Combining LLMs and Term Rewriting for Marking Algebraic Expressions in Physics Exams

Peter Baumgartner[1] , **Lachlan McGinness**[2]
[1]CSIRO/Data61 and Australian National University
[2] Australian National University and CSIRO/Data61
peter.baumgartner@data61.csiro.au, lachlan.mcginness@anu.edu.au

Abstract

We present our method for automatically marking Physics exams. The marking problem consists in assessing typed student answers for correctness with respect to a ground truth solution. This is a challenging problem that we seek to tackle using a combination of a computer algebra system, an SMT solver and a term rewriting system. A Large Language Model is used to interpret and remove errors from student responses and rewrite these in a machine readable format. Once formalized and language-aligned, the next step then consists in applying automated reasoning techniques for assessing student solution correctness. We consider two methods of automated theorem proving: off-the-shelf SMT solving and a term rewrite system tailored for physics problems involving trigonometric expressions. We report on experiments with these two systems on a rich pool of real-world student exam responses from the 2023 Australian Physics Olympiad.

1 Introduction

Many teachers across Australia are 'burning out' and leaving the profession due to excessive workload (Windle et al. 2022). As marking is one of the largest contributions to teacher workload, teachers are seeking AI marking solutions to make their workload more sustainable (Ogg 2024). Automated Essay Grading (Ramesh and Sanampudi 2022) and Automated Short Answer Grading (Weegar and Idestam-Almquist 2024) are longstanding areas of research.

Until recently, there were no effective strategies to mark free-form physics problems, as they could contain diverse inputs including text, equations and diagrams. Developments in generative AI have changed this and recent works have evaluated the potential of Large Language Models (LLMs) in grading physics exams (Kortemeyer 2023; Kortemeyer, Nöhl, and Onishchuk 2024; Mok et al. 2024; Chen and Wan 2025; McGinness and Baumgartner 2025a).

However, there are no guarantees of the correctness of LLM reasoning (Kambhampati et al. 2024). We propose a new framework, called *AlphaPhysics* (see Figure 1), that uses a combination of LLMs and automated reasoning engines. AlphaPhysics uses the strong pattern recognition abilities of LLMs to translate student responses into a standardized format before applying more rigorous reasoning engines, specifically *Term Rewriting Systems* (*TRS*s), to evaluate student responses.

Term rewriting is a well-established framework for equation-based theorem proving(Dershowitz and Jouannaud 1990; Baader and Nipkow 1998). We propose a tailored TRS for automating marking tasks in the pre-calculus physics domain. The main reasoning task is simplification of algebraic expressions to a normal form. Normalization then acts as a semantics-preserving "translation" service. The normal form of a student's expression and a solution can be compared to determine if they are semantically equivalent.

Our rule language features addition, multiplication, exponentiation and trigonometric functions (sine and cosine). The language supports *built-in arithmetic*, which is very useful in the physics domain. This poses challenges as it requires reasoning on infinite domains and in presence of commutativity and other axioms.

We describe our methods for dealing with necessarily incomplete equational logic over trigonometric and arithmetic operators. The theorem proving core of AlphaPhysics comprises four term rewrite systems (TRSs) that are chained for normalizing a given equation. We developed additional technical concepts beyond the scope of this paper for verifying termination and checking confluence of our TRSs.[1] In this paper we focus on the applications, grading student typed equations that may contain errors by comparing these to correct answers provided by a marking scheme.

Main contributions. In this paper we describe the current state of our *AlphaPhysics* framework (See Figure 1). We introduce our TRS tailored to pre-calculus physics with trigonometric expressions. We report on experiments with our implemented system using a generic TRS interpreter for our rule language. We also compare our results with those obtained by a state-of-the-art automated theorem prover Z3 (Bjørner and Nachmanson 2024).

Related work in Term Rewriting. Term rewriting with built-in operations and numeric constraints has a long tradition (Kaplan and Choppy 1989; Avenhaus and Becker 1994; Kop and Nishida 2013) but adding term order constraints for controlling termination is not commonly featured in available systems. As an alternative to term rewriting, one could

[1]Checking these properties is not trivial as our TRS language supports ordering constraints and infinite domains. We show our TRS is terminating and sound but not confluent nor complete.

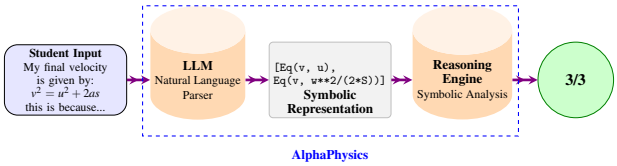


Figure 1: In the *AlphaPhysics* pipeline a student response is parsed by an LLM before a symbolic reasoning engine determines the corresponding grade.

consider first-order logic automated theorem proving (ATP) over built-in domains. For instance, the hierarchic superposition calculus (Baumgartner and Waldmann 2013) and its implementation features search space restrictions by means of term ordering constraints. While the termination and confluence analysis for a TRS is expected to be done once and forall as an offline step, superposition-based ATPs generate formulas for restoring confluence by deriving formulas during proof search. This involves inferences among the axioms, which typically is not finitely bounded.

The rest of this paper is structured as follows. Section 2 describes the use of LLMs for pre-processing student equations. In Section 3, we introduce the architecture of our TRS. Section 4 introduces the specific marking problem that we are trying to solve. Section 5 shows the results of experiments comparing our TRS to an SMT solver. Section 6 discusses the limitations of our system and how it could be extended to mark a wider variety of physics problems.

2 Large Language Model Pre-processing

AlphaPhysics is designed to meet the Australian AI Ethics Framework which include environmental well-being, transparency and privacy protection (Department of Industry Science and Resources 2024). To address these requirements we use local open source LLMs to extract features, such as equations, from student responses so they can be parsed by our TRS. This approach maintains student privacy by keeping data on a local system. Our proposed framework supports transparency as errors in the LLM feature extraction can be easily seen, contested and corrected by a student or teacher. The Alpha Physics approach makes teachers aware of the compute that they are using by running local models, exposing the normally hidden electricity usage and environmental cost of their usage.

The local LLM is given the task of extracting student equations from typed student responses and converting these into a standardised format. The student may make syntax errors or use incorrect or undefined variables. The LLM is expected to compensate for these errors in a way similar to a human marker in order to extract what the student intended to write. For example a student may write:

m1 x v0 + m2 = m1 x v1 + m2 x v2

which the LLM converts to correct SymPy syntax:

Eq(m_1*v_0 + m_2, m_1*v_1 + m_2*v_2)

A previous study (McGinness and Baumgartner 2025a) tested LLMs' capabilities to extract equations using the typed student responses to the 2023 Australian Physics Olympiad. A 14 billion parameter model, Phi 4 (Abdin and Zhang 2024), was able to complete the task of translating student equations with 73% accuracy by using an 'LLM-Modulo' (Kambhampati et al. 2024) prompting technique where Z3 is used as a syntax checker. If the equation provided by the LLM fails the Z3 check, then the LLM is prompted again with feedback on its previous response to repair the equations.

This paper focuses on the application of the *AlphaPhysics* TRS to the ground truth equations for each student response. The next section defines our TRS tailored for pre-calculus physics problems.

