A SYNTHESIZING TRAINING DATA

The generating algorithm takes as input a set of symbols `Sym` (e.g. all MitM-symbols for which an alignment to SMGLoM exists) and a starting symbol $s \in$ `Sym` (e.g. `nattimes`; binary multiplication on natural numbers). The algorithm then proceeds as follows:

1. If $s : T$ has a (simple or dependent) function type, we fill in the required arguments. For $s =$`nattimes`, our type is $T =$`Nat`$\to$`Nat`$\to$`Nat`, hence we need to find two arguments s_1, s_2 of type `Nat`. For each s_i of required type T_i we proceed as follows:
   (a) With probability p_{var}, we introduce a new variable $v : T_i$ from a list of allowed variable names (which include variants such as a, a', a_0 etc.) and let $s_i := v$.
   (b) With probability p_{fun}, we pick a symbol $f \in$ `Sym` with a function type with return type T_i (e.g. for $T_i =$`Nat`, we can pick `natplus`). In that case, we let $s := f$, recurse, and set s_i as the result.
   (c) With probability $p_{const} = 1 - p_{var} - p_{fun}$, we pick a constant symbol $c \in$ `Sym` of type T_i (e.g. for $T_i =$`Nat` we can pick 0) and return $s_i := c$.

   In order to avoid stack overflows, we reduce p_{fun} in each iteration by a certain factor < 1. As to not overuse certain symbols, we scale p_{fun} and p_{const} with the number of respectively suitable symbols available; if `Sym` contains no suitable function or constant symbols, we let $p_{fun} = 0$ (and/or $p_{const} = 0$, respectively).
2. If $s : T$ does *not* have a function type (or all its parameters have been filled in 1.), then s is well-typed and we return s with probability $1 - p_{up}$.
   With probability p_{up}, we instead pick a new symbol $s_f \in S$ of some function type such that some i-th parameter type of s_f is T. In that case, we let $s_i := s$ and $s := s_f$ and recurse.
   Again, in order to avoid stack overflows we reduce p_{up} by some factor with each iteration.

The algorithm also takes subtyping into account, e.g. whenever a term of type `Real` is required, terms of type `Int` or `Nat` are used with some probability.

In order to obtain a sentence in the sense of Section 5 providing context for disambiguation, we first translate t along alignments to SMGLoM (using a random `\symvariant`), collect the set V of all free variables of t and *verbalize* their types. For that, we associate each *type* with a set of *verbalizations* from which we choose randomly to produce a sentence that introduces the variables before using them in the generated expression. Figure 3 shows a few example verbalizations for a variable x of type `Nat` and generated sentences for the input symbol $s =$`realuminus`; the negation on real numbers.

The verbalizations are categorized as *prefixed* (e.g. "*a natural number* n") or *suffixed* (e.g. "n *a natural number*"), and *singular* or *plural*, and picked according to the number of variables of the same type and the surrounding sentence, which is also picked at random (e.g. "*Assume we have ...*" uses prefixed, whereas "*Let ...*" uses suffixed).

B EVALUATION TACTICS

For every LaTeX input S_{LaTeX}, expected label S_{sTeX} and returned sentence S_R, we employ the following strategies, the results of which are summarized in Figure 4:

`islatex` We parse S_R into an AST. Success implies that S_R is syntactically valid LaTeX. This might fail for "minor" reasons such as a missing closing bracket. It might yield false positives in cases where macros (not explicitly considered by our parser) occurring in S_R have a wrong number of arguments.

All subsequent evaluation strategies require `islatex` to succeed.

