as metavariables for non-constructions,
- $\alpha, \beta, \gamma, \ldots$ as metavariables for types,
- $x, y, z, \ldots$ as metavariables for (TIL) Variables.

3.1.2 Constructions and construction terms

We will conflate constructions and construction expressions (terms, diagrams,...), since distinguishing between them makes from a technical standpoint little difference. The argument goes as follows:

Tichý states:

> A linguistic expression serves as a symbolic diagram, or picture, of a definite construction.
>
> ([38], p. 203)
>
> A diagram is a graphic representation of a complex object. It need not resemble the object in appearance, but must exhibit the relationships between its various parts. (...) In general, a diagram represents, or stands for, a complex objects, and its parts stand for parts of that object.
>
> ([38], p. 10)

[2] Tichý then proceeds further and simplifies this into '$\lambda w \lambda t . A_{wt}$'. Analogously, he writes Compositions of the form $[[A\ W]\ T]$ as 'A_{WT}'. But we will not adopt this notation here, mainly because the appearance of what is essentially function arguments and variables in the subscript can be often unnecessarily confusing.

> Individual constituents of the expression are understood to represent parts of the referent, and the way the expression is composed from those constituents is understood to represent the way the referent is composed from its parts. An expression is an icon of what it stands for.
>
> ([38], p. 224)

Of course, Tichý's construction terms such as e.g., ([38], p. 203)

$$\lambda w \lambda t . \mathbf{A}_{wt}$$

are themselves nothing more than just linguistic expressions. Their structure, however, is suppose to closely represent the structure of the corresponding construction. As he puts it:

> The construction which is mirrored in the syntactic structure of an expression must be conceived not as a way of *arriving* at the referent but as the referent itself. Reference is a paradigmatic cognitive attitude. Hence, if, in general, cognitive attitudes are attitudes to constructions, so is reference. Constructions must be what we talk *about* and what expressions through which we communicate *stand for*.
>
> ([38], p. 224)

And at the end of *FFL* Tichý claims the following:

> It is true, however, that in the case of arithmetic, or (the 'classical') quantification theory, it is of little *practical* moment whether on insists that a formula expresses a construction which in turn determines an object or whether one thinks of the object as connected with formula directly. The notation of arithmetic as well as the notation of quantification theory was devised in the pre-modern era, when it still went without saying that the purpose of a formula is to represent a definite calculation. Thus every arithmetical or predicate-logic formula does, as a matter of fact, depict a calculation composed of the objects named by the symbols which appear in the formula. Thanks to this perfect fit it matters little whether the practitioner takes himself to be dealing with logical constructions or with the formulas which represent them (provided that he does not traffic in partial functions and does not need to *quantify* over constructions).
>
> ([38], p. 284)

To sum up the last quote up: since mathematical/logical expressions mirror the structure of the corresponding constructions ("a perfect fit"), we can deal only with the former and ignore the latter.[3]

But notice that we can apply the very same argument Tichý uses above to the expressions of TIL as well. After all, they were designed by Tichý himself to be mirroring constructions – to be a perfect fit, as he puts it. Hence, since TIL expressions reflect the structure of the corresponding constructions, we can deal just with the former and disregard the latter (or at least by Tichý's reasoning).

To summarize the whole argument:

P$_1$: If we have a perfect fit between notation and the subject matter, we can deal just with the notation.

P$_2$: We have a perfect fit between TIL notation and TIL subject matter (i.e., constructions).

C: Hence, we can deal just with TIL notation.

Hence, our investigation will be of syntactic nature: constructions will be conflated with the expressions representing them and not regarded as extra-linguistic entities as is common in TIL literature.

3.2 Judgements and type stacks

Before we get to the construction rules themselves, we have to first introduce two new notions that our rules make use of, namely, judgements and type stacks.

[3] Somewhat surprisingly, at the very end of *FFL* ([38], p. 284) Tichý openly admits that his framework makes little difference to a working mathematician or logician. In other words, if I am logician or mathematician that does not make use of partial functions and quantifying over constructions, TIL has virtually nothing to offer me and constructions become just dispensable proxies for formulas.

3.2.1 Judgements

In the previous chapter 2 we have encountered several issues connected with the lack of explicitness of TIL. Specifically, inability to talk directly about first-order objects, lack of explicit typing, and lack of assertoric force:

1. We can say that **5** constructs 5, but we cannot state this fact in TIL.
2. We can say that 5 has type $\mathbb{N}$, but we cannot state this fact in TIL.
3. We can say that $[\supset A\, A]$ constructs *true*, but we cannot state this fact in TIL.

All these limitations follow directly from the simple fact that constructions are procedures and procedures cannot be taken as statements (facts, declarative sentences,...).[4]

In order to amend these shortcomings, we introduce new notion of judgement. Judgements will take the form

$$\mathrm{A} : \alpha$$

which can be read as "object A (either non-construction or construction) has type α".[5] Judgements will occur as premisses and conclusions of our rules.[6]

Additionally, the fact that some construction A of type $\mathbb{C}$ was formed from some object a of type α will be written as '$A : \mathbb{C}^{\alpha}$'. The superscript labels will be used for better book-keeping during derivations, i.e., they will keep track of the way each construction was formed and consequently what it will construct upon execution.

Some examples of judgements: $5 : \mathbb{N}$, $\mathbf{5} : \mathbb{C}^{\mathbb{N}}$, $x : \mathbb{C}^{\mathbb{N}}$, etc.

[4] Recall that the argument was roughly as follows: if $[= [+ \mathbf{5}\ \mathbf{7}]\ \mathbf{12}]$ is to be considered as stating the fact that 5 plus 7 equals 12, then $[= [+ \mathbf{5}\ \mathbf{7}]\ \mathbf{11}]$ must considered as stating the fact 5 plus 7 equals 11, which is clearly incorrect. For more, see section 2.3.2.

[5] Not to confuse with Tichý's earlier notion of match written as 'a : A'. For more, see e.g., [22], [36].

[6] In pre-1988 TIL (i.e., before *FFL*) we can encounter expressions of the form A/α with similar informal meaning as $\mathrm{A} : \alpha$ (recall section 2.3.1). However, the important difference is that in our system such expressions belong to the object language (rules operate on them, etc.), they are not just metalanguage commentary as in the pre-1988 TIL.

3.2.1.1 Judgements and explicit typing

Once we adopt judgements we can properly formalize the information contained in Tichý's "typing tables". For example our judgement

$$succ : (\mathbb{N}\mathbb{N})$$

corresponds to

$$\mathsf{Suc} \quad (\alpha\alpha)$$

which appeared in the table discussed earlier in section 2.3.1. But the crucial difference is that $succ : (\mathbb{N}\mathbb{N})$ is a part of the system itself and hence rules can operate on it. It is not just a metatheory consideration. In other words, the fact that function Suc has type $(\alpha\alpha)$ is only implicitly assumed as a part of TIL metatheory and it cannot be declared directly in TIL.[7]

To demonstrate this further, the information contained in the following "typing" table ([38], p. 67)

entity	*type*	...
0	α	...
1	α	...
Suc	$(\alpha\alpha)$	...
⋮	⋮	⋮

can be formalized in our system as:

$$0 : \mathbb{N}$$
$$1 : \mathbb{N}$$
$$succ : (\mathbb{N}\mathbb{N})$$

Further, e.g, ([38], p. 202):

office	*type*	...
A	$\iota_{\tau\omega}$	...
Ai	$\iota_{\tau\omega}$	...
Sf	$(\iota_{\tau\omega})_{\tau\omega}$	...
P	$o_{\tau\omega}$	...
⋮	⋮	⋮

[7] Sometimes we can encounter notation '$\mathsf{Suc}_{(\alpha\alpha)}$' with similar informal meaning as our '$succ : (\mathbb{N}\mathbb{N})$' (see e.g., [2]). There is, however, important difference, while $\mathsf{Suc}_{(\alpha\alpha)}$ is just a function with subscript, $succ : (\mathbb{N}\mathbb{N})$ is a judgement. In other words, while latter declares that function *succ* has type $(\mathbb{N}\mathbb{N})$, the former presupposes that.

can be formalized as:

$$\begin{aligned}
&\mathit{authorWaverly} : ((\mathbb{IN})\mathbb{W})\\
&\mathit{authorIvanhoe} : ((\mathbb{IN})\mathbb{W})\\
&\mathit{scottFavourite} : ((((\mathbb{IN})\mathbb{W})\mathbb{N})\mathbb{W})\\
&\mathit{authorIvanhoePoet} : ((\mathbb{BN})\mathbb{W})
\end{aligned}$$

Remark Note that even equipped with judgements we are still unable to declare the statements in points (1) and (3) above, only (2). For expressing (1) and (3) we will need to introduce higher-level judgements, which will be discussed in later section 5.1.

3.2.2 Type stacks

The main idea behind the introduction of type stacks is to avoid type-flattening done by Composition, which we discussed earlier in section 2.2.5. Recall that the issue was that no matter how we form Composition (how many constituents it will have), it will always get the flat type $\mathbb{C}$. E.g., constructions [**Succ 0**] and [$+$ **5 7**] would be regarded as constructions of the same type $\mathbb{C}$.

Hence, we introduce new notion of type stack:

$$(\alpha, \beta_1, ..., \beta_m)$$

where α and $\beta_1, ..., \beta_m$ are types of first-order constructions.

More specifically, if α, $\beta_1, ..., \beta_m$ are types of first-order constructions, then $(\alpha, \beta_1, ..., \beta_m)$ is also a type of first-order constructions. Type stack is just a heterogenous list of types (i.e., list type) and it will replace Composition's nonspecific type $\mathbb{C}$. Sometimes we will use '$(\mathbb{C})$' as a metavariable for types and type stacks, i.e., in its place we can imagine type stacks as well as "ordinary" (i.e., non-stack) types.

Some examples: [**Succ 0**] : $(\mathbb{C}^{(\mathbb{NN})}, \mathbb{C}^{\mathbb{N}})$, [$+$ **5 7**] : $(\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})$, [$+$ [$+$ **5 7**] **3**] : $(\mathbb{C}^{(\mathbb{NNN})}, (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{N}})$, etc.

Type stacks will also allow us to type valuation-arrays. For example, the type of the previously discussed valuation-array v_1

$$\begin{pmatrix}
\mathit{true}_1^1 & \mathit{false}_2^1 & \cdots & & \\
1_1^2 & 2_2^2 & 3_3^2 & \cdots & \\
\mathit{Alice}_1^3 & \mathit{Bob}_2^3 & \mathit{Cecil}_3^3 & \mathit{Dana}_4^3 & \cdots
\end{pmatrix}$$

would be: $((\mathbb{B}^1_1, \mathbb{B}^1_2, ...), (\mathbb{N}^2_1, \mathbb{N}^2_2, ...), (\mathbb{I}^3_1, \mathbb{I}^3_2, ...))$. Thus

$$\begin{pmatrix} true^1_1 & false^1_2 & \cdots & & \\ 1^2_1 & 2^2_2 & 3^2_3 & \cdots & \\ Alice^3_1 & Bob^3_2 & Cecil^3_3 & Dana^3_4 & \cdots \end{pmatrix} : ((\mathbb{B}^1_1, \mathbb{B}^1_2, ...), (\mathbb{N}^2_1, \mathbb{N}^2_2, ...), (\mathbb{I}^3_1, \mathbb{I}^3_2, ...))$$

For brevity, we will sometimes write just '(v_1)' instead of

$$\begin{pmatrix} true^1_1 & false^1_2 & \cdots & & \\ 1^2_1 & 2^2_2 & 3^2_3 & \cdots & \\ Alice^3_1 & Bob^3_2 & Cecil^3_3 & Dana^3_4 & \cdots \end{pmatrix}$$

and non-specific '$array_1$' instead of

$$((\mathbb{B}^1_1, \mathbb{B}^1_2, ...), (\mathbb{N}^2_1, \mathbb{N}^2_2, ...), (\mathbb{I}^3_1, \mathbb{I}^3_2, ...))$$

to save some space. Hence, the above will become $(v_1) : array_1$, etc. Additionally, (v) and *array* will be used as metavariables for valuation-arrays and their types, respectively.

3.3 Formation and Execution rules

Aside from the introduction of judgements and type stacks, the last major depart from TIL will be the exclusion of Execution (as well as Double execution) from constructions.

However, that does not mean we get rid of it completely. We reintroduce Execution as one of the two principal rules governing constructions, namely Execution rules (or E-rules for short) and the other being Formation rules (F-rules). The rationale behind introduction of F/E-rules is quite simple: formation and execution seem to be the most basic operations associated with constructions: either we can form them or execute them.[8]

Thus, for each of the remaining construction (i.e., Variable, Trivialization, Composition, and Closure) we will have two types of rules:

- *Formation rules*, telling us how to form constructions (either from non-constructions or other constructions), and

[8] F/E-rules are loosely based on the intuitions behind Gentzen's Introduction and Elimination rules [12] and Martin-Löf's Formation and Computation rules [19].

- *Execution rules*, telling us how these constructions construct.

In other words, each construction will have pair of rules associated with it.

In the following sections we will explain why it makes sense to take formation and execution as the most basic operations associated with constructions, and hence why we base our dual rules on them.

3.3.1 Formation and execution: a motivation

One of the first instances of construction forming can be found in *FFL* [38] on p. 67. Tichý starts there by describing types of the "building material" he will later use

entity	*type*	*description:*
0	α	the number nought
1	α	the number one
Suc	$(\alpha\alpha)$	the successor mapping, i.e., the mapping which takes 0 to 1, 1 to 2, etc.
⋮	⋮	⋮

then he follows:[9]

> Where x is variable ranging over α, the following is an example of first-order construction: [**Suc** x]. If v is a valuation which assigns 0 to x, then this construction v-constructs 1.

Shortly put, construction [**Succ** x] v-constructs 1, if v assigns 0 to x.[10]

So, before we can introduce the construction [**Succ** x] (i.e., before we can form it), we have to be in possession of at least two things:

1. We have to have function Suc and know its type $(\alpha\alpha)$.

2. We have to have Variable x and know it ranges over type α.

[9] We paraphrase here slightly for reader's convenience. The original wording is: "Where x [...] [is variable] ranging over α [...], the following are examples of first-order constructions [...]: [**Suc** x]... (If v is a valuation which assigns 0 to x... [the mentioned construction] v-construct(s) 1...)" ([38], p. 67)

[10] Recall that '[**Suc** x]' is Tichý's notational variant for our '[**Succ** x]'.

Only after we know these things, we can form the Composition [**Succ** x] (via instructions specified in 2.2.5). These considerations are, however, not explicitly reflected in TIL and are left upon the reader and her grasp of TIL metatheory. Hence, although not explicitly stated, the process of getting to [**Succ** x] clearly consists of at least two reasoning steps: (I) typing objects (the building blocks for constructions – even if it is just in our mind) and (II) the formation of the construction itself.

Analogously, there are also at least two things we need to have (some valuation v and previously defined function Suc) and two reasoning steps we have to carry out, if we want to find out what this construction constructs when executed: (I) performing valuation v that assigns 0 to x and (II) knowing that executing **Succ** constructs function Suc and that its application to argument 0 yields number 1. Again, these reasoning steps are left out of TIL and require some insight on the side of the reader.

The most important thing to take away from all this is that all these crucial reasoning steps (specification of types, formation of construction, finding out what it constructs upon execution,...) are done in metatheory, i.e., outside of scope of TIL, where they are just implicitly assumed. Our goal is to make them explicit.

3.3.2 Formation and execution: a closer look

It is important to emphasize that the ideas that we have to form constructions and then execute them to get their results is nothing new injected into TIL. These notions are already present (and implicitly used) in *FFL*, however, never really fully elaborated on upon.

3.3.2.1 Constructions as being formed

The idea of construction formation is clearly stated in the following places in *FFL*:

> **15. Five modes of forming constructions**
>
> ...[W]e must first specify the modes of *forming* constructions (from non-constructions and other constructions). [emphasis author]
>
> ([38], p. 63)
>
> [...]

> This completes the list of the five modes of *forming* constructions. [emphasis author]
>
> ([38], p. 65)

3.3.2.2 Constructions as constructing upon execution

The way that Tichý specifies the behaviour of construction explicitly involves the notion of execution:

> To *carry out* [i.e., execute] ^{0}X, one starts with X and leaves it, so to speak, as it is.
>
> [...]
>
> The construction consisting in *executing*, or *carrying out*, construction X is clearly non other that X itself.
>
> [...]
>
> ...[C]ompound construction which consists in *executing* F, thus obtaining a mapping, then *executing* X, thus obtaining an argument...
>
> [...]
>
> ...[O]ne can *execute* X and go no and *execute* the result.
>
> [emphases author]
>
> ([38], pp. 63–64)

> ...*carrying* the calculation *out* to see which particular number it produces.
>
> ([38], p. 5)
>
> ...construction which consists in *carrying out* F...
>
> ([38], p. 10)

And implicitly it is also present in Tichý's deliberations on Variable construction on p. 60, [38].[11] So it seems safe to say that for Tichý constructions and executions were deeply interconnected notions. This is perhaps best evidenced by the second quote above:

[11] Tichý compares Variable to a machine that yields/produces certain output. Of course, machine does not produce anything on its own, first we have to run it, and hence, execute it.

> The construction consisting in executing, or carrying out, construction X is clearly non other that X itself.
>
> ([38], p. 63)

It could be perhaps paraphrased into: being a construction presupposes execution. Or alternatively, if it cannot be executed, then it is not a construction. Hence, for every construction Tichý seems to be implicitly assuming its execution.[12] So the idea that we have to carry out/execute construction to find out what it constructs seems to be shared between us and Tichý.

Someone might want to argue: if constructions really construct upon execution, why did Tichý never explicitly mention it? Well, in a way he did. Recall the specification of Execution construction. He states that there is no difference between construction of execution of X and construction X itself. This statement makes sense only when we assume that the execution is somehow already assumed with X. Hence, there is no reason to mention it every time explicitly.

Whatever the case, Tichý's constructions require execution in one way or another, that much is indisputable (see Tichý's original specifications of Composition, Trivialization,...). Hence, we say that constructions construct *upon* execution, which, we think, best describes Tichý's position as well. Or it is at least compatible with it.

Remark By execution we do not mean any concrete process localizable in space and time (i.e., concrete execution by some reasoner or machine, which requires certain resources such as time, memory, etc.), but the abstract notion of execution, i.e., how should the construction proceed – what are its steps (execution in abstract sense).

Remark Once we know that e.g., $[+\ \mathbf{5}\ \mathbf{7}]$ constructs upon execution 12, we can speak more generally and omit the explicit mention of execution and state simply that $[+\ \mathbf{5}\ \mathbf{7}]$ constructs 12. But it is important to remember that the second reading presupposes the first one.

[12] That, we think, is also the best way to make sense of the statement "^{1}X is X" we discussed earlier in section 2.2.3.

Remark We do not want to imply that results of constructions somehow depend on the fact whether someone or something executes them or not. Our point is simply that without the notion of execution stating that e.g., constructions (procedure, program, calculation,... i.e., something to be done) **5** constructs (yields, results, returns, produces,...) 5 makes very little sense. And Tichý seems to be well aware of this, as evidenced by his specifications of constructions.

Remark The position that constructions construct upon execution (i.e., that we have to execute them to get their result) is also present in [17].

> ...[R]unning or executing it [construction] to obtain whatever it constructs.
>
> ([17], p. 53)

Unfortunately, the explanation of Execution that follows suffers from the same flaws as Tichý's original specification. Horák writes:

> By execution of a construction C we obtain the same construction ($^1C = C$)...
>
> ([17], p. 53)

So, executing a construction yields its result (by the first quote), but execution of a construction yields back the same constructions (by the second quote – also note the use of the some word "obtain" in both quotes)? So executing construction is something different than execution of construction? How are we to understand this? Surely, these are just two stylistically different ways of stating one and the same thing.

3.3.3 Summary

To sum up, our dual F/E-rule rule scheme is based on the core principles of TIL (i.e., formation and execution of constructions) and tries to deal with them explicitly:

- Formation rules: they tell us how to form new constructions (either from non-constructions or other constructions),
- Execution rules: they tell us how these constructions construct upon execution.

3.4 Construction rules

Now we finally get to the rules themselves. The set of all the rules presented in the upcoming sections 3.4.1–3.4.4 will be denoted as $\mathcal{R}_{\text{eTIL}_{\mathbb{C}}}$ and it will constitute the basis of the system eTIL$_{\mathbb{C}}$.

As already mentioned above, $\mathcal{R}_{\text{eTIL}_{\mathbb{C}}}$ rules will come in pairs and are based on already existing mechanisms in TIL. These mechanisms, however, often lack formal character and also leave a lot of things unsaid. From this perspective, our set of rules just tries to make the informal and implicit parts of TIL formal and explicit, changing as little as possible along the way.

Remark Since the behaviour of these rules is directly lifted from Tichý's own specification discussed in previous chapter 2, they are, in a sense, self-justifying. In other words, we ought to recognize the validity of the rule once we understand it. This approach is similar in kind to that of Martin-Löf:

> The rules should be rules of immediate inference; we cannot further analyse them, but only explain them. However, in the end, no explanation can substitute each individual's understanding.
>
> ([19], p. 13)

3.4.1 Variable rules

$$\frac{\begin{array}{c}((v) : \mathit{array}) \\ \vdots \\ a^i_n : \alpha\end{array}}{{}^{v}|x^i|_n : \mathbb{C}^\alpha}\ \text{VF} \qquad \frac{{}^{v}|x^i|_n : \mathbb{C}^\alpha \qquad (v) : \mathit{array}}{a^i_n : \alpha}\ \text{VE}$$

Remark Recall that '**A**' stands for '^{0}A'. Also vertical ellipsis '⋮' will be used to indicate omitted derivation(s).

Variable Formation rule (VF): We set up new Variable in the following way: first, we assume some valuation-array (v) of type *array*, pick the desired object a^i_n of some type α, and then "abstract" away from it (i.e., we keep the superscript determining the type, but not the subscript determining concrete object),

thus forming a Variable ${}^{v}|x^{i}|_{n}$ of type $\mathbb{C}^{\alpha}$ for the type α, where ^{v} is the initial valuation. E.g., assuming 2 is row of natural numbers, by setting i to 2, we are effectively forming a Variable ${}^{v}|x^{2}|_{n}$ of type $\mathbb{C}^{\mathbb{N}}$ ranging over natural numbers.