3 Term Rewriting Framework

Preliminaries. We assume standard notions of first-order logic theorem proving and term rewriting, see (Harrison 2009; Baader and Nipkow 1998). A *signature* Σ is a collection of function and predicate symbols of given fixed and finite arities. 0-ary function symbols are also called constants. In this paper we fix "arithmetic" signatures Σ to comprise of the binary function symbols $+, \times, \hat{}$ (exponentiation), $/$ (division), unary function symbols sin, cos, a unary function symbol quote explained below, the constant π, and all integer and finite decimal concrete number constants, e.g., -1, 3.141.

Physics equations often contain universally quantified variables x, y, z, θ etc. In order not to confuse them with the meta-variables of logic, we consider extensions of Σ with finitely many (Skolem) constants Π representing these variables and call them *parameters*. We write Σ_Π for the extended signature.

Let $\Sigma_\Pi(V)$ denote the extension of Σ_Π by a denumerable set of variables. Unless noted otherwise, we assume the signature $\Sigma_\Pi(V)$, for some Π left unspecified. Terms are defined as usual, but we use infix notation and parentheses for writing terms made with arithmetic function symbols. A *parameter-free term* is a term without occurrences of parameters. We write $var(t)$ for the set of variables occurring in a term t. A *ground term* is a term t with $var(t) = \emptyset$. We use notions of *substitutions*, *instance*, *ground instance* and *(term) matching* in a standard way. The notation $t_p[u]$ means that t has a, not necessarily proper, subterm u at position p. The position index p is left away if clear from the context or not important. We use the letters γ, δ and σ for substitutions, and s and t for terms. A *ground substitution γ for t* is a substitution such that $t\gamma$ is ground. We often assume t is clear from the context and leave "for t" away. As for semantics, let $\mathcal{A}_\pi$ be a Σ_Π-Algebra with the reals as carrier set that maps all arithmetic function symbols in Σ to the expected functions over the reals, and maps every parameter in Π to some real number as per the *parameter mapping* π. An *assignment* α is a mapping from the variables to the reals. We write $\mathcal{A}_\pi(\alpha)$ for the usual interpretation function on terms, where variables are interpreted according to α. For ground terms we can unambiguously write $\mathcal{A}_\pi(t)$ instead of $\mathcal{A}_\pi(\alpha)(t)$.

Quoted terms and simplification. The signature includes the distinguished unary function symbol "quote". For better readability we write a quote-*term* quote(t) as $[\![t]\!]$ and call t

the *quoted term*. A term is quote-*free* if it is not a quoted term and does not contain a quote-term.

Quote-terms are our mechanism for building-in arithmetic on numbers. Ground quoted terms are simplified to a number constant (like $[\![2]\!]$ for $[\![1+1]\!]$), and non-ground quoted terms can be replaced by a semantically equivalent terms (like $[\![2\times x+1]\!]$ for $[\![x+x+1]\!]$). Formally, we define a *simplifier*[2] as a total function *simp* on quote-free and parameter-free terms such that

(i) if $simp(t) = s$ then, for all assignments α, $\mathcal{A}(\alpha)(t) = \mathcal{A}(\alpha)(s)$,

(ii) for every number n, $simp(n) = n$,

(iii) if t is ground then $simp(t)$ is a number.

Condition (i) is a soundness requirement and forbids *simp* to equate two semantically different terms ("no confusion"). Condition (iii) is needed for completeness of theorem proving by rewriting.

We extend *simp* homomorphically to all terms as follows:

$$t{\downarrow_{simp}} = \begin{cases} t & \text{if } t \in V \text{ or } t \in \Pi \\ [\![simp(u)]\!] & \text{if } t = [\![u]\!] \\ f(t_1{\downarrow_{simp}}, \ldots, t_n{\downarrow_{simp}}) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

We say that t is *(fully) simplified* iff $t = t{\downarrow_{simp}}$

By design of our rewrite systems, in all derivations from ground terms, all quote-terms in all rule instances are always parameter-free, ground and quote-free, hence can be simplified to a number. Non-ground quote-terms are needed only for confluence and termination analysis.

The vast majority of the literature on term-rewriting over built-in domains uses rules with constraints. What we achieve with a quote-term $[\![t]\!]$ can be expressed by replacing it in the rule with a *sorted* variable x and adding the constraint $x = t$. See (Kop and Nishida 2013) for a discussion and overview of rules with constraints. Notice that our approach makes explicit typing unnecessary as quoting achieves the same. We found that this simple approach works well for our use case.

Constrained rewrite rules and normal forms. A *(constrained rewrite) rule* ρ is of the form $l \to r \mid C$ where l and r are terms such that $var(r) \subseteq var(l)$, and the *constraint* C is a finite set of formulas whose free variables are contained in $var(l)$. Notice that $var(\rho)$, the set of (free) variables occurring in ρ, is just $var(l)$. If $C = \emptyset$ we just write $l \to r$. We assume the signature of the constraint language contains conjunction, so that constraints can be taken as the conjunction of their elements. We model evaluation of constraints by assuming a *satisfaction relation* $\mathcal{I}$ on constraints. We write $\mathcal{I} \models C$ instead of $C \in \mathcal{I}$. Notice that $\mathcal{I} \models \emptyset$, as expected. We require that constraint satisfaction is *stable under substitution*: if $\mathcal{I} \models C$ then $\mathcal{I} \models C\delta$.[3] If $\mathcal{I} \models C\delta$ we call the resulting unconstrained rule $l\delta \Rightarrow r\delta$ an *ordinary instance (of ρ) (via δ)*. (We choose $l\delta \Rightarrow r\delta$ as $l\delta \to r\delta$ is our convention for a rule with an empty constraint.)

The constraints C can be purely operational, in the NORM TRS, or term ordering constraints in the CANON and SIMP TRSs. As an example for the latter, one of our rules is $x + y \to y + x \mid x \succ y$ which orders $+$-terms but the ordering constraint prevents unbound application of the rule. The ordering $\succ$ is defined as both an extension (to the infinite domain of quote-terms) and an instantiation of a weighted path ordering (WPO) (Yamada, Kusakari, and Sakabe 2015).

Let $\rho = (l \to r \mid C)$ be a rule. We say that *s is obtained from t by (one-step) rewriting (with simplification)* and write $t \to_\rho s$ if $t = t_p[u]$ for some non-variable term u and position p, $u = l\delta$ for some substitution δ, $l\delta \Rightarrow r\delta$, and $s = t_p[(r\delta){\downarrow_{simp}}]$. A rewriting step is *ground* iff t is ground (and hence s is ground, too).

A *rewrite system* R is a finite set of rewrite rules. We define the *R-rewrite relation* $\to_R$ as $t \to_R s$ iff $t \to_\rho s$ for some $\rho \in R$. Let $\to_R^*$ be the transitive-reflexive closure of $\to_R$. We say that *s is an R-normal form of t* iff $t \to_R^* s$ but $s \not\to_R s'$ for any s'. We define the *R-normal form relation* $\downarrow_R$ as $t \downarrow_R s$ iff s is an R-normal form of t.

