Parsing Algebraic Word Problems into Equations

Rik Koncel-Kedziorski, Hannaneh Hajishirzi,
Ashish Sabharwal†, Oren Etzioni†, and Siena Dumas Ang
University of Washington, †Allen Institute for AI
{kedzior,hannaneh,sienaang}@uw.edu, {ashishs,orene}@allenai.org

Abstract

This paper formalizes the problem of solving multi-sentence algebraic word problems as that of generating and scoring equation trees. We use integer linear programming to generate equation trees and score their likelihood by learning local and global discriminative models. These models are trained on a small set of word problems and their answers, without any manual annotation, in order to choose the equation that best matches the problem text. We refer to the overall system as ALGES.

We compare ALGES with previous work and show that it covers the full gamut of arithmetic operations whereas Hosseini et al. (2014) only handle addition and subtraction. In addition, ALGES overcomes the brittleness of the Kushman et al. (2014) approach on single-equation problems, yielding a 15% to 50% reduction in error.

1 Introduction

Grade-school algebra word problems are brief narratives (see Figure 1). A typical problem first describes a partial world state consisting of characters, entities, and quantities. Next it updates the condition of an entity or explicates the relationship between entities. Finally, it poses a question about a quantity in the narrative.

An ordinary child has to learn the required algebra, but will easily grasp the narrative utilizing extensive world knowledge, large vocabulary, word-sense disambiguation, coreference resolution, mastery of syntax, and the ability to combine individual

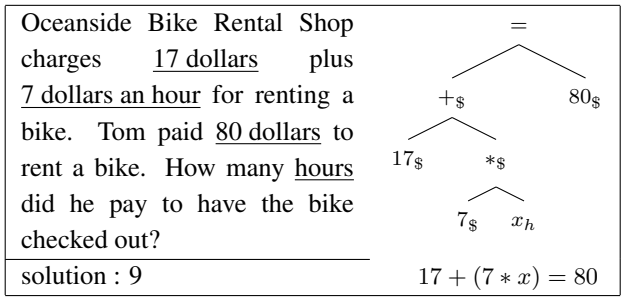

Figure 1: Example problem and solution

sentences into a coherent mental model. In contrast, the challenge for an NLP system is to "make sense" of the narrative, which may refer to arbitrary activities like renting bikes, collecting coins, or eating cookies.

Previous work coped with the open-domain aspect of algebraic word problems by relying on deterministic state transitions based on verb categorization (Hosseini et al., 2014) or by learning templates that cover equations of particular forms (Kushman et al., 2014). We have discovered, however, that both approaches are brittle, particularly as training data is scarce in this domain, and the space of equations grows exponentially with the number of quantities mentioned in the math problem.

We introduce ALGES,[1] which maps an unseen multi-sentence algebraic word problem into a set of possible equation trees. Figure 1 shows an equation tree alongside the word problem it represents.

ALGES generates the space of trees via Integer Linear Programming (ILP), which allows it to con-

[1]The code and data is publicly available at `https://gitlab.cs.washington.edu/ALGES/TACL2015`.

Transactions of the Association for Computational Linguistics, vol. 3, pp. 585–597, 2015. Action Editor: Regina Barzilay.
Submission batch: 7/2015; Revision batch: 10/2015; Published 12/2015.

strain the space of trees to represent type-consistent algebraic equations satisfying as many desirable properties as possible. ALGES learns to map spans of text to arithmetic operators, to combine them given the global context of the problem, and to choose the "best" tree corresponding to the problem. The training set for ALGES consists of unannotated algebraic word problems and their solution. Solving the equation represented by such a tree is trivial. ALGES is described in detail in Section 4.

ALGES is able to solve word problems with single-variable equations like the ones in Figure 1. In contrast to Hosseini et al. (2014), ALGES covers $+, -, *$, and $/$. The work of Kushman et al. (2014) has broader scope but we show that it relies heavily on overlap between training and test data. When that overlap is reduced, ALGES is 15% to 50% more accurate than this system.

Our contributions are as follows: (1) We formalize the problem of solving multi-sentence algebraic word problems as that of generating and ranking equation trees; (2) We show how to score the likelihood of equation trees by learning discriminative models trained from a small number of word problems and their solutions – without any manual annotation; and (3) We demonstrate empirically that ALGES has broader scope than the system of Hosseini et al. (2014), and overcomes the brittleness of the method of Kushman et al. (2014).

2 Previous Work

Our work is related to situated semantic interpretation, which aims to map natural language sentences to formal meaning representations (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Ge and Mooney, 2006; Kwiatkowski et al., 2010). More closely related is work on language grounding, whose goal is the interpretation of a sentence in the context of a world representation (Branavan et al., 2009; Liang et al., 2009; Chen et al., 2010; Bordes et al., 2010; Feng and Lapata, 2010; Hajishirzi et al., 2011; Matuszek et al., 2012; Hajishirzi et al., 2012; Artzi and Zettlemoyer, 2013; Koncel-Kedziorski et al., 2014; Yatskar et al., 2014; Seo et al., 2014; Hixon et al., 2015). However, while most previous work considered individual sentences in isolation, solving word problems often requires reasoning across the multi-sentence discourse of the problem text. Recent efforts in the math domain have studied number word problems (Shi et al., 2015), logic puzzle problems (Mitra and Baral, 2015), arithmetic word problems (Hosseini et al., 2014; Roy and Roth, 2015), algebra word problems (Kushman et al., 2014; Zhou et al., 2015), and geometry word problems (Seo et al., 2015). We discuss in more detail below two pioneering works closely related to our own.

Hosseini et al. (2014) solve elementary addition and subtraction problems by learning verb categories. They ground the problem text to a semantics of *entities* and *containers*, and decide if quantities are increasing or decreasing in a container based upon the learned verb categories. While relying only on verb categories works well for $+$ and $-$, modeling $*$ or $/$ requires going beyond verbs. For instance, "Tina has 2 cats. John has 3 more cats than Tina. How many cats do they have together?" and "Tina has 2 cats. John has 3 times as many cats as Tina. How many cats do they have together?" have identical verbs, but the indicated operation (+ and * resp.) is different. ALGES makes use of a richer semantic representation which facilitates deeper learning and a wider scope of application, solving problems involving the $+, -, /$, and $*$ operators (see Table 6).

