Hỏemūi: A semantics for Toaq

Robin Townsend (*Dỏaqrēa*)

Contents

1 A relational lambda calculus

Toaq is a rather high-level language, which is appropriate for regular conversation. However, for our purposes, it would be easier if we had a simpler, more regular way to represent Toaq sentences. So, in order to talk more rigorously about its semantics, we're going to create a formal symbolic language for which we will provide rules for translating to and from Toaq.

Specifically, we'll be creating a typed relational lambda calculus, which will enable us to use higher-order logic while not losing nice classical properties. That it is 'typed' means that all variables and constants will have a type associated with them, denoting them as one of three kinds of things: an individual (ι), a proposition (o), or a relation (denoted with parentheses, as in $(\iota\ \iota\ o)$). That it is a 'relational lambda calculus' means that it has syntax for creating anonymous relations on the fly with λ. This is a bit different from the classical lambda calculus, in that ours is deliberately built around relations, rather than functions. More on the difference later.

As an example, here's what the Toaq sentence *dåqmīq gâı zåq sa pỏq hóe da* will look like:

$$\begin{aligned}&\exists j^{(\iota)}.\ \exists h^{(\iota)}.\ \text{Jî}\ j \wedge \text{Hỏe}\ h \wedge \text{Rủaq}\ j\ \lambda r^{o}.\ r = \lambda.\\ &\quad \text{Dåqmīq}\ \lambda d^{o}.\ d = \lambda.\\ &\qquad \exists p^{(\iota)}.\ \text{Pỏq}\ p \wedge \Big(\lambda a^{(\iota)}\ b^{(\iota)}.\ \text{Gåı}\ a\ \lambda g^{o}.\ g = \lambda.\ \text{Zåq}\ b\Big)\ p\ h\end{aligned} \tag{1}$$

1.1 Syntax

So... let's create a language! We're going to need to lay down the syntax and typing rules.

First of all, we have a set of variables. These are single lowercase letters with or without subscripts: $\{a, a_0, a_1, \ldots, b, b_0, b_1, \ldots, z, z_0, z_1, \ldots\}$. We also have some constants, represented by capitalized Toaq words with type annotations to disambiguate them when needed, such as Hỏe, Chỉe, Jỏe$^{((o))}$, and Jỏe$^{((\iota)\ ((\iota)))}$.

Variables and constants form *terms*. Terms can have the following types:

- ι, the type of individuals
- o, the type of propositions
- $(\tau_1\ \ldots\ \tau_n)$, the type of n-ary relations between types $\tau_1, \ldots, \tau_n$

From these we build *formulas*, which are defined by the following rules (listed in order of precedence):

- *Atoms*: Words known as 'nullary predicates', such as Hỏejēaq and Tåqjȳ, are formulas by themselves.

- *Application*: If r is a term of type $(\tau_1 \ \ldots \ \tau_n)$ and $t_1, \ldots, t_n$ are terms of types $\tau_1, \ldots, \tau_n$, then $r \ t_1 \ \ldots \ t_n$ is a formula.
- *Equality*: If t_1 and t_2 are terms of type τ, $t_1 = t_2$ is a formula.
- *Negation*: If ϕ is a formula, $\neg\phi$ is a formula.
- *Modality*: If $\Box$ is a modal operator[1] and ϕ is a formula, $\Box\phi$ is a formula.
- *Conjunction*: If ϕ and ψ are formulas, $\phi \wedge \psi$ is a formula.
- *Disjunction*: If ϕ and ψ are formulas, $\phi \vee \psi$ is a formula.
- *Implication*: If ϕ and ψ are formulas, $\phi \to \psi$ is a formula.
- *Bi-implication*: If ϕ and ψ are formulas, $\phi \leftrightarrow \psi$ is a formula.
- *Existential quantification*: If v is a variable, τ is a type, and ϕ is a formula in which v has type τ, then $\exists v^\tau.\ \phi$ is a formula.
- *Universal quantification*: If v is a variable, τ is a type, and ϕ is a formula in which v has type τ, then $\forall v^\tau.\ \phi$ is a formula.

Precedence can of course be overridden with parentheses. By convention, $\to$ associates to the right, while all other connectives associate to the left.