3.1 Proving Physics Equations by Normalization

Theorem proving, the validity problem for $\Sigma(V)$-equations $\forall(s = t)$, is phrased in our setting as "does $\mathcal{A}_\pi(s_{sk}) = \mathcal{A}_\pi(t_{sk})$ hold for all parameter mappings π?", where s_{sk} and t_{sk} are Σ_Π-terms obtained from s and t by uniquely replacing every variable by a parameter from Π. This problem is, of course, not solvable in general. Our TRS method approximates solving it in an incomplete but sound way by combining four rewrite systems into one procedure for normalization. If the normalized versions of s and t are syntactically equal then the answer is "yes", otherwise "unknown".

The four rewrite systems are NORM, CANON, SIMP and CLEAN, collectively called the *ARI rewrite systems*. For normalization, the systems are chained, each exhaustively applying rewrite rules on the result of the previous one. More formally, we define the *ARI normalization* relation as $\to_{\text{ARI}} = {\downarrow_{\text{NORM}}} \circ {\downarrow_{\text{CANON}}} \circ {\downarrow_{\text{SIMP}}} \circ {\downarrow_{\text{CLEAN}}}$. The composition operator, $\circ$, stands for application from left to right. We say that a term u is an *ARI-normal form of s* iff $\text{norm}(s) \to_{\text{ARI}} u$. We say that s and t are *algebraically equal*, written as $s \approx t$, if s and t have a common ARI-normal form u. If the set of rewrite rules is complete then equations which can be converted from one to the other by standard algebraic operations should have the same normal form. In this case we say that the normal form is unique.

Let us explain the design and intention of the ARI rewrite systems and how they work together. The NORM system, see Table 1, implements several conceptually simple "preprocessing" operations. They are triggered by decorating a target term s as $\text{norm}(s)$.

NORM expands exponentiation terms $x\hat{}n$ where n is an integer number into a term $\text{pwr\_n}(x, \text{s}(\text{s}(\ldots([\![0]\!]))))$ where the second argument encodes n as n-fold "successor" of 0. Similar special cases are $\sin(n \times x)$ and $\cos(n \times x)$. These translate into similar terms with sin_n and cos_n, respectively. These patterns are recognized with type-checking

[2] A simplifier plays a similar role as a "canonizer" in (Shostak 1984).

[3] For soundness reasons; the free variables are used in a universal quantification context, so instances better be satisfied, too.

Table 1: The NORM rewrite system. The rule conditions in the rightmost column are in Python syntax.

N1.1	$\text{norm}(\llbracket x \rrbracket)$	$\to$	$\llbracket x \rrbracket$	
N1.2	$\text{norm}(x)$	$\to$	$\llbracket x \rrbracket$	$\text{is}(x, \text{number})$
N1.3	$\text{norm}(x)$	$\to$	$\llbracket 1 \rrbracket \times (x \hat{} \llbracket 1 \rrbracket)$	$\text{is}(x, \text{parameter})$
N2.1	$\text{norm}(\sin(n \times y))$	$\to$	$\text{norm}(\text{sin\_n}(n, y))$	if $\text{is}(n,\ \text{int})$ and $n \geq 0$
N2.2	$\text{norm}(\sin(x \times y))$	$\to$	$\sin(\text{norm}(x \times y))$	if not $(\text{is}(x,\ \text{int})$ and $x \geq 0)$
N2.3	$\text{norm}(\sin(x))$	$\to$	$\sin(\text{norm}(x))$	if not $(\text{is\_funterm}(x)$ and $x.\text{fun} == \times)$
N2.4	$\text{norm}(\cos(n \times y))$	$\to$	$\text{norm}(\text{cos\_n}(n, y))$	if $\text{is}(n,\ \text{int})$ and $n \geq 0$
N2.5	$\text{norm}(\cos(x \times y))$	$\to$	$\cos(\text{norm}(x \times y))$	not $(\text{is}(x,\ \text{int})$ and $x \geq 0)$
N2.6	$\text{norm}(\cos(x))$	$\to$	$\cos(\text{norm}(x))$	not $(\text{is\_funterm}(x)$ and $x.\text{fun} == \times)$
N3.1	$\text{norm}(\text{sin\_n}(n, x))$	$\to$	$\text{sin\_n}(\text{to\_succ}(n), \text{norm}(x))$	
N3.2	$\text{norm}(\text{cos\_n}(n, x))$	$\to$	$\text{cos\_n}(\text{to\_succ}(n), \text{norm}(x))$	
N4.1	$\text{norm}(x + y)$	$\to$	$\text{norm}(x) + \text{norm}(y)$	
N4.2	$\text{norm}(x \times y)$	$\to$	$\text{norm}(x) \times \text{norm}(y)$	
N4.3	$\text{norm}(x \hat{} y)$	$\to$	$\text{norm}(x) \hat{} \text{norm}(y)$	not $(\text{is}(y,\ \text{int})$ and $y \geq 0)$
N5.1	$\text{norm}(x \hat{} n)$	$\to$	$\text{norm}(\text{pwr\_n}(x, n))$	$\text{is}(n,\ \text{int})$ and $n \geq 0$
N5.2	$\text{norm}(\text{pwr\_n}(x, n))$	$\to$	$\text{pwr\_n}(\text{norm}(x), \text{to\_succ}(n))$	
N5.3	$\text{to\_succ}(0)$	$\to$	$\llbracket 0 \rrbracket$	
N5.4	$\text{to\_succ}(n)$	$\to$	$\text{s}(\text{to\_succ}(n-1))$	$n > 0$
N6.1	$\text{norm}(x - y)$	$\to$	$\text{norm}(x + \text{uminus}(y))$	
N6.2	$\text{norm}(\text{uminus}(x))$	$\to$	$\llbracket -1 \rrbracket \times \text{norm}(x)$	
N7.1	$\text{norm}(x / y)$	$\to$	$\text{norm}(x) \times (\text{norm}(y) \hat{} \llbracket -1 \rrbracket)$	

constraints that are evaluated by the host language Python. Unary minus and division are eliminated in terms with multiplication by -1 and exponentiation by -1, respectively. NORM also replaces every number n by $\llbracket n \rrbracket$, and every parameter a by $\llbracket 1 \rrbracket \times \text{a} \hat{} \llbracket 1 \rrbracket$.

The CANON and SIMP rewrite systems are defined in Tables 2 and 3, respectively. Their A1.2, A1.5, S1 and S2 rules have term ordering constraints of the form $x \succ y$ between variables. As an outlier, the rule T1.7 is the *only* rule not oriented with our term ordering. This circumstance did not lead to non-termination in our experiments but should certainly be addressed rigorously.

The CANON system (see Table 2) has rules for addition, multiplication, exponentiation, distributivity, and for evaluating quote-terms that are combined by arithmetic operators. It is not hard to see that every quoted term in every derivable term is initially given as a number (by NORM) or will be *simp*lified to a number as part of rewriting steps. CANON has rules for "sorting" factors of products in increasing order wrt. $\succ$, for example $x \times (y \times z) \to y \times (x \times z) \mid x \succ y$. Sorting is important for collecting like-terms as then only adjacent terms need to be considered. CANON has rules for trigonometric identities, and for expanding exponentiation with integer constants. For example $(\text{a} + \text{b}) \hat{} 3$ will be fully multiplied out in the obvious way.

