# Personalized Mathematical Word Problem Generation

**Oleksandr Polozov**
University of Washington
polozov@cs.washington.edu

**Eleanor O'Rourke**
University of Washington
eorourke@cs.washington.edu

**Adam M. Smith**
University of Washington
amsmith@cs.washington.edu

**Luke Zettlemoyer**
University of Washington
lsz@cs.washington.edu

**Sumit Gulwani**
Microsoft Research Redmond
sumitg@microsoft.com

**Zoran Popović**
University of Washington
zoran@cs.washington.edu

## Abstract

Word problems are an established technique for teaching mathematical modeling skills in K-12 education. However, many students find word problems unconnected to their lives, artificial, and uninteresting. Most students find them much more difficult than the corresponding symbolic representations. To account for this phenomenon, an ideal pedagogy might involve an individually crafted progression of unique word problems that form a personalized plot.

We propose a novel technique for automatic generation of personalized word problems. In our system, word problems are generated from general specifications using answer-set programming (ASP). The specifications include tutor requirements (properties of a mathematical model), and student requirements (personalization, characters, setting). Our system takes a logical encoding of the specification, synthesizes a word problem narrative and its mathematical model as a labeled logical plot graph, and realizes the problem in natural language. Human judges found our problems as solvable as the textbook problems, with a slightly more artificial language.

## 1 Introduction

Word problems are notoriously difficult for children and adults alike [Verschaffel, 1994]. This phenomenon is not always related to mathematical understanding; in fact, many people find word problems much more difficult than the corresponding symbolic representations. Children have been reported to perform up to 30% worse on word problems than on corresponding algebraic equations [Carpenter *et al.*, 1980]. Multiple studies have conjectured that this is caused by language understanding, conceptual knowledge, discourse comprehension, and other aspects required to build a mental representation of a word problem [Cummins *et al.*, 1988; Schumacher and Fuchs, 2012].

Moreover, many students find word problems artificial and irrelevant to their lives [Ensign, 1996]. This perception is known to be altered by introducing individual interest in a context of a word problem [Renninger *et al.*, 2002]. Many researchers have found that personalizing word problems raises understanding and engagement in a problem solving process, which, in turn, increases children's performance [Davis-Dorsey *et al.*, 1991; Hart, 1996]. However, personalizing a progression of word problems is impractical in a textbook, and would place unreasonable burden on teachers, who would need to maintain awareness of each student's interests.

According to observations above, an ideal pedagogy might involve an individually crafted progression of unique word problems that form a personalized plot. Educational scaffolding of such a progression should be able to vary multiple aspects of a word problem individually. Such aspects should include but are not limited to: concepts of a mathematical model, plot complexity, discourse structure, and language richness. Moreover, problem personalization should rely on the students' own preferences, and ideally should generate word problems automatically according to their requirements.

In this work, we present a system for automatic personalized word problem generation from general specifications. In our system, word problem generation is defined as a constrained synthesis of labeled logical graphs that represent abstract plots. The constraints are given by a tutor and a student independently as a set of mathematical and narrative requirements. Our system has the following properties:

- It is *automatic*: a mathematical model, a plot, and a discourse of a word problem are generated automatically from general specifications.
- It is *personalized*: students can set preferences for a problem's setting, characters, and their relationships.
- It is *sensible*: we enforce coherence in a synthesized plot using a novel technique called *discourse tropes*.
- It is *fit for scaffolding*: varying requirements to different layers of a word problem enables a tutor to scaffold a unique educational progression.

Synthesis of logical graphs is implemented with *answer-set programming* (ASP), a logic programming paradigm, well-suited for exploration of a huge space of possible models under declarative constraints [Gebser *et al.*, 2012]. The technical novelty of this approach lies in (a) application of ASP to a novel domain, and (b) using a relatively underexplored *saturation technique* [Eiter *et al.*, 2009] to solve the universally quantified problem of graph generation with discourse tropes.

The system also includes a natural language generation (NLG) module for generating a textual representation of a synthesized logical graph. It is built according to the classic guidelines of NLG systems [Reiter and Dale, 1997]. To reduce the impact of poor language clarity on problem complexity, the system includes an algorithm for unambiguous and non-repetitive *reference resolution* in problem text.

This paper makes the following contributions:

- We define the problem of *personalized word problem generation* from tutor and student requirements (§3.1).
- We formulate plot generation of a word problem as constrained synthesis of labeled graphs in ASP (§4).
- We build a system of *discourse tropes*, designed to enforce semantic coherence in a synthesized plot. We formulate the problem of graph synthesis with discourse tropes as an $\mathsf{NP}^{\mathsf{NP}}$ problem, and present its encoding in ASP using the *saturation technique* (§4.4).
- We present a method for translating abstract problem plots into unambiguous textual representations (§5).
- We report an evaluation of generated problems by comparing human judgements with textbook problems (§6). Our problems have slightly more artificial language, but they are generally comprehensible, and as solvable as the textbook problems.

## 2 Related Work

**Problem generation**

Automatic problem generation for various educational domains has been studied since the mid-1960s [Wexler, 1968]. Recently it has gained new interest with novel approaches in problem generation for natural deduction [Ahmed *et al.*, 2013], algebraic proof problems [Singh *et al.*, 2012], procedural problems [Andersen *et al.*, 2013], embedded systems [Sadigh *et al.*, 2012], etc. All of them apply a similar technique: they first generalize an existing problem into a template, and then explore a space of solutions that fit this template. However, the specific approaches vary. Ahmed et al. build templates automatically, Andersen et al. and Singh et al. do it semi-automatically, and Sadigh et al. write templates manually. Our system builds templates automatically w.r.t. the set of requirements, provided by tutors and students.

The underlying search space for possible problem models is usually explored with logical reasoning and exhaustive search, as opposed to combinatorial model space exploration in our approach. Andersen et al. [2013] use a state-of-the-art code coverage toolkit that is built on a SMT-based constraint solver. In this way, their approach is similar to our usage of ASP solvers for space exploration. However, the specific application of solvers to a new subject domain often requires significant technical insight, such as the usage of disjunctive ASP for enforcing narrative constraints for our domain.