Additionally, we have a couple of ways to create a term out of a formula by means of *abstraction*:

- *Proposition abstraction*: If ϕ is a formula, $\lambda.\ \phi$ is a term of type o.
- *Relation abstraction*: If $v_1, \ldots, v_n$ are variables, $\tau_1, \ldots, \tau_n$ are types, and ϕ is a formula in which $v_1, \ldots, v_n$ have types $\tau_1, \ldots, \tau_n$, then $\lambda v_1^{\tau_1} \ \ldots \ v_n^{\tau_n}.\ \phi$ is a term of type $(\tau_1 \ \ldots \ \tau_n)$.

And that's all the syntax we need! We have stated the typing rules here only in words instead of in the traditional inference rule style, but hopefully it should be obvious how they work.

1.2 Constants

Earlier we mentioned a few examples of the constants present in this language, which generally speaking represent Toaq predicates. However, since Toaq's predicates are often overloaded in their arity and type, we need to be more specific about how these are translated into our set of relation constants.

In the dictionary, predicates are given definitions for specific arities. However, as we know, any binary predicate will also have unary and nullary forms available, even if they are not explicitly stated. Normally these follow a predictable pattern of existential closure over the missing arguments—for example, *kůeq* corresponds to *lů kůeq hóa sa* when used as a unary predicate, and *kůeq sa sa* when used as a nullary predicate. But other predicates, such as *dů*, behave entirely differently

[1] We'll introduce the modal operators of this language later. For now, just know they exist.

at lower arities, having extra definitions with different argument structures that don't simply follow the existential closure pattern. We call these the *explicit arities* of a predicate, while the *implicit arities* are those than can be derived from an explicit, higher arity simply via existential closure.

A few examples:

- *bǐa* has an explicit arity 2, and implicit arities 1 and 0.
- *dǔ* has explicit arities 2 and 1, and an implicit arity 0.
- *hǒejēaq* has an explicit arity 0.
- *rǔqshūa* has explicit arities 2 and 0, and an implicit arity 1.[2]

In addition to arity information, we need to know what types these predicates may accept as arguments. Currently no such comprehensive list of type signatures is available, but it would be fairly straightforward to compile one, so we can pretend that agreed-upon type signatures exist. Note that some predicates, such as binary *jěo*, are in fact generic over the types of their arguments, giving infinitely many possible type signatures: $((\iota)\ ((\iota)))$, $((o)\ ((o)))$, $(((\iota\ o))\ (((\iota\ o))))$, ... We will have separate constants for each of these instances.

So, for every predicate in the dictionary, each of its explicit arities are assigned a relation constant, with generically-typed predicates receiving an infinite series of constants. If a predicate has only a single explicit arity with a single possible type, then its type signature is ommitted; otherwise, type signatures are added as a superscript to distinguish the different versions of an overloaded predicate. For predicates with an explicit arity of 0, they are added to the set of atomic formulas.

One more thing we must be careful of is that there are some Toaq predicates which aren't actually relations at all, such as *hǒa*, *pǒı*, and *pě*. These 'pseudo-predicates' are simply syntactic sugars that happen to have the grammar of predicates, and as such we will not provide relation constants for them—they will instead have special translation rules.

1.3 Deductive system

Now we come to the core of this language's purpose. In order to actually give meaning to the syntax we have established, we need to provide a *deductive system* as a way of obtaining proofs. From a proof-theoretic perspective, this is exactly what will determine our language's semantics.

We will be using a system of *natural deduction*, which is a common way to formulate proofs. We prove things by starting out with no assumptions, then

[2] 0 is an explicit arity of *rǔqshūa*, because while its binary and unary forms can talk about rain anywhere in the world, its nullary form supposedly only means "it is raining *here*". This is not the same as "something rains onto somewhere".

stating any relevant axioms and using *inference rules* to manipulate them and arrive at a conclusion. Here is an example of an inference rule:

$$\frac{\phi \quad \psi \quad \cdots}{\chi}\ \text{foo}$$

What this rule tells us is that, given the premises above the line (ϕ, ψ, etc.), we may infer the conclusion below the line (χ). The bit to the right of the line (foo) is simply the rule's name. If a rule requires *no* premises, then it is an axiom—something which we may assert in any context.

We will now give the inference rules necessary to define our language's logical connectives. These come in two varieties: *introduction* and *elimination* rules. These show how to introduce and eliminate the use of an operator, respectively.

First, the rules for implication:

$$\frac{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}{\phi \to \psi}\ \to_{\text{intro}} \qquad \frac{\phi \to \psi \quad \phi}{\psi}\ \to_{\text{elim}}$$

The elimination rule should be familiar—it is simply *modus ponens*. The introduction rule, on the other hand, is our equivalent of the deduction theorem. It says that if assuming ϕ allows us to derive ψ, we may conclude $\phi \to \psi$. The bit above the line can be thought of as a sub-proof in which we are allowed to make such an assumption, rather than a normal premise.