The main task of SIMP is to sort sums of monomials so that like-monomials can be collected, e.g., with $(\llbracket a \rrbracket \times x) + (\llbracket b \rrbracket \times x) \to \llbracket a + b \rrbracket \times x$ as one of these rules.[4]

Finally, CLEAN consists of the three rules $\llbracket x \rrbracket \to x$, $1 \times x \to x$ and $x \hat{} 1 \to x$ for a more simplified presentation of the final result.

[4] SIMP cannot be integrated with CANON as SIMP needs a right-to-left status for multiplication instead of left-to-right so that leading number coefficients are ignored for sorting. Their combination into one system leads to problems in proving termination.

Table 2: CANON rewrite rules

A1.1	$(x + y) + z$	$\to$	$x + (y + z)$	
A1.2.1	$x + y$	$\to$	$y + x$	$x \succ y$
A1.2.2	$x + (y + z)$	$\to$	$y + (x + z)$	$x \succ y$
A1.3.1	$\llbracket 0 \rrbracket + x$	$\to$	x	
A1.3.2	$\llbracket a \rrbracket + \llbracket b \rrbracket$	$\to$	$\llbracket a + b \rrbracket$	
A1.3.3	$\llbracket a \rrbracket + (\llbracket b \rrbracket + z)$	$\to$	$\llbracket a + b \rrbracket + z$	
A1.3.4	$(\llbracket a \rrbracket \times x) + (\llbracket b \rrbracket \times x)$	$\to$	$\llbracket a + b \rrbracket \times x$	
A1.3.5	$(\llbracket a \rrbracket \times x) + ((\llbracket b \rrbracket \times x) + z)$	$\to$	$(\llbracket a + b \rrbracket \times x) + z$	
A1.4	$(x \times y) \times z$	$\to$	$x \times (y \times z)$	
A1.5.1	$x \times y$	$\to$	$y \times x$	$x \succ y$
A1.5.2	$x \times (y \times z)$	$\to$	$y \times (x \times z)$	$x \succ y$
A1.6.1	$\llbracket 0 \rrbracket \times x$	$\to$	$\llbracket 0 \rrbracket$	
A1.6.2	$\llbracket a \rrbracket \times \llbracket b \rrbracket$	$\to$	$\llbracket a \times b \rrbracket$	
A1.6.3	$\llbracket a \rrbracket \times (\llbracket b \rrbracket \times z)$	$\to$	$\llbracket a \times b \rrbracket \times z$	
A1.7.1	$x \times (y + z)$	$\to$	$(x \times y) + (x \times z)$	
A1.7.2	$(y + z) \times x$	$\to$	$(y \times x) + (z \times x)$	
A1.8.1	$(x \hat{} y) \hat{} z$	$\to$	$x \hat{} (y \times z)$	
A1.8.2	$(x \hat{} y) \times (x \hat{} z)$	$\to$	$x \hat{} (y + z)$	
A1.8.3	$(x \hat{} y) \times ((x \hat{} z) \times v)$	$\to$	$(x \hat{} (y + z)) \times v$	
A1.9.1	$\llbracket a \rrbracket \hat{} \llbracket b \rrbracket$	$\to$	$\llbracket a \hat{} b \rrbracket$	
A1.9.2	$\text{pwr\_n}(x, \llbracket 0 \rrbracket)$	$\to$	$\llbracket 1 \rrbracket$	
A1.9.3	$\text{pwr\_n}(x, \text{s}(n))$	$\to$	$x \times \text{pwr\_n}(x, n)$	
A1.9.4	$(x + y) \hat{} \llbracket 1 \rrbracket$	$\to$	$x + y$	
A1.9.5	$(x \times y) \hat{} \llbracket a \rrbracket$	$\to$	$(x \hat{} \llbracket a \rrbracket) \times (y \hat{} \llbracket a \rrbracket)$	
A1.9.6	$x \hat{} \llbracket 0 \rrbracket$	$\to$	$\llbracket 1 \rrbracket$	
T1.1	$\sin(\llbracket -1 \rrbracket \times x)$	$\to$	$\llbracket -1 \rrbracket \times \sin(x)$	
T1.2	$\cos(\llbracket -1 \rrbracket \times x)$	$\to$	$\cos(x)$	
T1.3	$\sin(x_1 + x_2)$	$\to$	$\sin(x_1) \times \cos(x_2) + \cos(x_1) \times \sin(x_2)$	
T1.4	$\cos(x_1 + x_2)$	$\to$	$\cos(x_1) \times \cos(x_2) + \llbracket -1 \rrbracket \sin(x_1) \times \sin(x_2)$	
T1.5	$\text{cos\_n}(\text{s}(\text{s}(n)), x)$	$\to$	$\llbracket 2 \rrbracket \times (\cos(x) \times \text{cos\_n}(\text{s}(n), x)) + (\llbracket -1 \rrbracket \times \text{cos\_n}(n, x))$	
T1.6	$\text{sin\_n}(\text{s}(\text{s}(n)), x)$	$\to$	$\llbracket 2 \rrbracket \cos(x) \times \text{sin\_n}(\text{s}(n), x) + (\llbracket -1 \rrbracket \times \text{sin\_n}(n, x))$	
T1.7	$\sin(x) \hat{} \llbracket 2 \rrbracket$	$\to$	$\llbracket 1 \rrbracket + \llbracket -1 \rrbracket \times \cos(x) \hat{} \llbracket 2 \rrbracket$	Not oriented
T2.1	$\text{cos\_n}(\text{s}(\llbracket 0 \rrbracket), x)$	$\to$	$\cos(\llbracket 1 \rrbracket \times x)$	
T2.2	$\text{cos\_n}(\llbracket 0 \rrbracket, x)$	$\to$	$\cos(\llbracket 0 \rrbracket)$	
T2.3	$\text{sin\_n}(\text{s}(\llbracket 0 \rrbracket), x)$	$\to$	$\sin(\llbracket 1 \rrbracket \times x)$	
T2.4	$\text{sin\_n}(\llbracket 0 \rrbracket, x)$	$\to$	$\sin(\llbracket 0 \rrbracket)$	
T2.5	$\sin(\llbracket x \rrbracket)$	$\to$	$\llbracket \sin(x) \rrbracket$	
T2.6	$\cos(\llbracket x \rrbracket)$	$\to$	$\llbracket \cos(x) \rrbracket$	

Table 3: SIMP rewrite system rules.