Kushman et al. (2014) introduce a general method for solving algebra problems. This work can align a word problem to a system of equations with one or two unknowns. They learn a mapping from word problems to equation templates using global and local features from the problem text. However, the large space of equation templates makes it challenging for this model to learn to find the best equation directly, as a sufficiently similar template may not have been observed during training. Instead, our method maps word problems to equation trees, taking advantage of a richer representation of quantified nouns and their properties, as well as the recursive nature of equation trees. These allow ALGES to use a bottom-up approach to learn the correspondence between spans of texts and arithmetic operators (corresponding to intermediate nodes in the tree). ALGES then scores equations using global structure of the problem to produce the final result.

Our work is also related to research in using ILP to enforce global constraints in NLP appli-

cations (Roth and Yih, 2004). Most previous work (Srikumar and Roth, 2011; Goldwasser and Roth, 2011; Berant et al., 2014; Liu et al., 2015) utilizes ILP as an inference procedure to find the best global prediction over initially trained local classifiers. Similarly, we use ILP to enforce global and domain specific constraints. We, however, use ILP to form candidate equations which are then used to generate training data for our classifiers. Our work is also related to parser re-ranking (Collins, 2005; Ge and Mooney, 2005), where a re-ranker model attempts to improve the output of an existing probabilistic parser. Similarly, the global equation model designed in ALGES attempts to re-rank equations based on global problem structure.

3 Setup and Problem Definition

Given numeric quantities V and an unknown x whose value is the answer we seek, an *equation* over V and x is any valid mathematical expression formed by combining elements of $V \cup \{x\}$ using binary operators from $\mathcal{O} = \{+, -, *, /, =\}$ such that x appears exactly once. When each element of V appears at most once in the equation, it may naturally be represented as an *equation tree* where each operator is a node with edges to its two operands.[2] T denotes the set of all equation trees over V and x.

Problem Formulation. We address the problem of solving grade-school algebra word problems that map to single equations. Solving such a word problem w amounts to selecting an equation tree t representing the mathematical computation implicit in w. Figure 1 shows an example of w with quantities underlined, and the corresponding tree t. Formally, we use a joint probability distribution $p(t, w)$ that defines how "well" an equation tree $t \in T$ captures the mathematical computation expressed in w. Given a word problem w as input, our goal is to compute $\tilde{t} = \arg\max_{t \in T} p(t|w)$.

An exhaustive enumeration over T quickly becomes impractical as problem complexity increases and $n = |V \cup \{x\}|$ grows. Specifically, $|T| > h(n) = n!\,(n-1)!\,(n-1)\,2^{n-4}$, $h(4) = 432$, $h(6) > 1.7M$, $h(8) > 22B$, etc. This vast search space makes it challenging for a discriminative model to learn to find $\tilde{t}$ directly, as a sufficiently similar tree may not have been observed during training. Instead, our method first generates syntactically valid equation trees, and then uses a bottom-up approach to score equations with a *local* model trained to map spans of text to math operators, and a *global* model trained for coherence of the entire equation w.r.t. global problem text structure.

[2] Problems involving simultaneous equations require combining multiple equation trees, one per equation.

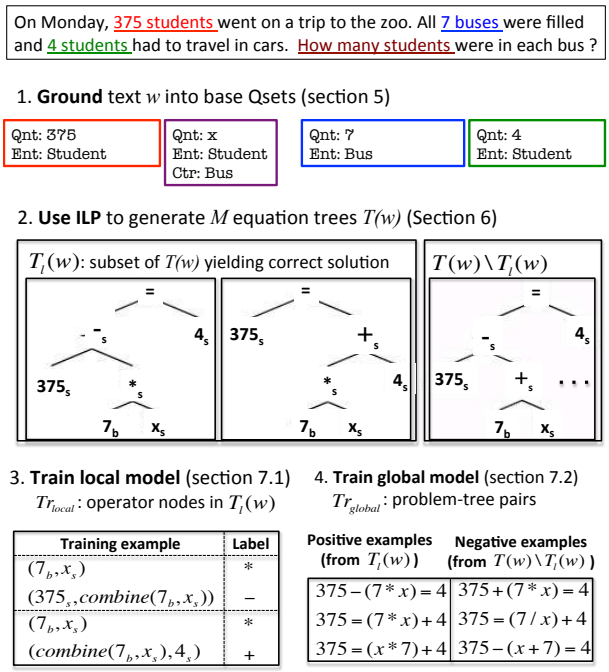

Figure 2: An overview of the process of learning for a word problem and its Qsets.

4 Overview of the Approach

Figure 2 gives an overview of our method, also detailed in Figure 3. In order to build equation trees, we use a compact representation for each node called a *Quantified Set* or Qset to model natural language text quantities and their properties (e.g., '375 students' in '7 buses'). Qsets are used for tracking and combining quantities when learning the correspondence between equation trees and text.

Definition 1. Given a math word problem w, let $\mathbb{S}$ be the set of all possible spans of text in w, ϕ denote the empty span, and $S^{\phi} = S \cup \{\phi\}$. A **Qset** for w is either a base Qset or a compound Qset. A *base Qset* is a tuple $(ent, qnt, adj, loc, vrb, syn, ctr)$ with:

- $ent \in \mathbb{S}$: *entity* or quantity noun (e.g., 'student');
- $qnt \in \mathbb{R} \cup \{x\}$: *number* or quantity (e.g., 4 or x);

Learning (word problems W, corresponding solutions L):

1. For every problem-solution pair (w_i, ℓ_i) with $w_i \in W, \ell_i \in L$
   (a) $S \leftarrow$ Base Qsets obtained by Grounding text w_i and Reordering the resulting Qsets (Section 5)
   (b) $T_i \leftarrow$ Top M type-consistent equation tree candidates generated by ILP(w_i) (Section 6)
   (c) $T_{\ell_i} \leftarrow$ Subset of T_i that yields the correct numerical solution ℓ_i
   (d) Add to Tr_{local} features $\langle s_1, s_2 \rangle$ with label *op* for each operator *op* combining Qsets s_1, s_2 in trees in T_{ℓ_i}
   (e) Add to Tr_{global} features $\langle w, t \rangle$ labeled positive for each $t \in T_{\ell_i}$ and labeled negative for each $t \in T \setminus T_{\ell_i}$
2. $\mathcal{L}_{local} \leftarrow$ Train a local Qset relationship model on Tr_{local} (Section 7.1)
3. $\mathcal{G}_{global} \leftarrow$ Train a global equation model on Tr_{global} (Section 7.2)
4. Output local and global models $(\mathcal{L}_{local}, \mathcal{G}_{global})$

Inference (word problem w, local set relation model $\mathcal{L}_{local}$, global equation model $\mathcal{G}_{global}$):

1. $S \leftarrow$ Base Qsets obtained by Grounding text w_i and Reordering the resulting Qsets (Section 5)
2. $T \leftarrow$ Top M type-consistent equation tree candidates generated by ILP(w) (Section 6)
3. $t^* \leftarrow \arg\max_{t_i \in T} \left(\prod_{t_j \in t} \mathcal{L}_{local}(t_j|w) \right) \times \mathcal{G}_{global}(t|w)$, scoring each tree $t_i \in T$ based on Equation 1
4. $\ell \leftarrow$ Numeric solution to w obtained by solving equation tree t^* for the unknown
5. Output (t^*, ℓ)

Figure 3: Overview of our method for solving algebraic word problems.