Often we will omit the assumption brackets as well as the vertical dots and write just:

$$(v) : array$$

$$\frac{a_n^i : \alpha}{{}^{v}|x^{i}|_{n} : \mathbb{C}^{\alpha}}\ \text{VF}$$

Variable Execution rule (VE): Informally, the rules takes a Variable with coordinates, the corresponding valuation-array, and retrieves the designated object (essentially the reverse of (VF)).

For example:

(3.1)

$$\begin{pmatrix} true_1^1 & false_2^1 & \cdots & & \\ 1_1^2 & 2_2^2 & 3_3^2 & \cdots & \\ Alice_1^3 & Bob_2^3 & Cecil_3^3 & Dana_4^3 & \cdots \end{pmatrix} : array_1$$

$$\frac{1_1^2 : \mathbb{N}}{{}^{v}|x^{2}|_{1} : \mathbb{C}^{\mathbb{N}}}\ \text{VF}$$

(3.2)

$$\frac{{}^{v}|x^{2}|_{1} : \mathbb{C}^{\mathbb{N}} \qquad \begin{pmatrix} true_1^1 & false_2^1 & \cdots & & \\ 1_1^2 & 2_2^2 & 3_3^2 & \cdots & \\ Alice_1^3 & Bob_2^3 & Cecil_3^3 & Dana_4^3 & \cdots \end{pmatrix} : array_1}{1 : \mathbb{N}}\ \text{VE}$$

In the rest of the thesis, we will omit the vertical bars as well as the coordinates in ${}^{v}|x^{i}|_{n}$, ${}^{v}|y^{i}|_{n}$, ${}^{v}|z^{i}|_{n}$,... and write Variables simply as $x, y, z, ...$

3.4.2 Trivialization rules

$$\begin{array}{c}(a:\alpha)\\ \vdots\\ \mathbf{A}:\mathbb{C}^{\alpha}\end{array}\ \text{TF} \qquad \frac{\mathbf{A}:\mathbb{C}^{\alpha}}{a:\alpha}\ \text{TE}$$

Trivialization Formation rule (TF): Informally, assuming a is an object of type α (either function or non-function type), we can form a construction $\mathbf{A}$ of type $\mathbb{C}^{\alpha}$ that constructs this object of said type. Regularly, we will omit the assumption brackets, the rule label as well as the ellipsis and write simply:

$$\begin{array}{c}a:\alpha\\ \mathbf{A}:\mathbb{C}^{\alpha}\end{array}$$

Note that the rule (TF) preserves types of the original objects. In TIL, all Trivializations of first-order objects such as e.g., 5, *Alice*, *true*, etc. would receive the same type $\mathbb{C}$, hence we would lose the information about their original types (i.e., $\mathbb{N}, \mathbb{I}, \mathbb{B}$, respectively).

Trivialization Execution rule (TE): Given that $\mathbf{A}$ is construction of type $\mathbb{C}$ constructing object of type α, i.e., $\mathbb{C}^{\alpha}$, we can execute this construction and get its result, i.e., object a of type α.

For example:

(3.3)

$$\begin{array}{c}5:\mathbb{N}\\ \mathbf{5}:\mathbb{C}^{\mathbb{N}}\end{array}$$

(3.4)

$$\frac{\mathbf{5}:\mathbb{C}^{\mathbb{N}}}{5:\mathbb{N}}\ \text{TE}$$

3.4.3 Composition rules

Composition is the first construction that can be formed only from other constructions.

$$\frac{A : \mathbb{C}^{(\alpha\beta_1...\beta_m)} \qquad B_1 : \mathbb{C}^{\beta_1} \quad ... \quad B_m : \mathbb{C}^{\beta_m}}{[A\,B_1...B_m] : (\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m})}\ \text{CF}$$

$$\frac{[A\,B_1...B_m] : (\mathbb{C}^{\alpha\beta_1...\beta_m}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m}) \qquad \begin{array}{c}(v) : array \\ [A\,B_1...B_m] : (\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m}) \\ [a\,b_1...b_m] : ((\alpha\beta_1...\beta_m), \beta_1, ..., \beta_m)\end{array}}{c : \alpha}\ \text{CE}$$

Composition Formation rule (CF): Informally, the rule takes construction A of a function of type $(\alpha\beta_1...\beta_m)$ and construction(s) $B_1, ..., B_m$ of its argument(s) of type $\beta_1, ..., \beta_m$ and puts it into a Composition $[A\,B_1...B_m]$ of type $(\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m})$.[13]

Remark The meaning of $\mathbb{C}^{(\alpha\beta_1...\beta_m)}$ in $(\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m})$ is the following: construction of type $\mathbb{C}^{(\alpha\beta_1...\beta_m)}$ constructs a function that takes m number of arguments of types $\beta_1, ..., \beta_m$ and returns object of type α. For example, let's have $(\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})$, here $m = 2$, $(\alpha\beta_1\beta_2) = (\mathbb{NNN})$, $\beta_1 = \mathbb{N}$, and $\beta_2 = \mathbb{N}$.[14]

Composition Execution rule (CE): The rule for execution is slightly more tricky. First, recall that Composition actually comprises of two operations: first it executes all its constituents, then it proceeds to perform function application of the results, hence, strictly speaking, Composition consists of two distinct processes carried out subsequently, that are, however, masked as one. However, in order to avoid splitting Composition into two separate rules, we integrate the batch execution step as a second premiss (under the assumption that the Composition was correctly formed, hence all constituent constructions construct what they should given the type signature).

[13] $\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m}$ can be either direct constructions of objects of types $(\alpha\beta_1...\beta_m), \beta_1, ..., \beta_m$ or other type stacks that construct these types of objects after their executions.

[14] For those already familiar with TIL type stacks might seem as unnecessary clutter. This is, however not the case. Their presence is necessitated by our F/E-rule scheme. More specifically, it is a technical device that allows us to track explicitly the structure of Composition constructions.

Thus, the second premiss can be read as follows: "Assuming $[A\,B_1...B_m]$ (and explicitly valuation-array (v) if there are any free Variables, otherwise it is omitted), we can construct $[a\,b_1...b_m]$" (this transition is justified via (CF) and (TE) rules and our starting premisses). Hence we execute all constituents of Composition at once, but not the Composition itself. So, strictly speaking, it is a "halfway-execution", sort of a middle step.

To put it all together, informally, the rule (CE) says the following: take a Composition and, under the assumption it was correctly formed, you execute it to get the function a and argument(s) $b_1,...,b_m$ and apply the former to the later, which gets you c , i.e., the result of applying a to $b_1,...,b_m$.[15]

For example, the formation of Composition $[+\ \mathbf{5}\ \mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})$ and its execution will look as follows (we suppress the type information at the assumptions of (CE) to save some space and gain more readability):

(3.5)

$$\dfrac{\dfrac{\dfrac{\begin{array}{ccc} add : (\mathbb{NNN}) & 5 : \mathbb{N} & 7 : \mathbb{N} \\ + : \mathbb{C}^{(\mathbb{NNN})} & \mathbf{5} : \mathbb{C}^{\mathbb{N}} & \mathbf{7} : \mathbb{C}^{\mathbb{N}} \end{array}}{[+\ \mathbf{5}\ \mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{CF} \qquad [+\ \mathbf{5}\ \mathbf{7}]}{[add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N})}}{12 : \mathbb{N}}\,\text{CE}$$

Note that the initial assumptions $add : (\mathbb{NNN})$, $5 : \mathbb{N}$, and $7 : \mathbb{N}$ of the derivation (3.5) are effectively "withdrawn" during the execution of the Composition and as a result we get back $12 : \mathbb{N}$.[16]

Remark Note that we use '*add*' for representing the addition function and '$+$' for representing the construction that constructs it (specifically, the trivialization of *add*). This is to better distinqusih between first-order objects and first-order constructions. Tichý uses '$+$' for the former and '$^0+$' for the latter, which, I think,

[15] Recall that eTIL$_{\mathbb{C}}$ deals with proper constructions only (and hence, only total functions are allowed). If we take into account improper constructions as well, we have to add condition that whenever any of the constructions $A, B_1, ..., B_m$ is improper, then application of the rule (CE) produces nothing. For more on the topic of improper constructions, see later section 5.3.

[16] The expression $[add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N})$ and others like this one (i.e., appearance of non-constructions inside square brackets, list type with no constructions) cannot appear outside of the (CE) rule, since it is not, strictly speaking, a proper judgement, just an intermediate step of (CE) rule. Hence, it cannot appear e.g., as premise or conclusion in any other place.

can easily lead to severe confusions. Although they look syntactically very similar, they are actually objects of completely different types, $(\mathbb{NNN})$, and $\mathbb{C}^{(\mathbb{NNN})}$, respectively, and even more importantly, the former is a non-construction, while the latter is a construction.

Another example, now with free Variable:

(3.6)

$$\dfrac{\dfrac{\dfrac{add : (\mathbb{NNN}) \quad 5 : \mathbb{N} \quad \dfrac{(v) : array_1 \quad a : \mathbb{N}}{x : \mathbb{C}^{\mathbb{N}}}\,\text{VF}}{+ : \mathbb{C}^{(\mathbb{NNN})} \quad \mathbf{5} : \mathbb{C}^{\mathbb{N}}}\,\text{CF}}{[+\,\mathbf{5}\,x] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})} \qquad \dfrac{(v_7^x) : array_1}{[+\,\mathbf{5}\,x]} \qquad [add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N})}{12 : \mathbb{N}}\,\text{CE}$$

Remark The notation '(v_7^x)' means that Variable x retrieves from valuation-array v the value 7.

Or simpler example:[17]

(3.7)

$$\dfrac{\dfrac{succ : (\mathbb{NN}) \quad 0 : \mathbb{N}}{\mathbf{Succ} : \mathbb{C}^{(\mathbb{NN})} \quad \mathbf{0} : \mathbb{C}^{\mathbb{N}}}\,\text{CF}}{[\mathbf{Succ\ 0}] : (\mathbb{C}^{(\mathbb{NN})}, \mathbb{C}^{\mathbb{N}})} \qquad \dfrac{[\mathbf{Succ\ 0}]}{[succ\ 0] : (\mathbb{NN}, \mathbb{N})}}{1 : \mathbb{N}}\,\text{CE}$$

Remark Derivations such as (3.7) or (3.5) might be at first slightly perplexing. We start with a function and argument(s), then "pack" them into constructions, then "unpack" them, and then finally apply the original function to the original argument(s). Aren't we back at were we initially started? What was the point of it all? Isn't it circular?

[17] Note that the $[succ\ 0]$ is again the result of "half-executing" [**Succ 0**], as it were, i.e., a function ready to be applied to its argument, which constitutes the second half of the execution of Composition [**Succ 0**].

Well, not really. Take e.g., derivation (3.7). Note that we start with *succ* : $(\mathbb{N}\mathbb{N})$ and $0 : \mathbb{N}$ and we end up with the application of the former to the latter, i.e., $[succ\ 0] : ((\mathbb{N}\mathbb{N}), \mathbb{N})$ – so in a sense the derivation is used, among other things, to glue together the function with its argument(s).[18] From this perspective, we can view constructions as providing a (non-functional, hyperintensional) framework for working with functions (and other first-order objects).[19]

Remark Note that execution rules are not, strictly speaking, computation rules. E.g., (CE) rule tells us only how Composition generally works, not how each separate function that can enter into it works. Recall e.g., derivation (3.5) – sure, we know that if we apply addition function to 5 and 7 we get 12, but we have never properly specified it in our system (we will return to this topic in section 5.5).

3.4.4 Closure rules

Closure, similarly to Composition, can be formed only from other constructions. And although Tichý allowed formation of Closure from any construction, for simplicity we will consider formation of Closure from Composition constructions only.

$$\frac{\begin{array}{cccc} & B_1 : \mathbb{C}^{\beta_1} & & B_m : \mathbb{C}^{\beta_m} \\ [A\,B_1 \ldots B_m] : (\mathbb{C}^{(\alpha\beta_1\ldots\beta_m)}, \mathbb{C}^{\beta_1}, \ldots, \mathbb{C}^{\beta_m}) & x_1 : \mathbb{C}^{\beta_1} & \cdots & x_m : \mathbb{C}^{\beta_m} \end{array}}{\lambda x_1 \ldots x_m\, [A\,x_1 \ldots x_m] : \mathbb{C}^{(\alpha\beta_1\ldots\beta_m)}}\ \text{ClF}$$

$$\frac{\lambda x_1 \ldots x_m\, [A\,x_1 \ldots x_m] : \mathbb{C}^{(\alpha\beta_1\ldots\beta_m)}}{f : (\alpha\beta_1\ldots\beta_m)}\ \text{ClE}$$

Closure Formation rule (ClF): Informally, the rules takes Composition $[A\,B_1 \ldots B_m]$ of type $(\mathbb{C}^{(\alpha\beta_1\ldots\beta_m)}, \mathbb{C}^{\beta_1}, \ldots, \mathbb{C}^{\beta_m})$, Variables constructing objects of the same type as the constructions of arguments that are to be abstracted away from, replaces

[18] Similar mechanisms for "explicit application" can be e.g., also found in Martin-Löf's CTT, specifically the operator Ap, see [19].

[19] This approach directly follows from the fundamental assumption of TIL that all calculations (computations,...) are constructions, thus we cannot have judgements such as e.g., $5 + 7 : \mathbb{N}$.

all the desired constructions of arguments by these Variables, and then finally λ-binds them. The type of resulting Closure is then the leftmost type in the leftmost type stack of the original Composition.[20]

Closure Execution rule (ClE): Briefly put, Closure upon execution returns the object (a function f) indicated by the superscript on the type $\mathbb{C}$, modulo the arguments for the constructed function that are already present.[21]

For example, we can form Closure from [**Succ 0**] in the following way:

(3.8)

$$\dfrac{\begin{array}{cc}\vdots & \mathbf{0} : \mathbb{C}^{\mathbb{N}} \\ [\mathbf{Succ\ 0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}) & x : \mathbb{C}^{\mathbb{N}}\end{array}}{\lambda x\, [\mathbf{Succ}\, x] : \mathbb{C}^{(\mathbb{N}\mathbb{N})}}\ \text{ClF}$$

The execution of the Closure $\lambda x\, [\mathbf{Succ}\, x]$ will be then:

(3.9)

$$\dfrac{\begin{array}{c}\vdots \\ \lambda x\, [\mathbf{Succ}\, x] : \mathbb{C}^{(\mathbb{N}\mathbb{N})}\end{array}}{succ : (\mathbb{N}\mathbb{N})}\ \text{ClE}$$

Another example:

(3.10)

$$\dfrac{\dfrac{\begin{array}{ccc} add : (\mathbb{N}\mathbb{N}\mathbb{N}) & 1 : \mathbb{N} & \dfrac{(v) : array_1}{\dfrac{a : \mathbb{N}}{x : \mathbb{C}^{\mathbb{N}}}\ \text{VF}} \\ + : \mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})} & \mathbf{1} : \mathbb{C}^{\mathbb{N}} & \end{array}}{[+\,\mathbf{1}\, x] : (\mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\ \text{CF} \qquad x : \mathbb{C}^{\mathbb{N}}}{\lambda x [+\,\mathbf{1}\, x] : \mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}}\ \text{ClF}$$

[20] There are exceptions to this when we are dealing with composed functions. For more, see later section 4.2.2.

[21] The reasons for including the second clause will become clear later, see e.g., derivation (4.1).

Remark Note that the second premise in (ClF) rule is, strictly speaking, unnecessary, since we already know that Variable x constructs objects of type $\mathbb{N}$. Consequently, no assumptions are needed, so the premise $x : \mathbb{C}^{\mathbb{N}}$ just reiterates what we already know in order the (ClF) rule can proceed on.

Hence, when unnecessary (i.e., when we already poses all the information required) we will omit the right premise of (ClF). Hence the above derivation would become:

(3.11)

$$\dfrac{\dfrac{\dfrac{add : (\mathbb{N}\mathbb{N}\mathbb{N}) \quad 1 : \mathbb{N} \quad \dfrac{(v) : array_1 \quad a : \mathbb{N}}{x : \mathbb{C}^{\mathbb{N}}}\,\text{VF}}{+ : \mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})} \quad \mathbf{1} : \mathbb{C}^{\mathbb{N}}}\,\text{CF}}{[+\,\mathbf{1}\,x] : (\mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{ClF}}{\lambda x [+\,\mathbf{1}\,x] : \mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}}$$

In this particular case the additional premise $x : \mathbb{C}^{\mathbb{N}}$ was not essential, because we already have in our disposition Variable x of type $\mathbb{C}^{\mathbb{N}}$.

Example of Composition with Closure:

(3.12)

$$\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\lambda x\,[\mathbf{Succ}\,x] : \mathbb{C}^{(\mathbb{N}\mathbb{N})} \quad \mathbf{1} : \mathbb{C}^{\mathbb{N}}}\,\text{CF}}{[\lambda x\,[\mathbf{Succ}\,x]\,\mathbf{1}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})} \quad [\lambda x\,[\mathbf{Succ}\,x]\,\mathbf{1}] \quad [succ\ 1] : ((\mathbb{N}\mathbb{N}), \mathbb{N})}\,\text{CE}}{2 : \mathbb{N}}$$

Remark Should the necessity arise we could additionally introduce rules for formation (and execution) of Closure from other constructions as well. E.g., in the case of Variable the rules might go as follows:

$$\dfrac{\dfrac{a : \alpha}{x : \mathbb{C}^{\alpha}}}{\lambda x\, x : \mathbb{C}^{(\alpha\alpha)}}\,\text{ClFV}$$

$$\dfrac{\lambda x\, x : \mathbb{C}^{(\alpha\alpha)}}{a : (\alpha\alpha)}\,\text{ClEV}$$

For example:

(3.13)

$$\dfrac{\begin{array}{c}\vdots\\ x : \mathbb{C}^{\mathbb{N}}\end{array}}{\lambda x\, x : \mathbb{C}^{(\mathbb{N}\mathbb{N})}}\ \text{ClFV}$$

(3.14)

$$\dfrac{\begin{array}{c}\vdots\\ \lambda x\, x : \mathbb{C}^{(\mathbb{N}\mathbb{N})}\end{array}}{id : (\mathbb{N}\mathbb{N})}\ \text{ClEV}$$

3.5 Concluding remarks on F/E-rules

What construction constructs is determined by the parts it was formed from. This, of course, makes intuitive sense. We cannot hope to construct anything, if we do not have the appropriate building blocks. E.g., we cannot construct number from truth values only, what we need are other numbers and/or functions returning numbers.

What is the general relationship between F/E-rules? Consider e.g., the derivation:

$$\dfrac{\dfrac{5 : \mathbb{N}}{\mathbf{5} : \mathbb{C}^{\mathbb{N}}}\ \text{TF}}{5 : \mathbb{N}}\ \text{TE}$$

It seems rather pointless, but—given what we said earlier—that is to be expected, since there isn't much we can construct with just 5. We need other building material as well to construct something more interesting. Similarly, we cannot build much with just one Lego brick, we need others to create something "noteworthy".

To demonstrate this point consider only slightly more complex derivation (3.15). Specifically, we add two new building blocks: addition function and one more number.

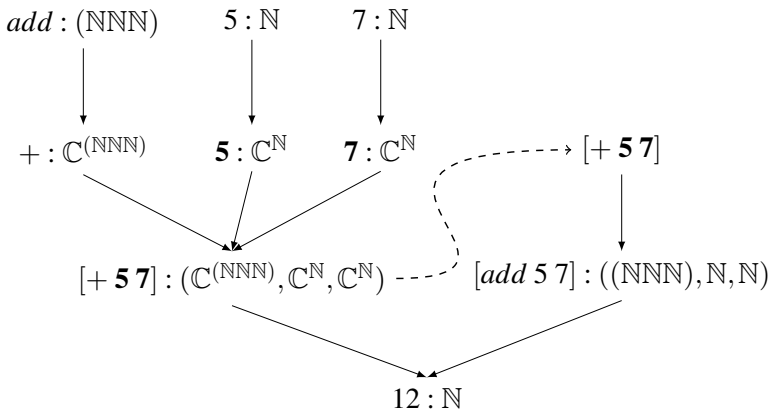

Figure 3.1: Derivation as manual for construction

(3.15)

$$\dfrac{\dfrac{\begin{array}{ccc} add : (\mathbb{NNN}) & 5 : \mathbb{N} & 7 : \mathbb{N} \\ + : \mathbb{C}^{(\mathbb{NNN})} & \mathbf{5} : \mathbb{C}^{\mathbb{N}} & \mathbf{7} : \mathbb{C}^{\mathbb{N}} \end{array}}{[+\,\mathbf{5}\,\mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{CF} \qquad \begin{array}{c} [+\,\mathbf{5}\,\mathbf{7}] \\ [add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N}) \end{array}}{12 : \mathbb{N}}\,\text{CE}$$

Suddenly, we have constructed something new, specifically number 12. In our Lego bricks analogy, we can think about our derivations as instruction manuals for putting these bricks together and thus creating something new. Constructions serving as the studs and tubes, i.e., the interlocking mechanism that keeps pieces "glued" together – see schema 4.1 where this is showed more explicitly. So what is the relationship between F/E-rules? Briefly put, F-rules supply material and E-rules glue it together.

3.5.1 F/E-derivations

Depending on the last rule used we will distinguish between F-derivation and E-derivation:

- F-derivation – derivation that ends up with formation rule (last rule applied is F-rule)

- E-derivation – derivation that ends up with execution rule (last rule applied is E-rule)

For example:

(3.16)

$$\dfrac{\dfrac{\begin{array}{ccc} add : (\mathbb{NNN}) & 5 : \mathbb{N} & 7 : \mathbb{N} \\ + : \mathbb{C}^{(\mathbb{NNN})} & \mathbf{5} : \mathbb{C}^{\mathbb{N}} & \mathbf{7} : \mathbb{C}^{\mathbb{N}} \end{array}}{[+\,\mathbf{5}\,\mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{CF} \qquad [add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N})}{12 : \mathbb{N}}\,\text{CE}$$

is E-derivation, while

(3.17)

$$\dfrac{\begin{array}{ccc} add : (\mathbb{NNN}) & 5 : \mathbb{N} & 7 : \mathbb{N} \\ + : \mathbb{C}^{(\mathbb{NNN})} & \mathbf{5} : \mathbb{C}^{\mathbb{N}} & \mathbf{7} : \mathbb{C}^{\mathbb{N}} \end{array}}{[+\,\mathbf{5}\,\mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{CF}$$

is F-derivation.

Remark We would like to reemphasize that with F/E-rules we do not derive constructions from other constructions, but judgements (about constructions or non-constructions) from other judgements (about constructions or non-constructions).