S1	$(\llbracket a \rrbracket \times x) + (\llbracket b \rrbracket \times y)$	$\to$	$(\llbracket b \rrbracket \times y) + (\llbracket a \rrbracket \times x)$	$x \succ y$
S2	$(\llbracket a \rrbracket \times x) + ((\llbracket b \rrbracket \times y) + z)$	$\to$	$(\llbracket b \rrbracket \times y) + ((\llbracket a \rrbracket \times x) + z)$	$x \succ y$
S3	$(\llbracket a \rrbracket \times x) + (\llbracket b \rrbracket \times x)$	$\to$	$\llbracket a + b \rrbracket \times x$	
S4	$(\llbracket a \rrbracket \times x) + ((\llbracket b \rrbracket \times x) + z)$	$\to$	$(\llbracket a + b \rrbracket \times x) + z$	

Example 1 (ARI-normal form)**.** Consider the following ARI-normal form computation $s \to_{\text{ARI}} t$ which has parameters $\text{b} \succ \text{a}$. It uses standard mathematical notation for better readibility; multiplication $\cdot$ is *left* associative. In the example we use $\cdot$ instead of $\times$ to save space.

$$\text{norm}(s) = \text{norm}(2 \cdot \text{b} \cdot 3 \cdot \text{a} \cdot 5 \cdot \text{b} + 5) \tag{1}$$

$$\to^*_{\text{NORM}} \llbracket 2 \rrbracket \cdot \llbracket 1 \rrbracket \cdot \text{b}^{\llbracket 1 \rrbracket} \cdot \llbracket 3 \rrbracket \cdot \llbracket 1 \rrbracket \cdot \text{a}^{\llbracket 1 \rrbracket} \cdot \llbracket 5 \rrbracket \cdot \llbracket 1 \rrbracket \cdot \text{b}^{\llbracket 1 \rrbracket} + \llbracket 5 \rrbracket \tag{2}$$

$$\to^*_{\text{CANON}} \llbracket 5 \rrbracket + \llbracket 6 \rrbracket \cdot \llbracket 5 \rrbracket \cdot \llbracket 1 \rrbracket \cdot \llbracket 1 \rrbracket \cdot \text{a}^{\llbracket 1 \rrbracket} \cdot \llbracket 1 \rrbracket \cdot \text{b}^{\llbracket 1 \rrbracket} \cdot \text{b}^{\llbracket 1 \rrbracket} \tag{3}$$

$$\to^*_{\text{CANON}} \llbracket 5 \rrbracket + \llbracket 30 \rrbracket \cdot \text{a}^{\llbracket 1 \rrbracket} \cdot \text{b}^{\llbracket 2 \rrbracket} \tag{4}$$

$$\to^*_{\text{SIMP}} \llbracket 5 \rrbracket + \llbracket 30 \rrbracket \cdot \text{a}^{\llbracket 1 \rrbracket} \cdot \text{b}^{\llbracket 2 \rrbracket} \tag{5}$$

$$\to^*_{\text{CLEAN}} 5 + 30 \cdot \text{a} \cdot \text{b}^2 = t \tag{6}$$

Line (2) demonstrates the replacements (a) and (b) mentioned above with NORM. It achieves that every monomial over parameters is either a number or a product of (at least one) number and parameters with exponents. This form is assumed and exploited by the CANON rules by moving all numbers to the left with aggregated multiplication, and sorting all parameters with exponents in increasing order wrt.

$\succ$. The ordering $\succ$ is such that every quote-term is smaller than every non-quote term, e.g., $\mathsf{a} \succ [\![1]\!]$. Line (3) is a snapshot of the CANON process before reaching CANON-normal form on line (4). Notice that two occurrences of the $\mathsf{b}^{\wedge}[\![1]\!]$ term have been like-collected into $\mathsf{b}^{\wedge}[\![2]\!]$. The SIMP rewrite system has no effect in this example, see line (5). Finally, line (6) simplifies and unquotes numbers. ARI-normal form computation hence uses quoting only as an intermediate device for triggering built-in evaluation of arithmetic terms.

Some examples for normalization of trigonometric terms:

$$\sin(\mathsf{a}+\mathsf{b}) \to_{\text{ARI}} \cos(\mathsf{a})\cdot\sin(\mathsf{b})+\cos(\mathsf{b})\cdot\sin(\mathsf{a})$$
$$\sin(3\cdot\mathsf{c}) \to_{\text{ARI}} -1\cdot\sin(\mathsf{c})+4\cdot\cos(\mathsf{c})\cdot\cos(\mathsf{c})\cdot\sin(\mathsf{c})$$
$$\sin(\pi/2-\phi) \to_{\text{ARI}} -1\cdot\sin(\phi)\cdot\cos(0.5\cdot\pi)+\cos(\phi)\cdot\sin(0.5\cdot\pi)$$

Note 2 (Uniqueness and confluence)**.** The intention behind CANON is to sort the parameters with exponents by their bases only, for collecting like-terms. Unfortunately, this is not always possible. For example, $\mathsf{b}^{\mathsf{d}} \succ \mathsf{a}^{\mathsf{e}}$ is sorted as intended, where $\mathsf{b} \succ \mathsf{a}$. However, $\mathsf{a}^{\mathsf{b}^{\mathsf{d}}} \succ \mathsf{b}^{\mathsf{d}}$ is not. This phenomenon can lead to non-confluence. Notice, it required an exponent b^{d} that is equal to (or greater than) another factor. Luckily, such cases are rare in physics exams.

3.2 Termination of Rewriting

A rewrite system R is *terminating* if there is no infinite rewrite derivation. This means there is no sequence of one-step rewrites $t \to_R t_1 \to_R t_2 \to_R \cdots$, for any term t. A standard way to prove termination of rewrite systems over a finite signature is to define a *reduction ordering* $\succ$ such that all rules are *oriented*, i.e., $l \succ r$ for every standard rule $l \to r$. The counterpart for our constrained rules is to show that every rule ρ (in ARI) is *oriented wrt. ordinary instances (for $\succ$)*, i.e., that every ordinary instance $s \Rightarrow t$ of ρ satisfies $s \succ t$. This requires specific analysis for each constrained rule. For example, $x+y \to y+x \mid x \succ y$ is oriented in this sense if the symbol "+" has left-to-right lexicographic status: it holds that, for every ordinary instance $s+t \Rightarrow t+s$, where $s \succ t$, the term $s+t$ is greater than $t+s$ wrt. $\succ$. The ordinary instance is not oriented if the order in the tuple were reversed.

Our concrete reduction ordering used for showing orientedness is both an extension (by quote-terms) and instantiation of Weighted Path Ordering (WPO) (Yamada, Kusakari, and Sakabe 2015).

4 The Dataset

We examined student responses to the Australian Physics Olympiad exam of 2023. There were a total of 1526 typed student responses (including blank responses) to each of the 45 questions. The marking team's grade for each student response was also provided. Approximately 10% of the responses were hand-written, scanned and attached as images. The hand-written responses were not considered in this study and therefore we changed students scores for these responses to 0.

We focus on grading two questions. Question 25 requires flexible marking, as there were multiple forms of the correct answer. Question 26 requires students to use trigonometric functions.

4.1 Question 25

Question 25 is worded as follows:

> *Consider two solid, spherical masses, one with mass m_1, and one with mass m_2. Assuming that m_2 is initially at rest, and that m_1 is incident on m_2 with some energy E_0, the particles will scatter with final energies E_1 and E_2 respectively (as shown). Write an equation for conservation of energy for this process. Only include the following variables in your answer:*
>
> - *m_1 – the mass of the incoming particle*
> - *m_2 – the mass of the originally stationary target*
> - *E_0 – the kinetic energy of m_1 before the collision*
> - *E_1 – the kinetic energy of m_1 after the collision*
> - *E_2 – the kinetic energy of m_2 after the collision*

The question has a simple correct answer: $E_0 = E_1 + E_2$. However many students are taught to substitute expressions for kinetic energy ($KE = \frac{mv^2}{2}$) when writing conservation of energy equations. Therefore many students gave answers such as $m_1 v_0^2 = m_1 v_1^2 + m_2 v_2^2$ which are equivalent to the correct answer and markers would be able to determine the physical/algebraic equivalence of this student response.