- $adj \subseteq \mathbb{S}^{\phi}$: *adjectives* for ent in w;
- $loc \in \mathbb{S}^{\phi}$: *location* of ent (e.g., 'in the drawer');
- $vrb \in \mathbb{S}^{\phi}$: *governing verb* for ent (e.g., 'fill');
- syn: *syntactic* and positional information for ent (e.g., 'buses' is in subject position) ;
- $ctr \subseteq \mathbb{S}^{\phi}$: *containers* of ent (e.g., 'Bus' is a container for the 'students' Qset).

Properties being ϕ indicates these optional properties are unspecified. A *compound Qset* is formed by combining two Qsets with a non-equality binary operator as discussed in section 5.

Qsets can be further combined with the equality operator to yield a *semantically augmented equation tree*.[3] The example in Figure 2 has four base Qsets extracted from problem text. Each possible equation tree corresponds to a different recursive combination of these four Qsets.

Given w, ALGES first extracts a list of n base Qsets $S = \{s_1, \ldots, s_n\}$ (Section 5). It then uses an ILP-based optimization method to combine extracted Qsets into a list of type-consistent candidate equation trees (Section 6). Finally, ALGES uses discriminative models to score each candidate equation, using both local and global features (Section 7).

Specifically, the recursive nature of our representation allows us to decompose the likelihood function $p(t, w)$ into local scoring functions for each internal node of t followed by scoring the root node:

$$p(t|w) \propto \left(\prod_{t_j \in t} \mathcal{L}_{local}(t_j|w) \right) \times \mathcal{G}_{global}(t|w) \quad (1)$$

where the local function $\mathcal{L}_{local}(t_j|w)$ scores the likelihood of the subtree t_j, modeling pairwise Qset relationships, while the global function $\mathcal{G}_{global}(t|w)$ scores the likelihood of the root of t, modeling the equation in its entirety.

Learning. ALGES learns in a *weakly supervised* fashion, using word problems w_i and only their correct answer ℓ_i (not the corresponding equation tree) as training data $\{(w_i, \ell_i)\}_{i \in \{1,\ldots,N\}}$. We ground each w_i into ordered Qsets and generate a list of type-consistent candidate training equations T_{ℓ_i} that yield the correct answer ℓ_i.

We build a *local* discriminative model $\mathcal{L}_{local}$ to score the likelihood that a math operator $op \in \mathcal{O}$ can correctly combine two Qsets s_1 and s_2 based on their semantics and intertextual relationships. For example, in Figure 2 this model learns that $*$ has a high likelihood score for '7 buses' and 'x students'. The training data consists of feature vectors $\langle s_1, s_2 \rangle$ labeled with op, derived from the equation trees that yield the correct solution.

We also build a *global* discriminative model that scores equation trees based on the global problem structure: $\mathcal{G}_{global} = \psi^{\intercal} f_{global}(w, t)$ where f_{global}

[3] Inspired by Semantically Augmented Parse Trees (Ge and Mooney, 2005) adapted to equational logic.

represents global features of w and t, and ϕ are parameters to be learned. The training data consists of feature vectors $\langle w, t \rangle$ for equation trees that yield the correct solution as positive examples, and the rest as negatives (Figure 2). The details of learning and inference steps are described in Section 7.

5 Grounding and Combining Qsets

We discuss how word problem text is grounded into an ordered list of Qsets. A Qset is a compact representation of the properties of a quantity as described in a single sentence. The use of Qsets facilitates the building of semantically augmented equation trees. Additionally, by tracking certain properties of text quantities, ALGES can resolve pronominal references or elided nouns to properties of previous Qsets. It can also combine information about quantities referenced in different sentences into a single semantic structure for further use.

Grounding. ALGES translates the text of the problem w into interrelated base Qsets $\{s_1, \ldots, s_n\}$, each associated with a quantity in the problem text w. The properties of each Qset (Definition 1) are extracted from the dependency parse relations present in the sentence where the quantity is referred to according to the rules described in Table 1.

Additionally, ALGES assigns a single target Qset s_x corresponding to the question sentence. The properties of the target Qset are also extracted according to the rules of the Table 1. In particular, the qnt property is set to unknown, the ent is set to the noun appearing after the words *what, many* or *much* in the target sentence, and the other properties are extracted as listed in Table 1.

Reordering. In order to reduce the space of possible equation trees, ALGES reorders Qsets $\{s_1, \ldots, s_n\}$ according to semantic and textual information and enforces a constraint that Qsets can only combine with adjacent Qsets in the equation tree. In Figure 2, the target Qset corresponding to the unknown (x 'students') is moved from its textual location at the end of the problem and placed adjacent to the Qset with entity 'buses'. This move is triggered by the relationship between the target entity 'student' and its container 'bus' that is quantified by *each* in the last sentence. In addition to the container match rule, we employ three other rules to move the target Qset as described in Table 2.[4]

For each quantity mentioned in the text, properties $(qnt, ent, ctr, adj, vrb, loc)$ of the corresponding Qset are extracted as follows:
1. qnt (quantity) is a numerical value or determiner found in the problem text, or a variable.
2. ent (entity) is a noun related to the qnt in the dependency parse tree. If qnt is a numerical value, ent is the noun related by the *num, number,* or *prep_of* relations. If qnt is a determiner, ent is the noun related via the *det* relation. When such a noun does not exist due to parse failure or pragmatic recoverability, ent is the noun that is closest to qnt in the same sentence or the ent associated with the most recent Qset.
3. ctr (container) is the subject of the verb governing ent, except in two cases: when this subject is a pronominal reference, the ctr is set to the ctr of the closest previous Qset; if ent is related to another Qset whose qnt is one of *each, every, a, an, per,* or *one*, ctr is set to the ent of that Qset.
4. adj (adjectives) is a list of adjectives related to ent by the *amod* relation.
5. vrb (verb) is a governing verb, either related to ent by *nsubj* or *dobj*
6. loc (location) is a noun related to *ent* by *prep_on, prep_in,* or *prep_at* relations.