In the word problem domain, the most common automation approach is a database of worksheets with templates, prepared in advance. This approach has proved itself valuable for assessment, but its personalization level is insufficient for engaging education. Deane and Sheehan [2003] built one of the rare projects that performs automatic word problem generation. They focus on natural language generation using Frame Semantics [Fillmore, 1976] with the standard NLG architecture introduced by Reiter and Dale [1997], and explore distance/speed word problems as an example domain. Our work builds upon the same architecture for NLG, but it also includes automatic generation of the word problem logic.

**Declarative content generation using ASP**

A number of systems have explored procedural content generation (PCG) via declarative specification of desired properties and constraints. Tutenel *et al.* [2009] describe a rule-based interior generation system for building design systems. Brain and Schanda [2009] present a declarative *Warzone 2100* map generator. Smith *et al.* [2012] generate levels for an educational puzzle game. Many of such projects use ASP as a paradigm for declarative specification of content constraints, and rely on state-of-the-art ASP solvers for combinatorial model space exploration [Smith and Mateas, 2011].

Our system also uses ASP for automatic generation of a word problem. However, it also pushes the approach further, using a saturation technique for automatic verification of a universally quantified property. Smith *et al.* [2013] productively applied this technique in puzzle generation where they used universally quantified constraints to require key concepts across all possible puzzle solutions.

## 3 Overview

This section defines the problem of *personalized mathematical word problem generation* and gives a general overview of our solution at a high level. We describe our implementation in details in §4 and §5. Throughout the paper, we use the following simple addition problem as our running example:

**Example 1.** *Knight Alice has 30 chalices. Dragon Elliot has 9 chalices. Alice slays the dragon, and takes his chalices. How many chalices does she have now?*

### 3.1 Problem Definition

The word problem generation procedure takes as input a set of *requirements* $R$ to the problem and produces as output a *textual representation* of the problem. Out of many requirements that a progression designer would want to include in the set $R$, we study two requirement classes that define pedagogical and narrative constraints on the generated problem: *tutor requirements* $R_T$ and *student requirements* $R_S$.

*Tutor requirements* provide pedagogical constraints on the problem's mathematical model. They vary from general constraints ("Should include multiplication") to equation patterns like "$x = ? + 6 \times ?$", where "?" stands for any subexpression. One might imagine more complex requirements like constraining any solution to include certain steps [Smith *et al.*, 2013].

*Student requirements* provide personalized narrative constraints on the problem plot according to the student's literary preferences. Student requirements can be of three kinds:

- *Setting requirements* – define the literary setting of the problem plot, e.g. "Science Fiction," "Fantasy."
- *Character requirements* – define characters to be included in the plot, along with their names and genders.

- *Relationship requirements* – define narrative relationships between the characters, e.g. "Amanda and Billy are friends." The word problem plot should explicitly showcase these relationships, and should not include any plot elements that logically contradict these relationships.

The output of the system is a textual representation of the word problem. Its plot and mathematical model should satisfy all the requirements in $R$.

## 3.2 Architecture

Figure 1 shows the architecture of our solution to the word problem generation problem. The process consists of two phases: *logic generation* and *natural language generation*.

**Logic generation**

The first phase constructs a logical representation of a word problem plot, given the requirements $R$. It builds the mathematical and the narrative layers of a word problem. The result is a *logical graph* of actors, actions, and entities.

The system starts with *equation generation* (§4.2). It builds a mathematical model of the problem (an abstract equation $E$), taking into account tutor requirements $R_T$.

Given a specific equation $E$, the system generates a problem plot (§4.3, §4.4). This involves finding setting-dependent interpretations for every variable in the equation $E$, according to the setting in the student requirements $R_S$. Plot actors and actions, generated from the ontology according to the rest of the requirements $R_S$, complete the logical graph of the word problem. At this step the generator also gives concrete values to any constants present in the equation $E$ according to the constraints of their setting-dependent logical types.

**Natural language generation**

The NLG phase takes a generated logical graph, and realizes it into a concrete textual representation (§5). First, it makes use of primitive sentence templates that form a placeholder for the word problem text. The language produced by such templates is unnatural: it is repetitive, every entity is referenced identically every time it appears, and every sentence describes exactly one action. Thus, the rest of the NLG phase is a post-processing of the template-generated text to make the generated language more natural, before passing it to the *surface realizer* to produce technically valid English.

At the sentence ordering step, the sentences produced by templates are ordered into a linear narrative according to temporal and causal relationships between the sentences.

At the reference resolution step, every entity reference in the text is realized into its textual representation. Elements of representations include pronouns, articles, type determiners (*knight*, *dragon*), etc. We present a reference resolution algorithm that finds the shortest unambiguous non-repetitive representation for every reference, given its discourse context.

# 4 Logic Generation

We use ASP to implement the logic generation phase of the process. ASP is a logic programming paradigm where programs declaratively describe a set of models, called *answer sets*. In this section, we describe our usage of ASP for word problem generation. We start with a general description of ASP, and then proceed to explanation of logic generation.

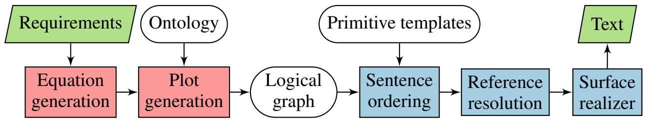


Figure 1: Word problem generation. Red blocks represent logic generation steps, and blue blocks represent NLG steps.

## 4.1 Answer-set programming

In ASP [Gebser *et al.*, 2012], programs are composed of facts and rules in a first-order logic representation (syntactically similar to Prolog). After propositionalization of this program (called *grounding*), answer set solvers search the space of truth assignments associated with each logical statement. Each satisfying solution, called an *answer set*, is a set of self-consistent statements identifying a different possible world.

Conventionally, a program in ASP consists of three parts that correspond to three different types of rules. The "Generate" part consists of *choice rules*. They allow the solver to *guess* facts that might be true. The "Define" part consists of *deductive rules*. They allow the solver to *deduce* new facts from the established ones. The "Test" part consists of *integrity constraints*. They *forbid* solutions that have certain properties. These three parts together define a general formulation of a problem, called a general *problem encoding*. It is later combined with a specific *problem instance* (a set of *facts* that describe input data), and passed to the solver.