Often, however, we will be sloppy and say things like "$\mathbf{5} : \mathbb{C}^{\mathbb{N}}$ constructs $5 : \mathbb{N}$". Strictly speaking, this is incorrect, since $\mathbf{5} : \mathbb{C}^{\mathbb{N}}$ is a judgement, not a construction, hence it cannot construct anything. The proper way of phrasing it would be: "from $\mathbf{5} : \mathbb{C}^{\mathbb{N}}$ is derivable $5 : \mathbb{N}$ via (TE) rule", "$\mathbf{5} : \mathbb{C}^{\mathbb{N}}$ yields upon application of execution rule $5 : \mathbb{N}$", "$\mathbf{5} : \mathbb{C}^{\mathbb{N}}$ entails $5 : \mathbb{N}$" or alternatively just "**5** constructs 5".

Remark Note that '$\mathbb{C}^{\mathbb{N}}$' can be read as both "type of construction that was formed from $\mathbb{N}$" and "type of construction that constructs $\mathbb{N}$". It depends on context in the derivation, i.e., whether the corresponding construction was just formed using F-rule, or it is about to be executed by applying E-rule.

Remark Our $\mathcal{R}_{\text{eTIL}_{\mathbb{C}}}$ rules of eTIL$_{\mathbb{C}}$ can be perhaps thought of as some kind of metarules for TIL. We are not against such view, however, much more suitable would be to think of them as rules that try to capture the mechanisms behind TIL,

and thus provide new way of looking at TIL itself via the rules that govern it. In other words, we are interested in the logic behind TIL and our system tries to bring it into the foreground. That being said, they can be considered as metarules for TIL in the sense that they deal explicitly with information that was previously accessible only in metatheory.

Chapter 4

Case studies: eTIL$_{\mathbb{C}}$

Summary This chapter will be devoted to concrete examples of application of eTIL$_{\mathbb{C}}$ system, demonstrating it in practice. Specifically, we will be showing how various TIL analyses are to be translated into our system.

4.1 Natural language analysis

So far we have been interested almost exclusively in analysis of simple mathematical examples. However, TIL as well as our system of rules can deal with empirical/natural language examples as well.

In [9] is utilized so called three-step method of logical analysis of language, which consists in:

1. type-theoretical analysis (type assignment),
2. synthesis (construction formation), and
3. type-checking,

respectively. For example, the three-stage analysis of the English sentence

"Alice is a girl."

would go roughly as follows:[1]

[1] We retain the original notation from [9], p. 78, where '$(o\iota)_{\tau\omega}$' stands for the type $(((o\iota)\tau)\omega)$ and '$^0isGirl_{wt}$' for the construction $[[^0isGirl\ w]\ t]$. In TIL, τ represents type of real numbers, ω type of possible worlds.

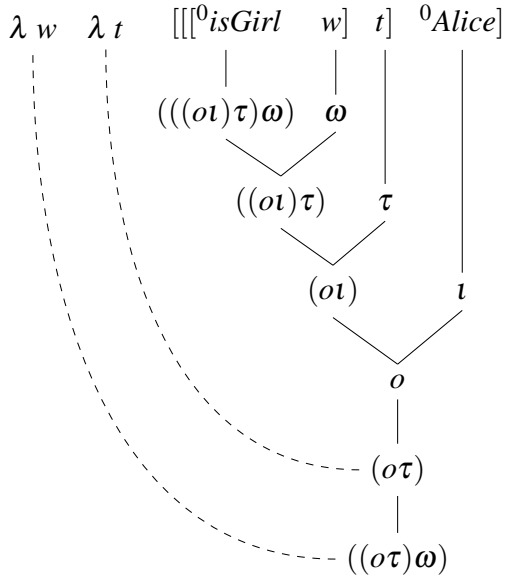

Figure 4.1: Type-checking tree

1. Type assignment	**2. Synthesis**	**3. Type checking**
$Alice/\iota$ $isGirl/(o\iota)_{\tau\omega}$	$\lambda w \lambda t\ [^0isGirl_{wt}\ ^0Alice]$	See Figure 4.1.

At first glimpse it is all quite straightforward, however, these three steps are rather separate and informal.

For example, (Q1) what exactly are the expressions '$Alice/\iota$' and '$isGirl/(o\iota)_{\tau\omega}$'? Their intuitive meaning is clear enough, but it is far from obvious what they are in correspondence to TIL language as a whole. As it is, in [9] they are just metalanguage commentary.

Secondly, (Q2) how exactly do we perform the synthesis? No explicit rules are given to guide the formation of the corresponding construction.

Thirdly, (Q3) what exactly is the type-checking tree figure? No explicit definition is given.

And what is possibly the most important, all three steps are performed in isolation, in its own respective sub-systems, so to speak. We have a system for typing objects, a system for forming constructions, and a system for building type-checking trees.

Our system combines all these informal sub-systems into a single formal one. For demonstration, we show that the whole original analysis above can be recreated (with lot more additional information) in our system eTIL$_{\mathbb{C}}$ as follows (we use $\mathbb{N}$ as a stand-in for the original type τ, i.e., type of real numbers; also note the addition of a new type of possible worlds $\mathbb{W}$, originally denoted as 'ω'):[2]

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{isGirl} : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}{\mathbf{isGirl} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}} \quad \dfrac{\mathit{world} : \mathbb{W}}{\mathbf{w} : \mathbb{C}^{\mathbb{W}}}}{[\mathbf{isGirl}\ \mathbf{w}] : (\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}})}\,\mathrm{CF} \quad \dfrac{\mathit{time} : \mathbb{N}}{\mathbf{t} : \mathbb{C}^{\mathbb{N}}}}{[[\mathbf{isGirl}\ \mathbf{w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\,\mathrm{CF} \quad \dfrac{\mathit{Alice} : \mathbb{I}}{\mathbf{Alice} : \mathbb{C}^{\mathbb{I}}}}{[[[\mathbf{isGirl}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Alice}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}})}\,\mathrm{CF} \quad \dfrac{\mathbf{t}}{t : \mathbb{C}^{\mathbb{N}}}}{\lambda t\ [[[\mathbf{isGirl}\ \mathbf{w}]\ t]\ \mathbf{Alice}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\,\mathrm{ClF} \quad \dfrac{\mathbf{w}}{w : \mathbb{C}^{\mathbb{W}}}}{\lambda w \lambda t\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\,\mathrm{ClF}}{\mathit{isGirlAlice} : ((\mathbb{B}\mathbb{N})\mathbb{W})}\,\mathrm{ClE} \tag{4.1}$$

Task of all three of the above mentioned sub-systems (i.e., type assignment, synthesis, and type checking) are effectively done by a single derivation in our system. Furthermore, it also provides answers to our three questions from earlier (i.e., (A1) judgements, (A2) with $\mathcal{R}_{\mathrm{eTIL}_{\mathbb{C}}}$, (A3) a derivation tree constructed via $\mathcal{R}_{\mathrm{eTIL}_{\mathbb{C}}}$).

Remark Recall that '$\lambda w \lambda t\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}$' stands for '$[\lambda w[\lambda t\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}]]] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}$', but we omit these extra brackets to simplify the notation. Also remember that the form of the type stack in the previously formed judgement $[[[\mathbf{isGirl}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Alice}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}})$ is $(((\mathbb{C},\mathbb{C}),\mathbb{C}),\mathbb{C})$, the superscripts are just labels that help us keep track of additional information.

Now assume that Alice is indeed a girl at world **w** and time **t**. Then the initial Composition [[[**isGirl w**] **t**] **Alice**] would construct:

$$\dfrac{\begin{matrix}\vdots \\ [[[\mathbf{isGirl}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Alice}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}})\end{matrix} \quad \vdots}{\mathit{true} : \mathbb{B}}\,\mathrm{CE} \tag{4.2}$$

[2] About the type of the conclusion: because one of the arguments for the constructed function is already present, specifically *Alice* of type $\mathbb{I}$ constructed via **Alice** of type $\mathbb{C}^{\mathbb{I}}$, we construct just function of type $((\mathbb{B}\mathbb{N})\mathbb{W})$ instead of $(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})$ (recall section 3.4.4).

Of course, we could get to the same result if we compose the Closure from the previous derivation with newly supplanted **w** and **t** (i.e., *world*, *time*, respectively – in both of which Alice is still a girl):

(4.3)

$$\dfrac{\dfrac{\dfrac{\vdots \quad \lambda w\lambda t\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \qquad \begin{matrix} world : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{matrix}}{[\lambda w\lambda t\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}]]\ \mathbf{w}] : (\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}})}\ \text{CF} \qquad \begin{matrix} time : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{matrix}}{[[\lambda w\lambda t\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}]]\ \mathbf{w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\ \text{CF} \qquad \vdots}{true : \mathbb{B}}\ \text{CE}$$

Now what would the Composition containing free Variables

$$[[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}})$$

construct? That would depend on specific valuation-array (v). For example, assuming it supplants our earlier *world* and *time*, we get again:

(4.4)

$$\dfrac{\begin{matrix} \vdots \\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}}) \end{matrix} \qquad \begin{matrix} (v^{w,t}_{world,time}) : array_1 \\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}] \\ [isGirlAlice\ world\ time] : (((\mathbb{B}\mathbb{N})\mathbb{W}), \mathbb{W}, \mathbb{N}) \end{matrix}}{true : \mathbb{B}}\ \text{CE}$$

Remark The notation '$(v^{w,t}_{world,time})$' can be read as "valuation v assigns to Variable w value *world* and to t value *time*" (or alternatively, Variables w, t retrieve from valuation array (v) values, *world* and *time*, respectively).

4.2 Construction formation

In this section we will demonstrate explicit formation of various constructions from *FFL* in eTIL$_{\mathbb{C}}$ system.

4.2.1 Example 1

Construction ([38], p. 211)

$$\lambda w \lambda t.\mathsf{U}_{wt}\mathsf{S}$$

which is equivalent in our notation to

$$\lambda w \lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]$$

will get the following F-derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c} isPoet : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \\ \mathbf{isPoet} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \end{array} \quad \begin{array}{c} world : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{array}}{[\mathbf{isPoet\ w}] : (\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}})}\text{CF} \quad \begin{array}{c} time : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{array}}{[[\mathbf{isPoet\ w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\text{CF} \quad \begin{array}{c} Scott : \mathbb{I} \\ \mathbf{Scott} : \mathbb{C}^{\mathbb{I}} \end{array}}{[[[\mathbf{isPoet\ w}]\ \mathbf{t}]\ \mathbf{Scott}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}})}\text{CF} \quad \begin{array}{c} \mathbf{t} \\ t : \mathbb{C}^{\mathbb{N}} \end{array}}{\lambda t\ [[[\mathbf{isPoet\ w}]\ t]\ \mathbf{Scott}] : \mathbb{C}^{((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\text{ClF} \quad \begin{array}{c} \mathbf{w} \\ w : \mathbb{C}^{\mathbb{W}} \end{array}}{\lambda w \lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\text{ClF} \tag{4.5}$$

Notice the often repeated processes of first forming Compositions of the form [[... **w**] **t**] via application of (CF) rules and then forming the corresponding Closures of the form $\lambda w \lambda t$ [[... w] t] via application of (ClF) rules. Sometimes we will condense these derivation steps into a single one and indicate this compression with double inference line '==========='. Hence, e.g.,

$$\dfrac{\dfrac{\begin{array}{c} isPoet : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \\ \mathbf{isPoet} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \end{array} \quad \begin{array}{c} world : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{array}}{[\mathbf{isPoet\ w}] : (\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}})}\text{CF} \quad \begin{array}{c} time : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{array}}{[[\mathbf{isPoet\ w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\text{CF}$$

becomes

$$\begin{array}{c} \begin{array}{c} isPoet : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \\ \mathbf{isPoet} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \end{array} \quad \begin{array}{c} world : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{array} \quad \begin{array}{c} time : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{array} \\ \hline\hline [[\mathbf{isPoet\ w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}) \end{array}\ \text{CF}$$

However, it is important to remember that this is just syntactic “short-cut” and not a proper derivation step. Additionally, we will also omit redundant premises for both Composition and Closure formations (see e.g., later derivation (4.8)).

E.g., the whole above derivation (4.5) can be shortened (assuming the mentioned sub-derivations were already carried out) to

(4.6)

$$\dfrac{\dfrac{\dfrac{\begin{matrix} \mathit{isPoet} : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \\ \mathbf{isPoet} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \end{matrix} \quad \begin{matrix} \mathit{world} : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{matrix} \quad \begin{matrix} \mathit{time} : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{matrix}}{[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\ \text{CF} \quad \begin{matrix} \mathit{Scott} : \mathbb{I} \\ \mathbf{Scott} : \mathbb{C}^{\mathbb{I}} \end{matrix}}{[[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Scott}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}})}\ \text{CF} \quad \begin{matrix} \mathbf{t} \\ t : \mathbb{C}^{\mathbb{N}} \end{matrix} \quad \begin{matrix} \mathbf{w} \\ w : \mathbb{C}^{\mathbb{W}} \end{matrix}}{\lambda w \lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\ \text{ClF}$$

When is the double inference line shorthand used for Composition or for Closure formation will be obvious from the context of the derivation as well as from the accompanying rule labels. E.g., in the above derivation, the first double line condenses two applications of (CF) rule into a single step, the second one condenses two applications of (ClF) rule into a single step.

To slim the derivation even further down, we can optionally omit the construction types to get

(4.7)

$$\dfrac{\dfrac{\dfrac{\begin{matrix} \mathit{isPoet} : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \\ \mathbf{isPoet} \end{matrix} \quad \begin{matrix} \mathit{world} : \mathbb{W} \\ \mathbf{w} \end{matrix} \quad \begin{matrix} \mathit{time} : \mathbb{N} \\ \mathbf{t} \end{matrix}}{[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]}\ \text{CF} \quad \begin{matrix} \mathit{Scott} : \mathbb{I} \\ \mathbf{Scott} : \mathbb{C}^{\mathbb{I}} \end{matrix}}{[[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Scott}]}\ \text{CF} \quad \begin{matrix} \mathbf{t} \\ t \end{matrix} \quad \begin{matrix} \mathbf{w} \\ w \end{matrix}}{\lambda w \lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]}\ \text{ClF}$$

But this is generally not advised, since types carry crucial information. Also it can lead to unnecessary confusions (rules operate on judgements, not on constructions). However, sometimes it can be useful just for illustrative purposes.

4.2.2 Example 2

Construction ([38], p. 212)

$$\lambda w \lambda t . \neg [\lambda w \lambda t . \mathbf{U}_{wt} \mathbf{S}]_{wt}$$

which is equivalent in our notation to

$$\lambda w \lambda t \; [\neg [[\lambda w \lambda t \; [[[\mathbf{isPoet}\; w]\; t]\; \mathbf{Scott}]]\; w]\; t]$$

is established via the F-derivation (4.9) (or its compressed version (4.10)).

And finally, the most condensed version, where we omit even the reoccurring premises for (CF) and (ClF) rules (specifically, $\begin{matrix} world : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{matrix}$, $\begin{matrix} time : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{matrix}$, $\begin{matrix} \mathbf{t} \\ t : \mathbb{C}^{\mathbb{N}} \end{matrix}$, and $\begin{matrix} \mathbf{w} \\ w : \mathbb{C}^{\mathbb{W}} \end{matrix}$):

(4.8)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{isPoet : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}{\mathbf{isPoet} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}} \quad \dfrac{world : \mathbb{W}}{\mathbf{w} : \mathbb{C}^{\mathbb{W}}} \quad \dfrac{time : \mathbb{N}}{\mathbf{t} : \mathbb{C}^{\mathbb{N}}}}{[[\mathbf{isPoet}\; \mathbf{w}]\; \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\,\text{CF} \quad \dfrac{Scott : \mathbb{I}}{\mathbf{Scott} : \mathbb{C}^{\mathbb{I}}}}{[[[\mathbf{isPoet}\; \mathbf{w}]\; \mathbf{t}]\; \mathbf{Scott}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}})}\,\text{CF} \quad \dfrac{\mathbf{t}}{t : \mathbb{C}^{\mathbb{N}}} \quad \dfrac{\mathbf{w}}{w : \mathbb{C}^{\mathbb{W}}}}{\lambda w \lambda t \; [[[\mathbf{isPoet}\; w]\; t]\; \mathbf{Scott}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\,\text{ClF} \quad neg : (\mathbb{B}\mathbb{B})}{[[\lambda w \lambda t \; [[[\mathbf{isPoet}\; w]\; t]\; \mathbf{Scott}]\; \mathbf{w}]\; \mathbf{t}] : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\,\text{CF} \quad \neg : \mathbb{C}^{(\mathbb{B}\mathbb{B})}}{\neg [[\lambda w \lambda t \; [[[\mathbf{isPoet}\; w]\; t]\; \mathbf{Scott}]\; \mathbf{w}]\; \mathbf{t}] : (\mathbb{C}^{(\mathbb{B}\mathbb{B})}, ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}))}\,\text{CF}}{\lambda w \lambda t \; [\neg [[\lambda w \lambda t \; [[[\mathbf{isPoet}\; w]\; t]\; \mathbf{Scott}]\; w]\; t]] : (\mathbb{C}^{(\mathbb{B}\mathbb{B})}, \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})})}\,\text{ClF}$$

Remark Note that the final construction, somewhat unexpectedly, has type stack $(\mathbb{C}^{(\mathbb{B}\mathbb{B})}, \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})})$. The reason for this is that although the construction constructs a function from w and t to truth values, it still has to negate the resulting value accordingly to the $\neg$, hence the additional $\mathbb{C}^{(\mathbb{B}\mathbb{B})}$. It cannot be just $\mathbb{C}^{(\mathbb{B}\mathbb{B})}$, as might be expected from our initial specification of (ClE) rule, because the resulting function still expects the world and time arguments, not just another truth value. Essentially, we are constructing a composed function ("get world and time, return truth value, and then negate it"), which necessitates the appearance of type stacks.

$$\frac{\begin{array}{cc} \mathit{isPoet}:(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) & \mathit{world}:\mathbb{W} \\ \mathbf{isPoet}:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} & \mathbf{w}:\mathbb{C}^{\mathbb{W}} \end{array}}{[\mathbf{isPoet}\ \mathbf{w}]:(\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}})}\ \mathrm{CF}$$

$$\frac{[\mathbf{isPoet}\ \mathbf{w}]:(\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}) \quad \begin{array}{c}\mathit{time}:\mathbb{N}\\ \mathbf{t}:\mathbb{C}^{\mathbb{N}}\end{array}}{[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}})}\ \mathrm{CF}$$

$$\frac{[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}) \quad \begin{array}{c}\mathit{Scott}:\mathbb{I}\\ \mathbf{Scott}:\mathbb{C}^{\mathbb{I}}\end{array}}{[[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Scott}]:(((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}),\mathbb{C}^{\mathbb{I}})}\ \mathrm{CF}$$

$$\frac{[[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Scott}]:(((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}),\mathbb{C}^{\mathbb{I}}) \quad \begin{array}{c}\mathbf{t}\\ t:\mathbb{C}^{\mathbb{N}}\end{array}}{\lambda t\ [[[\mathbf{isPoet}\ \mathbf{w}]\ t]\ \mathbf{Scott}]:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\ \mathrm{ClF}$$

$$\frac{\lambda t\ [[[\mathbf{isPoet}\ \mathbf{w}]\ t]\ \mathbf{Scott}]:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \quad \begin{array}{c}\mathbf{w}\\ w:\mathbb{C}^{\mathbb{W}}\end{array}}{\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\ \mathrm{ClF}$$

$$\frac{\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \quad \begin{array}{c}\mathit{world}:\mathbb{W}\\ \mathbf{w}:\mathbb{C}^{\mathbb{W}}\end{array}}{[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]:(\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}})}\ \mathrm{CF}$$

$$\frac{[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]:(\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}) \quad \begin{array}{c}\mathit{time}:\mathbb{N}\\ \mathbf{t}:\mathbb{C}^{\mathbb{W}}\end{array}}{[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}})}\ \mathrm{CF}$$

$$\frac{[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}) \quad \begin{array}{c}\mathit{neg}:(\mathbb{B}\mathbb{B})\\ \neg:\mathbb{C}^{(\mathbb{B}\mathbb{B})}\end{array}}{\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:(\mathbb{C}^{\mathbb{B}\mathbb{B}},((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}))}\ \mathrm{CF}$$

$$\frac{\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:(\mathbb{C}^{\mathbb{B}\mathbb{B}},((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}})) \quad \begin{array}{c}\mathbf{t}\\ t:\mathbb{C}^{\mathbb{N}}\end{array}}{\lambda t\ [\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ t]]:(\mathbb{C}^{(\mathbb{B}\mathbb{B})},\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})})}\ \mathrm{ClF}$$

$$\frac{\lambda t\ [\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ t]]:(\mathbb{C}^{(\mathbb{B}\mathbb{B})},\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}) \quad \begin{array}{c}\mathbf{w}\\ w:\mathbb{C}^{\mathbb{W}}\end{array}}{\lambda w\lambda t\ [\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ w]\ t]]:(\mathbb{C}^{\mathbb{B}\mathbb{B}},\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})})}\ \mathrm{ClF} \tag{4.9}$$

$$\frac{\begin{array}{ccc} \mathit{isPoet}:(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) & \mathit{world}:\mathbb{W} & \mathit{time}:\mathbb{N} \\ \mathbf{isPoet}:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} & \mathbf{w}:\mathbb{C}^{\mathbb{W}} & \mathbf{t}:\mathbb{C}^{\mathbb{N}} \end{array}}{[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}})}\ \mathrm{CF}$$

$$\frac{[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}) \quad \begin{array}{c}\mathit{Scott}:\mathbb{I}\\ \mathbf{Scott}:\mathbb{C}^{\mathbb{I}}\end{array}}{[[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Scott}]:(((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}),\mathbb{C}^{\mathbb{I}})}\ \mathrm{CF}$$

$$\frac{[[[\mathbf{isPoet}\ \mathbf{w}]\ \mathbf{t}]\ \mathbf{Scott}]:(((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}),\mathbb{C}^{\mathbb{I}}) \quad \begin{array}{c}\mathbf{t}\\ t:\mathbb{C}^{\mathbb{N}}\end{array} \quad \begin{array}{c}\mathbf{w}\\ w:\mathbb{C}^{\mathbb{W}}\end{array}}{\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}\ \mathrm{ClF}$$

$$\frac{\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]:\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}}{[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}})}\ \mathrm{CF}$$

$$\frac{[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}) \quad \begin{array}{c}\mathit{neg}:(\mathbb{B}\mathbb{B})\\ \neg:\mathbb{C}^{(\mathbb{B}\mathbb{B})}\end{array}}{\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:(\mathbb{C}^{(\mathbb{B}\mathbb{B})},((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}))}\ \mathrm{CF}$$

$$\frac{\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ \mathbf{w}]\ \mathbf{t}]:(\mathbb{C}^{(\mathbb{B}\mathbb{B})},((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})},\mathbb{C}^{\mathbb{W}}),\mathbb{C}^{\mathbb{N}}))}{\lambda w\lambda t\ [\neg[[\lambda w\lambda t\ [[[\mathbf{isPoet}\ w]\ t]\ \mathbf{Scott}]\ w]\ t]]:(\mathbb{C}^{\mathbb{B}\mathbb{B}},\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})})}\ \mathrm{ClF} \tag{4.10}$$