Table 1: The process of forming a single Qset.

Combining. Two Qsets and an arithmetic operator can be combined via the $combine$ function to form a third Qset, alternately referred to as a *compound*. Because of this, we can represent intermediate nodes in the equation tree as Qsets themselves. The recursive combination of Qsets allows us to effectively decompose equation trees into a collection of local operations over identical abstractions. This enables learning features of Qsets and text that indicate particular operations from both leaf and intermediate nodes. The mechanics of $c \leftarrow combine(a, b, op)$ are detailed below.

For $op = +$, the properties of either Qset a or b suffice to define c. ALGES always forms c using the properties of b in these situations. For $op = -$, the properties of the left operand a define the resultant set, as evidenced by the subtraction operations present in the first problem in Table 9. To determine

[4]These reordering rules are intentionally minimal, but do provide some gain over both preserving the text ordering of quantities or setting ordering as a soft constraint. See Table 7.

1. Move Qset s_i to immediately after Qset s_j if the container of s_i is the entity of s_j and is quantified by *each*.
2. Move target Qset to the front of the list if the question statement includes keywords *start* or *begin*.
3. Move target Qset to the end of the list if the question statement includes keywords *left, remain,* and *finish*.
4. Move target Qset to the textual location of an intermediate reference with the same *ent* if its *num* property is the determiner *some*.

Table 2: Rules for reordering Qsets.

the stickers in Luke's possession, we need to track stickers related to the left Qset with the verb 'got'.

For $op = *$, the Qset relationship is captured by the container and entity properties: the one whose properties preserve after multiplication has the other's entity as its container. In Figure 2, the 'bus' Qset is the container of 'students'. When these are combined with the $*$ operator, the result is of entity type 'student'. For $op = /$, we use the properties of the left operand to encourage a distinction between division and multiplication.

6 Generating Equation Trees with ILP

We use an ILP optimization model to generate equation trees involving n base Qsets. These equation trees are used for both learning and inference steps. ALGES generates an ordered list of M of the most desirable candidate equations for a given word problem w using an ILP, which models global considerations such as type consistency and appropriate low expression complexity. To facilitate generation of equation trees, we represent them in parenthesis-free *postfix* or *reverse Polish* notation, where a binary operator immediately follows the two operands it operates on (e.g., $abc{+}{*}x{=}$).

Given a word problem w with n base Qsets (cf. Table 3 for notation), we build an optimization model ILP(w) over the space of postfix equations $E = e_1 e_2 \ldots e_L$ of length L involving k numeric constants, $k' = n - k$ unknowns, r possible binary operators, and q "types" of Qsets, where type corresponds to the *entity* property of Qsets and determines which binary relationships are permitted between two given Qsets. For single variable equations over binary operators $\mathcal{O}$, $k' = 1, r = |\mathcal{O}| = 5$, and $L = 2n - 1$. For brevity, define $m = n + r$ and let $[j]$ denote $\{1, \ldots, j\}$. Expression E can be evaluated by considering $e_1, e_2, \ldots, e_L$ in order, pushing non-operator symbols on to a stack σ, and, for operator symbols, popping the top two elements of σ, applying the operator to them, and pushing the result back on to σ. The *stack depth* of the e_i is the stack size after e_i has been processed this way.

INPUT	
w	input math word problem
n	number of base Qsets
k	number of numeric constants
k'	number of unknowns (1 for single-var. eqns.)
r	number of binary operators ($r = \|\mathcal{O}\| = 5$)
m	number of possible symbols ($n + r$)
$type_j$	type of j-th base Qset
M	desired number of candidate equation trees
L	desired length of postfix equations ($2n - 1$)
OUTPUT	
E	postfix equation to be generated
e_i	i-th element of E; $i \in [L]$
VARIABLES for $i \in [L]$	
x_i	main ILP variable for i-th symbol of E
c_i	indicator variable: e_i is a numeric constant
u_i	indicator variable: e_i is an unknown
o_i	indicator variable: e_i is an operator
d_i	postfix stack depth of e_i; $d_i \in [L]$
t_i	type of e_i (corresponds to Qset entity); $t_i \in [q]$

Table 3: ILP notation for candidate equations model

Variables. Integer variables $x_1, x_2, \ldots, x_L$ encode which symbol each e_i refers to. Their domain, $[m]$, represents the k numeric constants in the same order as their respective Qsets, followed by the k' unknowns, and finally operators in the order $+, -, *, /, =$. Binary variables c_i, u_i, and o_i indicate whether e_i is a numeric constant, unknown, or operator, resp. Variables d_i with domain $[L]$ equal the postfix stack depth of e_i. Finally, variables t_i with domain $[q]$ indicate the type of e_i. For $j \in [n]$, i.e., for the k constants and k' unknowns, $type_j \in [q]$ denotes the respective Qset entity. Uncertainty in object types may be incorporated easily by treating $type_j$ as a (potentially weighted) subset of $[q]$.

Constraints and Objective Function. Constraints in ILP(w) include syntactic validity, type consistency, and domain specific simplicity considerations. We describe them briefly here, leaving details to the Appendix. The *objective function* minimizes the sum of the weights of violated soft constraints.

Below, (H) denotes hard constraints, (W) weighted soft constraints, and (P) post-processing steps.

Definitional Constraints (H): Constraints over indicator variables c_i, u_i, and o_i ensure they represent their intended meaning, including the invariant $c_i + u_i + o_i = 1$. For stack depth variables, we add $d_1 = 1$ and $d_i = d_{i-1} - 2o_i + 1$ for $i > 1$.

Syntactic Validity (H): Validity of the postfix expression is enforced easily through constraints $o_1 = 0$ and $d_L = 1$. In addition, we add $x_L = m$ and $x_i < m$ for $i < L$ to ensure equality occurs exactly once and as the top-level operator.