## 4.2 Equation generation

In this section, we describe the equation generation phase in ASP. It takes tutor requirements $R_T$ as a problem instance, and generates an equation $E$ that satisfies constraints in $R_T$.

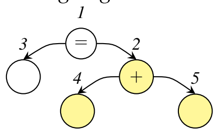
**Example 2.** *The set of requirements $R$ for our running example in ASP syntax. Line 2 shows a tutor requirement.*

```
1 require_setting(fantasy).
2 require_math(plus(any, any)).  % "? + ?" subexpression
3 require_character(cAlice, ("Alice", female)).
4 require_character(cElliot, ("Elliot", male)).
5 require_relationship(adversary, cAlice, cElliot).
```

The ASP formulation of equation generation follows the conventional approach, described in §4.1. The *key idea* is to encode an equation $E$ as an expression tree. The encoding first *guesses* a tree shape (of given maximal size) and an assignment of mathematical operators to internal binary nodes. Then it *deduces* whether every mathematical requirement in $R_T$ is *covered* by some subexpression of the guessed tree. Finally, it *forbids* the solutions that do not cover all of the requirements in $R_T$ or do not represent a valid equation.

**Example 3.** *One of many possible answer sets that represent our running example is shown below, along with its graphical representation as an expression tree. The tree has five nodes, labeled 1 through 5. The subtree that covers a tutor requirement* plus(any, any) *from Example 2 is highlighted.*

```
node_op(1, eq). node_op(2, plus).
node_arg(1,1,3). node_arg(1,2,2).
node_arg(2,1,4). node_arg(2,2,5).
```



## 4.3 Plot generation

Given an equation $E$, the following phase of logic generation synthesizes a word problem plot that models $E$. A plot is described with an *ontology* that consists of *types*, *relations*, and *discourse tropes*. A *logical graph* of a word problem instance consists of *entities* that are annotated with ontology types, *facts* that are annotated with ontology relations, and *connectives* that are annotated with discourse tropes. We first give a formal definition of entities and facts in a logical graph, and describe their meaning. We defer the discussion of discourse tropes and connectives to §4.4.

**Definition 1.** A *logical graph* $\mathcal{G}$ is a tuple $\langle \mathcal{E}, \mathcal{F}, \mathcal{C} \rangle$ where:

• $\mathcal{E}$ is a set of *entities*. Every entity $e \in \mathcal{E}$ has a corresponding *ontology type* $\tau$, denoted $e \colon \tau$. Types form a hierarchy tree: every type $\tau$ except for the root has a *parent type* $\mathsf{par}(\tau)$. We write $\tau \preccurlyeq \tau'$ to denote that $\tau$ is a subtype of $\tau'$.

• $\mathcal{F}$ is a set of *facts*. Every fact $f \in \mathcal{F}$ has a corresponding *ontology relation* $\mathcal{R} = \mathsf{relation}(f)$. Every relation $\mathcal{R}$ has a set of named *arguments* $\mathsf{args}(\mathcal{R})$. For a fact $f \in \mathcal{F}$ in a logical graph $\mathcal{G}$, every argument $a \in \mathsf{args}(\mathsf{relation}(f))$ is associated with an entity $e \in \mathcal{E}$. We denote this as $f = \mathcal{R}(a_1 = e_1, \ldots, a_n = e_n)$ or simply $f = \mathcal{R}(e_1, \ldots, e_n)$.

Facts are required to type-check w.r.t. their associated relations. Each relation argument $(a_j \colon \tau) \in \mathsf{args}(\mathcal{R})$ is annotated with its required ancestor type $\tau$. The corresponding entities $e_j \colon \tau'$ in the fact instances of this relation should have $\tau' \preccurlyeq \tau$.

• $\mathcal{C}$ is a set of *temporal* ($\mathsf{T}$) and *causal* ($\mathsf{C}$) *connectives*. A connective $c \in \mathcal{C}$ is a tuple $\langle t, f_1, f_2 \rangle$, where tag $t \in \{\mathsf{T}, \mathsf{C}\}$.

**Example 4.** *In our example, $\mathcal{G}$ consists of the following sets:*

$\mathcal{E} = \{k_1, d_1, c_k, c_d, c_u\}$ *where:*

$$k_1 \colon \mathsf{TKnight} \quad d_1 \colon \mathsf{TDragon} \quad c_k, c_d, c_u \colon \mathsf{TChalice}$$

$\mathcal{F} = \{owns_k, owns_d, owns_u, slays, acq, total, unk\}$ *where:*

$$owns_k = \mathsf{Owns}(\mathsf{owner} = k_1, \mathsf{item} = c_k)$$
$$owns_d = \mathsf{Owns}(\mathsf{owner} = d_1, \mathsf{item} = c_d)$$
$$slays = \mathsf{Slays}(\mathsf{slayer} = k_1, \mathsf{victim} = d_1)$$
$$acq = \mathsf{Acquires}(\mathsf{receiver} = k_1, \mathsf{item} = c_d)$$
$$total = \mathsf{TotalCount}(\mathsf{total} = c_u, \mathsf{count1} = c_k, \mathsf{count2} = c_d)$$
$$owns_u = \mathsf{Owns}(\mathsf{owner} = k_1, \mathsf{item} = c_u)$$
$$unk = \mathsf{Unknown}(\mathsf{unknown} = c_u)$$

$$\mathcal{C} = \{\langle \mathsf{T}, owns_k, slays\rangle, \langle \mathsf{T}, owns_d, slays\rangle, \langle \mathsf{C}, slays, acq\rangle\}$$

Some of the relations $\mathcal{R}$ in the ontology are annotated with their corresponding mathematical operations. We say that a logical graph $\mathcal{G}$ *models* an equation $E$, if its subgraph generated by such operations is isomorphic to the expression tree of $E$. For example, the relation TotalCount represents an operation $\mathsf{total} = \mathsf{count1} + \mathsf{count2}$ over its arguments. This operation is isomorphic to $E$, hence $\mathcal{G}$ models $E$.