4.2.3 Example 3

Construction ([38], p. 204)

$$\lambda x.+[Gx][Hx]$$

which is equivalent in our notation to

$$\lambda x\,[+\,[\mathbf{G}\,x][\mathbf{H}\,x]]$$

is derivable as

(4.11)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{add:(\mathbb{NNN})}}{+:\mathbb{C}^{(\mathbb{NNN})}}\quad\dfrac{\dfrac{\dfrac{}{G:(\mathbb{NN})}}{\mathbf{G}:\mathbb{C}^{(\mathbb{NN})}}\quad\dfrac{\dfrac{\dfrac{}{(v):array_1}}{a:\mathbb{N}}\,\mathrm{VF}}{x:\mathbb{C}^{\mathbb{N}}}\,\mathrm{VF}}{[\mathbf{G}\,x]:(\mathbb{C}^{(\mathbb{NN})},\mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}}{[+\,[\mathbf{G}\,x]:(\mathbb{C}^{(\mathbb{NNN})},(\mathbb{C}^{(\mathbb{NN})},\mathbb{C}^{\mathbb{N}}),(\mathbb{C}^{(\mathbb{NN})},\mathbb{C}^{\mathbb{N}}))}\quad\dfrac{\dfrac{\dfrac{}{H:(\mathbb{NN})}}{\mathbf{G}:\mathbb{C}^{(\mathbb{NN})}}\quad\dfrac{\dfrac{\dfrac{}{(v):array_1}}{a:\mathbb{N}}\,\mathrm{VF}}{x:\mathbb{C}^{\mathbb{N}}}\,\mathrm{VF}}{[\mathbf{H}\,x]:(\mathbb{C}^{(\mathbb{NN})},\mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}}{[+\,[\mathbf{G}\,x]\,[\mathbf{H}\,x]]:(\mathbb{C}^{(\mathbb{NNN})},(\mathbb{C}^{(\mathbb{NN})},\mathbb{C}^{\mathbb{N}}),(\mathbb{C}^{(\mathbb{NN})},\mathbb{C}^{\mathbb{N}}))}\,\mathrm{CF}\qquad x:\mathbb{C}^{\mathbb{N}}}{\lambda x\,[+\,[\mathbf{G}\,x]\,[\mathbf{H}\,x]]:(\mathbb{C}^{(\mathbb{NNN})},\mathbb{C}^{\mathbb{N}},\mathbb{C}^{\mathbb{N}})}\,\mathrm{ClF}$$

Remark Note that we do not have to form new Variable x "from scratch" for the Closure formation, because we already have it at our disposal, we just need to bind it.

4.2.4 Example 4

Construction ([38], p. 203)

$$\lambda w\lambda t.\mathbf{H}_{wt}\mathbf{A}_{wt}$$

which is equivalent in our notation to

$$\lambda w\lambda t\;[[[\mathbf{isOfheight}\;w]\;t][[\mathbf{authorOfWaverly}\;w]\;t]]$$

is derivable as (4.12) (or its condensed version (4.13)).

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \genfrac{}{}{0pt}{}{isOfheight : (((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}{\mathbf{isOfheight} : \mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}}
        \quad
        \genfrac{}{}{0pt}{}{world : \mathbb{W}}{\mathbf{w} : \mathbb{C}^{\mathbb{W}}}
      }{[\mathbf{isOfheight\ w}] : (\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}})}\text{CF}
      \quad
      \genfrac{}{}{0pt}{}{time : \mathbb{N}}{\mathbf{t} : \mathbb{C}^{\mathbb{N}}}
    }{[[\mathbf{isOfheight\ w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\text{CF}
    \quad
    \dfrac{
      \dfrac{
        \genfrac{}{}{0pt}{}{authorOfWaverly : ((\mathbb{I}\mathbb{N})\mathbb{W})}{\mathbf{authorOfWaverly} : \mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})}}
        \quad
        \genfrac{}{}{0pt}{}{world : \mathbb{W}}{\mathbf{w} : \mathbb{C}^{\mathbb{W}}}
      }{[\mathbf{authorOfWaverly\ w}] : (\mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}})}\text{CF}
      \quad
      \genfrac{}{}{0pt}{}{time : \mathbb{N}}{\mathbf{t} : \mathbb{C}^{\mathbb{N}}}
    }{[[\mathbf{authorOfWaverly\ w}]\ \mathbf{t}] : ((\mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\text{CF}
  }{
    \dfrac{[[[\mathbf{isOfheight\ w}]\ t]\ [[\mathbf{authorOfWaverly\ w}]\ t]] : (((\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), ((\mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}))
    \quad
    \genfrac{}{}{0pt}{}{\mathbf{t}}{t : \mathbb{C}^{\mathbb{N}}}}{\lambda t\ [[[\mathbf{isOfheight\ w}]\ \mathbf{t}]\ [[\mathbf{authorOfWaverly\ w}]\ \mathbf{t}]] : (\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})})}\text{ClF}
  }\text{CF}
  \quad
  \genfrac{}{}{0pt}{}{\mathbf{w}}{w : \mathbb{C}^{\mathbb{W}}}
}{\lambda w \lambda t\ [[[\mathbf{isOfheight}\ w]\ t]\ [[\mathbf{authorOfWaverly}\ w]\ t]] : (\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})})}\text{ClF}
\tag{4.12}
$$

$$
\dfrac{
  \dfrac{
    \dfrac{
      \genfrac{}{}{0pt}{}{isOfheight : (((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}{\mathbf{isOfheight} : \mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}}
      \quad
      \genfrac{}{}{0pt}{}{world : \mathbb{W}}{\mathbf{w} : \mathbb{C}^{\mathbb{W}}}
      \quad
      \genfrac{}{}{0pt}{}{time : \mathbb{N}}{\mathbf{t} : \mathbb{C}^{\mathbb{N}}}
    }{[[\mathbf{isOfheight\ w}]\ \mathbf{t}] : ((\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\text{CF}
    \quad
    \dfrac{
      \genfrac{}{}{0pt}{}{authorOfWaverly : ((\mathbb{I}\mathbb{N})\mathbb{W})}{\mathbf{authorOfWaverly} : \mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})}}
      \quad
      \genfrac{}{}{0pt}{}{world : \mathbb{W}}{\mathbf{w} : \mathbb{C}^{\mathbb{W}}}
      \quad
      \genfrac{}{}{0pt}{}{time : \mathbb{N}}{\mathbf{t} : \mathbb{C}^{\mathbb{N}}}
    }{[[\mathbf{authorOfWaverly\ w}]\ \mathbf{t}] : ((\mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\text{CF}
  }{[[[\mathbf{isOfheight\ w}]\ \mathbf{t}]\ [[\mathbf{authorOfWaverly\ w}]\ \mathbf{t}]] : (((\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), ((\mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}))}\text{CF}
  \quad
  \genfrac{}{}{0pt}{}{\mathbf{t}}{t : \mathbb{C}^{\mathbb{N}}}
  \quad
  \genfrac{}{}{0pt}{}{\mathbf{w}}{w : \mathbb{C}^{\mathbb{W}}}
}{\lambda w \lambda t\ [[[\mathbf{isOfheight}\ w]\ t]\ [[\mathbf{authorOfWaverly}\ w]\ t]] : (\mathbb{C}^{(((\mathbb{N}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{((\mathbb{I}\mathbb{N})\mathbb{W})})}\text{ClF}
\tag{4.13}
$$

4.2.5 Example 5

Construction ([38], p. 263)

$$\&[\mathbf{Dt}_{wt}\ x][\mathbf{Ps}_{wt}\ x]$$

which is equivalent in our notation to

$$[\wedge\ [\textbf{isDetective}\ w]\ t]\ [\textbf{smokesPipe}\ w]\ t]]$$

can be derived as (4.14) (or its condensed version (4.15)).

4.3 Construction execution: logical connectives

In TIL Tichý specifies explicitly behaviour of only one logical connective, negation (see [38], p. 67):

entity	*type*	*description:*
$\neg$	(oo)	negation, i.e., the mapping which takes T to F and F to T

Tichý intents this to be just an example, however, that does not change the fact that this is the only explicit definition at our disposal of how function $\neg$ should behave in TIL.

Of course, it is not difficult to guess what the rest of the table should look like:

entity	*type*	*description:*
$\supset$	(ooo)	implication, i.e., the mapping which takes T and T to T, T and F to F, F and T to T, and F and F to T
$\wedge$	(ooo)	conjunction, i.e., the mapping which takes T and T to T, T and F to F, F and T to F, and F and F to F
$\vee$	(ooo)	disjunction, i.e., the mapping which takes T and T to T, T and F to T, F and T to T, and F and F to F

(4.14)

$$\dfrac{\dfrac{\dfrac{\dfrac{\textit{isDetective} : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \quad \mathbf{w}}{\mathbf{isDetective} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \quad w : \mathbb{C}^{\mathbb{W}}}\,\text{CF} \quad \mathbf{t}}{[\mathbf{isDetective}\ w] : (\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}) \quad t : \mathbb{C}^{\mathbb{N}}}\,\text{CF} \quad \dfrac{\dfrac{\dfrac{\textit{smokesPipe} : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \quad \mathbf{w}}{\mathbf{smokesPipe} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \quad w : \mathbb{C}^{\mathbb{W}}}\,\text{CF} \quad \mathbf{t}}{[\mathbf{smokesPipe}\ w] : (\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}) \quad t : \mathbb{C}^{\mathbb{N}}}\,\text{CF} \quad \textit{and} : (\mathbb{B}\mathbb{B}\mathbb{B})}{[[\mathbf{isDetective}\ w]\ t] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}) \quad [[\mathbf{smokesPipe}\ w]\ t] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}) \quad \wedge : \mathbb{C}^{(\mathbb{B}\mathbb{B}\mathbb{B})}}\,\text{CF}$$

$$[\wedge\ [[\mathbf{isDetective}\ w]\ t]\ [[\mathbf{smokesPipe}\ w]\ t]] : (\mathbb{C}^{(\mathbb{B}\mathbb{B}\mathbb{B})}, ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}))$$

(4.15)

$$\dfrac{\dfrac{\dfrac{\textit{isDetective} : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \quad \mathbf{w} \quad \mathbf{t}}{\mathbf{isDetective} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \quad w : \mathbb{C}^{\mathbb{W}} \quad t : \mathbb{C}^{\mathbb{N}}}\,\text{CF} \quad \dfrac{\textit{smokesPipe} : (((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W}) \quad \mathbf{w} \quad \mathbf{t}}{\mathbf{smokesPipe} : \mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})} \quad w : \mathbb{C}^{\mathbb{W}} \quad t : \mathbb{C}^{\mathbb{N}}}\,\text{CF} \quad \textit{and} : (\mathbb{B}\mathbb{B}\mathbb{B})}{[[\mathbf{isDetective}\ w]\ t] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}) \quad [[\mathbf{smokesPipe}\ w]\ t] : ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}) \quad \wedge : \mathbb{C}^{(\mathbb{B}\mathbb{B}\mathbb{B})}}\,\text{CF}$$

$$[\wedge\ [[\mathbf{isDetective}\ w]\ t]\ [[\mathbf{smokesPipe}\ w]\ t]] : (\mathbb{C}^{(\mathbb{B}\mathbb{B}\mathbb{B})}, ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), ((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}))$$

To sum it up, the behaviour of classical connectives can be described as follows

- $v([\neg A]) = true$ iff $v(A) = false$
- $v([A \wedge B]) = true$ iff $v(A) = true$ and $v(B) = true$
- $v([A \vee B]) = true$ $v(A) = true$ or $v(B) = true$
- $v([A \supset B]) = true$ $v(A) = false$ or $v(B) = true$

'$v([\neg A]) = true$' can be read as "construction $[\neg A]$ constructs $true$ if valuation v assigns value $false$ to Variable A". Analogously in all other cases.

Note, however, that this specification is still largely informal, since it is not really explained what exactly is the '$v([A \wedge B]) = true$'. As it stands it is just metalanguage commentary.[3]

Often we will shorten $v(A) = true$ and $v(B) = true$ as '$(v^{A,B}_{true,true})$'. Analogously in other cases. Hence, we will get

- $v(A) = true$ iff (v^{A}_{true})
- $v([\neg A]) = true$ iff (v^{A}_{false})
- $v([A \wedge B]) = true$ iff $(v^{A,B}_{true,true})$
- $v([A \vee B]) = true$ iff $(v^{A;B}_{true;false})$
- $v([A \supset B]) = true$ iff $(v^{A;B}_{true;true})$

where '$true, true$" shortens "$true$ and $true$", '$true; false$" shortens "$true$ or $false$". Analogously in the cases of A, B and $A; B$. The whole '$(v^{A,B}_{true,true})$' can be read as "valuation v assigns $true$ to A and $true$ to B'.

For example, the execution of the construction $[\supset\ A\ B]$ with $v(A) = true$ and $v(B) = false$ would go as follows:

[3] More proper formalization will be offered later in section 6.2

(4.16)

$$\dfrac{\dfrac{imp : (\mathbb{BBB})}{\supset : \mathbb{C}^{(\mathbb{BBB})}} \quad \dfrac{\dfrac{(v) : array_1}{true : \mathbb{B}}\,\text{VF}}{A : \mathbb{C}^{\mathbb{B}}} \quad \dfrac{\dfrac{(v) : array_1}{true : \mathbb{B}}\,\text{VF}}{B : \mathbb{C}^{\mathbb{B}}}}{[\supset\ A\ B] : (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}})}\,\text{CF} \qquad \dfrac{(v^{A,B}_{true,false}) : array_1}{[\supset\ A\ B]}}{false : \mathbb{B}}\,\text{CE}$$

Remark Note that the letters 'A', 'B' are used here as Variables, not as metavariables.

Chapter 5

Generalizing eTIL$_{\mathbb{C}}$ into eTIL

Summary In this chapter we further generalize system eTIL$_{\mathbb{C}}$ by including higher-order constructions, higher-level judgements and rules, and improper constructions. The resulting system will be called eTIL. Additionally, this generalization will permit us to introduce new notions and provide new analyses previously unobtainable due to our restrictions to first-order constructions only.

5.1 Higher-level judgements

Let's take a look at one of our earlier derivations (3.5):

$$\dfrac{\dfrac{\begin{array}{ccc} add : (\mathbb{NNN}) & 5 : \mathbb{N} & 7 : \mathbb{N} \\ + : \mathbb{C}^{(\mathbb{NNN})} & \mathbf{5} : \mathbb{C}^{\mathbb{N}} & \mathbf{7} : \mathbb{C}^{\mathbb{N}} \end{array}}{[+\,\mathbf{5}\,\mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{CF} \qquad [+\,\mathbf{5}\,\mathbf{7}]}{[add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N})}\,\text{CE}$$
$$12 : \mathbb{N}$$

Note that this derivation essentially establishes that construction

$$[+\,\mathbf{5}\,\mathbf{7}] \text{ of type } (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}}) \text{ constructs } 12 \text{ of type } \mathbb{N}.$$

This piece of information, however, cannot be directly stated in our system of rules $\mathcal{R}_{\text{eTIL}_{\mathbb{C}}}$. In order to amend this we introduce higher-level judgements (or H-judgements for short). Shortly put, it is a judgement about the derivations established via $\mathcal{R}_{\text{eTIL}_{\mathbb{C}}}$ such e.g., (3.5). Their general form will be

$$A : \alpha \Rightarrow a : \beta$$

which can be read as "construction A of type α constructs upon execution object a of type β". Consequently, judgements of the form A : a will be retroactively called lower-level judgements (or L-judgements).

Depending on context (i.e., if it does not lead to any confusion) this can be shortened to just

$$A \Rightarrow a : \beta$$

where the type of construction is suppressed, or further into

$$A \Rightarrow a$$

where the types of both construction and its result are omitted. '$A \Rightarrow a$' can be read as "A constructs a" or more properly as "A constructs upon execution a".

For example, the information conveyed by the above derivation (3.5) can be stated by the following higher-level judgement

$$[+\ \mathbf{5}\ \mathbf{7}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}}) \Rightarrow 12 : \mathbb{N}$$

which can be shortened to

$$[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12 : \mathbb{N}$$

or even further to

$$[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12$$

Other examples:

$$[\supset\ A\ A] : (\mathbb{C}^{(\mathbb{B}\mathbb{B}\mathbb{B})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}}) \Rightarrow \mathit{true} : \mathbb{B}$$

$$[\supset\ A\ A] \Rightarrow \mathit{true} : \mathbb{B}$$

$$[\supset\ A\ A] \Rightarrow \mathit{true}$$

In case the result of construction depends on some valuation-array (v) (i.e., if it contains free Variable(s)), we write $A : \alpha \Rightarrow_v B : \beta$.

Consider e.g., the derivation

(5.1)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{add : (\mathbb{NNN})} \quad \dfrac{}{5 : \mathbb{N}} \quad \dfrac{\dfrac{}{(v) : array_1}\ \dfrac{a : \mathbb{N}}{x : \mathbb{C}^{\mathbb{N}}}\ \text{VF}}{}}{+ : \mathbb{C}^{\mathbb{NNN}} \quad \mathbf{5} : \mathbb{C}^{\mathbb{N}}}\ \text{CF}}{[+\ \mathbf{5}\ x] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})} \quad \dfrac{}{(v_7^x) : array_1}}{[add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N})}\ \text{CE}}{12 : \mathbb{N}}$$

or simpler

(5.2)

$$\dfrac{\dfrac{\vdots}{[\mathbf{Succ}\ y] : (\mathbb{C}^{(\mathbb{NN})}, \mathbb{C}^{\mathbb{N}})} \quad \dfrac{\dfrac{}{(v_0^y) : array_1}}{[\mathbf{Succ}\ y]}}{[succ\ 0] : ((\mathbb{NN}), \mathbb{N})}\ \text{CE}}{1 : \mathbb{N}}$$

The corresponding H-judgements will be

$$[+\ \mathbf{5}\ x] \Rightarrow_{v_7^x} 12 : \mathbb{N}$$

and

$$[\mathbf{Succ}\ \ y] \Rightarrow_{v_0^y} 1 : \mathbb{N}$$

respectively. The notation '$_{v_7^x}$' is a shorthand for "valuation v assigns 0 to y".

Derivations such as (3.5) operating with lower-level judgements will be regarded as proofs of the corresponding higher-level judgements (e.g., $[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12 : \mathbb{N}$).

For example, the corresponding H-judgements for the derivations (4.1), (4.2), and (4.4) from section 4.1 would be, respectively:

$$\lambda w \lambda t\ [[[\mathbf{isGirl}\ w]\ t]\ \mathbf{Alice}] : \mathbb{C}^{(((\mathbb{BI})\mathbb{N})\mathbb{W})} \Rightarrow isGirlAlice : ((\mathbb{BN})\mathbb{W})$$

$$[[[\textbf{isGirl w}]\ \textbf{t}]\ \textbf{Alice}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}}) \Rightarrow true : \mathbb{B}$$

$$[[[\textbf{isGirl } w]\ t]\ \textbf{Alice}] : (((\mathbb{C}^{(((\mathbb{B}\mathbb{I})\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}}) \Rightarrow_{v^{w,t}_{\mathbf{w},\mathbf{t}}} true : \mathbb{B}$$

In other words, these three higher-level judgements will be regarded as proved via derivations (4.1), (4.2), and (4.4), respectively.[1]

Remark Note that expressions of the form $A \Rightarrow a : \beta$ allow us to work more explicitly with constructions. Constructions are often thought of as determined by (1) the object it constructs and (2) the way it constructs this object (see e.g., [32], p. 17). Both of these aspects are explicitly present in our H-judgements.

Remark We might be tempted to interpret '$A \Rightarrow a$' as "A evaluates to a" or 'A reduces to a". This view is, however, incorrect, because A is object of different type than a. (We will return to this topic in the later section 5.5.)

5.2 Higher-level rules

The addition of higher-level judgements opens up the possibility of formulating higher-level rules (or H-rules for short) operating on them. Consequently, $\mathcal{R}_{\text{eTIL}_{\mathbb{C}}}$ rules will be retroactively called lower-level rules (or L-rules).

The general form of higher-level rules will be:

$$\frac{A : \alpha \Rightarrow a : \beta \qquad B : \gamma \Rightarrow b : \delta}{C : \varepsilon \Rightarrow c : \zeta}$$

In case we are not interested in the types of constructions (or it is obvious from the context), we can simply write

$$\frac{A \Rightarrow a : \alpha \qquad B \Rightarrow b : \beta}{C \Rightarrow c : \gamma}$$

or just

$$\frac{A \Rightarrow a \qquad B \Rightarrow b}{C \Rightarrow c}$$

[1] We will return to this topic in section 5.2.1.

if we wish to suppress the type information all together. In contrast to L-rules, H-rules will not come in pairs of formation and execution rules, but in more common pairs of introduction and elimination rules (Intro-rules and Elim-rules for short).

Derivations established via such higher-level rules will be called higher-level derivations (or H-derivations). Consequently, derivations established via $\mathcal{R}_{\text{eTIL}_{\mathbb{C}}}$ will be retroactively called lower-level derivations (or L-derivations).

Remark Sometimes in TIL literature (see e.g., [9]) we can encounter expressions of the form $C/*_1 \rightarrow_v \alpha$ (read as: "construction C of type $*_1$ v-constructs object of type α"), which, more or less, corresponds to our $A : \alpha \Rightarrow_v b : \beta$. But there are three crucial differences: first, we explicitly state the constructed object, not just its type, secondly, $A : \alpha \Rightarrow_v b : \beta$ is an expression of our object language, while $C/*_1 \rightarrow_v \alpha$ is metalanguage commentary, and thirdly and most importantly, H-judgments of the form $A : \alpha \Rightarrow_v b : \beta$ can be derived via L-derivations.