Operand Access (H): The second operand of an operator symbol e_i is always e_{i-1}. Its first operand, however, is defined instead by the stack-based evaluation process. $\mathrm{ILP}(w)$ encodes it using an alternative characterization: the first operand of e_i is e_j iff $j \leq i-2$ and j is the largest index such that $d_i = d_j$.

Type Consistency (W): Suppose T_1 and T_2 are the types of the two operands of an operator o, whose type is T_o. Addition and subtraction preserve the type of their operands, i.e., if o is $+$ or $-$, then $T_o = T_1 = T_2$. Multiplication inherits the type of one of its operands, and division inherits the type of its first operand. In both cases, the two operands must be of different types. Formally, if o is $*$, then $T_o \in \{T_1, T_2\}$ and $T_1 \neq T_2$; if o is $/$, then $T_o = T_1 \neq T_2$.

Domain Considerations (H,W): We add a few domain specific constraints based on patterns observed in a small subset of the questions. These include an upper bound on the stack depth, which helps avoid overly complex expressions unsuitable for grade-school algebra, and reducing redundancy by, e.g., disallowing the numeric constant 0 to be an operand of $+$ or $-$ or the second operand of $/$.

Symmetry Breaking (H,W): If a commutative operator is preceded by two numeric constants (e.g., $ab+$), we require the constants to respect their Qset ordering. Every other pair of constants that disrespects its Qset ordering incurs a small penalty.

Negative and Fractional Answers (P): Rather than imposing non-negativity as a complex constraint in $\mathrm{ILP}(w)$, we filter out candidate expressions yielding a negative answer as a post-processing step. Similarly, when all numeric constants are integers, we filter out expressions yielding a fractional answer, again based on typical questions in our datasets.

7 Learning

Our goal is to learn a scoring function that identifies the best equation tree t^* corresponding to an unseen word problem w. Since our dataset consists only of problem-solution pairs $\{(w_i, \ell_i)\}_{i=1,\ldots,N}$, training our scoring models requires producing equation trees matching ℓ_i. For every training instance (w_i, ℓ_i), we use $\mathrm{ILP}(w_i)$ to generate M type-consistent equation tree candidates T_i. To train our local model (section 7.1), we filter out trees from T_i that do not evaluate to ℓ_i, extract all (s_1, s_2, op) triples from the remaining trees, and use feature vectors capturing (s_1, s_2) and labeled with op as training data (see Figure 2). For the global model, we use for training data a subset of T_i with an equal number of correct and incorrect equation trees (section 7.2). Once trained, we use Equation 1 to combine these models to compute a score for each candidate equation tree generated for an unseen word problem at inference time (see Figure 3).

7.1 Local Qset Relationship Model

We train a *local model* of a probability distribution over the math operators that may be used to $combine$ a pair of Qsets. The idea is to learn the correspondence between spans of texts and math operators by examining such texts and the Qsets of the involved operands. Given Qsets s_1 and s_2, the local scoring function scores the probability of each $op \in \{+, -, *, /\}$, i.e., $\mathcal{L}_{local} = \theta^{\intercal} f_{local}(s_1, s_2)$ where f_{local} is a feature vector for s_1 and s_2. Note that either Qset may be a compound (the result of a combine procedure). The goal is to learn parameters θ by maximizing the likelihood of the operators between every two Qsets that we observe in the training data. We model this as a multi-class SVM with an RBF kernel.

Features. Given the richness of the textual possibilities for indicating a math operation, the features are designed over semantic and intertextual relationships between Qsets, as well as domain-specific lexical features. The feature vector includes three main feature categories (Table 4).

First, *single set features* include syntactic and positional features of individual Qsets. For example, they include indicator features for whether elements of a short lexicon of math-specific terms such as

1. Single Qset Features (repeated for B)
   - what argument of its governing verb is A?
   - is A a subset of another set?
   - is A a compound?
   - math keywords found in context of A
   - verb Lin distance from known verb categories (B only)
2. Relational features between Qsets A and B
   - entity match
   - adjective overlap
   - location match
   - distance in text
   - Lin similarity between verbs governing A and B
   - is one a subtype of the other?
   - does one contain the other?
3. Target Quantity features
   - A/B is target Qset
   - A/B entity matches target entity
   - math keywords in target context

4. Root node features
   - # of ILP constraints violated by equation
   - Scores of left and right subtrees of root

Figure 4: Features used for local and global models, for left Qset A and right Qset B

'add' and 'times' appear in the vicinity of the set reference in the text. Also, following Hosseini et al. (2014), we include a vector that captures the distance between the verbs associated with each Qset and a small collection of verbs found to be useful in categorizing arithmetic operations in that work, based upon their Lin Similarity (Lin, 1998).

Second, *relationships between Qsets* are described w.r.t. various Qset properties described in section 4. These include binary features like whether one Qset's *container* property matches the other Qset's *entity* (a strong indicator of multiplication), or the distance between the verbs associated with each set based upon their Lin Similarity.

Third, *target quantity features* check the matching between the target Qset and the current Qset as well as math keywords in the target sentence.

7.2 Global Equation Model

We also train a *global* model that scores equation trees based on the global structure of the tree and the problem text. The global model scores the compatibility of the tree with the soft constraints introduced in Section 6 as well as its correspondence with the problem text. We use a discriminative model: $\mathcal{G}_{global} = \psi^{\intercal} f_{global}(w, t)$ where f_{global} are the features capturing trees and their correspondences with the problem text. We train a global classifier to relate these features through parameters ψ.

Features f_{global} are explained in Table 4. They include the number of violated soft constraints in the ILP, the probabilities of the left and right subtrees of the root as provided by the local model, and global lexical features. Additionally, the three local feature sets are applied to the left and right Qsets.

7.3 Inference

For an unseen problem w, we first extract base Qsets from w. The goal is to find the most likely equation tree with minimum violation of hard and soft constraints. Using $\mathrm{ILP}(w)$ over these Qsets, we generate M candidate equation trees ordered by the sum of the weights of the constraints they violate. We compute the likelihood score given by Eqn. (1) for each candidate equation tree t, use this as an estimate of the likelihood $p(t|w)$, and return the candidate tree t^* with the highest score. In Eqn. (1), the score of t is the product of the likelihood scores given by the local classifier for each operand in t and the Qsets over which it operates, multiplied by the likelihood score given by the global classifier for the correctness of t. If the resulting equation provides the correct answer for w, we consider inference successful.