We describe the ontology with ASP facts like:

```
% Type TWarrior with par(TWarrior) = TPerson belongs to a fantasy setting.
type(setting(fantasy), t_warrior, t_person).
% Slays(slayer: TWarrior, victim: TMonster) belongs to a fantasy setting.
relation(setting(fantasy), p_slays(t_warrior, t_monster)).
% Arguments slayer and victim in Slays relation can only be adversaries in R_S.
only_relationship(p_slays, adversary(1, 2)).
% TotalCount(total: TCountable, count1: TCountable, count2: TCountable)
relation(setting(common),
    p_total_count(t_countable, t_countable, t_countable)).
math_skeleton(p_total_count, eq(1, plus(2, 3))).
```

We say that $\mathcal{G}$ *fits* a set of student requirements $R_S$, if (a) every type and relation in $\mathcal{G}$ are either setting-neutral, or belong to a setting from $R_S$, (b) every character in $R_S$ corresponds to a distinct entity $e \in \mathcal{E}$, and (c) facts $f \in \mathcal{F}$ that include character entities as arguments do not violate relationship requirements from $R_S$ semantically.

The goal of plot generation phase is: given an equation $E$, student requirements $R_S$, and an ontology, generate any logical graph $\mathcal{G}$ from types, relations, and tropes in the ontology, such that it models the equation $E$, and fits the requirements $R_S$. In ASP, we solve this problem using a generalization of equation generation technique from §4.2, sketched below. We first *guess* the graph topology, then *deduce* whether it type-checks w.r.t. the ontology, and whether it models the equation $E$, and then *forbid* graphs that do not fit the requirements $R_S$.

```
% Guess a single type for each entity.
1 { entity_type(E, T): concrete_type(T) } 1 :- entity(E).
instanceof(E, T1) :- entity_type(E, T), subtype(T, T1).
% Guess a relation and an assignment of typed arguments for each fact.
1 { fact_relation(F, R): relation(R) } 1 :- fact(F).
1 { fact_argument(F, K, E): instanceof(E,T) } 1 :-
    fact_relation(F, R), K = 1..@arity(R), argument_type(R, K, T).

% Deduce whether a logical graph models an equation.
models(Eq, F) :- fact_relation(F, R), math_skeleton(R, S),
                 shape_matches(Eq, F, S).
shape_matches(Eq, F, S) :- . . .  % Deduce inductively from arguments.
% Forbid solutions that do not model the required equation.
:- equation(Eq), #count { F: matches(Eq, F) } == 0.
```

The main part of the deduction is the inductive verification that the graph topology, guessed in the first section of our ASP program, models the equation $E$. For that, the encoding recursively follows the argument edges in $E$ and in the mathematical operations of ontology relations in parallel. This process is denoted by the predicate `shape_matches(Eq, F, S)` above. Here `Eq` denotes the equation $E$, `F` denotes the current fact $f$ under verification, and `S` denotes the *skeleton* of the mathematical operation against which $f$ is being matched. Our encoding implements `shape_matches` as follows:

1. For each numbered placeholder $k$ in the skeleton `S`, recursively follow the paths from the root of `S` and the root of $E$ in parallel, arriving at the *equation argument* node $n_k \in E$. Fail if the paths in `S` and $E$ are not isomorphic.
2. Map each equation argument $n_k$ to the $k^{\text{th}}$ plot argument $e_k$ of the fact $f = \mathcal{R}(e_1, \ldots, e_n)$ under verification.
3. Fail if the deduced mapping between $E$ and $\mathcal{E}$ is non-injective or is not a function, succeed otherwise.

**Example 5.** *Matching the skeleton* `eq(1, plus(2, 3))` *of the relation* $\mathsf{TotalCount}$ *against the nodes of equation $E$ from Example 3, we obtain the following corresponding equation nodes: $n_1 = 3, n_2 = 4, n_3 = 5$. They are respectively mapped to the arguments of the fact "total": $n_1 \to c_u, n_2 \to c_k, n_3 \to c_d$. This mapping is functional and injective, hence the logical graph $\mathcal{G}$ from Example 4 models the equation $E$.*

Multiple integrity constraints forbid insensible logical graphs (not shown in the snippet above). They require the graph to be connected, to include an entity for every character in $R_S$, to comply with character relationships, to have a single "unknown" value (that the students recover by solving the problem), to use types and relations only from the required setting or from common life, etc.

## 4.4 Discourse tropes

Relying on the ontology type system is sufficient for generating plausible logical situations, but insufficient for generating engaging plots. Word problems are primarily short stories, and every short story needs a developed narrative. This aspect is even more important for our setting of *personalized* word problems, where the story is centered around student-defined characters and their relationships.

An ASP solver, when presented with an encoding above, generates *any* logical graph $\mathcal{G}$ that fits the requirements, models the equation, and is logically sound w.r.t. the ontology type system. However, requiring $\mathcal{G}$ to represent a coherent narrative is a more challenging problem. Consider the logical graph in Example 4. One might ask, why did the solver not choose any subset of $\mathcal{F}$ instead? Or why did the solver not add a fact $slays' = \mathsf{Slays}(\mathsf{slayer} = d_1, \mathsf{victim} = k_1)$? In both of these situations, the resulting graph would still be logically sound, but it would either lack plot-driving details, or include unnecessary statements. To forbid such answer sets, we describe such narrative constraints in *discourse tropes*.

**Definition 2.** A *discourse trope* $\mathcal{D}$ is a constraint on a logical graph $\mathcal{G}$ of form $\forall \vec{x} \subset \mathcal{E}\, [\Phi(\vec{x}) \Rightarrow \exists\, \vec{y} \subset \mathcal{E} \colon \Psi(\vec{x}, \vec{y})]$. Here $\Phi, \Psi$ are quantifier-free formulas that make use of ontology relations as logical predicates. $\vec{x}, \vec{y}$ are groups of distinct entities in $\mathcal{G}$ that participate in the constraints $\Phi, \Psi$.

A subgraph of $\mathcal{G}$ generated by facts corresponding to predicates in $\Phi, \Psi$ and their arguments is called *an instance* of $\mathcal{D}$ if both $\mathcal{D}$ and $\Phi(\vec{x})$ hold on this subgraph.