The rules for conjunction, disjunction, and bi-implication are straightforward. Note that each of them come with multiple introduction or elimination rules, since connectives represent multiple possibilities.

$$\frac{\phi \quad \psi}{\phi \wedge \psi}\ \wedge_{\text{intro}} \qquad \frac{\phi \wedge \psi}{\phi}\ \wedge_{\text{elim 1}} \qquad \frac{\phi \wedge \psi}{\psi}\ \wedge_{\text{elim 2}}$$

$$\frac{\phi}{\phi \vee \psi}\ \vee_{\text{intro 1}} \qquad \frac{\psi}{\phi \vee \psi}\ \vee_{\text{intro 2}} \qquad \frac{\phi \vee \psi \quad \begin{array}{c}\phi\\ \vdots\\ \chi\end{array} \quad \begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}{\chi}\ \vee_{\text{elim}}$$

$$\frac{\phi \to \psi \quad \psi \to \phi}{\phi \leftrightarrow \psi}\ \leftrightarrow_{\text{intro}} \qquad \frac{\phi \leftrightarrow \psi}{\phi \to \psi}\ \leftrightarrow_{\text{elim 1}} \qquad \frac{\phi \leftrightarrow \psi}{\psi \to \phi}\ \leftrightarrow_{\text{elim 2}}$$

Now for negation:

$$\frac{\begin{array}{c}\phi\\ \vdots\\ \psi \wedge \neg\psi\end{array}}{\neg\phi}\ \neg_{\text{intro}} \qquad \frac{\begin{array}{c}\neg\phi\\ \vdots\\ \psi \wedge \neg\psi\end{array}}{\phi}\ \neg_{\text{elim}}$$

The introduction rule tells us that in order to prove the negation of ϕ, we first assume ϕ, and then show how this leads to a contradiction ($\psi \wedge \neg\psi$). This gives

us the law of non-contradiction for free:

Rule. $\dfrac{}{\neg(\phi \wedge \neg\phi)}\ NC$

Derivation. $\dfrac{\phi \wedge \neg\phi}{\neg(\phi \wedge \neg\phi)}\ \neg_{\text{intro}}$ □

The elimination rule is also rather important, as it is what makes our system classical rather than intuitionistic. From it, we can prove *ex falso quodlibet*, double negation elimination, and the law of excluded middle.

Rule. $\dfrac{\phi \wedge \neg\phi}{\psi}\ EFQ$

Derivation. $\dfrac{\phi \wedge \neg\phi}{\psi}\ \neg_{\text{elim}}$ □

Rule. $\dfrac{\neg\neg\phi}{\phi}\ \neg\neg_{elim}$

Derivation. $\dfrac{\dfrac{\neg\phi \qquad \neg\neg\phi}{\neg\phi \wedge \neg\neg\phi}\ \wedge_{\text{intro}}}{\phi}\ \neg_{\text{elim}}$ □

Rule. $\dfrac{}{\phi \vee \neg\phi}\ XM$

Derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg\phi}{\phi \vee \neg\phi}\ \vee_{\text{intro 2}} \qquad \neg(\phi \vee \neg\phi)}{(\phi \vee \neg\phi) \wedge \neg(\phi \vee \neg\phi)}\ \wedge_{\text{intro}}}{\phi}\ \neg_{\text{elim}} \qquad \dfrac{\dfrac{\dfrac{\phi}{\phi \vee \neg\phi}\ \vee_{\text{intro 1}} \qquad \neg(\phi \vee \neg\phi)}{(\phi \vee \neg\phi) \wedge \neg(\phi \vee \neg\phi)}\ \wedge_{\text{intro}}}{\neg\phi}\ \neg_{\text{intro}}}{\phi \wedge \neg\phi}\ \wedge_{\text{intro}}}{\phi \vee \neg\phi}\ \neg_{\text{elim}}$$

□

So far, these have all been rules of propositional logic—that is, not involving terms. To work with terms, we need to provide rules for quantifiers, which gets a little tricky. First, universal quantification:

$$\dfrac{\phi}{\forall v^{\tau}.\ \phi}\ \forall_{\text{intro}} \qquad \dfrac{\forall v^{\tau}.\ \phi}{\phi^{t/v}}\ \forall_{\text{elim}}$$

The introduction rule is fairly straightforward—given ϕ (which may contain v as a free variable), we can state that ϕ holds for all values of type τ. For the elimination rule, $\phi^{t/v}$ is the same as ϕ, except with some term t of type τ substituted for all free occurrences of v. Thus, we may go from a general statement to a specific one, or simply free v from its binding.