8 Experiments

This section reports on three experiments: a comparison of ALGES with Kushman et al. (2014)'s template-based method, a comparison of ALGES with Hosseini et al. (2014)'s verb-categorization methods, and ablation studies. The experiments are complicated by the fact that ALGES is limited to single equations, and the verb categorization method can only handle single-equations without multiplication or division. Our main experimental result is to show an improvement over the template-based method on single-equation algebra word problems. We further show that the template-based method depends on lexical and template overlap between its training and test sets. When these overlaps are reduced, the method's accuracy drops sharply. In contrast, ALGES is quite robust to changes in lexical and template overlap (see Tables 4 and 5).

Experimental Setup. We use the Stanford Dependency Parser in CoreNLP 3.4 (De Marneffe et al., 2006) to obtain syntactic information used for grounding and feature computation. For the ILP model, we use CPLEX 12.6.1 (IBM ILOG, 2014) to generate the top $M = 100$ equation trees with a maximum stack depth of 10, aborting exploration upon hitting 10K feasible solutions or 30 seconds.[5] We use Python's SymPy package for solving equations for the unknown. For the local and global models, we use the LIBSVM package to train SVM classifiers (Chang and Lin, 2011) with RBF kernels that return likelihood estimates as the score.

Dataset. This work deals with grade-school algebra word problems that map to single equations with varying length. Every equation may involve multiple math operations including multiplication, division, subtraction, and addition over non-negative rational numbers and one variable. The data is gathered from `http://math-aids.com`, `http://k5learning.com`, and `http://ixl.com` websites and a subset of the data from Kushman et al. (2014) that maps word problems to single equations. We refer to this dataset as SINGLEEQ (see Table 9 for example problems). The SINGLEEQ dataset consists of 508 problems, 1,117 sentences, and 15,292 words.

Baselines. We compare our method with the template-based method (Kushman et al., 2014) and the verb-categorization method (Hosseini et al., 2014). For the template-based method, we use the fully supervised setting, providing equations for each training example.

8.1 Comparison with Template-based Method

We first compare ALGES with the template-based method over SINGLEEQ. We evaluate both systems on the number of correct answers provided and report the average of a 5-fold cross validation. ALGES achieves 72% accuracy whereas the template-based method achieves 67% accuracy, a 15% relative reduction in errors (first columns in Tables 4 and 5). This result is statistically significant with a p-value of 0.018 under a paired t-test.

[5]These hyper-parameters were chosen based on experimentation with a small subset of the questions. A more systematic choice may improve overall performance.

Template Overlap	10.4	7.7	6.3	2.1
ALGES	**0.72**	**0.66**	**0.66**	**0.63**
Template-based	0.67	0.60	0.46	0.26
Error reduction	15%	15%	33%	50%

Table 4: Decreasing template overlap: Accuracy of ALGES versus the template-based method on single-equation algebra word problems. The first column corresponds to the SINGLEEQ dataset, and the other columns are for subsets with decreasing template overlap.

Lexical Overlap	4.3	3.3	2.6	2.5
ALGES	**0.72**	**0.66**	**0.66**	**0.63**
Template-based	0.67	0.60	0.46	0.26
Error reduction	15%	15%	33%	50%

Table 5: Decreasing lexical overlap: Accuracy of ALGES versus the template-based method on single-equation algebra word problems. The first column corresponds to the SINGLEEQ dataset, and the other columns are for subsets with decreasing lexical overlap.

Lexical Overlap. By further analyzing SINGLEEQ, we noted that there is substantial overlap between the content words (common noun, adjective, adverb, and verb lemmas) in different problems. For example, many problems ask for the total number of seashells collected by two people on a beach, with only the names of the people and the number of seashells that each found changed. To analyze the effect of this repetition on the learning methods evaluated, we define a *lexical overlap* parameter as the total number of content words in a dataset divided by the number of unique content words. The two "seashell problems" have a high lexical overlap.

Template Overlap. We also noted that many problems in SINGLEEQ can be solved using the same *template*, or equation tree structure above the leaf nodes. For example, a problem which corresponds to the equation $(9 * 3) + 7$ and a different problem that maps to $(4 * 5) + 2$ share the same template. We introduce a *template overlap* parameter defined as the average number of problems with the same template in a dataset.

Results. In our data, template overlap and lexical overlap co-vary. To demonstrate the brittleness of the template-based method simply, we picked three subsets of SINGLEEQ where both parame-

ters were substantially lower than in SINGLEEQ and recorded the relative performance of the template-based method and of ALGES in Tables 4 and 5. The data used in both tables is the same, but the tables are separated for readability. The first column reports results for the SINGLEEQ dataset, and the other columns report results for the subsets with decreasing template and lexical overlaps. The subsets consist of 254, 127, and 63 questions respectively. We see that as the lexical overlap drops from 4.3 to 2.5 and as the template overlap drops from 10.4 to 2.1, the relative advantage of ALGES over the template methods goes up from 15% to 50%.

While the template-based method is able to solve a wider range of problems than ALGES, its accuracy falls off significantly when faced with fewer repeated templates or less spurious lexical overlap between problems (from 0.67 to 0.26). The accuracy of ALGES also declines from 0.72 to 0.63 across the table, which needs to be investigated further. In future work, we also need to investigate additional settings for the two parameters and to attempt to "break" their co-variance. Nevertheless, we have uncovered an important brittleness in the template-based method and have shown that ALGES is substantially more robust.

8.2 Comparison with Verb-Categorization

The verb-categorization method learns to solve addition and subtraction problems, while ALGES is capable of solving multiplication and division problems as well. We compare against their method over our dataset as well as the dataset provided by that work, here referred to as ADDSUB. ADDSUB consists of addition and subtraction word problems with the possibility of irrelevant distractor quantities in the problem text. The verb categorization method uses rules for handling irrelevant information. An example rule is to remove a Qset whose adjective is not consistent with the adjective of the target Qset. We augment ALGES with rules introduced in this method for handling irrelevant information in ADDSUB.

Results, reported in Table 6, show comparable accuracy between both methods on Hosseini et al. (2014) data. Our method shows a significant improvement versus theirs on the SINGLEEQ dataset due to the presence of multiplication and division operators, as 40% of the problems in our dataset include these operators.

Method	ADDSUB	SINGLEEQ
ALGES	0.77	**0.72**
Verb-categorization	0.78	0.48
Error reduction	-	53%

Table 6: Accuracy of ALGES compared to verb categorization method.