**Example 6.** *A fantasy trope "A warrior slays a monster only if the monster has some treasures" is described logically as:*

$$\mathcal{D} = \forall\, w, m \in \mathcal{E}\, [\mathsf{Slays}(w, m) \Rightarrow \exists\, t \in \mathcal{E} \colon \mathsf{Owns}(m, t)]$$

*In Example 4 a subgraph* $\{slays, owns_d\}$ *is an instance of* $\mathcal{D}$. *The snippet below shows* $\mathcal{D}$ *in our ontology syntax:*

```
discourse( forall(m, w), premise(p_slays(w, m)),
           exists(t), conclusion(p_owns(m, t)),
           temporal(p_owns(m, t), p_slays(w, m)) ).
```

The syntax describes arbitrary first-order logical formulas of the form above with predicates `forall`, `exists`, and, etc. Distinctly named variables are assumed to match distinct entities. Expressions `premise` and `conclusion` represent constraints $\Phi$ and $\Psi$, respectively. The last argument of `discourse` predicate instructs a solver to generate connectives $\mathcal{C}$ between some pairs of facts when they belong to an instance of this trope in $\mathcal{G}$. These connectives are later used by the NLG phase to order problem sentences (§5).

Our plot generator takes the ontology discourse tropes into account by requiring every fact in the generated logical graph $\mathcal{G}$ to *either be mathematical or belong to an instance of some discourse trope*. Thus, any solution that does not represent a coherent literary narrative will be forbidden by the solver.

### Saturation technique

Verifying a discourse trope on a logical graph is non-trivial. ASP is well-suited for solving search problems with existentially quantified goal conditions, which lie in the complexity class NP. However, the problem of plot generation with discourse tropes has a $\exists \forall \exists$ structure. Here the first quantifier represents a search for a graph $\mathcal{G}$ that satisfies both the problem encoding and the set of tropes in the ontology. The last two quantifiers represent validation of $\mathcal{G}$ w.r.t. $\mathcal{D}$ according to Definition 2. A naïve encoding of such a problem in ASP would cause an exponential blowup in grounding time.

In our setting, the innermost $\exists$ quantifier is expanded during grounding using *skolemization* [Benedetti, 2005]. It causes only a polynomial blowup in grounding time, because every trope in our ontology has at most 3 existentially quantified variables $\vec{y}$. After that, we are left with a 2QBF problem, which lies in the complexity class $\mathsf{NP}^{\mathsf{NP}}$. In order to express a 2QBF problem in ASP without an exponential blowup, we leverage an extension of classic ASP with a special fourth rule type in a relatively underexplored *saturation technique*.

*Disjunctive rules*, not supported by many ASP solvers, include disjunction in rule heads, allowing the solver to deduce any of the facts in the head, given the body. The solution is then chosen according to *subset minimality semantics*: no answer set is emitted as a solution if any of its subsets is also a valid solution [Gebser *et al.*, 2013].

Consider for simplicity the problem of searching for a graph $\mathcal{G}$, constrained with a single trope $\forall \vec{x}\, \Phi(\vec{x})$. In the snippet below, we briefly demonstrate the saturation technique for this problem (see [Eiter *et al.*, 2009] for a detailed overview):

```
1 % Example discourse trope: ∀a, b ∈ 𝓔: Owns(a, b) ∧ Acquires(a, b)
2 discourse(forall(a,b), premise(owns(a,b), acquires(a,b))).
3 % Assign each formal variable V ∈ {"a", "b"} to some entity e ∈ 𝓔.
4 bind(V, E): entity(E) :- var(V).
5 sat(Xs, F) :- . . .   % Deduced if Φ(x⃗) holds under the current assignment x⃗.
6 valid :- discourse(Xs, F), sat(Xs, F).
7 bind(V, E) :- valid, var(V), entity(E).
8 :- not valid.
```

We want to enforce the property $\forall \vec{x}\, \Phi(\vec{x})$. For that, we nondeterministically choose an assignment of formal variables $\vec{x}$ to a subset of entities, using a disjunctive rule in line 4. If the assignment is a valid counterexample to $\Phi(\vec{x})$, `valid` will not be deduced, and the integrity constraint in line 8 will eliminate this answer set. On the other hand, if the assignment is not a counterexample, line 7 will *saturate* the answer set, including every possible `bind(V, E)` fact into it. According to subset minimality semantics, a valid answer set is emitted *only if all of its subsets are not valid.* However, for a saturated answer set, *every other answer set* is its subset. Thus, the solver will emit a solution only after a thorough check of all other variable assignments, which is equivalent to enforcing a $\forall \vec{x}\, \Phi(\vec{x})$ property.

## 5 Natural Language Generation

After generating a logical graph of a problem, our system realizes it into a textual representation. First, it approximates the text with sentences produced using *primitive templates*, and then it *orders* sentences produced by templates, *resolves* entity references unambiguously and non-repetitively, and *realizes* the result into valid English.

### Primitive Templates

Figure 2 shows an example of a primitive template. It matches a subset of facts in $\mathcal{F}$, and produces a syntactic tree of a sentence, where some NP (noun phrase) subtrees are replaced

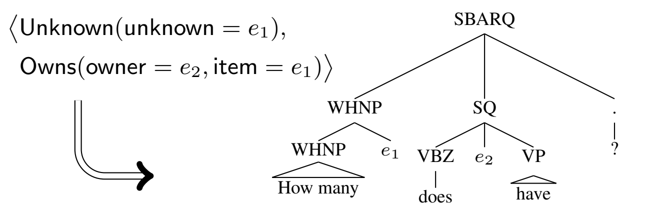

⟨Unknown(unknown = $e_1$),
Owns(owner = $e_2$, item = $e_1$)⟩
SBARQ
WHNP
SQ
.
WHNP
$e_1$
VBZ
$e_2$
VP
?
How many
does
have

Figure 2: A primitive template that matches 2 facts from $\mathcal{F}$.

with entities they describe. As a result, we obtain a first approximation to the word problem text – an unordered bag of sentences $\mathcal{S}$ with somewhat unnatural, repetitive language.