Now for existential quantification:

$$\frac{\phi^{C/v}}{\exists v^\tau.\ \phi}\ \exists_{\text{intro}} \qquad \frac{\exists v^\tau.\ \phi \qquad \begin{array}{c}\phi^{C/v}\\ \vdots \\ \psi\end{array}}{\psi}\ \exists_{\text{elim}}$$

For both rules, $\phi^{C/v}$ is the same as ϕ, except with some constant C substituted for all free occurrences of v. In the elimination rule, C must be a new constant not already in scope—one that we make up simply for the purpose of argument within the sub-proof. Thus, this constant must not occur in our conclusion ψ.

It follows that these quantifiers are duals:

Rule. $\dfrac{\forall v^\tau.\ \phi}{\neg\exists v^\tau.\ \neg\phi}\ \forall_{dual}$

Derivation.
$$\dfrac{\exists v^\tau.\ \neg\phi \qquad \dfrac{\dfrac{\dfrac{\forall v^\tau.\ \phi}{\phi^{C/v}}\ \forall_{\text{elim}} \qquad \neg\phi^{C/v}}{\phi^{C/v} \wedge \neg\phi^{C/v}}\ \wedge_{\text{intro}}}{\psi \wedge \neg\psi}\ \text{EFQ}}{\dfrac{\psi \wedge \neg\psi}{\neg\exists v^\tau.\ \neg\phi}\ \neg_{\text{intro}}}\ \exists_{\text{elim}}$$
□

Rule. $\dfrac{\exists v^\tau.\ \phi}{\neg\forall v^\tau.\ \neg\phi}\ \exists_{dual}$

Derivation.
$$\dfrac{\exists v^\tau.\ \phi \qquad \dfrac{\dfrac{\phi^{C/v} \qquad \dfrac{\forall v^\tau.\ \neg\phi}{\neg\phi^{C/v}}\ \forall_{\text{elim}}}{\phi^{C/v} \wedge \neg\phi^{C/v}}\ \wedge_{\text{intro}}}{\psi \wedge \neg\psi}\ \text{EFQ}}{\dfrac{\psi \wedge \neg\psi}{\neg\forall v^\tau.\ \neg\phi}\ \neg_{\text{intro}}}\ \exists_{\text{elim}}$$
□

Another thing we will want to do with terms is the substitution of equals. Here are the rules for equality, which tell us how to do just that:

$$\frac{}{t = t}\ =_{\text{intro}} \qquad \frac{t_1 = t_2 \qquad \phi^{t_1/v}}{\phi^{t_2/v}}\ =_{\text{elim}}$$

The introduction rule says that equality is reflexive. Any equivalence relation should also be symmetric and transitive, which we can indeed prove from these rules.

Rule. $\dfrac{t_1 = t_2}{t_2 = t_1}\ =_{sym}$

Derivation. $\dfrac{t_1 = t_2 \qquad \dfrac{}{t_1 = t_1}\ =_{\text{intro}}}{t_2 = t_1}\ =_{\text{elim}}$ □

Rule. $\dfrac{t_1 = t_2 \qquad t_2 = t_3}{t_1 = t_3}\ =_{trans}$

Derivation. $\dfrac{\dfrac{t_1 = t_2}{t_2 = t_1}\ =_{\text{sym}} \qquad t_2 = t_3}{t_1 = t_3}\ =_{\text{elim}}$ □

Lastly, we define relation abstraction. Here we use a double line as a shorthand to indicate that the rule is reversible, giving both the introduction and elimination rules in a condensed form.

$$\frac{\phi^{t_1/v_1\ \ldots\ t_n/v_n}}{(\lambda v_1^{\tau_1}\ \ldots\ v_n^{\tau_n}.\ \phi)\ t_1\ \ldots\ t_n}\ \lambda_{\text{intel}}$$

Going in the introduction direction, the rule states that given any formula containing the terms $t_1, \ldots, t_n$, we may abstract the relation between them into a λ-term applied to the original arguments. The elimination direction, which substitutes the terms back in, is the equivalent of β-reduction from the classical lambda calculus. We can also rename λ-bound variables by applying α-conversion, as in the classical lambda calculus.