To allow our system to mimic this grader behavior, the substitutions in Table 4 were applied to the student expressions exhaustively as the final step of pre-processing. After this, both Z3 (an SMT solver) and our TRS determined the equivalence of the student answer and the correct solution and assigned a grade to the student.

Table 4: Expressions for energy and momentum of particles before and after an interaction

Original Quantity	New Expression	Description
$E_0 \to$	$\frac{m_1 v_0^2}{2}$	Initial kinetic energy of particle 1
$p_0 \to$	$m_1 v_0$	Initial momentum of particle 1
$E_1 \to$	$\frac{m_1 v_1^2}{2}$	Final kinetic energy of particle 1
$p_1 \to$	$m_1 v_1$	Final momentum of particle 1
$E_2 \to$	$\frac{m_2 v_2^2}{2}$	Final kinetic energy of particle 2
$p_2 \to$	$m_2 v_2$	Final momentum of particle 2

The Z3 Grading process of Question 25 is as follows. We define the *trigonometric axiom set* T as a set of equations corresponding to trigonometric axioms where x is a variable implicitly universally quantified within each formula. See Section 5.1 for an example of set T.

Let $D_i = \{d_{i1}, d_{i2}, \ldots, d_{ip}\}$ be the set of equations contained in the ith student's response and $C = \{c_1, c_2, \ldots, c_q\}$ be the set of equations required in the marking scheme. We generate a set of additional inequalities, which we call *non-zero constraints*, $Z = \{z_1, z_2, \ldots, z_l\}$ which prevent expressions from being undefined. For example, expressions in denominators are not allowed to equal 0.

We define two equations d and c as *z3-equivalent*, $d \equiv_{Z3} c$, if $T \models \forall(Z \to (d \equiv c))$. We then break the equivalence checking task into two unsatisfiability checks for Z3.

For Question 25, C consists only of one equation, $C = \{E_0 = E_1 + E_2\}$ to which the Table 4 rules are applied. We refer to this equation as c_1. The Z3 assigned mark for the ith student is given by:

$$M_{Z3i} = \begin{cases} 1 & \text{if } \exists d_{ip} \in D_i \text{ such that } d_{ip} \equiv_{Z3} c_1 \\ 0 & \text{otherwise} \end{cases}$$

The TRS grading procedure for Question 25 was different. We define *solving an equation e for a parameter x* as the process of performing valid algebraic operations to the equation such that the LHS consists only of x and the RHS does not contain x. In general, not all equations are solvable.

We made the choice to solve all student equations d_{ij} for the parameter v_0 because the pre-processing substitutions in Table 4 guarantee that a correct answer must contain this parameter. Therefore equations were expressed in the form $v_0 = e_j$ where e_j is an expression containing numbers, parameters and well founded function symbols but not v_0.

We define a *solving function, f*, such that:

$$f(d_{ij}) = \begin{cases} e_j & \text{if } d_{ij} \text{ can be solved by our computer} \\ & \text{algebra system for } v_0 \\ \text{NaN} & \text{otherwise} \end{cases}$$

Where *NaN* is a special expression with the property $\text{NaN} \approx c = \bot$ any expression c. See Section 3.1 for definition of algebraically equal, $\approx$.

We implemented the solving function $f()$ using the SymPy algebra system (Meurer et al. 2024). We note that there were no cases where SymPy was unable to solve for v_0. The rewrite system mark for the ith student was given by:

$$M_{\text{TRS}i} = \begin{cases} 1 & \text{if } \exists d_{ij} \in D_i | f(d_{ij}) \approx f(c_1) \\ 0 & \text{otherwise} \end{cases}$$

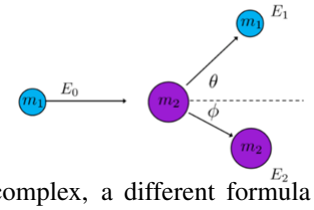
Intuitively this means that students were awarded a mark if any of the equations that they wrote were equivalent to the correct answer. We note that this may not be an appropriate way to mark complex questions which require students to demonstrate understanding through correct working, however it is sufficient for this one-mark question.

4.2 Question 26

Question 26 was worded as follows:

Write two equations for conservation of momentum, accounting for the scattering angles ϕ and θ.

The same LLM pre-processing was applied to student responses to Question 26 to standardize their format and remove syntax errors.



However as the marking scheme for Question 26 was more complex, a different formula was used to grade the student responses.

The Z3 Grading Procedure for Question 26 was as follows. Let $W = \{w_1, w_2, ..., w_q\}$ be the *marking weights* corresponding to each required equation $c \in C$. We define the *identity function*, $\mathbb{I}(\phi)$ as:

$$\mathbb{I}(\phi) = \begin{cases} 1 & \phi = \top \\ 0 & \phi = \bot \end{cases}$$

The Z3 mark assigned to the student is given by:

$$M_{Z3i} = \sum_{k=1}^{q} w_k \cdot \mathbb{I}(\exists d_{ij} \in D_i | d_{ij} \equiv_{Z3} c_k)$$

Note that for Question 26 $q = 2$ and $w_1 = w_2 = 0.5$. Intuitively, students were awarded 0.5 marks if any of their equations matched the correct x-momentum equation, and 0.5 marks if any of their equations matched the correct y-momentum equation. In Sections 5.1 and 5.2 we describe how this technique enabled Z3 to assign the correct marks to most of the student responses.

The TRS grading procedure for Question 26 was similar. The rewrite system awarded points to the students in Question 26 as follows:

$$M_{\text{TRS}i} = \sum_{k=1}^{q} w_k \cdot \mathbb{I}(\exists d_{ij} \in D_i | f(d_{ij}) \approx f(c_k))$$

This means that students were awarded marks according to the weight on the marking scheme for each equation they wrote which was algebraically equivalent to a corresponding article on the scheme. This mirrors the behavior of the majority of markers for most physics questions. Both the term rewriting system and Z3 were applied to these questions and the results are shown in Section 5.

5 Results and Discussion

A series of experiments were performed to evaluate the performance of our TRS using Z3 as a control. The results of these experiments are described in detail in Sections 5.1, 5.2 and 5.3. A summary of all of the experimental results can be found in Table 5. If the marking method failed to assign a grade or assigned a student grade different to the ground truth grade, this was counted as a fail.

All experiments were run on a local machine with an Intel Core-i9-13900K CPU (3-5.8GHz), 64GB of DDR5 (4800MT/s) and an NVIDIA GeForce RRTX 4060Ti GPU with 16GB of dedicated GPU memory.