To briefly demonstrate the usefulness of higher-level judgements and rules consider the following example. Suppose we want to prove that

$$[\supset A\ [\supset\ [\supset\ A\ B]\ B]] \Rightarrow \mathit{true}$$

in other words, that $[\supset A\ [\supset\ [\supset\ A\ B]\ B]]$ is a theorem. Since this is a H-judgement its proof will be a L-derivation (recall that lower-level derivations can serve as proofs for higher-level judgements), and more specifically E-derivation.

The corresponding proof would look roughly as follows:[2]

(5.3)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathit{imp} : (\mathbb{BBB})}\quad \dfrac{\dfrac{}{(v) : \mathit{array}}\quad \dfrac{\mathit{true} : \mathbb{B}}{A : \mathbb{C}^{\mathbb{B}}}\,\text{VF}}{\supset : \mathbb{C}^{(\mathbb{BBB})}}\quad \dfrac{\dfrac{}{(v) : \mathit{array}}\quad \dfrac{\mathit{true} : \mathbb{B}}{B : \mathbb{C}^{\mathbb{B}}}\,\text{VF}}{[\supset\ A\ B] : (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}})}\,\text{CF}}{[\supset\ [\supset\ A\ B]\ B] : (\mathbb{C}^{(\mathbb{BBB})}, (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}}), \mathbb{C}^{\mathbb{B}})}\,\text{CF}}{\vdots}\quad \dfrac{\dfrac{}{(v^{A,B}_{i,j}) : \mathit{array}}}{[\supset A\ [\supset\ [\supset\ A\ B]\ B]]}}{[\supset A\ [\supset\ [\supset\ A\ B]\ B]] : (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, (\mathbb{C}^{(\mathbb{BBB})}, (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}}), \mathbb{C}^{\mathbb{B}}))}\quad \vdots}{\mathit{true} : \mathbb{B}}\,\text{CE}$$

[2]Demonstrating this in full is rather lengthy task, hence we will skip it here and return to it later in dedicated section 6.2.

Although there is nothing wrong with this proof, it is a lot of tedious work to prove just one simple theorem.

This is where our newly introduced notion of higher-level rules comes in handy. We can simplify the whole task of proving $[\supset A\ [\supset\ [\supset\ A\ B]\ B]] \Rightarrow true$ by introducing two new higher-level logical rules for implication (borrowed from Natural Deduction, also note the change from prefix notation for $\supset$ to infix notation.)

$$\frac{\begin{array}{c}(A \Rightarrow true)\\ \vdots \\ B \Rightarrow true\end{array}}{[A \supset B] \Rightarrow true}\ \supset\text{Intro} \qquad \frac{[A \supset B] \Rightarrow true \qquad A \Rightarrow true}{B \Rightarrow true}\ \supset\text{Elim}$$

and prove the initial theorem much more simply (from a higher-level, so to speak) as

(5.4)

$$\dfrac{\dfrac{\dfrac{[A \supset B] \Rightarrow true \qquad A \Rightarrow true}{B \Rightarrow true}\ \supset\text{Elim}}{[[A \supset B] \supset B] \Rightarrow true}\ \supset\text{Intro}}{[A \supset [[A \supset B] \supset B]] \Rightarrow true}\ \supset\text{Intro}$$

And, of course, we can import all the other rules from Natural Deduction as well, not just ($\supset$I) and ($\supset$E) rules. (We will return to the topic of logical higher-level rules and all the accompanying notions later in section 6.2.)

Remark Note that since we are starting with TIL on "sub-atomic" level, our higher-level logical rules correspond to ordinary logical rules. In other words, what is usually considered as sub-atomic level (see e.g., [43]) is for us atomic level.

Remark Can we somehow reconcile our system of rules with the fact that Tichý advocated the so called two-dimensional inference (see [38], chapter 13 or [27] and [25])? There are two basic options (1) we are dealing with calculus of TIL, not for TIL (as mentioned already in chapter 1), hence we do not need to feel tied by this requisite and/or (2) expressions of the form $A \Rightarrow a$ (i.e., H-judgements) can be understood as sort of entailments in their own right. After all, stating that

A yields *a* (in the sense that the former constructs the latter) is not that different on intuitive level from stating that *A* yields *a* (in the sense that the latter can be inferred from the latter). In both cases it is something that always holds no matter what. Interestingly, Tichý himself used the word "yield" in both of these senses, i.e., yielding-as-constructing (see e.g., [38], p. 60, p. 87, p. 207, p. 225) and yielding-as-resulting from inference (see e.g., [38], p. 119, p. 235).

5.2.1 Semantic vs. syntactic rules

The general reasoning behind higher-level rules can be described roughly as follows: "Assuming these constructions construct these results, we can derive that this construction will construct this result". In other words, we assume from higher level results on the lower level (via H-judgements) and use them to reason to other, further results on the lower level (via H-rules). For example, we do not need to derive on lower level that some construction *A* constructs *true*, we can just directly assume it on higher level and use it to derive other conclusions.

Note, however, that this also means that the meaning of higher-level judgements is dependent on lower-level derivations. In other words, H-judgements are "semantically" justified by the L-derivations. From this perspective, we can view H-rules which operate with H-judgements as syntactic rules (which follow the Intro/Elim-rule scheme) and L-rules (which adhere to the F/E-rule scheme) as semantic rules.

For example, the meaning of construction **5** is portrayed by the following F-derivation and E-derivation, respectively

$$\frac{5 : \mathbb{N}}{\mathbf{5} : \mathbb{C}^{\mathbb{N}}}\ \mathrm{TF}$$

$$\frac{\begin{array}{c}\vdots\\ \mathbf{5} : \mathbb{C}^{\mathbb{N}}\end{array}}{5 : \mathbb{N}}\ \mathrm{TE}$$

or if we connect them together

$$\frac{\dfrac{5 : \mathbb{N}}{\mathbf{5} : \mathbb{C}^{\mathbb{N}}}\ \mathrm{TF}}{5 : \mathbb{N}}\ \mathrm{TE}$$

This derivation tells us everything we need to know about construction **5**, i.e., how it was formed, what type it has, and what does it construct.

The terminology of semantic (lower-level) rules vs. syntactic (higher-level) rules might be slightly confusing at first. But the main idea is quite simple: it just means that first we have to have the first kind of rules to make precise sense of the latter kind.

Of course, that does not mean that syntactic rules are completely void of any meaning – they offer (proof-theoretic like) meaning on higher-level. They tell us, what can we do with higher-level judgements, but not what these higher-level judgements are, which is the task for lower-level rules.[3] To put it shortly, L-derivations fix the meaning for H-judgements, but not for H-rules themselves. H-rules portray the meaning in the fashion similar to proof-theoretic semantics (see e.g., [10], [35]) by following the general Introduction and Elimination rule scheme.

E.g., we do not need to know the exact meaning of

$$A \Rightarrow true$$

to understand the rule:

$$\frac{A \Rightarrow true \qquad B \Rightarrow true}{[A \wedge B] \Rightarrow true}\ \wedge\text{Intro}$$

In other words, ($\wedge$Intro) rule can convey the meaning of connective $\wedge$ but it cannot convey the meaning of H-judgements it operates on. That's why we call H-rules such as this one syntactic rules.

To sum up, F/E-rules are semantic in respect to H-judgements, which are in turn used by H-rules, which are meaning portraying in their own right. So in a sense, both levels or rules (i.e, L-rules as well as H-rules) are semantic and syntactic. It all depends on the point of view we take. Hence, we will prefer the more adequate terminology lower-level vs. higher-rules as oppose to semantic vs. syntatic rules, which might be unnecessarily confusing.

[3] Recall that the "meaning" of the crucial arrow '$\Rightarrow$' in H-judgements that H-rules operate on is established by the L-derivations which were derived via L-rules.

5.2.2 Beyond logical fragment

So far we have been dealing with logical H-rules only. The natural question seems to be: can we utilize higher-level judgements and rules even elsewhere?

The general relationship between L-derivations and H-judgements stays the same even outside the realm of logical connectives. Consider e.g., the following higher-level judgement:

$$[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12$$

The corresponding lower-level derivation would be:

(5.5)

$$\dfrac{\dfrac{\begin{array}{cccc} add : (\mathbb{NNN}) & 5 : \mathbb{N} & 7 : \mathbb{N} & \\ + : \mathbb{C}^{(\mathbb{NNN})} & \mathbf{5} : \mathbb{C}^{\mathbb{N}} & \mathbf{7} : \mathbb{C}^{\mathbb{N}} & \end{array}}{[+\ \mathbf{5}\ \mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{CF} \qquad \begin{array}{c} [+\ \mathbf{5}\ \mathbf{7}] \\ [add\ 5\ 7] : ((\mathbb{NNN}), \mathbb{N}, \mathbb{N}) \end{array}}{12 : \mathbb{N}}\,\text{CE}$$

So even in arithmetical cases, L-derivations can be seen as establishing the meaning of higher-level judgements.

But what about the relationship between L-derivations and H-derivations? Can H-judgements such as $[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12$ sensibly appear as premises for some H-rules as well? As it turns out we can formulate such H-rules, but it will not be as straightforward as in the logical case. Specifically, we will first need to introduce new kind of judgement, so called congruency judgement. (We will return to this in section 6.3).

Remark Note that with H-judgements we can also formulate explicit rules for handling so called type inference in Composition.

Consider e.g., the following derivation

$$\dfrac{\mathbf{Succ} \Rightarrow succ : (\mathbb{NN}) \qquad \mathbf{0} \Rightarrow 0 : \mathbb{N}}{[\mathbf{Succ}\ \mathbf{0}] \Rightarrow 1 : \mathbb{N}}$$

which, among other things, tells us that if we apply function of type $(\mathbb{NN})$ to argument of type $\mathbb{N}$ we get result of type $\mathbb{N}$.

If we generalize it, we get the rule:

$$\dfrac{A \Rightarrow a : (\alpha\beta_1, ...\beta_m) \qquad B_1 \Rightarrow b_1 : \beta_1 \qquad \ldots \qquad B_m \Rightarrow b_m : \beta_m}{[A\ B_1 ... B_m] \Rightarrow c : \alpha}$$

5.3 Improper constructions

Tichý distinguishes between proper and improper constructions:

> A construction other than a variable may *v*-construct nothing at all. Such a construction will be briefly called *v-improper*.
>
> ([38], p. 62)

Shortly put, (*v*-)improper construction is a construction that fails to construct anything. Sometimes we will also call them abortive constructions. If a construction is not *v*-improper, than it is a proper construction.

As an example of improper construction Tichý offers the following Composition:

$$[\div \mathbf{3}\ \mathbf{0}]$$

Since the function constructed by $\div$ is not defined on arguments 3 and 0 constructed by **3** and **0** (i.e., we cannot divide by 0), this construction produces no result upon execution, hence it is an improper construction. Utilizing H-judgements, we can write this information down as

$$[\div \mathbf{3}\ \mathbf{0}] \Rightarrow \varnothing$$

So far so good. But our question is: how did we managed to form this construction? Recall that Tichý clearly states that all constructions (with the exception of Variable) are formed from non-constructions and other constructions. This Composition is clearly formed from other three constructions $\div$, **3**, and **0** of types $\mathbb{C}^{(\mathbb{NNN})}$, $\mathbb{C}^{\mathbb{N}}$, and $\mathbb{C}^{\mathbb{N}}$, which were in turn formed from non-constructions *div*, 3, and 0 of types $(\mathbb{NNN})$, $\mathbb{N}$, and $\mathbb{N}$, respectively.

How does it happen then that when we put it all together (i.e., three proper constructions), we end up with improper construction? Mind you, the type of the above Composition is $(\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})$, which means that upon execution it should construct object of type $\mathbb{N}$.

Of course, we know what went wrong – we cannot divide by zero. However, we know this from the very beginning of formation of the construction, yet the type of the function (i.e., $(\mathbb{NNN})$) completely conceals this crucial piece of information. To pinpoint the problem: TIL deceives us, the type of *div* function is not

really $(\mathbb{N}\mathbb{N}\mathbb{N})$, but rather something like

$$((\textit{maybe}\ \mathbb{N})\mathbb{N}\mathbb{N})$$

Take a look at the following table from *FFL* ([38], p. 67):

entity	*type*	*description:*
⋮	⋮	⋮
$-$	$(\alpha\alpha\alpha)$	the subtraction mapping (partial)
$\div$	$(\alpha\alpha\alpha)$	the division mapping (partial)
⋮	⋮	⋮

So even though Tichý explicitly mentions the fact that division is a partial function, he omits to include this information in the type of the function.[4] It is simply not the case that for every two arguments of type α (or $\mathbb{N}$ in our notation) we get another object of the same type as would the type $(\alpha\alpha\alpha)$ suggest.

We should, however, account for such possible failures that are known to us in advance. After all, when we are forming the construction $\div$ from the corresponding partial function *div* we already know that any subsequent constructions containing $\div$ might turn out to be improper, so why not to include this important information in the type? In other words, when we are dealing with abortive constructions (and by extension, partial functions), error handling (e.g., that we cannot divide by zero in case of "non-empirical" constructions, or ascribing some property to an individual that might turn out to be actually non-existent in case of "empirical" constructions) should be of utmost importance.

So to be as prepared as possible to potential failures we propose, as already mentioned above, new type *maybe*. The value of type *maybe A* has two possible values: either *just A* or *nothing*.[5] Thus, properly specified for TIL, the function *div* should look e.g., like this

entity	*type*	*description:*
⋮	⋮	⋮
$\div$	$((\textit{maybe}\ \alpha)\alpha\alpha)$	the division mapping (partial)
⋮	⋮	⋮

[4] Recall that Tichý used 'α' to denote type of natural numbers.

[5] We borrow this type from Haskell programming language [24].

or in eTIL

$$sDiv : ((maybe\ \mathbb{N})\mathbb{N}\mathbb{N})$$

which is a function that returns nothing if the second argument is 0, otherwise it divides the first argument by the second.

The notation '$(maybe\ \alpha)$' can be written down in more economic way simply as '$\bar{\alpha}$'. Hence, e.g., '$((maybe\ \mathbb{N})\mathbb{N}\mathbb{N})$' becomes '$(\bar{\mathbb{N}}\mathbb{N}\mathbb{N})$', etc. (We will return to the topic of improper constructions in later section 6.4.)

Remark Since we got rid of Single and Double execution as constructions, the only way to arrive at improper construction is via Composition,[6] and, consequently, via partial functions (see the second clause of Tichý's definition of Composition: "...or if X_0 does not *v*-construct a mapping which is defined at *m*-tuple of entities *v*-constructed by $X_1, .., X_m$").

5.4 Higher-order constructions

So far we have been dealing only with first-order constructions, i.e., constructions constructing first-order objects. But in principal there is nothing that prevents us from introducing into eTIL even higher-level constructions (see appendix A).

When it comes down to higher-order constructions, the crucial moment takes place between the transition from first-order constructions to second-order constructions (i.e., from type $\mathbb{C}_1$ to type $\mathbb{C}_2$). It is then when mentioning of constructions, i.e., hyperintensionality, enters the picture for the first time. The consequent steps from second-order constructions to third-order constructions and further up then proceed in uniform way.

To better mark the significance of second-order constructions we will use new type symbol '$\mathbb{K}$' instead of the more expected '$\mathbb{C}_2$'. After that the type notation will go as expected, i.e., '$\mathbb{C}_3$', '$\mathbb{C}_4$', etc.

[6] In TIL constructions can be improper without invoking partial functions, e.g., $^1 3$ is considered to be improper construction. In eTIL this is not the case due to our exclusion of Execution from constructions.

Thus, the type of second-order construction formed from first-order construction will be written as

$$\mathbb{K}^{\mathbb{C}}$$

which is, as noted above, just notation convention for '$\mathbb{C}_2^{\mathbb{C}_1}$'.

The formation and execution of constructions of type $\mathbb{K}$ (including type stacks) proceeds in analogous manner as constructions of type $\mathbb{C}$.

Remark Recall that initially we used '**Succ**' as a shorthand for '^{0}Succ'. Once we enter the realm of higher-order constructions we can form constructions of the form $^0(^0\text{Succ})$ and so on. Here our bold font convention clearly fails.[7] For this reason we adopt new convention for marking Trivialization of constructions, which we denote by double square brackets '$[\![\ ]\!]$'. Hence '$[\![\mathbf{Succ}]\!]$' will be shorthand for '$^0(^0\text{Succ})$'. Analogously, '$[\![[\![\mathbf{Succ}]\!]]\!]$' will correspond to '$^0(^0(^0\text{Succ}))$' and so on. To sum up, construction in bold font represent constructions of first-order objects, constructions in double square brackets represent second-order constructions of first-order constructions, etc.

For example, from construction **Succ** of type $\mathbb{C}^{(\mathbb{N}\mathbb{N})}$ we can form higher-order (second-order, to be specific) construction $[\![\mathbf{Succ}]\!]$ of type $\mathbb{K}^{\mathbb{C}^{(\mathbb{N}\mathbb{N})}}$ in the following straightforward way:

(5.6)

$$\frac{\begin{array}{c}\vdots\\ \mathbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})}\end{array}}{[\![\mathbf{Succ}]\!] : \mathbb{K}^{\mathbb{C}^{(\mathbb{N}\mathbb{N})}}}\ \text{TF}$$

Execution also proceeds in the analogous manner.

(5.7)

[7] Technically, we could make it twice, thrice as bold, etc., but the result would be rather confusing than helpful.

$$\frac{\begin{array}{c}\vdots\\ [\![\mathbf{Succ}]\!] : \mathbb{K}^{\mathbb{C}^{(\mathbb{NN})}}\end{array}}{\mathbf{Succ} : \mathbb{C}^{(\mathbb{NN})}}\ \text{TE}$$

Remark Even though the notation such as

$$\mathbb{K}^{\mathbb{C}^{(\mathbb{NNN})}}$$

might seem daunting at first it just states that we are dealing with construction of type $\mathbb{K}$ that was formed from construction of type $\mathbb{C}$ which was in turn formed from object of type $(\mathbb{NNN})$. Or alternatively (in case of E-derivation) that we are dealing with construction of type $\mathbb{K}$ that constructs upon execution construction of type $\mathbb{C}$ which in turn constructs upon another execution object of type $(\mathbb{NNN})$. Hence $A : \mathbb{K}^{\mathbb{C}^{(\mathbb{NNN})}}$ can be read as both

$$A : \mathbb{K} \text{ was formed from } B : \mathbb{C} \text{ which was formed from } c : (\mathbb{NNN})$$

and

$$A : \mathbb{K} \text{ constructs } B : \mathbb{C} \text{ which constructs } c : (\mathbb{NNN})$$

Also recall that the type is just $\mathbb{K}$, the superscripts are only labels that help us to keep track of what was the construction formed from/what it constructs.

Remark It is important to note that formation of higher-order construction does not always proceed in a straightforward manner. Consider e.g., we want to form higher-order construction from [**Succ** x]. After applying (TF) rule to the corresponding judgement, we would get construction $[\![\mathbf{Succ}\ x]\!]$. In this Composition, however, the Variable x is no longer free, but bound by the Trivialization (see corresponding definitions in appendix A).

Other more complex examples:

(5.8)

$$\frac{\begin{array}{ccc} calc : (\mathbb{BI}(\mathbb{C})) & Alice : \mathbb{I} & \dfrac{\begin{array}{ccc} add : (\mathbb{NNN}) & 5 : \mathbb{N} & 7 : \mathbb{N} \\ + : \mathbb{C}^{(\mathbb{NNN})} & \mathbf{5} : \mathbb{C}^{\mathbb{N}} & \mathbf{7} : \mathbb{C}^{\mathbb{N}} \end{array}}{[+\ \mathbf{5}\ \mathbf{7}] : (\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\ \text{TF} \\ \mathbf{Calc} : \mathbb{K}^{(\mathbb{BI}(\mathbb{C}))} & \mathbf{Alice} : \mathbb{C}^{\mathbb{I}} & [\![+\ \mathbf{5}\ \mathbf{7}]\!] : \mathbb{K}^{(\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})} \end{array}}{[\mathbf{Calc}\ \mathbf{Alice}\ [\![+\ \mathbf{5}\ \mathbf{7}]\!]] : (\mathbb{K}^{(\mathbb{BI}(\mathbb{C}))}, \mathbb{C}^{\mathbb{I}}, \mathbb{K}^{(\mathbb{C}^{(\mathbb{NNN})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})})}\ \text{CF}$$

(5.9)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{sDiv : (\tilde{\mathbb{N}}\mathbb{N}\mathbb{N}) \quad 3 : \mathbb{N} \quad 0 : \mathbb{N}}{\div : \mathbb{C}^{(\tilde{\mathbb{N}}\mathbb{N}\mathbb{N})} \quad \mathbf{3} : \mathbb{C}^{\mathbb{N}} \quad \mathbf{0} : \mathbb{C}^{\mathbb{N}}}\,\text{CF}}{[\div\ \mathbf{3}\ \mathbf{0}] : (\mathbb{C}^{(\tilde{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\text{TF} \qquad prop : (\mathbb{B}(\mathbb{C}))}{[\![\div\ \mathbf{3}\ \mathbf{0}]\!] : \mathbb{K}^{(\mathbb{C}^{(\tilde{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})} \qquad \mathbf{Pr} : \mathbb{K}^{(\mathbb{B}\mathbb{C})}}\,\text{CF} \qquad [\mathbf{Pr}\ [\![\div\ \mathbf{3}\ \mathbf{0}]\!]]}{[\mathbf{Pr}\ [\![\div\ \mathbf{3}\ \mathbf{0}]\!]] : (\mathbb{K}^{(\mathbb{B}(\mathbb{C}))}, \mathbb{K}^{(\mathbb{C}^{(\tilde{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}) \qquad [prop\ [\div\ \mathbf{3}\ \mathbf{0}]] : (\mathbb{B}(\mathbb{C}), (\mathbb{C}^{(\tilde{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}}))}\,\text{CE}$$
$$false : \mathbb{B}$$

Remark Recall that $(\mathbb{C})$ serves us as a placeholder for construction of any type (including type stacks). So in this particular case $(\mathbb{B}(\mathbb{C}))$ is a type of function that takes any first-order construction and returns either $true$ or $false$.

