Bridging the gap: A complete axiomatization of Gregorian date arithmetic and scheduling logic for automated theorem provers

Adam Pease[1], Pantelimon Stănică[2]

[1]Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943, USA
[2]Department of Applied Mathematics, Naval Postgraduate School, Monterey, CA 93943, USA

Abstract

Reasoning about calendar time combines integer arithmetic, irregular calendar structure, and cyclic weekday constraints. Although each component admits a straightforward formalisation, their interaction poses significant challenges for automated theorem proving (ATP) in first-order logic with Uninterpreted Functions and Linear Integer Arithmetic (UFLIA). We present a stratified first-order axiomatization of Gregorian calendar arithmetic organised into three strata: structural calendar laws ($\mathcal{T}_1$), arithmetic normalization predicates ($\mathcal{T}_2$), and a finite set of ground *semantic anchor* axioms ($\mathcal{T}_3$), where $\mathcal{T}_3$ is a conservative extension of $\mathcal{T}_2$ for non-ground formulas. We prove determinacy of the normalization predicates, conservativity of the anchor stratum, and a precise compression property: whenever a benchmark subgoal is itself available as an anchor, the corresponding recursive normalization derivation can be replaced by a bounded number of inference steps independent of the offset magnitude. We validate the approach on a heterogeneous 200-conjecture benchmark suite: Vampire achieves 100% proof coverage under a three-tier implementation of TPTP files, and Z3 independently validates all 200 conjectures.

1. Introduction

The integration of arithmetic reasoning within automated theorem provers has remained a persistent challenge. While modern provers such as Vampire [6], E [13], and SPASS [17] excel at pure first-order reasoning, combining symbolic reasoning with integer arithmetic over irregular domains stretches their capabilities. Calendar arithmetic is a canonical example: the Gregorian calendar mixes linear arithmetic with conditional leap-day rules, month-length irregularities, and a cyclic weekday constraint of period seven. Individually, each feature is tractable; together they create proof-search pathologies that block off-the-shelf provers. The root cause, as we will show, is not the calendar domain itself but the conflation of three semantically distinct phenomena in a flat, undifferentiated axiom set.

Motivating example. Consider: *"What is the date of the second Thursday of November 2025?"* Answering it requires (i) computing that November 1, 2025 is a Saturday, (ii) finding the offset to the first Thursday (five days, landing on November 6), and (iii) adding seven more days to obtain November 13. The answer is a *calendar date*; the query is non-trivial precisely because November 1 can fall on any weekday depending on the year. Formally this is the conjecture

$$\exists D.\ \mathsf{nth\_weekday\_date}(2,\ \mathsf{thursday},\ 11,\ 2025,\ D),$$

which requires coordinating date arithmetic, Zeller's congruence for weekday computation, and month-boundary reasoning simultaneously. This kind of multi-aspect interaction is characteristic of the entire benchmark suite.

The 10th Workshop on Practical Aspects of Automated Reasoning, as part of the Federated Logic Conference (FLoC) 2026, Lisbon, Portugal, July 25, 2026
adam.pease@nps.edu (A. Pease); pstanica@nps.edu (P. Stănică)
0000-0001-9772-1266 (A. Pease); 0000-0002-8622-7120 (P. Stănică)

Three sources of difficulty. We identify three interacting phenomena that, when combined without discipline in a single axiom set, lead to pathological proof-search behaviour.
(1) Unbounded arithmetic traversal. A backward query such as $\exists D.\ \mathsf{calc\_date}(-1460, 1, 2028, D)$ asks for the date 1460 days before January 1, 2028. The recursive normalization predicate works month by month: each step crosses one month boundary, so the derivation depth is proportional to the number of months traversed (roughly $|O|/30$ steps for offset $|O|$). For large offsets this chain is prohibitively long for saturation-based provers. The problem is entirely independent of the calendar semantics: it is a consequence of how the arithmetic is *presented* to the prover.
(2) Semantic discontinuities. Month lengths vary (28–31 days), and leap days appear conditionally under the Gregorian 4-100-400 rule. Encoded as arithmetic case splits, these interact destructively with offset reasoning, multiplying proof branches. The century exceptions are particularly delicate: 2100 is *not* a leap year despite $4 \mid 2100$, and no simple modular test can distinguish it from 2000.
(3) Cyclic constraints. Weekday reasoning operates on a finite cycle of period seven independent of the linear structure of time. Entangling this with arithmetic traversal forces the prover to reason simultaneously about linear growth and modular arithmetic, leading to combinatorial explosion.