5.1 Initial Z3 experiments

One of the first problems when applying Z3 is to find an appropriate trigonometric axiom set T that allows Z3 to understand the trigonometric functions but does not quickly lead to time-out or memory limits. By default we use trigono-

Table 5: Summary of experimental results for Questions 25 and 26.

Method	Q25		Q26	
	CPU Time (seconds)	Number of fails	CPU Time (seconds)	Number of fails
Z3 + trig. axioms[a]	550	18	5000	355
Z3 + custom sqrt tactic[b]	**16**	**0**	**40**	5
TRS[c]	1140	**0**	612	1
TRS + T' [d]	1210	**0**	870	**0**

[a] Fails if student equation contains trigonometric functions or square root symbols. In most of the observed failure cases returns timeout error.
[b] Fails on non-linear combinations of trigonometric expressions and cases requiring angle-addition formulas. In four of the five observed failure cases returns `unknown` so the user is aware of the fail.
[c] There was a failure case because the system was missing trigonometric identities, resulting in a false negative.
[d] There were no observed failures after the extra T' axioms were added.

metric axioms that correspond to our rewrite rules:

$$T = \begin{cases} \sin(-x) & = -\sin(x) \\ \cos(-x) & = \cos(x) \\ \sin^2(x) & = 1-\cos^2(x) \\ \sin(x_1+x_2) & = \sin(x_1)\cos(x_2)+\cos(x_1)\sin(x_2) \\ \cos(x_1+x_2) & = \cos(x_1)\cos(x_2)-\sin(x_1)\sin(x_2) \\ \cos((n+2)x) & = 2\cos(x)\cos((n+1)x)-\cos(nx) \\ \sin((n+2)x) & = 2\cos(x)\sin((n+1)x)-\sin(nx) \end{cases}$$

These rules were used to mark the student responses to Question 25. Z3 was able to process the 1526 examples in approximately 10 minutes, successfully classifying all but 18 results. In these cases Z3 was not able to return sat or unsat within a 100 second timeout. Further investigation shows that there were two distinct failure cases.

The first type of failure case occurred when students included trigonometric functions in their responses, for example $m_1v_0 = m_1v_1\cos(\theta) + m_2v_2\cos(\phi)$. In these 14 cases the reason Z3 gives for its unknown result is 'timeout'. The solver statistics for each of these problems show that the universal quantifiers in the trigonometric rules caused the number of quantified instantiations, added equations and clauses to approach the millions. In these cases the Z3 memory usage quickly grew to tens of gigabytes before hitting hard resource limits.

The second type of failure case occurred when square root symbols or fractional powers were included in student responses. For example:

$$v_2 = (((m_1v_0^2) - (m_1v_1^2))/m_2)^{1/2}$$

In these cases, Z3's reason for the unknown result was that the SMT tactic was incomplete, and Z3 failed quickly (less than 0.01 seconds) using less than 20MB of memory. Looking at the solver statistics from these cases shows that Z3 attempts to use Gröbner Bases and other Non-Linear Arithmetic (NLA) tools. These tools fail to reduce the problem to a state where Z3 is able to prove unsatisfiability or find a satisfying example.

The results were similar for Question 26; Z3 gave a timeout error for any expression which contained a trigonometric expression. This resulted in 355 fails. For Z3 to progress it is clear that a simpler set of trigonometric axioms would need to be provided. We note that in each of these examples Z3 fails gracefully, providing an unknown result to alert the user.

5.2 Further Z3 Experiments

The universally quantified T axioms for the uninterpreted functions sin and cos force Z3 to solve a very difficult problem. We performed further experiments where we removed some of the axioms from T to try to improve its grading performance.

Removing these axioms meant that Z3 would not be able to show the equivalence of some expressions such as $\sin(2\theta)$ and $2\sin(\theta)\cos(\theta)$. However, if most student answers do not require operations to be applied to the arguments of the trig functions then Z3 would be able to correctly grade the responses.

First we consider applying a reduced set of axioms to Question 25. When the final four of the original seven T axioms are removed, timeout errors no longer occur. However the $\sin^2(x) = 1-\cos^2(x)$ axiom causes the system to fail quickly when student responses contain trig functions. In these cases Z3 recognizes that these cases contain non-linear arithmetic and therefore its solvers are incomplete. Removing the third axiom fixes this problem and Z3 was quickly able to provide a sat response for the Question 25 examples that contained trigonometry. Note that such (sat) counter examples become increasingly unreliable as axioms are removed.

Radical Elimination: After removing the last five axioms, only the four examples which contained square roots caused problems. In these cases, the NLA solver quickly decides that the system is incomplete and gives up. To prevent this, we implement a radical elimination procedure as a pre-processing step. Each square root is replaced with a new variable and two additional constraints, thus converting the problem to be polynomial which can be handed to the NLA solver. For example, consider this student response:

$$E_0 = m_1\cos(\theta) + (m_2 - m_1)^{1/2}$$

In this case $(m_2-m_1)^{1/2}$ is identified as a radical and the system creates a new variable to represent it, r_0. Then we remove the original equation from the solver and replace it with the following three: $r_0 \geq 0$, $r_0 \times r_0 = (m_2-m_1)$, $E_0 = m_1\cos(\theta) + r_0$.

Once implemented, this allowed Z3 to correctly grade all responses in Question 25. This same approach was applied to Question 26 and resulted in only five fails. The first was that no points were awarded for: $m_1v_0 = \sin(\pi/2 -$

$\phi)m_2v_2 + \cos(\theta)m_1v_1$. This fail occurred because the angle addition axioms were removed. Note that the following response would have resulted in a similar fail: $m_1v_1 = m_1v_1\cos(\theta) + m_2v_2\sin(\pi/2-\phi)$, except this response was incorrect as it contains a v_1 in the place of v_0.

The other four fails were for expressions such as

$$v_2 = (m_2(v_0\cos(\theta))^2 + (v_0\sin(\theta))^2)^{1/2}$$

This example contains non-linear combinations of uninterpreted functions, a type of problem where Z3 is incomplete. In these cases the solver statistics show that only 20MB of memory was used and Z3 makes approximately 5 quantifier instantiations with a number of calls to NLA components. This indicates that in these instances Z3 is not 'blowing-up' and exhausting resources, but after creating a few quantifier instantiations stopped because none of them helped progress the proof. This is possibly because each new instantiation adds a term that contains an uninterpreted function (trig function) which are treated by the solver as fresh variables.

Since very few properties of the trigonometric functions were given to Z3, it is unable to solve non-linear combinations of these functions. Overall the reduced axiom set allowed Z3 to quickly find unsat, but as soon as non-identical comparisons of trig functions are required, it lacks the axioms to resolve these instances.