8.3 Ablation Study

In order to determine the effect of various components of our system on its overall performance, we perform the following ablations:

No Local Model: Here, we test our method absent the local information (Section 7.1). That is, we generate equations using all ILP constraints, and score trees solely on information provided by the global model: $p(t|w) \propto \mathcal{G}_{global}(w, t)$.

No Global Model: Here, we test our method without the global information (Section 7.2). That is, we generate equations using only the hard constraints of ILP and score trees solely on information provided by the local model: $p(t|w) \propto \prod_{t_i \in t} \mathcal{L}_{local}(w, t_i)$.

No Qset Reordering: We test our method without the deterministic Qset reordering rules outlined in Section 5. Instead, we allow the ILP to choose the top M equations regardless of order.

Results in Table 7 show that each component of ALGES contributes to its overall performance on the SINGLEEQ corpus. We find that both the Global and Local models contribute significantly to the overall system, demonstrating the significance of a bottom-up approach to building equation trees.

Importance of Features. We also evaluate the accuracy of the local Qset relationship model (Section 7.1) on the task of predicting the correct operator for a pair of Qsets $\langle s_1, s_2 \rangle$ over the SINGLEEQ dataset using a 5-fold cross validation. Table 8 shows the value of each feature group used in the local classifier, and thus the importance of details of the Qset representation.

Method	Accuracy
ALGES	**0.72**
No Local Model	0.50
No Global Model	0.49
No Qset Reordering	0.68

Table 7: Ablation study of each component of ALGES.

Method	Accuracy
Local classifier: Full Feature set	**0.84**
No Single Set Features	0.81
No Set Relation Features	0.75
No Target Features	0.79

Table 8: Accuracy of local classifier in predicting the correct operator between two Qsets and ablating feature sets.

8.4 Qualitative Examples and Error Analysis.

Table 9 shows some examples of problems solved by our method. We analyzed 72 errors made by ALGES on the SINGLEEQ dataset. Table 10 summarizes five major categories of errors.

Problems and equations
Luke had 20 stickers. He bought 12 stickers from a store in the mall and got 20 stickers for his birthday. Then Luke gave 5 of the stickers to his sister and used 8 to decorate a greeting card. How many stickers does Luke have left? $((20 + ((12 + 20) - 8)) - 5) = x$
Maggie bought 4 packs of red bouncy balls, 8 packs of yellow bouncy balls, and 4 packs of green bouncy balls. There were 10 bouncy balls in each package. How many bouncy balls did Maggie buy in all? $x = (((4 + 8) + 4) * 10)$
Sam had 79 dollars to spend on 9 books. After buying them he had 16 dollars. How much did each book cost? $79 = ((9 * x) + 16)$
Fred loves trading cards. He bought 2 packs of football cards for $2.73 each, a pack of Pokemon cards for $4.01, and a deck of baseball cards for $8.95. How much did Fred spend on cards? $((2 * 2.73) + (4.01 + 8.95)) = x$

Table 9: Examples of problems solved by ALGES together with the returned equation.

Parsing errors cause a wrong grounding into the designed representation. For example, the parser treats 'vanilla' as a noun modified by the number '19', leading our system to treat 'vanilla' as the entity of a Qset rather than 'cupcake'. Despite the improvements that come from ALGES, a portion of errors are attributed to grounding and ordering issues. For instance, the system fails to correctly distinguish between the sets of wheels, and so does not get the movement-triggering container relationships right. Semantic limitations are another source of errors. For example, ALGES does not model the semantics of 'three consecutive numbers'. The fourth category refers to errors caused due to lack of world knowledge (e.g., 'week' corresponds to '7 days'). Finally, ALGES is not able to infer quantities when they are not explicitly mentioned in the text. For example, the number of people should be inferred by counting the proper names in the problem.

Error type	Example
Parsing Issues (12%)	Randy needs 53 cupcakes for a birthday party. He already has 7 chocolate cupcakes and 19 vanilla cupcakes. How many more cupcakes should Randy buy?
Grounding & Ordering (19%)	There are 24 bicycles and 14 tricycles in the storage at Danny's apartment building. Each bicycle has 2 wheels and each tricycle has 3 wheels. How many wheels are there in all?
Semantic Limitation (19%)	The sum of three consecutive even numbers is 162. What is the smallest of these numbers?
Lack of Knowledge (32%)	A restaurant sold 63 hamburgers last week. How many hamburgers on average were sold each day?
Inferring quantities (18%)	Sara, Keith, Benny, and Alyssa each have 96 baseball cards. How many dozen baseball cards do they have in all?

Table 10: Examples of different error categories and relative frequencies. Sources of errors are underlined.

9 Conclusion

In this work we have outlined a method for solving grade school algebra word problems. We have empirically demonstrated the value of our approach versus state-of-the-art word problem solving techniques. Our method grounds quantity references, utilizes type-consistency constraints to prune the search space, learns which algebraic operators are indicated by text, and ranks equations according to a global objective function. ALGES is a hybrid of previous template-based and verb categorization state-based methods for solving such problems. By learning correspondences between text and mathematical operators, we extend the method of state updates based on verb categories. By learning to re-rank equation trees using a global likelihood model, we extend the method of mapping word problems to

equation templates.

Different components of ALGES can be adapted to other domains of language grounding that require cross-sentence reasoning. Future work involves extending ALGES to solve higher grade math word problems including simultaneous equations. This can be accomplished by extending the variable grounding step to allow multiple variables, and training the global equation model to recognize which quantities belong to which equation. The code and data for ALGES are publicly available.

Acknowledgments: This research was supported by the Allen Institute for AI (66-9175), Allen Distinguished Investigator Award, and NSF (IIS-1352249). We thank Regina Barzilay, Luke Zettlemoyer, Aria Haghighi, Mark Hopkins, Ali Farhadi, and the anonymous reviewers for their helpful comments.

References

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *TACL*, 1(1):49–62.

Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D. Manning. 2014. Modeling biological processes for reading comprehension. In *EMNLP*.

Antoine Bordes, Nicolas Usunier, and Jason Weston. 2010. Label ranking under ambiguous supervision for learning semantic correspondences. In *ICML*, pages 103–110.

S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *ACL/AFNLP*, pages 82–90.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27.

David L. Chen, Joohyun Kim, and Raymond J. Mooney. 2010. Training a multilingual sportscaster: Using perceptual context to learn language. *JAIR*, 37:397–435.