Rule. $\dfrac{(\lambda v_1^{\tau_1}\ \ldots\ v_n^{\tau_n}.\ \phi)\ t_1\ \ldots\ t_n}{\left(\lambda w_1^{\tau_1}\ \ldots\ w_n^{\tau_n}.\ \phi^{w_1/v_1\ \ldots\ w_n/v_n}\right)\ t_1\ \ldots\ t_n}\ \lambda_{\alpha\text{-}conv}$

Derivation. $\dfrac{\dfrac{(\lambda v_1^{\tau_1}\ \ldots\ v_n^{\tau_n}.\ \phi)\ t_1\ \ldots\ t_n}{\phi^{t_1/v_1\ \ldots\ t_n/v_n}}\ \lambda_{\text{elim}}}{\left(\lambda w_1^{\tau_1}\ \ldots\ w_n^{\tau_n}.\ \phi^{w_1/v_1\ \ldots\ w_n/v_n}\right)\ t_1\ \ldots\ t_n}\ \lambda_{\text{intro}}$ □

These inference rules cover everything but modality, proposition abstraction, and the behaviors of constants. Since those topics take longer to cover, we'll be amending our deductive system with rules for them as we go along.

2 Propositions and truth

$$\frac{\text{True } \lambda.\ \neg\phi}{\text{False } \lambda.\ \phi}\ \text{False}_{\text{intel}}$$

$$\frac{\text{True } \lambda.\ \phi \qquad \text{True } \lambda.\ \psi}{\text{True } \lambda.\ \phi \wedge \psi}\ \text{True}_{\wedge\ \text{intro}}$$

$$\frac{\text{True } \lambda.\ \phi \wedge \psi}{\text{True } \lambda.\ \phi}\ \text{True}_{\wedge\ \text{elim 1}} \qquad \frac{\text{True } \lambda.\ \phi \wedge \psi}{\text{True } \lambda.\ \psi}\ \text{True}_{\wedge\ \text{elim 2}}$$

$$\frac{\text{False } \lambda.\ \phi}{\text{False } \lambda.\ \phi \wedge \psi}\ \text{False}_{\wedge\ \text{intro 1}} \qquad \frac{\text{False } \lambda.\ \psi}{\text{False } \lambda.\ \phi \wedge \psi}\ \text{False}_{\wedge\ \text{intro 2}}$$

$$\frac{\text{True } \lambda.\ \phi \to \psi \qquad \text{True } \lambda.\ \phi}{\text{True } \lambda.\ \psi}\ \text{True}_{\to\ \text{elim}}$$

$$\frac{\exists v^{\tau}.\ \text{False } \lambda.\ \phi}{\text{False } \lambda.\ \forall v^{\tau}.\ \psi}\ \text{False}_{\forall\ \text{intro}}$$

$$\frac{\text{True } t \qquad \text{False } t}{\phi}\ \text{True}_{\text{EFQ}}$$

$$\frac{\begin{array}{c}\vdots\\ \phi\end{array}}{\text{True } \lambda.\ \phi}\ \text{True}_{\text{intro}} \qquad \frac{\begin{array}{c}\vdots\\ \text{True } \lambda.\ \phi\end{array}}{\phi}\ \text{True}_{\text{elim}}$$

$$\frac{\forall v^{o}.\ t\ v \to \text{True } v}{\text{Jẻo}^{((o))}\ t}\ \text{Jẻo}_{\text{intel}}^{((o))}$$

$$\frac{\forall v_1^{\tau}.\ \forall v_2^{(\tau)}.\ t_1\ v_1 \to t_2\ v_2 \to v_2\ v_1}{\text{Jẻo}^{((\tau)\ ((\tau)))}\ t_1\ t_2}\ \text{Jẻo}_{\text{intel}}^{((\tau)\ ((\tau)))}$$

$$\frac{\forall v^{o}.\ t\ v \to \text{False } v}{\text{Bů}^{((o))}\ t}\ \text{Bů}_{\text{intel}}^{((o))}$$

$$\frac{\forall v_1^{\tau}.\ \forall v_2^{(\tau)}.\ t_1\ v_1 \to t_2\ v_2 \to \neg v_2\ v_1}{\text{Bů}^{((\tau)\ ((\tau)))}\ t_1\ t_2}\ \text{Bů}_{\text{intel}}^{((\tau)\ ((\tau)))}$$