**Sentence ordering**

The *key idea* in turning the sentences $\mathcal{S}$ into a linear narrative is leveraging connectives $\mathcal{C}$ that were added to $\mathcal{G}$ from discourse tropes during plot generation. Each connective $c \in \mathcal{C}$ specifies a temporal or causal relationship between two facts $f_1, f_2 \in \mathcal{F}$. Consequently, corresponding sentences $s_1, s_2 \in \mathcal{S}$ that describe $f_1, f_2$ should follow the same ordering in a narrative. The set $\mathcal{C}$ thus defines a *partial ordering* on $\mathcal{S}$. Any total ordering of sentences in $\mathcal{S}$ that satisfies this partial ordering constitutes a valid narrative.

**Example 7.** *After this phase, the template-generated problem text of our running example turns into the following text. The subtrees are marked with corresponding entities from $\mathcal{E}$.*

*"[Knight Alice]$_{k_1}$ has [30 chalices]$_{c_k}$. [Dragon Elliot]$_{d_1}$ has [9 chalices]$_{c_d}$. [Knight Alice]$_{k_1}$ slays [Dragon Elliot]$_{d_1}$, and takes [9 chalices]$_{c_d}$. How many [chalices]$_{c_u}$ does [knight Alice]$_{k_1}$ have?"*

**Reference resolution**

At the reference resolution step, we convert every entity-marked text node into some textual representation for this entity. Our method is inspired by classic NLG guidelines [Krahmer and Van Deemter, 2012], and is based on *type attributes*.

Every type $\tau$ has a set of named attributes. We divide them disjointly into *implicit* and *explicit*, denoted $\mathsf{imp}(\tau)$ and $\mathsf{exp}(\tau)$. Every textual reference to $e\colon \tau$ implicitly or explicitly mentions all attributes in $\mathsf{imp}(\tau)$ to the reader. For example, gender is an implicit attribute for sentient entities: once $e$ is introduced, every reference to it preserves the memory of its gender to the reader. Names, types, and quantities can be mentioned only explicitly, and hence belong to $\mathsf{exp}(\tau)$.

Given a reference to an entity $e\colon \tau$, a subset of its attributes $A \subset \mathsf{exp}(\tau)$, and its discourse context, we can generate a textual representation of $e$ that mentions exactly the attributes in $A \cup \mathsf{imp}(\tau)$ to the reader. The representation also depends on some contextual properties, e.g. whether $e$ is a subject or an object in the sentence, or whether $e$ is a definite reference (i.e. has it been introduced before). For example, $A = \emptyset$ represents a pronoun only if the reference is definite.

The *key idea* of our reference resolution algorithm is to choose the smallest set of attributes for every reference that makes it unambiguous w.r.t. the preceding discourse. For every reference to $e\colon \tau$ after its introduction, the algorithm enumerates the subsets of $\mathsf{exp}(\tau)$, and chooses the smallest one that *does not share at least one attribute value with other entities in the preceding discourse*. To avoid repetition, we often choose a second or third unambiguous representation instead.

**Example 8.** *Consider a reference to $c_d$ on the third line of Example 7. The algorithm first checks $A = \emptyset$, which corresponds to the pronoun representation "them." However, it is ambiguous because* $\mathsf{imp}(c_d) = \{\mathsf{plural}\}$*, and the value of this attribute is equal to the value of the same attribute of a previously processed reference to $c_k$. In several iterations, the algorithm finds an unambiguous subset* $A = \{\mathsf{owner}, \mathsf{type}\}$*, corresponding to the representation "$[d_1]$'s chalices."*

## 6 Evaluation

In this work, we chose not to evaluate the *personalization* aspect of our system, since a complete study of its pedagogical value on the students is beyond the scope of this paper. Instead, we evaluate our *generation techniques* by assessing the *content* of the produced word problems w.r.t. its language comprehensibility and its mathematical applicability.

**User Study**

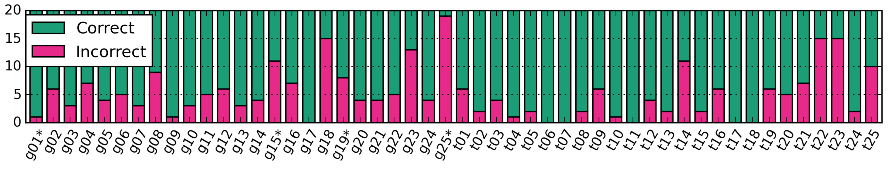
We prepared an ontology of 100-200 types, relations, and tropes in three literary settings: "Fantasy," "Science Fiction," "School of Wizardry." This one-time initial setup of the system took about 1-2 person-months. From it, we randomly generated 25 problems in the domains of age, counting, and trading, with the solutions requiring 2-4 primitive arithmetic operations. We sampled the problems with sufficient linguistic variability to evaluate the overall text quality. Although the ASP solving has exponential complexity, every problem was generated in less than 60 s, which is a feasible time limit for realistic problems within our range of interests.

We selected 25 textbook problems from the Singapore Math curriculum [Publications, 2009] with the equivalent distribution of complexity (solution lengths), and conducted two studies using Mechanical Turk. Study A assessed language aspects of the problems. It asked the subjects 4 questions (shown in Figure 3) on a forced-choice Likert scale. Study B assessed mathematical applicability of the problems. It asked the subjects to solve a given problem, and measured solving time and correctness. For both studies, each problem was presented to 20 different native English speakers (1000 total).

Our experimental hypotheses were: **(a)** the generated problems should be rated equally or slightly less comprehensible than the textbook problems (by the nature of our problems being artificial), **(b)** the language of our problems should remain generally comprehensible with possible slight oddities (with a mean $> 3$ on the forced-choice Likert scale converted to 1-4), and **(c)** problems should remain solvable with no statistical difference from the textbook problems.