5.3 Our TRS Results

Using the method described in Section 4.1, our rewrite system was able to correctly grade all 1526 examples for Question 25 in a total CPU time of approximately 1140 seconds. It took 612 seconds for our system to grade Question 26, with one fail occurring. The fail response contained this equation:

$$m_1v_0 = \sin(\pi/2-\phi)m_2v_2 + \cos(\theta)m_1v_1$$

This equation matches the x-momentum equation. Our rewrite system contains the rule:

$$\sin(X_1+X_2) \to \sin(X_1)\cos(X_2) + \cos(X_1)\sin(X_2).$$

Which should reduce $\sin(\pi/2-\phi)$ to $\sin(\pi/2)\cos(\phi) - \cos(\pi/2)\sin(\phi)$. However to make the final step $\sin(\pi/2)\cos(\phi) - \cos(\pi/2)\sin(\phi) \to \cos(\phi)$ would require our system to know that $\sin(\pi/2) = 1$ and $\cos(\pi/2) = 0$. To solve this issue, the following axioms, T' were added:

$$T' = \begin{cases} \sin(\pi) \to 0, & \sin(\pi/2) \to 1 \\ \cos(\pi) \to 1, & \cos(\pi/2) \to 0 \end{cases}$$

Adding these extra trigonometric axioms increased the approximate CPU time to 870 seconds but allowed our TRS to correctly grade all student responses.

One advantage of our TRS compared to SMT solvers is that adding additional unused axioms only has a minimal impact on the run time. We reran the Question 25 set with the additional four T' axioms, this increased the CPU run-time from 1140 seconds to 1210, an approximately 6% increase. This is a stark contrast to Z3 where adding the additional axioms caused the system to timeout and fail.

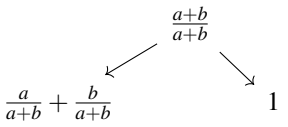
6 Conclusions and Future Work

In this paper, we presented our approach for supporting marking physics exams by combining LLMs, computer algebra systems, and a custom term rewriting system.

Our TRS is able to successfully normalize student equations which contain addition, multiplication, exponentiation and trigonometric functions. The TRS approach scales well when redundant rules are added, with minor penalty in CPU time for normalization. In contrast, the SMT solver performance degraded when we added redundant axioms.

Some limitations and ideas for future work. Currently the LLM stage of the pipeline is only able to achieve 73% accuracy, even using techniques which provide the model with feedback. To improve, larger LLMs could be used or models could be fine-tuned to improve performance.

Our TRS was sufficiently terminating, confluent and complete for grading two 2023 Australian Physics Olympiad problems. However, this is not enough for all physics problems. A natural example for expected convergence where our system fails to join is on the right.



One significant drawback of the TRS is that in general there is no well defined canonical form for arbitrary equations. This means that the TRS is not able to prove that a student's answer is incorrect and leaves the possibility of false negatives. A clear example of this is the response that was incorrectly graded in Question 26, requiring the T' axioms to be added. In future we will specify the exact functional forms of equations which our TRS can reduce to a unique canonical form, this will define the situations where the system will fail.

Our TRS could be further improved by adding axioms for complicated expressions involving exponentiation by variables, logarithms and inverse trigonometric functions. The scope of problems which can be graded could also be widened, beyond simply checking for algebraic equivalence, to verify student proofs.

Currently our system requires SymPy to first solve the equation for a specific parameter before applying the TRS. This means our system is only able to determine the equivalence of two equations if Sympy is able to solve the equation for at least one variable. This means that the rewrite system can only be as good as the chosen algebra system's solving capabilities.

Future work will use a confluence analysis tool to examine critical pairs and add rules to improve confluence. Finally, our system is implemented in Python, a slow interpreted language, and in a non-optimized way. Performance was sufficient for our purpose. Re-implementation in a faster compiled language and/or using efficient data structures, for example term indexing.

Acknowledgments The authors would like to thank Australian Science Innovations for access to data from the 2023 Australian Physics Olympiad. This research was supported by a scholarship from CSIRO's Data61. The ethical aspects

of this research have been approved by the ANU Human Research Ethics Committee (Protocol 2023/1362).

References

Abdin, M. I., and Zhang, Y. 2024. Phi-4 Technical Report. Microsoft Research. https://www.microsoft.com/en-us/research/publication/phi-4-technical-report/

Avenhaus, J., and Becker, K. 1994. Operational specifications with built-ins. In *STACS 94*. 263–274. Springer.

Baader, F., and Nipkow, T. 1998. *Term Rewriting and All That*. Cambridge University Press.

Baumgartner, P., and Waldmann, U. 2013. Hierarchic Superposition with Weak Abstraction. In Bonacina, M. P., ed., CADE-24, 39–57. Springer.

Bjørner, N., and Nachmanson, L. 2024. Arithmetic Solving in Z3. In Gurfinkel, A., and Ganesh, V., eds., *CAV*, volume 14681. Springer. 26–41.

Chen, Z., and Wan, T. 2025. Grading explanations of problem-solving process and generating feedback using large language models at human-level accuracy. *Physical Review Physics Education Research* 21(1):010126.

Department of Industry Science and Resources. 2024. Australia's AI Ethics Principles.

Dershowitz, N., and Jouannaud, J.-P. 1990. Rewrite Systems. In *Formal Models and Semantics*, Handbook of Theoretical Computer Science. Elsevier. 243–320.

Harrison, J. 2009. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *ICML*, 22895–22907. PMLR.

Kaplan, S., and Choppy, C. 1989. Abstract rewriting with concrete operators. In *Rewriting Techniques and Applications*, volume 355. Springer. 178–186.

Kop, C., and Nishida, N. 2013. Term Rewriting with Logical Constraints. In *FroCoS*, 343–358. Springer.

Kortemeyer, G.; Nöhl, J.; and Onishchuk, D. 2024. Grading assistance for a handwritten thermodynamics exam using artificial intelligence: An exploratory study. *Physical Review Physics Education Research* 20(2):020144.

Kortemeyer, G. 2023. Toward AI grading of student problem solutions in introductory physics: A feasibility study. *Physical Review Physics Education Research* 19(2):020163.

McGinness, L., and Baumgartner, P. 2025a. Can Large Language Models Correctly Interpret Equations with Errors? arXiv:2505.10966 [physics.ed-ph]

Meurer, A.; Smith, C.; Paprocki, M.; and Scopatz, A. 2024. SymPy: Symbolic computing in Python. https://www.sympy.org/en/index.html.

Mok, R.; Akhtar, F.; Clare, L.; Li, C.; Ida, J.; Ross, L.; and Campanelli, M. 2024. Using AI Large Language Models for Grading in Education: A Hands-On Test for Physics. arXiv:2411.13685 [physics.ed-ph]

Ogg, M. 2024. Brisbane AI edtech Edexia accepted into Y Combinator. http://www.businessnewsaustralia.com.html.

Ramesh, D., and Sanampudi, S. K. 2022. An automated essay scoring systems: A systematic literature review. *Artificial Intelligence Review* 55(3):2495–2527.

Shostak, R. E. 1984. Deciding Combinations of Theories. *J. ACM* 31(1):1–12.

Weegar, R., and Idestam-Almquist, P. 2024. Reducing Workload in Short Answer Grading Using Machine Learning. *International Journal of Artificial Intelligence in Education* 34(2):247–273.

Windle, J.; Morrison, A.; Sellar, S.; Squires, R.; Kennedy, J.; and Murray, C. 2022. *Teachers at Breaking Point*. University of South Australia.

Yamada, A.; Kusakari, K.; and Sakabe, T. 2015. A unified ordering for termination proving. *Science of Computer Programming* 111:110–134.