Notice that we can utilize H-judgements with second-order constructions in the same manner as we did it with first-order constructions. E.g., the fact that construction $[\![\mathbf{Succ}]\!] : \mathbb{K}^{\mathbb{C}^{(\mathbb{N}\mathbb{N})}}$ constructs upon execution construction **Succ** can be written down simply as:

$$[\![\mathbf{Succ}]\!] : \mathbb{K}^{\mathbb{C}^{(\mathbb{N}\mathbb{N})}} \Rightarrow \mathbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})}$$

Or in shortened versions

$$[\![\mathbf{Succ}]\!] \Rightarrow \mathbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})}$$

or

$$[\![\mathbf{Succ}]\!] \Rightarrow \mathbf{Succ}$$

where the types are obvious from the context, since we are now dealing only with first- and second-order constructions. (In section 6.4 we will examine more examples.)

Remark Note the difference between

$$[\div\ \mathbf{3}\ \mathbf{0}] \Rightarrow \varnothing$$

and

$$[\![\div\ \mathbf{3}\ \mathbf{0}]\!] \Rightarrow [\div\ \mathbf{3}\ \mathbf{0}]$$

So while improper constructions constructs nothing upon execution, "trivialized" improper construction constructs upon execution the improper construction itself.

Remark If we allow in the system eTIL$_{\mathbb{C}}$ all the generalizations discussed previously in this chapter (i.e., higher-order constructions, improper constructions, higher-level judgements and higher-level rules) we get system eTIL which is almost as expressive as TIL. To close this gap, we need to adopt additionally Double execution rule (section 6.5) and congruency judgements with corresponding rules (section 6.3).

5.5 Computation rules

Remark From this section onward we will be discussing subject matter that goes mostly beyond the topics covered in *FFL*. In other words, we will be departing from the canonical TIL.

5.5.1 Preliminary notes

The most basic mechanism in the background of TIL can be described as follows:

> Construction A of type α constructs with respect to valuation v object B of type β.

which is often shortened to just:

> Construction A v-constructs B.

The phrase "construction constructs" is the staple of TIL. But what does it actually mean? How or in what sense do constructions actually construct?

Let's approach this more generally. For example, in propositional logic we can say things like:

$$\text{Proof} \quad \dfrac{\dfrac{\dfrac{[A \supset B]^1 \qquad [A]^2}{B}\supset \mathrm{E}}{(A \supset B) \supset B}\supset \mathrm{I}^1}{A \supset ((A \supset B) \supset B)}\supset \mathrm{I}^2 \quad \text{proves } A \supset ((A \supset B) \supset B).$$

And it should be generally quite clear how it is proved.

Or, alternatively, we can say that:

Truth table

A	B	$(A \supset B)$	$((A \supset B) \supset B)$	$A \supset ((A \supset B) \supset B)$
1	1	1	1	1
1	0	0	1	1
0	1	1	1	1
0	0	1	0	1

shows that $A \supset ((A \supset B) \supset B)$ is a tautology.

And again, it should be obvious how is this result achieved.

Analogously, in e.g., Peano arithmetic (or CTT, lambda calculus,...), we can say things as

$$5+7 \text{ is } 12$$

because $5+7$ reduces to $S(5+6)$, which is in the end reduced to

$$S(S(S(S(S(S(S(S(S(S(S(S(0)))))))))))$$

which is taken to be synonymous with 12. Again, it is easy to see how we can get from $5+7$ to 12 once we get acquainted with the definition of $+$ operation.

However, if we say

Construction $[A \supset [[A \supset B] \supset B]$ constructs $true$.

or

Construction $[+ \mathbf{5}\ \mathbf{7}]$ constructs 12.

it is not quite clear how to check it in the context of TIL.

E.g., take the case of "$[+ \mathbf{5}\ \mathbf{7}]$ constructs 12". In what way do constructions $+$, **5**, and **7** construct 12? Sure, we can say that $+$ constructs function *add*, **5** constructs 5, and **7** constructs 7. But then analogous question appears: in what way do objects *add*, 5, and 7 lead to 12? As mentioned above, both lambda calculus and CTT can answer this.

5.5.2 Computation in TIL

Lambda calculus originally developed by Alonzo Church [2] can be regarded as the most universal tool for expressing computations (see [40]). Single computational step then corresponds to β-reduction (i.e., function application), which is

purely syntactic operation based on substitution.[8] In other words, computation is understood as reduction of terms (i.e., syntactic transformation). Once we reach irreducible form, the computation stops.

If we were to import this parallelism from lambda calculus to TIL, it would mean that computation in TIL is also just a syntactic transformation. But this is problematic because constructions themselves should represent computations, not syntactic transformations of thereof.[9] After all, we are told that e.g., [**Succ 0**] (as a result of β-reducing [λx [**Succ** x] **0**]) *constructs* number 1, not that it *is* number 1 because it is further irreducible (and hence not computable). To put it differently, the problem is that β-reduction is usually viewed as computation (to compute means essentially to reduce). This, however, goes against TIL principles, where constructions should stand for computations.

It follows that the step from [**Succ 0**] to 1 cannot be computational step (or at least not in the sense that computation is understood nowadays). We can perhaps called it a *constructive* step, which essentially amounts to the move from the construction to the object it constructs (wrt some valuation v). E.g., the step from **5** to 5 (via (TE) rule) can be understood as an example of constructive step.

But however we call it, the end result is the same. In TIL implicitly coexist two different notions of "computation":

1. syntactic computation (corresponds to standard lambda calculus computation, i.e., *computational step* coincides with β*-reduction* which coincides with *function application*) – takes place on the syntactic level.

2. semantic/constructional computation (unclear to what it corresponds, i.e., *computational step* does not coincide with β*-reduction* which does not coincide with *function application*) – takes place at the semantic level of constructions.

The syntactic notion of computation is well known and explored, TIL notion, however, remains rather elusive. The basic question we have to ask is: in what sense

[8] The topic of conversions will be discussed in more length in the next section 5.6.

[9] In *FFL* [38] Tichý explicitly identifies constructions only with calculations (see e.g., p. 7, p. 12, p. 20, p. 31, p. 82, p. 222, p. 281), not with computations in general, but it is obvious from the way he uses constructions that this is just a matter of terminology.

does e.g., [**Succ 0**] construct number 1? The problem is—at least as it appears to me—that we cannot at the moment offer better answer than "it just does".[10]

The other important question is: what is the relation between the above mentioned notions of computation? Is there actually any meaningful difference? Because e.g., saying that [**Succ 0**] constructs 1 or that it is 1, seems—at least in this case—just as difference of terminology. In other words, at this basic level there seems to be no technical difference whether we view '[**Succ 0**]' as a representation of a construction that in some further unspecified way constructs number 1, or as a direct representation the number 1 itself.

To be even more specific consider the following case. The standard way of dealing with calculations such as

$$0+1$$

formally is to first define numbers via 0 and successor function S (as in Peano arithmetic), hence we get

$$0+S(0)$$

Next order of business is to define recursively the function $+$, which is most often done in the following way[11]

$$a+0=a$$

$$a+S(b)=S(a+b)$$

Our calculation clearly falls under the second case, so we rewrite (reduce, compute,...) it into new form

$$S(0+0)$$

This new form of calculation falls under the first case, so we get the further irreducible

$$S(0)$$

[10] It seems that it is not actually the constructions themselves that do the constructing in TIL, but our minds. In other words, the construction [**Succ 0**] constructs 1 only because we, as reasoners, already know that when we put *succ* and 0 together we get 1. The construction itself offers no instructions on how to actually do it and reader's intuitive insight is required in order to accomplish this task.

[11] Note that the '=' symbol essentially prescribes reductions (and hence computations), i.e., it tells us that we can rewrite the term on the left side of '=' to the term on the right side.

which is, of course, nothing else than

$$1$$

Notice that the whole computation can be then written as

$$0+S(0) = S(0+0) = S(0)$$

i.e., all the steps were equivalent transformations. Alternatively we could write this down as

$$0+S(0) \rightsquigarrow S(0+0) \rightsquigarrow S(0)$$

where '$\rightsquigarrow$' represents "computes to". The important thing to notice here is that all three involved expressions, i.e., $0+S(0)$, $S(0+0)$, and $S(0)$ are taken to be objects of the same type.

However, in TIL if we write e.g.,

$$[+\ \mathbf{0}\ \mathbf{1}] \Rightarrow 1$$

where '$\Rightarrow$' is suppose to represent "constructs", the entities on each side of the arrow are objects of different types.

Of course, we could introduce into TIL reductions such as e.g.,

$$[\mathbf{Succ}\ [+\ \mathbf{0}\ \mathbf{0}]] \rightsquigarrow [\mathbf{Succ}\ \mathbf{0}]$$

and hence get constructions, i.e., objects of the same type, on both sides of the "computation arrow". However, in this scenario '$\rightsquigarrow$' cannot be understood as "constructs" but it must be interpreted e.g., as "is reducible to", "can be syntactically transformed", "can be rewritten into", etc.

The problem is that if we allow this notion of computation to enter into TIL, we are effectively eliminating the whole "constructional" dimension of TIL – that constructions suppose to be computations and that they should construct various objects upon execution relative to valuations v.[12] To put it differently, with the notion of computation represented by '$\rightsquigarrow$' above we are just doing syntactic

[12]Either that or we would have to say that e.g., β-reduction rule is actually some sort of metacomputation rule that takes us from one computation into another computation. However, such notion of "metacomputation" is far from being clear.

transformations and the whole idea of constructions constructing does not need to enter the picture at all.

However, that is exactly the price we have to pay if we want to have explicit computation rules. In other words, if we are to have any success at providing computation rules for TIL, they have to be transformation rules on constructions.

E.g., rules for handling Composition that contains addition function could be in our system defined as roughly as follows:

$$\frac{x : (\mathbb{C})}{[+\, x\, \mathbf{0}] \rightsquigarrow x : (\mathbb{C})}\ \text{Add1}$$

$$\frac{x : (\mathbb{C}) \qquad y : (\mathbb{C})}{[+\, x\, [\mathbf{Succ}\, y]] \rightsquigarrow [\mathbf{Succ}\, [+\, x\, y]] : (\mathbb{C})}\ \text{Add2}$$

To sum up, we can include computation rules into TIL, but they have to be based on construction transformations. This goes against the core principle of TIL that constructions themselves should be computations and not the manipulations with them.

5.5.2.1 Summary

Computation rules for TIL are problematic. They have to operate only on constructions, but then the whole concept of constructing becomes rather superfluous. Not to mention that constructions should represent computations in the first place, so it is not at all clear, what we are doing when we are rewriting one construction into another. Thus, we end up with constructions that suppose to represent computations, yet when we actually compute we just move symbols around and the whole notion of constructing can be disregarded.

In TIL we could never say that $[\lambda x\ [\mathbf{Succ}\ x]\ \mathbf{0}]$ constructs $[\mathbf{Succ}\ \mathbf{0}]$. Yet in lambda calculus it would be perfectly fine to say that $\lambda x.succ(x)\ 0$ computes to $succ(0)$. Hence, constructing and computing are very different notions. Analogously, the resulting expression $S(0)$ from above example is not possible to compute anymore (it cannot be reduced any further), but in TIL the "corresponding" construction $[\mathbf{Succ}\ \mathbf{0}]$ is still viewed as a computation that produces the number 1.

5.6 β-, η-, α-conversions

Although β-, η-, α-conversions are nowadays considered common part of TIL, in *FFL* they are not mentioned at all. Whatever the reason was there are couple of problematic aspects connected with them in the framework of TIL:[13]

- *β-reduction* – in ordinary lambda calculus β-reduction captures the idea of function application and it is essentially nothing else than substitution. So β-reduction = function application = substitution. In TIL this is, however, not the case. Function application is one of the implicit subroutines of Composition construction.

  Consider e.g., β-reduction

  $$[\lambda x\ [\mathbf{Succ}\ x]\ \mathbf{0}] \rightsquigarrow_{\beta} [\mathbf{Succ}\ \mathbf{0}]$$

  We are essentially transforming Composition of Closure and Trivialization into a different Composition of two Trivializations.

  So this transformation effectively changes the kinds of involved constructions (Trivialization in general behaves differently than Closure, e.g., results of Trivialization never depend on valuations).

- *β-abstraction* – classically used to capture the idea of lambda abstraction (i.e., creating an anonymous function – by rewriting expression to an application of a lambda abstraction to an argument expression). Again, in TIL this cannot be the case, because function abstraction as such does not exist in TIL (we have Closure construction).

  Consider e.g., β-abstraction

  $$[\mathbf{Succ}\ \mathbf{0}] \rightsquigarrow_{\beta} [\lambda x\ [\mathbf{Succ}\ x]\ \mathbf{0}]$$

  Clearly same troubles appear as above, i.e., we are rewriting one kind of construction into a completely different kind of construction.

[13] Some of the troubles connected with β-reduction were already mentioned in the previous section 5.5.2.

- η*-abstraction* – classically used to capture the idea of adding abstraction over a function. In TIL again problematic for the same reasons as stated above.

  Consider e.g., η-abstraction

  $$\mathbf{Succ} \rightsquigarrow_{\eta} \lambda x\, [\mathbf{Succ}\, x]$$

  In short, it is an operation that transforms Trivilization into a different kind of construction, namely Closure.

- η*-reduction* – classically used to capture the idea of dropping lambda abstraction.

  E.g., consider η-reduction

  $$\lambda x\, [\mathbf{Succ}\, x] \rightsquigarrow_{\eta} \mathbf{Succ}$$

  In TIL again problematic, since we are transforming Closure into Trivialization.[14]

- α*-conversion* – renaming of Variables, the only non-problematic conversion due to the fact that we are not transforming one kind of construction into a different kind. In other words, α-conversion just takes us from a Variable to a different Variable.

  E.g., α-conversion

  $$\lambda x\, [\mathbf{Succ}\, x] \rightsquigarrow_{\alpha} \lambda y\, [\mathbf{Succ}\, y]$$

Remark Note all the problematic aspects discussed above stem from the fact that in TIL "lambda terms" are considered to be constructions that can upon execution construct (wrt to valuation) objects. Hence the behaviour of constructions significantly differs from the behaviour of terms from classical lambda calculus that has no *v*-constructing, no executions, etc., which leads to these discrepancies.

To sum up, β- and η-conversion cannot play the same role in TIL as they do in lambda calculus due to the distinct nature of constructions. The best recourse seems to be to take them as transformations of constructions.

[14] These difficulties are, of course, not limited just to Closure and Trivialization. Consider e.g., construction $\lambda x\, [s\ x]$ with free Variable s. After applying η-reduction we are effectively reducing Closure to Variable.

5.6.1 β, η, α-conversion rules

Despite the problems discussed in the previous section we can still provide basic β, η, α-conversion rules for eTIL. However, it is important to keep in mind that their role in TIL is different from the role they play in ordinary lambda calculus.

5.6.1.1 β-conversions

$$\frac{A(B_1/x_1,...,B_m/x_m) : (\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m})}{[\lambda x_1...x_m\, A\, B_1...B_m] : (\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m})}\ \beta\text{-abs}$$

$$\frac{[\lambda x_1...x_m\ \ A\, B_1...B_m] : (\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m})}{A(B_1/x_1,...,B_m/x_m) : (\mathbb{C}^{(\alpha\beta_1...\beta_m)}, \mathbb{C}^{\beta_1}, ..., \mathbb{C}^{\beta_m})}\ \beta\text{-red}$$

Remark The notation '$A(B_1/x_1,...,B_m/x_m)$' can be read as "substitute $B_1,...,B_m$ for $x_1,...,x_m$ in A".[15]

For example:

(5.10)

$$\frac{\begin{array}{c}\vdots\\ [\mathbf{Succ}\ \mathbf{0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})\end{array}}{[\lambda x\, [\mathbf{Succ}\, x]\ \mathbf{0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})}\ \beta\text{-abs}$$

(5.11)

$$\frac{\begin{array}{c}\vdots\\ [\lambda x\, [\mathbf{Succ}\, x]\ \mathbf{0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})\end{array}}{[\mathbf{Succ}\ \mathbf{0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})}\ \beta\text{-red}$$

(5.12)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{cc} succ : (\mathbb{N}\mathbb{N}) & 0 : \mathbb{N} \\ \mathbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})} & \mathbf{0} : \mathbb{C}^{\mathbb{N}}\end{array}}{[\mathbf{Succ}\ \mathbf{0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})}\ \text{CF}}{[\lambda x\, [\mathbf{Succ}\, x]\ \mathbf{0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})}\ \beta\text{-abs}}{[\mathbf{Succ}\ \mathbf{0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})}\ \beta\text{-red} \qquad \begin{array}{c}[\mathbf{Succ}\ \mathbf{0}] \\ [succ\ 0] : ((\mathbb{N}\mathbb{N}), \mathbb{N})\end{array}}{1 : \mathbb{N}}\ \text{CE}$$

[15] For more on substitution, see appendix A or [32]

5.6.1.2 η-conversions

$$\frac{\lambda x_1 \ldots x_m\ [A\ x_1 \ldots x_m] : \mathbb{C}^{(\alpha\beta_1 \ldots \beta_m)}}{A : \mathbb{C}^{(\alpha\beta_1 \ldots \beta_m)}}\ \eta\text{-red}$$

$$\frac{A : \mathbb{C}^{(\alpha\beta_1,\ldots,\beta_m)}}{\lambda x_1 \ldots x_m\ [A\ x_1 \ldots x_m] : \mathbb{C}^{(\alpha\beta_1 \ldots \beta_m)}}\ \eta\text{-abs}$$

For example:

(5.13)

$$\frac{\begin{array}{c}\vdots\\ \lambda x\ [\mathbf{Succ}\ x] : \mathbb{C}^{(\mathbb{N}\mathbb{N})}\end{array}}{\mathbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})}}\ \eta\text{-red}$$

(5.14)

$$\frac{\begin{array}{c}\vdots\\ \mathbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})}\end{array}}{\lambda x\ [\mathbf{Succ}\ x] : \mathbb{C}^{(\mathbb{N}\mathbb{N})}}\ \eta\text{-abs}$$

5.6.1.3 α-conversion

$$\frac{\lambda x_1 \ldots x_m\ [A\ x_1 \ldots x_m] : \mathbb{C}^{(\alpha\beta_1 \ldots \beta_m)}}{\lambda y_1 \ldots y_m\ [A(y_1/x_1, \ldots, y_m/x_m)] : \mathbb{C}^{(\alpha\beta_1 \ldots \beta_m)}}\ \alpha\text{-con}$$

For example:

(5.15)

$$\frac{\begin{array}{c}\vdots\\ \lambda x\ [\mathbf{Succ}\ x] : \mathbb{C}^{(\mathbb{N}\mathbb{N})}\end{array}}{\lambda y\ [\mathbf{Succ}\ y] : \mathbb{C}^{(\mathbb{N}\mathbb{N})}}\ \alpha\text{-con}$$

Chapter 6

Case studies: eTIL

Summary Analogously to chapter 4 the main purpose of this chapter is to demonstrate new notions introduced in eTIL (higher-order constructions, improper constructions, higher-level rules,...) in practice.

6.1 Higher-order construction formation

6.1.1 Example 1

Construction ([38], p. 72)

$$[\neg[\mathsf{Pr}_1[\div\,\mathbf{3}\,\mathbf{0}]]]$$

which is equivalent in our notation to

$$[\neg[\mathbf{Pr}\,[\![\div\,\mathbf{3}\,\mathbf{0}]\!]]]$$

is derivable as:

(6.1)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{sDiv : (\bar{\mathbb{N}}\mathbb{N}\mathbb{N}) \quad 3 : \mathbb{N} \quad 0 : \mathbb{N}}{\div : \mathbb{C}^{(\bar{\mathbb{N}}\mathbb{N}\mathbb{N})} \quad \mathbf{3} : \mathbb{C}^{\mathbb{N}} \quad \mathbf{0} : \mathbb{C}^{\mathbb{N}}}\,\mathrm{CF}}{[\div\,\mathbf{3}\,\mathbf{0}] : (\mathbb{C}^{(\bar{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\mathrm{TF} \quad prop : (\mathbb{B}(\mathbb{C}))}{[\![\div\,\mathbf{3}\,\mathbf{0}]\!] : \mathbb{K}^{(\mathbb{C}^{(\bar{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})} \quad \mathbf{Pr} : \mathbb{K}^{(\mathbb{B}(\mathbb{C}))}}\,\mathrm{CF} \quad neg : (\mathbb{B}\mathbb{B})}{[\mathbf{Pr}\,[\![\div\,\mathbf{3}\,\mathbf{0}]\!]] : (\mathbb{K}^{(\mathbb{B}(\mathbb{C}))}, \mathbb{K}^{(\mathbb{C}^{(\bar{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}) \quad \neg : \mathbb{C}^{(\mathbb{B}\mathbb{B})}}\,\mathrm{CF}$$
$$[\neg[\mathbf{Pr}\,[\![\div\,\mathbf{3}\,\mathbf{0}]\!]]] : (\mathbb{C}^{(\mathbb{B}\mathbb{B})}, (\mathbb{K}^{(\mathbb{B}(\mathbb{C}))}, \mathbb{K}^{(\mathbb{C}^{(\bar{\mathbb{N}}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}))$$

Remark Recall that, technically speaking, the type of $[\neg[\mathbf{Pr}\ [\![\div\ \mathbf{3}\ \mathbf{0}]\!]]]$ is $(\mathbb{C},(\mathbb{K},\mathbb{K}))$, the superscripts are just labels that help us track all the important information.

6.1.2 Example 2

Construction ([38], p. 222)

$$\lambda w\lambda t.\mathbf{B}^{1}_{wt}\mathbf{G}[\lambda w\lambda t.=[+\mathbf{11}]\mathbf{2}]$$

which is in our notation equivalent to

$$\lambda w\lambda t\ [[[\mathbf{Bel\ w}]\ \mathbf{t}]\ \mathbf{G}\ [\![\lambda w\lambda t\ [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!]]$$

and in eTIL it can be established via the F-derivation (6.2) (or its compressed version (6.3)).

At first glance the derivation (6.2) might seem excessive, but it is important to realize that it contains the same amount of reasoning steps necessary to get to the end construction as does the original. It just makes these steps more formal and explicit. In other words, the derivation is no more complex than the original construction displayed above, the only difference is that all the work is laid out explicitly in front of us and not just assumed on the side of the reader.