Figure 3 presents the results of the studies A and B on the aggregated dataset of 50 problems. The hypotheses (a) and (b) were confirmed: the judges rated the textbook problems significantly better than our problems across all 4 questions with moderate effect sizes (for the most general Q1 we had $\chi^2 = 193.52, p < 0.001, V = 0.44$), but our problems were generally comprehensible with occasional oddities (mean rating ranged from 3.36 to 3.50). The original results invalidated hypothesis (c): participants correctly solved textbook problems significantly more often than the generated problems ($\chi^2 = 9.474, p < 0.005, V = 0.01$). However, since

Q1: How comprehensible is the problem? How well did you understand the plot?
Q2: How logical/natural is the sentence order?
Q3: When the problem refers to an actor (e.g. with a pronoun, a name), is it clear who is being mentioned?
Q4: Do the numbers in the problem fit its story (e.g. it wouldn't make sense for a knight to be 5 years old)?



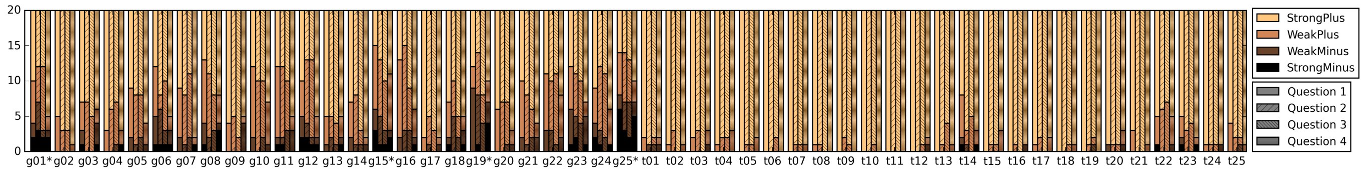


Figure 3: *Top-left:* the forced-choice Likert scale questionnaire for the study A. *Bottom and top-right:* evaluation results for the studies A and B, respectively. Generated/textbook problems are prefixed with g/t. The outliers are marked with an asterisk.

| **Start:** $R_T = \{(? \times 3) - ?\}$, $R_S$ as in Example 2 | **Step 1:** $R_T = \{(? \times 3) - ?\}$, $R_S$: "adversaries" → "friends" | **Step 2:** $R_T = \{(? \times 3) + ?\}$, $R_S$ as in Step 1 | **Step 3:** $R_T = \{(? \times 3) + ?\}$, $R_S$: "Fantasy" → "Wizardry" | **Step 4:** $R_T = \{(? \times x) + ?\}$, $R_S$ as in Step 3 |
|---|---|---|---|---|
| **Duchess Alice** leads **3 squads** of **12 mounted knights** each in a brave attack upon **duke Elliot**'s camp. Scouts have reported that there are **17 mounted knights** in **his** camp. How many more **mounted knights** does **Elliot** need? | **Duchess Joan**'s countryside consists of **11 towers**, surrounded by **3 villages** each. **She** and **baron Elliot** are at war. **He** has already occupied **16 villages** with the help of **wizard Alice**. How many **villages** are still unoccupied by **Elliot**? | **Orc Bob** has **11 chests**. Inspired by recent advances in burglary, **dwarf Alice** steals **chests** from **the orc**. **They** have **3 gold bars** each. **She** gets a honorable reward of **15 gold bars** from **the master thief Elliot**. How many **gold bars** does **the dwarf** have? | **Professor Alice** assigns **Elliot** to make a **luck potion**. **He** had to spend **9 hours** first reading the recipe in the textbook. **He** spends **several hours** brewing **11 portions** of **it**. **The potion** has to be brewed for **3 hours** per portion. How many **hours** did **Elliot** spend in total? | **Professor Elliot** assigns **Alice** to make a **sleep potion**. **She** had to spend **5 hours** first reading the recipe in the textbook. **It** has to be brewed for **9 hours** per portion. **She** spends **several hours** brewing **several portions** of **it**. The total time spent was **59 hours**. How many **portions** did **Alice** make? |

Figure 4: A series of independently generated word problems. For demonstration purposes, each step makes a change of a single aspect in the requirements. The last step demonstrates an unknown variable requirement. Entity references are highlighted.

a Cramer's $V$ effect size of $0.01$ is considered very small, we conjectured that the result was driven by a handful of problems with poor nondeterministic choices during the NLG phase (§5). To validate this hypothesis, we excluded *outlier* problems from the analysis (those with a mean answer $< 3$ to any Study A question). As Figure 3 shows, there were only 4 outliers out of 25 generated problems. A new analysis of the resulting filtered dataset found no statistically significant difference in solving performance ($\chi^2 = 2.439$, *n.s.*). The participants solved textbook problems correctly 78% of the time, compared to 73% of the time for generated problems. There was also no statistically significant difference in the solving times for generated (mean = 232 s, SD = $367.2$) and textbook (mean = 220 s, SD = $378.7$) problems ($t(918) = 0.473$, *n.s.*). The mean Q1-4 rating of non-outlier generated problems was 3.45-3.65, as compared to 3.90-3.92 for textbook problems.

Study B also exposed two generated problems with clear language but mostly incorrect solutions (g18 and g23). This happened because of incomplete semantics of the participating ontology relations (e.g. an intermediate result assuming less than 24 hours/day). One way to resolve it is a richer ontology with commonsense background knowledge such as Cyc [Lenat, 1995], which is an important area of future work.

### Examples

Figure 4 demonstrates how various problem requirements correspond to different aspects of word problem complexity. It shows a series of generated word problems, where each step changes one requirement at a time, thereby producing a different problem with respect to a single complexity aspect. The aspects are, in order: character relationships, mathematical model, literary setting, unknown value. We highlighted entity references in each sentence to demonstrate how our reference resolution impacts language clarity. This aspect, together with sentence ordering in the last two steps, demonstrates variation in linguistic complexity of word problems.

## 7 Conclusions and Future Work

In this work, we defined and explored the problem of personalized mathematical word problem generation. We presented a system that focuses on synthesis of single word problems from general specifications, provided by a tutor and a student independently. Our usage of constrained logic programming and our coherence enforcement with universally quantified discourse tropes allows a tutor to define the word problem structure in the form of its abstract concepts. Word problems, as a domain with multiple independent layers of complexity, also entail a personalized approach, where students express personal preferences to the generated content. In that case, the student's engagement often overcomes the impact of semantic and linguistic problem complexity on their performance.