`stexcheck` We heuristically check whether S_R is in $\mathcal{L}_{\text{sTeX}}$ – unlike `islatex`, this requires that all sTeX macros occurring in S_R have the right number of arguments. Success does *not* tell us that the input has been disambiguated *correctly*, but *does* imply that is *has* been disambiguated *at all*. False negatives can occur if S_R (and thus likely S_{LaTeX} as well)

	Generated sTEX	PDF output
Verbalizations	`$\inset{x}{\NaturalNumbers}$` a positive integer `$x$` an integer `$\intmethan{x}{0}$` a natural number `$x$`	$x \in \mathbb{N}$ a positive integer x an integer $x \geq 0$ a natural number x
Sentences	`Assume we have some $\inset{y'}{\NaturalNumbers}$ and arbitrary $\inset{\mathcal F}{\IntegerNumbers}$. It follows that $\realuminus{\realuminus{\inttimes[x]{\mathcal F,y',y'}}}$.`	Assume we have some $y' \in \mathbb{N}$ and arbitrary $\mathcal{F} \in \mathbb{Z}$. It follows that $--(\mathcal{F} \times y' \times y')$.
	`Let $\natmorethan n{0}$. Then consider $\realuminus{\realuminus{\natsucc{\natsucc n}}}$.`	Let $n > 0$. Then consider $--S(S(n))$.
	`Whenever we have some positive natural number $\varepsilon$, any integer $\ell$ and a real number $\livar{\mathcal C}{2}$, then it follows that $\realtimes{\livar{\mathcal C}{2},\livar{\mathcal C}{2},\realplus{\realuminus{\ell},\natsucc{\varepsilon}}}$.`	Whenever we have some positive natural number ε, any integer ℓ and a real number $\mathcal{C}_2$, then it follows that $\mathcal{C}_2\mathcal{C}_2(-\ell + S(\varepsilon))$.

Figure 3: Example Verbalizations for x :Nat and Generated Sentences

contains complex variable names, or if S_R contains e.g. an equality symbol "=" instead of the corresponding sTEX macro, which LaTeXML could recover.

`eval_latex` All sTEX macros occurring in S_R are expanded and S_R is normalized as described in Section 5. The result is string-compared to S_{LATEX}. Success thus implies, that the notational presentation in PDF output of S_{LATEX} and S_R will coincide. False negatives can occur due to minor differences e.g. in not strictly necessary brackets.

`omdoc` S_R is translated to OMDOC using LaTeXML and imported to MMT. Success guarantees syntactic well-formedness of S_R. Since both the LaTeXML-OMDOC export and the subsequent MMT-import are somewhat brittle, this can easily lead to false negatives.

`translated` The import from `omdoc` is translated to the typed MitM library. This entails that all symbols used in S_R are aligned with MitM symbols and S_R is amenable for formal knowledge management services.

`inferred` The translation to MitM obtained from `translated` is type checked by MMT by having its type inferred. Success guarantees that S_R is well-typed.

Notably, if S_R is a mere variable (e.g. the expression `$n$`), it does not actually have an inferrable type, but succeeds trivially. This accounts for 60 of the entries in our evaluation set, i.e. 37%.

`provided_stex` Both the expected label S_{sTEX} and S_R are normalized and string-compared. Success implies that S_R is definitely the correct translation. False negatives can easily occur due to non-semantic differences between S_{sTEX} and S_R however, such as bracketing, nested applications in S_R (e.g. `$\natplus{\natplus{a,b},c}$` vs. `$\natplus{a,b,c}$`), etc.

`stex_as_omdoc` S_{sTEX} is translated to OMDOC via LaTeXML and directly compared to the OMDOC-term obtained from `omdoc`. Like `provided_stex`, success implies that S_R is correct, but it is more fault-tolerant with respect to the precise syntax of S_R, while being *less* fault tolerant due to the issues mentioned in `omdoc`.

The first three evaluations can always be applied; from the remaining, all but `provided_stex` require a working installation of LaTeXML and its sTEX-Plugin. The last two require a known correct translation.

Total inputs	161
`islatex`	96.9%
`stexcheck`	60.2%
`eval_latex`	64.0 %
`omdoc`	76.4%
`translated`	63.5%
`inferred`	59.6%
`provided_stex`	47.2 %
`stex_as_omdoc`	53.4 %

Figure 4: Results on our Evaluation Document

A detailed log file on our evaluation document with the individual results for each input and evaluation is available in the associated git repository.