Remark In (6.2) we can notice something that could be called vacuous Closure: even though we are binding the Variable t (and later w) it itself is not a part of the construction. Nevertheless, it has to appear in the type because it is a (TIL) proposition, i.e., object of type $((\mathbb{B}\mathbb{N})\mathbb{W})$.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\begin{array}{c} believeC : (((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W}) \\ \mathbf{Bel} : \mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})} \end{array}
\quad
\begin{array}{c} world : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{array}
}{[\mathbf{Bel\ w}] : (\mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}})}\,\mathrm{CF}
\quad
\begin{array}{c} time : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{array}
}{[[\mathbf{Bel\ w}]\ \mathbf{t}] : ((\mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}
\quad
\begin{array}{c} GeorgeIV : \mathbb{I} \\ \mathbf{G} : \mathbb{C}^{\mathbb{I}} \end{array}
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\begin{array}{c} add : (\mathbb{N}\mathbb{N}\mathbb{N}) \\ + : \mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})} \end{array}
\quad
\begin{array}{c} 1 : \mathbb{N} \\ \mathbf{1} : \mathbb{C}^{\mathbb{N}} \end{array}
\quad
\begin{array}{c} 1 : \mathbb{N} \\ \mathbf{1} : \mathbb{C}^{\mathbb{N}} \end{array}
}{[+\ \mathbf{1}\ \mathbf{1}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}
\quad
\begin{array}{c} equal : (\mathbb{B}\mathbb{N}\mathbb{N}) \\ = : \mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})} \end{array}
\quad
\begin{array}{c} 2 : \mathbb{N} \\ \mathbf{2} : \mathbb{C}^{\mathbb{N}} \end{array}
}{[=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}] : (\mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}, (\mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}
\quad
\begin{array}{c} \mathbf{t} \\ t : \mathbb{C}^{\mathbb{N}} \end{array}
}{\lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}] : \mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}}\,\mathrm{ClF}
\quad
\begin{array}{c} \mathbf{w} \\ w : \mathbb{C}^{\mathbb{W}} \end{array}
}{\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}] : \mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}}\,\mathrm{ClF}
}{[\![\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!] : \mathbb{K}^{\mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}}}\,\mathrm{TF}
}{[[[\mathbf{Bel\ w}]\ \mathbf{t}]\ \mathbf{G}\ [\![\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!]] : (((\mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}}, \mathbb{K}^{\mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}})}\,\mathrm{CF}
\quad
\begin{array}{c} \mathbf{t} \\ t : \mathbb{C}^{\mathbb{N}} \end{array}
}{\lambda t\, [[[\mathbf{Bel}\ w]\ t]\ \mathbf{G}\ [\![\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!]] : \mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}}\,\mathrm{ClF}
\quad
\begin{array}{c} \mathbf{w} \\ w : \mathbb{C}^{\mathbb{W}} \end{array}
}{\lambda w \lambda t\, [[[\mathbf{Bel}\ w]\ t]\ \mathbf{G}\ [\![\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!]] : \mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}}\,\mathrm{ClF}
\tag{6.2}
$$

$$
\dfrac{
\dfrac{
\dfrac{
\begin{array}{c} believeC : (((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W}) \\ \mathbf{Bel} : \mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})} \end{array}
\quad
\begin{array}{c} world : \mathbb{W} \\ \mathbf{w} : \mathbb{C}^{\mathbb{W}} \end{array}
\quad
\begin{array}{c} time : \mathbb{N} \\ \mathbf{t} : \mathbb{C}^{\mathbb{N}} \end{array}
}{[[\mathbf{Bel\ w}]\ \mathbf{t}] : ((\mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}
\quad
\begin{array}{c} GeorgeIV : \mathbb{I} \\ \mathbf{G} : \mathbb{C}^{\mathbb{I}} \end{array}
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\begin{array}{c} add : (\mathbb{N}\mathbb{N}\mathbb{N}) \\ + : \mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})} \end{array}
\quad
\begin{array}{c} 1 : \mathbb{N} \\ \mathbf{1} : \mathbb{C}^{\mathbb{N}} \end{array}
\quad
\begin{array}{c} 1 : \mathbb{N} \\ \mathbf{1} : \mathbb{C}^{\mathbb{N}} \end{array}
}{[+\ \mathbf{1}\ \mathbf{1}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}
\quad
\begin{array}{c} equal : (\mathbb{B}\mathbb{N}\mathbb{N}) \\ = : \mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})} \end{array}
\quad
\begin{array}{c} 2 : \mathbb{N} \\ \mathbf{2} : \mathbb{C}^{\mathbb{N}} \end{array}
}{[=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}] : (\mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}, (\mathbb{C}^{(\mathbb{N}\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}}, \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{N}})}\,\mathrm{CF}
\quad
\begin{array}{c} \mathbf{t} \\ t : \mathbb{C}^{\mathbb{N}} \end{array}
\quad
\begin{array}{c} \mathbf{w} \\ w : \mathbb{C}^{\mathbb{W}} \end{array}
}{\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}] : \mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}}\,\mathrm{ClF}
}{[\![\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!] : \mathbb{K}^{\mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}}}\,\mathrm{TF}
}{[[[\mathbf{Bel\ w}]\ \mathbf{t}]\ \mathbf{G}\ [\![\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!]] : (((\mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}, \mathbb{C}^{\mathbb{W}}), \mathbb{C}^{\mathbb{N}}), \mathbb{C}^{\mathbb{I}}, \mathbb{K}^{\mathbb{C}^{(\mathbb{B}\mathbb{N}\mathbb{N})}})}\,\mathrm{CF}
}{\lambda w \lambda t\, [[[\mathbf{Bel}\ w]\ t]\ \mathbf{G}\ [\![\lambda w \lambda t\, [=[+\ \mathbf{1}\ \mathbf{1}]\ \mathbf{2}]]\!]] : \mathbb{K}^{(((\mathbb{B}\mathbb{I}(\mathbb{C}))\mathbb{N})\mathbb{W})}}\,\mathrm{ClF}
\tag{6.3}
$$

6.2 Higher-level logical rules

In order to prove

$$[\supset A\ [\supset\ [\supset\ A\ B]\ B]] \Rightarrow true$$

we want to show that it constructs *true* for all possible valuations of Variables A and B.[1] This, in turn, means deriving $true : \mathbb{B}$ from $[\supset A\ [\supset\ [\supset\ A\ B]\ B]]$ under all four possible valuations $(v^{A,B}_{true,true}) : array$, $(v^{A,B}_{true,false}) : array$, $(v^{A,B}_{false,true}) : array$, and $(v^{A,B}_{false,false}) : array$. This, of course, requires four separate derivations.

The first one with $(v^{A,B}_{true,true}) : array$ will look as follows

(6.4)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{imp : (\mathbb{BBB})}{\supset : \mathbb{C}^{(\mathbb{BBB})}} \quad \dfrac{\dfrac{(v) : array}{true : \mathbb{B}}}{A : \mathbb{C}^{\mathbb{B}}}\,\mathrm{VF} \quad \dfrac{\dfrac{(v) : array}{true : \mathbb{B}}}{B : \mathbb{C}^{\mathbb{B}}}\,\mathrm{VF}}{[\supset\ A\ B] : (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}})}\,\mathrm{CF} \quad \supset : \mathbb{C}^{(\mathbb{BBB})} \quad B : \mathbb{C}^{\mathbb{B}}}{[\supset\ [\supset\ A\ B]\ B] : (\mathbb{C}^{(\mathbb{BBB})}, (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}}), \mathbb{C}^{\mathbb{B}})}\,\mathrm{CF} \quad \supset : \mathbb{C}^{(\mathbb{BBB})} \quad A : \mathbb{C}^{\mathbb{B}}}{[\supset A\ [\supset\ [\supset\ A\ B]\ B]] : (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, (\mathbb{C}^{(\mathbb{BBB})}, (\mathbb{C}^{(\mathbb{BBB})}, \mathbb{C}^{\mathbb{B}}, \mathbb{C}^{\mathbb{B}}), \mathbb{C}^{\mathbb{B}}))}\,\mathrm{CF} \quad \begin{array}{c}(v^{A,B}_{true,true}) : array\\ [\supset A\ [\supset\ [\supset\ A\ B]\ B]]\end{array}}{true : \mathbb{B}}\,\mathrm{CE} \quad [imp\ true\ [imp\ [imp\ true\ true]\ true]]$$

Now, we have to show that we get the same result (i.e., $true : \mathbb{B}$) for all the remaining three valuations as well, i.e., $(v^{A,B}_{true,false}) : array$, $(v^{A,B}_{false,true}) : array$, and $(v^{A,B}_{false,false}) : array$. Of course, the proofs would proceed in the same way, just the final step would differ (i.e., application of CE rule):

(6.5)

$$\dfrac{\begin{array}{c}(v^{A,B}_{true,false}) : array\\ [\supset A\ [\supset\ [\supset\ A\ B]\ B]]\\ \vdots \qquad [imp\ true\ [imp\ [imp\ true\ false]\ false]]\end{array}}{true : \mathbb{B}}\,\mathrm{CE}$$

(6.6)

$$\dfrac{\begin{array}{c}(v^{A,B}_{false,true}) : array\\ [\supset A\ [\supset\ [\supset\ A\ B]\ B]]\\ \vdots \qquad [imp\ false\ [imp\ [imp\ false\ true]\ true]]\end{array}}{true : \mathbb{B}}\,\mathrm{CE}$$

[1] Recall our definition of logical connectives from section 4.3.

(6.7)

$$\dfrac{\begin{array}{c}(v^{A,B}_{false,false}) : array\\ [\supset A\ [\supset\ [\supset\ A\ B]\ B]]\\ \vdots\qquad [imp\ false\ [imp\ [imp\ false\ false]\ false]]\end{array}}{true : \mathbb{B}}\ \text{CE}$$

So in the total, four distinct derivations are needed to properly establish that the construction $[\supset A\ [\supset\ [\supset\ A\ B]\ B]]]$ constructs always *true*.

In general, if such derivations can be established, we say that A is semantically derivable in the system of lower-level rules (i.e., L-derivable) and write it as $\models^{L}_{\mathbb{C}} A$.

Hence, '$\models^{L}_{\mathbb{C}}$' denotes the (semantic) derivability in the system of lower-level rules, where $\mathbb{C}$ is the highest order of constructions. Semantic derivability means that we can demonstrate via F/E-rules that some construction A constructs upon execution with all possible valuations the value *true*. So $\models^{L}_{\mathbb{C}} A$ essentially just states that construction A constructs upon execution with all possible valuations the value *true*.

Remark Note that the above fourfold proof is in a way nothing more than ostentatious truth table, however, it is necessitated by the Tichý's core principles of TIL. Fortunately for us, we will not need to use them in practice. For more elegant logical proofs we have the higher-level rules. To put it differently, L-derivations provide essential theoretical foundation. Without them there would be a hole in our theory, so to speak (recall that they establish the meaning of H-judgements). Hence, it is necessary to lay them down, but once we do we rarely need to return to them ever again.

In the previous chapter, we mentioned that we can import other Natural Deduction rules to our system, not just ($\supset$Intro) and ($\supset$Elim) rule. The result would look as follows (rules are based on [29]):[2]

$$\dfrac{A \Rightarrow true \qquad B \Rightarrow true}{[A \wedge B] \Rightarrow true}\ \wedge\text{Intro} \qquad \dfrac{[A \wedge B] \Rightarrow true}{A \Rightarrow true}\ \wedge\text{Elim}$$

$$\dfrac{[A \wedge B] \Rightarrow true}{B \Rightarrow true}\ \wedge\text{Elim}$$

[2]The symbol '$\perp$' stands for contradiction.

$$\frac{A \Rightarrow true}{[A \vee B] \Rightarrow true}\ \vee\text{Intro} \qquad \frac{B \Rightarrow true}{[A \vee B] \Rightarrow true}\ \vee\text{Intro}$$

$$\frac{[A \vee B] \Rightarrow true \qquad \begin{matrix}(A \Rightarrow true)\\ C \Rightarrow true\end{matrix} \qquad \begin{matrix}(B \Rightarrow true)\\ C \Rightarrow true\end{matrix}}{C \Rightarrow true}\ \vee\text{Elim}$$

$$\frac{\begin{matrix}(A \Rightarrow true)\\ \vdots\\ B \Rightarrow true\end{matrix}}{[A \supset B] \Rightarrow true}\ \supset\text{Intro} \qquad \frac{[A \supset B] \Rightarrow true \qquad A \Rightarrow true}{B \Rightarrow true}\ \supset\text{Elim}$$

$$\frac{\bot}{A \Rightarrow true}\ \bot_{\mathrm{I}} \qquad \frac{\begin{matrix}([\neg A] \Rightarrow true)\\ \bot\end{matrix}}{A \Rightarrow true}\ \bot_{\mathrm{C}}$$

Special cases of implication rules:

$$\frac{\begin{matrix}(A \Rightarrow true)\\ \bot\end{matrix}}{[\neg A] \Rightarrow true}\ \neg\text{Intro} \qquad \frac{A \Rightarrow true \qquad [\neg A] \Rightarrow true}{\bot}\ \neg\text{Elim}$$

Furthermore, recall that in the previous chapter we proved $[\supset A\ [\supset\ [\supset\ A\ B]\ B]] \Rightarrow true$ using H-derivation:

(6.8)

$$\cfrac{\cfrac{\cfrac{[A \supset B] \Rightarrow true \qquad A \Rightarrow true}{B \Rightarrow true}\ \supset\text{Elim}}{[[A \supset B] \supset B] \Rightarrow true}\ \supset\text{Intro}}{[A \supset [[A \supset B] \supset B]] \Rightarrow true}\ \supset\text{Intro}$$

In general, if such derivation can be established we say that A is syntactically derivable in the system of higher-level rules (i.e., H-derivable) and write it as $\vdash^{H}_{\mathbb{C}} A \Rightarrow true$.

Hence, '$\vdash^{H}_{\mathbb{C}}$' denotes the (syntactic) derivability in the system of higher-level rules, where $\mathbb{C}$ is the highest order of constructions. Syntactic derivability means that we can demonstrate via higher-level logical rules (as presented above) that some construction A constructs $true$. So $\vdash^{H}_{\mathbb{C}} A \Rightarrow true$ essentially just states that construction A always constructs $true$.

The system of higher-level logical rules presented above behaves as ordinary Natural Deduction for propositional logic. Consequently, the proofs of soundness

of completeness of this propositional fragment of $\text{eTIL}_{\mathbb{C}}$ would proceed in usual manner, i.e., by structural induction on the complexity of formulas.

The only slight deviation is the definition of semantics of lower-level rules. Recall that for us valuation is not just a function, rather part of execution rules for constructions with free Variables – see our above derivations (3.5), (6.5), (6.6), and (6.7) as well as section 4.3 where we discussed the definition of logical connectives.

Note, however, that since semantics and syntax is not really two sides of the same coin here, rather difference of scope (lower-level rules vs. higher-level rules), any completeness results are rather awkward, although they still make sense. It is no longer what follows vs. what can be shown to follow, but rather what can be shown to follow "semantically" vs. what can be shown to follow "syntactically" (remember section 5.2.1). In other words, the notions of L-derivability and H-derivability should coincide. In the propositional fragment of our system, we should not be able to derive with L-rules something that is not derivable with H-rules and vice versa.

Remark Note that with the help of higher-level judgements, we can explicitly formulate the semantics for our logical constants from chapter 4.3:

- $\bot \Rightarrow false$
- $A \Rightarrow true$ iff (v^A_{true})
- $[\neg A] \Rightarrow true$ iff $A \Rightarrow false$
- $[A \wedge B] \Rightarrow true$ iff $A \Rightarrow true$ and $B \Rightarrow true$
- $[A \vee B] \Rightarrow true$ iff $A \Rightarrow true$ or $B \Rightarrow true$
- $[A \supset B] \Rightarrow true$ iff $A \Rightarrow false$ or $B \Rightarrow true$

6.3 Higher-level non-logical rules

As we hinted in previous section 5.2.2 the utility of having higher-level rules goes beyond just the matters of convenience in regard to logical proofs.

For example, we can use higher-level rules for dealing with other key notion of TIL, which is congruency:

> Two constructions will be called *v-congruent* if they *v*-construct one and the same object or are both *v*-improper. Moreover, they will be called *congruent* if they are *v*-congruent for any *v*.
>
> ([38], p. 62)

First, we need to introduce new form of judgement, which is reserved for constructions only:

$$A \cong B : (\mathbb{C})^{\alpha}$$

It can be read as "construction A is congruent with construction B of type $(\mathbb{C})^{\alpha}$". Analogously, $A \cong_v B : (\mathbb{C})^{\alpha}$ will be read as "A is v-congruent with B of type $(\mathbb{C})^{\alpha}$".

Remark Note that congruency judgements of the form $A \cong B : (\mathbb{C})^{\alpha}$ are H-judgements, even though the missing arrow '$\Rightarrow$' might imply otherwise.

The corresponding Intro/Elim-rules (recall that we are no longer on the lower-level governed by F/E-rules) can be then defined as follows:

$$\frac{A \Rightarrow a : \alpha \qquad B \Rightarrow a : \alpha}{A \cong B : (\mathbb{C})^{\alpha}} \text{ Con-Intro}$$

Informally, two constructions A, B are congruent if they construct the same object.

$$\frac{A \Rightarrow_v a : \alpha \qquad B \Rightarrow_v a : \alpha}{A \cong_v B : (\mathbb{C})^{\alpha}} \text{ Con-Intro2}$$

Two constructions A, B are v-congruent if they construct the same object at the same valuation.

$$\frac{A \Rightarrow \varnothing \qquad B \Rightarrow \varnothing}{A \cong B : (\mathbb{C})^{\alpha}} \text{ Con-Intro3}$$

Two constructions A, B are v-congruent if they are both v-improper.

The Elimination rules:

$$\frac{A \Rightarrow a : \alpha \qquad A \cong B : (\mathbb{C})^{\alpha}}{B \Rightarrow a : \alpha} \text{ Con-Elim}$$

Informally, if construction A constructs object a of type α and A is congruent with B, we can derive that B constructs the very same object.

$$\frac{A \Rightarrow_v a : \alpha \qquad A \cong_v B : (\mathbb{C})^{\alpha}}{B \Rightarrow_v a : \alpha}\ \text{Con-Elim2}$$

Same as above, but relative to valuation v.

For example:

$$\frac{[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12 : \mathbb{N} \qquad [-\ \mathbf{13}\ \mathbf{1}] \Rightarrow 12 : \mathbb{N}}{[+\ \mathbf{5}\ \mathbf{7}] \cong [-\ \mathbf{13}\ \mathbf{1}] : (\mathbb{C})^{\mathbb{N}}}\ \text{Con-Intro}$$

$$\frac{[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12 : \mathbb{N} \qquad [+\ \mathbf{5}\ \mathbf{7}] \cong [-\ \mathbf{13}\ \mathbf{1}] : (\mathbb{C})^{\mathbb{N}}}{[-\ \mathbf{13}\ \mathbf{1}] \Rightarrow 12 : \mathbb{N}}\ \text{Con-Elim}$$

Remark Interpreting the premise $[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12$ of above H-derivation as "Assume that $[+\ \mathbf{5}\ \mathbf{7}] \Rightarrow 12$" might sound odd: $5+7$ equals 12, we do not need to assume anything you might want to say. That is true, however, remember that $+$, **5**, and **7** are not addition function and numbers, but constructions. And since we do not know how they were formed (for that we need L-derivations), we can only assume what they will produce. For example, the $+$ construction could have been formed not from function *add* but from function *multiply* – because it looks syntactically like plus operation does not mean it is plus operation. So assuming results still makes sense in this setup.

Being in possession of these rules can be very useful. Consider e.g., that we have two constructions A and B and know only the result of A, but we also know that they are congruent. Hence, we can get the result of the second construction B for free, so to speak – we do not have to calculate it.

For example:

$$\frac{[+\ \mathbf{1}\ \mathbf{2}] \Rightarrow 3 : \mathbb{N} \qquad [+\ \mathbf{1}\ \mathbf{2}] \cong [-\ [\times\ \mathbf{1909}\ \mathbf{2}]\ [\div\ \mathbf{19075}\ \mathbf{5}]] : (\mathbb{C})^{\mathbb{N}}}{[-\ [\times\ \mathbf{1909}\ \mathbf{2}]\ [\div\ \mathbf{19075}\ \mathbf{5}]] \Rightarrow 3 : \mathbb{N}}\ \text{Con-Elim}$$

In other words, we do not need to execute the construction

$$[-\ [\times\ \mathbf{1909}\ \mathbf{2}]\ [\div\ \mathbf{19075}\ \mathbf{5}]]$$

at all to get its result. It suffices to know that it is congruent with $[+\ \mathbf{1}\ \mathbf{2}]$ and that this construction constructs 3.

Remark Tichý in *FFL* relies only on congruency, but in principle there is nothing preventing us from introducing new judgement forms capturing additional information about constructions. E.g., we could introduce β-convertibility (equality) judgements, that would look e.g., as $[\mathbf{Succ}\ \mathbf{0}] =_\beta [\lambda x\ [\mathbf{Succ}\ x]\ \mathbf{0}] : \mathbb{C}^{(\mathbb{NN})}$, etc.

With our congruency judgements we can formalize Tichý's assertions such as ([38], p. 226):

$$\mathbf{J}^1_{wt}x \quad \text{is congruent with} \quad \exists_{*_1}\lambda c.\&[\mathbf{Z}^1_{wt}xc][\mathbf{Q}^1_{wt}c]$$

as[3]

$$[[[\mathbf{J}^1\ \mathbf{w}]\ \mathbf{t}]\ x] \cong \exists_{\mathbb{C}}\lambda c\ [\wedge\ [[[[\mathbf{Z}^1\ \mathbf{w}]\ \mathbf{t}]\ x]\ c]\ [[[\mathbf{Q}^1\ \mathbf{w}]\ \mathbf{t}]\ c] : (\mathbb{C})$$

Remark With H-judgements and H-rules we can emulate rules and consequently derivations presented in [32]. To briefly demonstrate this consider e.g., the following derivation via (^{0}-INST) rule ([32], p. 238)

$$\frac{{}^0\top : [p \wedge q],\ x : {}^05 \Rightarrow {}^0\top : p}{{}^0\top : [p \wedge q] \Rightarrow {}^0\top : p}$$

which essentially establishes that if ${}^0\top : p$ follows from premise ${}^0\top : [p \wedge q]$ and some additional assumption $x : {}^05$ (i.e., that Trivialization constructions ^{05} is proper), then it follows even without this additional assumption.[4]

The corresponding derivation would look in our system roughly as follows (assuming adoption of the corresponding rule together with sequent style of H-rules instead of Natural Deduction style utilized so far):

$$\frac{[p \wedge q] \Rightarrow \mathit{true},\ \mathbf{5} \Rightarrow 5 \vdash\ p \Rightarrow \mathit{true}}{[p \wedge q] \Rightarrow \mathit{true} \vdash\ p \Rightarrow \mathit{true}}$$

[3] '$\exists\lambda x\,A$' is shorthand for '$[^0\exists\,[\lambda x\,A]]$'. Also note that the binding is done solely via Closure.