Automatic problem generation under individually tailored conceptual constraints could be a valuable tool for instructional scaffolding in any educational domain. Since problem generation through synthesis of labeled logical graphs is a domain-independent idea, it allows a wide spectrum of applications. For instance, replacing mathematical constraints with logical implications enables generation of language comprehension problems, notoriously confusing in elementary school. Augmenting an ontology with background knowledge enables problem generation in physics or programming. Problem generation in such domains is not only beneficial for education or assessment, but also valuable for data-driven research on the individual aspects of problem complexity. This work takes an important step towards realizing the vision of providing personalized pedagogy to support the needs of teachers and the interests of students simultaneously.

## 8 Acknowledgments

This work was supported by the Office of Naval Research grant N00014-12-C-0158, the Bill and Melinda Gates Foundation grant OPP1031488, the Hewlett Foundation grant 2012-8161, Adobe, and Microsoft. We also thank Roy Szeto and Robert Duisberg for helpful discussions, and the anonymous reviewers for their feedback.

## References

[Ahmed *et al.*, 2013] Umair Z. Ahmed, Sumit Gulwani, and Amey Karkare. Automatically generating problems and solutions for natural deduction. In *IJCAI*, pages 1968–1975. AAAI Press, 2013.

[Andersen *et al.*, 2013] Erik Andersen, Sumit Gulwani, and Zoran Popović. A trace-based framework for analyzing and synthesizing educational progressions. In *CHI*, pages 773–782. ACM, 2013.

[Benedetti, 2005] Marco Benedetti. Evaluating QBFs via symbolic skolemization. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 285–300. Springer, 2005.

[Brain and Schanda, 2009] Martin Brain and Florian Schanda. DIORAMA (*Warzone 2100* map tools). http://warzone2100.org.uk/, 2009.

[Carpenter *et al.*, 1980] Thomas P Carpenter, Mary Kay Corbitt, Henry S Kepner, Mary Montgomery Linquist, and Robert E Reys. Solving verbal problems: Results and implications from national assessment. *Arithmetic Teacher*, 28(1):8–12, 1980.

[Cummins *et al.*, 1988] Denise Dellarosa Cummins, Walter Kintsch, Kurt Reusser, and Rhonda Weimer. The role of understanding in solving word problems. *Cognitive psychology*, 20(4):405–438, 1988.

[Davis-Dorsey *et al.*, 1991] Judy Davis-Dorsey, Steven M Ross, and Gary R Morrison. The role of rewording and context personalization in the solving of mathematical word problems. *Journal of Educational Psychology*, 83(1):61, 1991.

[Deane and Sheehan, 2003] Paul Deane and Kathleen Sheehan. Automatic item generation via frame semantics: Natural language generation of math word problems. In *Annual meeting of the National Council on Measurement in Education, Chicago, IL*, 2003.

[Eiter *et al.*, 2009] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web. Semantic Technologies for Information Systems*, pages 40–110. Springer, 2009.

[Ensign, 1996] Jacque Ensign. *Linking life experiences to classroom math*. PhD thesis, University of Virginia, 1996.

[Fillmore, 1976] Charles J Fillmore. Frame semantics and the nature of language. *Annals of the New York Academy of Sciences*, 280(1):20–32, 1976.

[Gebser *et al.*, 2012] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238, 2012.

[Gebser *et al.*, 2013] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Advanced conflict-driven disjunctive answer set solving. In *IJCAI*, pages 912–918. AAAI Press, 2013.

[Hart, 1996] Janis M Hart. The effect of personalized word problems. *Teaching Children Mathematics*, 2(8):504–505, 1996.

[Krahmer and Van Deemter, 2012] Emiel Krahmer and Kees Van Deemter. Computational generation of referring expressions: A survey. *Computational Linguistics*, 38(1):173–218, 2012.

[Lenat, 1995] Douglas B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[Publications, 2009] Frank Schaffer Publications. *Singapore Math 70 Must-Know Word Problems, Level 3 Grade 4*. Carson-Dellosa Publishing, LLC, 2009.

[Reiter and Dale, 1997] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[Renninger *et al.*, 2002] KA Renninger, L Ewen, and AK Lasher. Individual interest as context in expository text and mathematical word problems. *Learning and Instruction*, 12(4):467–490, 2002.

[Sadigh *et al.*, 2012] Dorsa Sadigh, Sanjit A Seshia, and Mona Gupta. Automating exercise generation: A step towards meeting the MOOC challenge for embedded systems. In *Proceedings of the Workshop on Embedded Systems Education (WESE), ESWeek*, 2012.

[Schumacher and Fuchs, 2012] Robin F Schumacher and Lynn S Fuchs. Does understanding relational terminology mediate effects of intervention on compare word problems? *Journal of experimental child psychology*, 111(4):607–628, 2012.

[Singh *et al.*, 2012] Rohit Singh, Sumit Gulwani, and Sriram K Rajamani. Automatically generating algebra problems. In *AAAI*, 2012.

[Smith and Mateas, 2011] Adam M Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):187–200, 2011.

[Smith *et al.*, 2012] Adam M Smith, Erik Andersen, Michael Mateas, and Zoran Popović. A case study of expressively constrainable level design automation tools for a puzzle game. In *FDG*, pages 156–163. ACM, 2012.

[Smith *et al.*, 2013] Adam M Smith, Eric Butler, and Zoran Popović. Quantifying over play: Constraining undesirable solutions in puzzle design. In *FDG*, pages 221–228, 2013.

[Tutenel *et al.*, 2009] Tim Tutenel, Rafael Bidarra, Ruben M Smelik, and Klaas Jan de Kraker. Rule-based layout solving and its application to procedural interior generation. In *CASA Workshop on 3D Advanced Media In Gaming And Simulation*, 2009.

[Verschaffel, 1994] Lieven Verschaffel. Using retelling data to study elementary school children's representations and solutions of compare problems. *Journal for Research in Mathematics Education*, pages 141–165, 1994.

[Wexler, 1968] Jonathan D Wexler. A self-directing teaching program that generates simple arithmetic problems. *Computer Sciences Technical Report*, 19, 1968.