Michael Collins. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D. Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Yansong Feng and Mirella Lapata. 2010. How many words is a picture worth? automatic caption generation for news images. In *ACL*, pages 1239–1249.

Ruifang Ge and Raymond J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Conference on Computational Natural Language Learning*, pages 9–16.

Ruifang Ge and Raymond J. Mooney. 2006. Discriminative reranking for semantic parsing. In *ACL*.

D. Goldwasser and D. Roth. 2011. Learning from natural instructions. In *IJCAI*.

Hannaneh Hajishirzi, Julia Hockenmaier, Erik T. Mueller, and Eyal Amir. 2011. Reasoning about robocup soccer narratives. In *UAI*, pages 291–300.

Hannaneh Hajishirzi, Mohammad Rastegari, Ali Farhadi, and Jessica K. Hodgins. 2012. Semantic understanding of professional soccer commentaries. In *UAI*.

Ben Hixon, Peter Clark, and Hannaneh Hajishirzi. 2015. Learning knowledge graphs for question answering through conversational dialog. In *NAACL*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, pages 523–533.

IBM ILOG. 2014. IBM ILOG CPLEX Optimization Studio 12.6.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, and Ali Farhadi. 2014. Multi-resolution language grounding with weak supervision. In *EMNLP*.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *ACL*, pages 271–281.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *EMNLP*, pages 1223–1233.

Percy Liang, Michael I. Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *ACL/AFNLP*, pages 91–99.

Dekang Lin. 1998. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 2015. Toward abstractive summarization using semantic representations. In *NAACL*.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. 2012. Learning to parse natural language commands to a robot control system. In *Proc. of the 13th International Symposium on Experimental Robotics (ISER)*, June.

Arindam Mitra and Chitta Baral. 2015. Learning to automatically solve logic grid puzzles. In *EMNLP*.

Dan Roth and Wen-tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In Hwee Tou Ng and Ellen Riloff, editors, *CoNLL*, pages 1–8. Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *EMNLP*.

Min Joon Seo, Hannaneh Hajishirzi, Ali Farhadi, and Oren Etzioni. 2014. Diagram understanding in geometry questions. In *AAAI*.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *EMNLP*.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *EMNLP*.

Vivek Srikumar and Dan Roth. 2011. A joint model for extended semantic role labeling. In *EMNLP*, Edinburgh, Scotland.

Mark Yatskar, Lucy Vanderwende, and Luke Zettlemoyer. 2014. See no evil, say no evil: Description generation from densely labeled images. *Lexical and Computational Semantics (* SEM 2014)*, page 110.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI*, pages 1050–1055.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, pages 658–666.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *EMNLP*.

Appendix: ILP Model Details

Figure 5 summarizes various constraints of our ILP model for generating candidate equations. $op1_i^{idx}$ is an auxilary variable whose value, when x_i is an operator, is the index in the postfix expression of the first operand of x_i. If $op1_i^{idx} = j$, auxiliary variables $op1_i^x$, $op1_i^t$, $op1_i^o$, and $op1_i^u$ mirror x_j, t_j, o_j, and u_j, respectively. s_e denotes the corresponding constant or operator symbol e (e.g., '+', '=', '0', etc.) in the postfix expression being constructed. H and W, as before, represent hard and weighted soft constraints.

Definitional Constraints (H) :

$c_i = \mathbb{I}(x_i \leq k), i \in [L]$

$o_i = \mathbb{I}(x_i > n), i \in [L]$

$c_i + u_i + o_i = 1, i \in [L]$

$d_1 = 1; \quad d_i = d_{i-1} - 2o_i + 1, 2 \leq i \leq L$

$op1_i^{idx} = \max_{j \leq i-2} \{j \mid d_j = d_i\}, 3 \leq i \leq L$

$\mathbb{I}(op1_i^{idx} = j) \Rightarrow \mathbb{I}(op1_i^x = x_j), \mathbb{I}(op1_i^t = t_j),$
$\mathbb{I}(op1_i^o = o_j), \mathbb{I}(op1_i^u = u_j), i, j \in [L]$

$\mathbb{I}(x_i = j) \Rightarrow \mathbb{I}(t_i = type_j), i \in [L], j \in [q]$

$o_1 = 0; \quad d_L = 1$, Postfix validity (H)

$x_L = m; \quad x_i < m, 1 \leq i < L$, Equation tree structure (H)

$\mathbb{I}(x_i = x_j) \leq o_i, 1 \leq i < j < L$, Single use of constants (H)

$c_i \Rightarrow \mathbb{I}(x_i < x_j), 1 \leq i < j < L$, Perserve text ordering (W)

$\sum_{i \in L} u_i = 1$, Single unknown (H)

Type consistency (W) :

$\mathbb{I}(x_i \in \{s_+, s_-\}) \Rightarrow \mathbb{I}(t_i = t_{i-1} = op1_i^t), i \in [L]$

$\mathbb{I}(x_i = s_*) \Rightarrow \mathbb{I}(t_i \in \{t_{i-1}, op1_i^t\}), i \in [L]$

$\mathbb{I}(x_i = s_/) \Rightarrow \mathbb{I}(t_i = op1_i^t), i \in [L]$

$\mathbb{I}(x_i \in \{s_*, s_/\}) \Rightarrow \mathbb{I}(t_{i-1} \neq op1t_i), i \in [L]$

Non-redundancy (H), Symmetry breaking (H) :

$\mathbb{I}(x_i \in \{s_+, s_-\}) \Rightarrow \mathbb{I}(x_{i-1} \neq s_0, op1_i^x \neq s_0), i \in [L]$

$\mathbb{I}(x_i = s_/ \Rightarrow \mathbb{I}(x_{i-1} \notin \{s_0, s_1\}), i \in [L]$

$\mathbb{I}(x_i \in \{s_+, s_-\}, c_{i-1} = c_{i-2} = 1) \Rightarrow$
$\mathbb{I}(x_{i-2} < x_{i-1}), 3 \leq i \leq L$

Simplicity (H), Equality/Unknown first or last (W) :

$d_i \leq maxStackDepth, i \in [L]$

$op1_L^o + o_{L-1}, \leq 1$

$u_{L-1} + \mathbb{I}(u_1 = 1, \forall i \in \{2, \ldots, L-1\} : d_i \geq 2) \geq 1$

Equality next to unknown (W) :

$\mathbb{I}(x_i = s_=) \leq u_{i-1} + op1_i^u, 3 \leq i \leq L$

Figure 5: ILP model for generating candidate equations