[4] Expressions of the form a : A are called matches and they are essentially used to indirectly state the results of constructions. E.g., (satisfied) match ${}^0\top : [p \wedge q]$ states that construction $[p \wedge q]$ constructs truth value *true* (or more precisely, that both $[p \wedge q]$ and ${}^0\top$ v-construct the same object *true*). For more, see [32].

6.4 Higher-level judgements and improper constructions

In previous section 5.3 we discussed improper constructions. Recall that with the help of H-judgements we can explicitly state that some construction A is improper as

$$A \Rightarrow \varnothing$$

For example:

$$[\div \mathbf{3}\ \mathbf{0}] \Rightarrow \varnothing$$

states that Composition $[\div \mathbf{3}\ \mathbf{0}]$ is improper.

Analogously

$$[\div \mathbf{3}\ x] \Rightarrow_v^{x,0} \varnothing$$

states that Composition $[\div \mathbf{3}\ x]$ is improper given that valuation assigns 0 to Variable x.

With judgements such as these ready, we can now formulate rules specifically for dealing with improper constructions. E.g., some of the associated rules might look as follows:

$$\frac{A \Rightarrow a \qquad [A\ B] \Rightarrow \varnothing}{B \Rightarrow \varnothing}\ \text{Imp1}$$

$$\frac{B \Rightarrow b \qquad [A\ B] \Rightarrow \varnothing}{A \Rightarrow \varnothing}\ \text{Imp2}$$

These rules state that if we have construction A that is not improper and we compose it with some other construction B and the resulting Composition is improper, then we can infer that B itself is an improper construction. Analogously the second rule.

6.5 Double execution rule

Since our earlier restriction to first-order constructions only has been lifted off, we can now attempt to tackle Double execution construction as well. Recall that Tichý defines Double execution as follows:

Definition 5 (Double execution).

If what is constructed by X is itself a constructions, one can execute X and go on and execute the result. We shall speak of this two-stage construction as the *double execution* of X and symbolize it as ^{2}X. For any entity X, the construction ^{2}X is *v*-improper (i.e., yields, relative to *v*, nothing at all) if X is not itself a construction, or if it does not *v*-construct a construction, or if it *v*-constructs a *v*-improper construction. Otherwise ^{2}X *v*-constructs what is *v*-constructed by what is *v*-constructed by X.

([38], p. 64)

So Double execution executes constructions twice over in succession. More specifically, it executes the result of the first execution. Of course, in our system Execution is no longer a construction, but a construction rule. Naturally, this should extend to Double execution as well. Hence, we will have to devise new double execution rule for constructions.

Before we get to the rule itself, let's take a step back and look at the executions again. We already know that we can write down the fact that e.g., "construction [**Succ 0**] constructs upon execution object 1 of type $\mathbb{N}$" as

$$[\textbf{Succ 0}] \Rightarrow 1 : \mathbb{N}$$

After the introduction of higher-order constructions, we also know that we can state the fact that "construction $[\![\textbf{Succ 0}]\!]$ constructs upon execution object [**Succ 0**] of type $(\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})$" as[5]

$$[\![\textbf{Succ 0}]\!] \Rightarrow [\textbf{Succ 0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})$$

Now, with double execution, we are essentially going to introduce new kind of H-judgement stating "construction $[\![\textbf{Succ 0}]\!]$ constructs upon double execution object [**Succ 0**] of type $(\mathbb{C}^{(\mathbb{N}\mathbb{N})}, \mathbb{C}^{\mathbb{N}})$" that we will write down as

$$[\![\textbf{Succ 0}]\!] \Rightarrow\Rightarrow 1 : \mathbb{N}$$

which virtually amounts to[6]

$$[\![\textbf{Succ 0}]\!] \Rightarrow [\textbf{Succ 0}] \Rightarrow 1 : \mathbb{N}$$

[5] Recall that '$[\![\textbf{Succ 0}]\!]$' stands for '^{0}[**Succ 0**]'.

[6] This is, of course, not a valid form of a judgement (notice the symbol '$\Rightarrow$' appears twice), we are just using it to better explain the reasoning behind double execution.

Hence, the basic idea of double execution is the following: it conceals one execution (or rather, runs it in the background) and yields back the result of executing the result of this "implicit" execution. Hence, if we "unpack" it correctly, we should rather end up with something like this

$$[\![\textbf{Succ 0}]\!] \Rightarrow [\textbf{Succ 0}] \text{ and } [\textbf{Succ 0}] \Rightarrow 1 : \mathbb{N}$$

Let's try to capture this behaviour of double execution in the form of a H-rule:

$$\frac{A : (\mathbb{K}) \Rightarrow B : \mathbb{C} \qquad B : (\mathbb{C}) \Rightarrow b : \alpha}{A : (\mathbb{K}) \Rightarrow\Rightarrow b : \alpha} \text{ DE}$$

Note that the rule essentially expresses certain form of "transitivity": "if A constructs B, and B constructs b (on single execution), then A constructs b (on double execution)".

So for the case above the corresponding derivation would look as follows:

$$\frac{[\![\textbf{Succ 0}]\!] : \mathbb{K}^{(\mathbb{C}^{(\mathbb{N}\mathbb{N})},\mathbb{C}^{\mathbb{N}})} \Rightarrow [\textbf{Succ 0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})},\mathbb{C}^{\mathbb{N}}) \qquad [\textbf{Succ 0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})},\mathbb{C}^{\mathbb{N}}) \Rightarrow 1 : \mathbb{N}}{[\![\textbf{Succ 0}]\!] : \mathbb{K}^{(\mathbb{C}^{(\mathbb{N}\mathbb{N})},\mathbb{C}^{\mathbb{N}})} \Rightarrow\Rightarrow 1 : \mathbb{N}} \text{ DE}$$

Note that the rule (DE) is general enough, so we do not need to introduce specific double execution rule for each construction (as was required on the lower-level). This is due to the fact that (DE) is higher-level rule, not a lower-level rule.

For example, consider the case of second-order Variable s that constructs construction **Succ**:

$$\frac{s : \mathbb{K}^{\mathbb{C}^{(\mathbb{N}\mathbb{N})}} \Rightarrow_v \textbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})} \qquad \textbf{Succ} : \mathbb{C}^{(\mathbb{N}\mathbb{N})} \Rightarrow succ : (\mathbb{N}\mathbb{N})}{s : \mathbb{K}^{\mathbb{C}^{(\mathbb{N}\mathbb{N})}} \Rightarrow\Rightarrow_v succ : (\mathbb{N}\mathbb{N})} \text{ DE}$$

Remark The (DE) rule can be applied only to constructions of type $\mathbb{K}$ and higher. Consider e.g., we try to double execute first-order construction:

$$\frac{[\textbf{Succ 0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})},\mathbb{C}^{\mathbb{N}}) \Rightarrow 1 : \mathbb{N} \qquad 1 : \mathbb{N} \Rightarrow \varnothing}{[\textbf{Succ 0}] : (\mathbb{C}^{(\mathbb{N}\mathbb{N})},\mathbb{C}^{\mathbb{N}}) \Rightarrow\Rightarrow \varnothing} \text{ DE}$$

This derivation is, however, not possible in our system, because the second premise of the (DE) rule requires object of type $\mathbb{C}$, i.e., a construction. In other words, we do not allow non-executable objects to be executed, which seems as reasonable restriction. In TIL, however, derivations such as this one are theoretically possible due to the presence of constructions such as ${}^1 1$ (i.e., execution of non-construction 1).

Remark It is possible to generalize the (DE) rule further beyond second-order constructions. The crucial information to remember is that both premises have to be always constructions, while the construction that appears in the second premise to the left of '⇒' has to be always of one order lower than the construction that appears in the first premise to the left of '⇒'. Schematically:[7]

$$\frac{A : (\mathbb{C})_n \Rightarrow B : (\mathbb{C})_{n-1} \qquad B : (\mathbb{C})_{n-1} \Rightarrow C : (\mathbb{C})_{n-2}}{A : (\mathbb{C})_n \Rightarrow\Rightarrow C : (\mathbb{C})_{n-2}}\ \text{DE2}$$

where n is construction order. Of course, this schema would work only for constructions of type $\mathbb{C}_3$ and higher. Once A is of type $\mathbb{C}_1$, i.e., once we reach the limit case where constructions start to construct non-constructions, we have to switch to the earlier rule (DE).

[7] Remember that '$\mathbb{C}$' and '$\mathbb{K}$' stand for '$\mathbb{C}_1$' and '$\mathbb{C}_2$'.

Chapter 7

Concluding remarks

Tichý wrote:

> In modern logic and metamathematics it has become customary to study constructions in an indirect way. Instead of dealing with constructions themselves, one deals with *formulas* which serve to express those constructions in a particular notational system. The notion of construction has dropped, in fact, from the conceptual system of modern logic altogether.
>
> ([22], p. 62)

Consequently, he let only constructions into his language. But, in a way, he was guilty of the same oversight he criticized in others. He condemned them for their "no constructions" approach, while he himself championed "only constructions" approach.

We think there is a middle ground to be found, which we tried to show in this work. System eTIL presented here advocates a moderate position of having in language explicitly both constructions and non-constructions, which seems to be ultimately the more general approach (not being able to talk about procedures seems to be just as restrictive as not being able to speak about the results of these procedures). In doing so, however, we depart greatly from the philosophy behind TIL.

Štěpán recently wrote:[1]

[1] In Czech original: "Srovnáme-li inferencialismus s transparentní intenzionální logikou, máme tu dva extrémy. TIL preferuje analýzu a má pro ni dobrý aparát. Pokud jde ale o dedukci, můžeme

> If we compare inferentialism and transparent intensional logic, we have two extremes. TIL prefers analysis and has a good apparatus for it. But when it comes down to deduction, we can always appeal back to Pavel Tichý's dictum: "Once we know what we are talking about, we will also know what follows from what." Therefore entailment remains mainly on the intuitive level. But maybe it would be fruitful to somehow connect these two conceptions.
>
> ([42], pp. 121–122)

We agree with both of his last two points: (1) entailment in TIL as presented in *FFL* indeed remains mostly just a matter of intuition only and (2) tipping the scales away from analysis and more towards deduction would be indeed beneficial. After all, eTIL is attempting exactly this, i.e., finding balance between these two so called extremes.

But not only that. The work offered here is also an attempt to open up TIL to new audiences by employing new method of explicit rules. A goal whose fulfilment can be ultimately judged only by the individual readers. That said, however, with eTIL we are not trying something new just for the sake of being new. There are important aspects of TIL (How do we form constructions? What is exactly execution? What it means that construction constructs something? etc. discussed at length in chapters 2, 3, and 5) that require more elucidation if we wish to move TIL as a whole forward. And it is these features that initially sparked the creation of eTIL – a system that tries to reconstruct TIL from ground up by our new method of explicit rules.

With eTIL we proposed formal platform with enough rules to take TIL from *FFL* off the ground. However, to turn it into truly full-fledged system for analysis of natural and formal languages yet even more is needed. As it currently stands, it is still only a little more than proof of concept. The main focus of future development should be especially in computations rules (including so called conversions rules, partiality, substitution, and binding of Variables) and supplying additional higher-level rules tailored specifically for dealing with more intricate aspects natural language analysis.

---

se vždy odvolat na prohlášení Pavla Tichého: "Když budeme vědět, o čem mluvíme, budeme také vědět, co z čeho vyplývá." Tedy vyplývání zde zůstává především na intuitivní úrovni. Možná by ale bylo plodné nějaké spojení obou koncepcí."

Appendix A

Definitions from *FFL*

Key definitions from *FFL*. We keep the original list labels for better cross-referencing.

Definition 16.1. ([38], p. 66)

Let B be a base.

1. (t_1i) Every member of B is a *type of order* 1 *over* B.

   ($t_1$ii) If $0 < m$ and $\alpha, \beta_1 ... \beta_m$ are *types of order* 1 *over B* then the collection $(\alpha\beta_1 ... \beta_m)$ of all m-ary (total and partial) mappings from $\beta_1 ... \beta_m$ into α is also a *type of order* 1 *over* B.

   ($t_1$iii) Nothing is a *type of order* 1 *over* B unless it so follows from (t_1i) and ($t_1$ii).

2. (c_ni) Let τ be any *type of order n over* B. Every variable ranging over τ is a *construction of order n over* B. If A is of (i.e., belongs to) type τ then ^{0}A, ^{1}A, and ^{2}A are *constructions of order n over* B. Every variable ranging over τ is a *construction of order n over* B.

   ($c_n$ii) If $0 < m$ and $X_0, X_1, ..., X_m$ are *constructions of order n* then $[X_0\, X_1 ... X_m]$ is a *construction of order n over* B. If $0 < m$, τ is a *type of order n over* B, and Y as well as the distinct variables $x_1, ..., x_m$ are *constructions of order n over* B, then $[\lambda_\tau x_1 ... x_m\ Y]$ is a *construction of order n over* B.

   ($c_n$iii) Nothing is a *construction of order n over* B unless it so follows from (c_ni) and ($c_n$ii).

Let $*_n$ be the collection of *constructions of order n over* B. The collection of *types of order* $n+1$ *over* B is defined as follows:

(t_{n+1}i) $*_n$ and every *type of order n* is a *type of order* $n+1$.

(t_{n+1}ii) If $0 < m$ and $\alpha, \beta_1, ..., \beta_m$ are *types of order* $n+1$ *over* B then the collection $(\alpha\beta_1...\beta_m)$ of all m-ary (total and partial) mappings from $\beta_1, ..., \beta$ into α is also a *type of order* $n+1$ *over* B.

(t_{n+1}iii) Nothing is a *type of order* $n+1$ *over* B unless it so follows from (t_{n+1}i) and (t_{n+1}ii).

Definition 17.1 (Freedom of Variables). ([38], pp. 73–74)

Let d be a variable.

1. d is *free* in d.
2. If d is *free in* A then d is *free in* ^{1}A and ^{2}A. If d is *free in* X_0 or X_1 or ... or X_m, then d is *free in* $[X_0X_1...X_m]$. If d is *free in* Y and is distinct from the variables x_1,..., x_m then d is *free in* $[\lambda_\tau x_1....x_m Y]$.
3. d is not *free in* any construction unless it so follows from 1 and 2.

The substitution operation can be now defines as follows:

Definition 17.2 (Substitution). ([38], p. 74)

Let C and D be constructions and d a variable. If d is not free in C then *the result of substituting* D *for* d *in* C is C. Assume, therefore, that d is free in C.

1. If C is d then *the result of substituting* D *for* d *in* C is D. If C is ^{1}A or ^{2}A then *the result of substituting* D *for* d *in* C is ^{1}B and ^{2}B respectively, where B is the *result of substituting* D *for* d *in* A.
2. If C is $[X_0X_1...X_m]$ then *the result of substituting* D *for* d *in* C is $[Y_0Y_1...Y_m]$, where Y_0, Y_1,..., and Y_m are *the results of substituting* D *for* d *in* X_0, X_1,..., and X_m, respectively. Now let C be of the form $[\lambda_\tau x_1...x_m Y]$; for $1 \leq i \leq m$, let y_i be x_i if x_i is not free in D, and otherwise the first variable of the same type as x_i, not occurring in C, not free in D, and distinct from y_1,..., y_{i-1};

then *the result of substituting* D *for* d *in* C is $[\lambda_\tau y_1 ... y_m Z]$, where Z is *the result of substituting* D *for* d *in the result of substituting* y_i for x_i $(1 \leq i \leq m)$ *in* Y.

Bibliography

[1] ANDREWS, P. B. *Transfinite Type Theory with Type Variable*. North-Holland Series on Logic and the Foundations of Mathematics. North-Holland Publishing Company, 1965.

[2] CHURCH, A. A formulation of the simple theory of types. *Journal of Symbolic Logic 5*, 2 (06 1940), 56–68.

[3] CURRY, H., AND FEYS, R. *Combinatory Logic*, vol. 1 of *Combinatory Logic*. North-Holland Publishing Company, 1958.

[4] DALRYMPLE, M. *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. Bradford books. Bradford Books, 1999.

[5] DE BRUIJN, N. Automath, a language for mathematics. Tech. rep., Department of Mathematics, Eindhoven University of Technology.

[6] DUŽÍ, M. The paradox of inference and the non-triviality of analytic information. *Journal of Philosophical Logic 39*, 5 (2010), 473–510.

[7] DUŽÍ, M., AND JESPERSEN, B. Transparent quantification into hyperintensional contexts de re. *Logique et Analyse 55*, 220 (2012), 513–554.

[8] DUŽÍ, M., AND JESPERSEN, B. Transparent quantification into hyperintensional objectual attitudes. *Synthese 192*, 3 (2015), 635–677.

[9] DUŽÍ, M., JESPERSEN, B., AND MATERNA, P. *Procedural Semantics for Hyperintensional Logic: Foundations and Applications of Transparent Intensional Logic*. Logic, Epistemology, and the Unity of Science. Springer, 2010.

[10] FRANCEZ, N. *Proof-theoretic Semantics*. College Publications, 2015.

[11] GABBAY, D. M. *Labelled Deductive Systems*. No. v. 1 in Oxford Logic Guides. Oxford University Press, USA, 1996.

[12] GENTZEN, G. Untersuchungen über das logische schließen. I. *Mathematische Zeitschrift 39*, 1 (1935), 176–210.

[13] GENTZEN, G., AND SZABO, M. E. *The collected papers of Gerhard Gentzen*. Studies in logic and the foundations of mathematics. North-Holland Pub. Co., 1969.

[14] GOTTLOB, F. Über sinn und bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, 100 (1892), 25–50.

[15] HEIJENOORT, J. V. Logic as calculus and logic as language. *Synthese 17*, 3 (1967), 324–330.

[16] HINTIKKA, J. *Lingua Universalis vs. Calculus Ratiocinator: An Ultimate Presupposition of Twentieth-Century Philosophy*. Jaakko Hintikka Selected Papers. Springer Netherlands, 1997.

[17] HORÁK, A. *The Normal Translation Algorithm in Transparent Intensional Logic for Czech [online]*. Phd thesis, Masaryk University, Faculty of Informatics, 2002 [cit. 2014-05-18].

[18] HOWARD, W. A. The formulae-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, H.B., J. Hindley, and J. Seldin, Eds. Academic Press, 1980.

[19] MARTIN-LÖF, P. *Intuitionistic type theory*. Studies in proof theory. Bibliopolis, 1984.

[20] MATERNA, P. *Concepts and Objects*. Acta Philosophica Fennica. Helsinki: Philosophical Society of Finland, Vol. 63. 1998.

[21] MATERNA, P. *Conceptual systems*. Logische Philosophie. Logos, 2004.

[22] ODDIE, G., AND TICHÝ, P. The logic of ability, freedom and responsibility. *Studia Logica 41*, 2-3 (1982), 227–248.

[23] PEANO, G. *Arithmetices Principia: Nova Methodo Exposita*. Fratres Bocca, 1889.

[24] PEYTON JONES, S. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

[25] PEZLAR, I. Tichý's two-dimensional conception of inference. *Organon F: Supplementary Issue 2 20* (2013), 54–65.

[26] PEZLAR, I. Logic as a toolbox. In *The Future of Philosophy*, L. Zámečník, Ed. Palacký University Olomouc, Olomouc, 2014, pp. 73–85.

[27] PEZLAR, I. Towards a more general concept of inference. *Logica Universalis 8*, 1 (2014), 61–81.

[28] PEZLAR, I. Desire-based model of reasoning. In *Beyond Artificial Intelligence: The Disappearing Human-Machine Divide (Topics in Intelligent Engineering and Informatics)* (2015), E. K. J. Romportl, Jan; Zackova, Ed., Springer International Publishing, pp. 143–157.

[29] PRAWITZ, D. *Natural Deduction: A Proof-theoretical Study*. Dover Books on Mathematics Series. Dover Publications, Incorporated, 2006.

[30] RACLAVSKÝ, J. *Jména a deskripce: logicko-sémantická zkoumání*. Olomouc: Nakladatelství, Olomouc, 2009.

[31] RACLAVSKÝ, J., AND KUCHYŇKA, P. Conceptual and derivation systems. *Logic and Logical Philosophy 20*, 1-2 (2011).

[32] RACLAVSKÝ, J., KUCHYŇKA, P., AND PEZLAR, I. *Transparentní intenzionální logika jako characteristica universalis a calculus ratiocinator [Transparent Intensional Logic as Characteristica Universalis and Calculus Ratiocinator]*. Brno: Masarykova univerzita (Munipress), Brno, 2015.

[33] RANTA, A. *Type-theoretical Grammar*. Indices. Clarendon Press, 1994.

[34] SCHROEDER-HEISTER, P. Definitional reasoning in proof-theoretic semantics and the square of opposition. In *In J.-Y. Béziau & A. Costa-Leite (Eds.), Square of Opposition: Proceedings of the International Congress* (2008).

[35] SCHROEDER-HEISTER, P. Proof-theoretic semantics. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., spring 2013 ed. 2013.

[36] TICHÝ, P. Foundations of partial type theory. *Reports on Mathematical Logic*, 14 (1982), 59–72.

[37] TICHÝ, P. Indiscernibility of identicals. *Studia Logica 45*, 3 (1986), 251–273.

[38] TICHÝ, P. *The Foundations of Frege's Logic*. Foundations of Communication. de Gruyter, 1988.

[39] TICHÝ, P., SVOBODA, V., JESPERSEN, B., AND CHEYNE, C. *Pavel Tichý's collected papers in logic and philosophy*. Filosofia, 2004.

[40] TURING, A. M. Computability and λ-definability. *The Journal of Symbolic Logic 2* (12 1937), 153–163.

[41] VAN HEIJENOORT, J. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Source Books in the History of the Sciences. Harvard University Press, 1977.

[42] ŠTĚPÁN, J. O inferencialismu pro Pavla Maternu. *Studia philosophica 62*, 2 (2015), 120–122.

[43] WIĘCKOWSKI, B. Rules for subatomic derivation. *Review of Symbolic Logic 4*, 2 (2011), 219–236.

[44] WIĘCKOWSKI, B. Refinements of subatomic natural deduction. *Journal of Logic and Computation* (2014).