The three phenomena compound: a mixed scheduling conjecture requires all three to be handled coherently in a single proof. Reasoning in UFLIA is inherently hard [4]; the calendar layered on top amplifies each source of difficulty.

Prior approaches and their limitations. Temporal logics [11, 1] address ordering and duration but abstract away from concrete arithmetic. SMT solvers [2, 3] provide efficient calendar support via specialised theory solvers, but typically target quantifier-free fragments; completeness under quantifiers and uninterpreted functions is not guaranteed [4], and proofs in the FOL sense are generally not produced. Procedural libraries such as Joda-Time [5] and the Java 8 Time API handle calendar computations correctly but provide no formal proofs and no path to integration with logical reasoning systems such as SUMO [8, 9], which prior to this work lacked the axioms to compute that adding 400 minutes to a timestamp may advance the calendar day. The TimeML specification [12] captures temporal vocabulary but lacks the axiomatization required for theorem proving.

Our approach: stratification. The key insight is that the three sources of difficulty play *distinct semantic roles*. Structural calendar facts (month lengths, leap rules) are independent of any particular reasoning task. Arithmetic normalization connects integer offsets to calendar positions through recursive definitions. Semantic anchors are simply pre-computed consequences that collapse long but uninteresting traversals into single inference steps. Separating these roles into three strata eliminates the interference that makes monolithic axiomatizations pathological.

We present a stratified TPTP Typed First-order form with Arithmetic (TFA) axiomatization of Gregorian calendar arithmetic organised into three *strata*, where each stratum is an independent module and $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$ denotes their union (with $\mathcal{T}_1 \subsetneq \mathcal{T}_1 \cup \mathcal{T}_2 \subsetneq \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$ as axiom sets):

- $\mathcal{T}_1$ (*structural*): universal axioms for month lengths (all four cases including both 30-day months and February), leap-year rules, and weekday cyclicity via Zeller's congruence.
- $\mathcal{T}_2$ (*normalization*): axioms relating integer offsets to canonical date/time representations via relational normalization predicates (calc_date, normalize_time).
- $\mathcal{T}_3$ (*semantic anchors*): a finite, benchmark-driven set of ground consequences of $\mathcal{T}_1 \cup \mathcal{T}_2$, acting as admissible lemmas that collapse deep normalization paths without altering the universal theory.

We establish determinacy and canonical bounds for the normalization predicates; soundness preservation across tiers; conservativity of $\mathcal{T}_3$ (Theorem 4); and a formal bounded-proof-depth result (Proposition 1) that precisely characterises *why* the stratification works. Experimentally, Vampire achieves 100% proof coverage over a 200-conjecture benchmark suite, with each tier solving exactly the conjectures predicted by the theoretical analysis, and Z3 independently validates all results. The complete axiomatization, benchmarks, and Python validation scripts are at [10, 14].

Paper organisation. Section 2 presents the formal language and axiomatization. Section 3 states and proves the theoretical properties. Section 4 describes the benchmark suite and explains why the queries are challenging for ATPs. Section 5 reports experimental results with both Vampire and Z3. Section 6 draws out implications. Appendices contain full TPTP listings, the anchor catalogue, benchmark samples, and the FAST accelerator analysis.

2. Formal Language and Stratified axiomatization

2.1. Many-Sorted Signature

We work in many-sorted FOL with interpreted integer arithmetic. The base signature provides $\mathbb{Z}$ with operations $(+, -, \times, \mathsf{div}_e, \mathsf{rem}_e)$ and comparisons $(<, \leq, =, \neq)$, where div_e and rem_e are Euclidean quotient and remainder. Three uninterpreted sorts extend the base: τ_{date}, τ_{time}, and τ_{dname}. Constructor functions populate the calendar sorts: $\mathsf{ymd} : \mathbb{Z}^3 \to \tau_{date}$ constructs dates from year–month–day triples; $\mathsf{dt} : \mathbb{Z}^5 \to \tau_{time}$ constructs datetime values; and constants $\mathsf{sunday}, \ldots, \mathsf{saturday} : \tau_{dname}$ enumerate weekdays. Terms $\mathsf{ymd}(Y, M, D)$ are *not* required to be syntactically normalised; normalization is enforced semantically by the axioms in $\mathcal{T}_2$.

Definition 1. The core predicates mediating arithmetic–calendar interaction are:

$$\mathsf{is\_leap\_year}(Y) : \mathbb{Z} \to \mathbb{B}$$
$$\mathsf{is\_days\_in\_month}(N, M, Y) : \mathbb{Z}^3 \to \mathbb{B} \text{ [month } M \text{ in year } Y \text{ has } N \text{ days]}$$
$$\mathsf{calc\_date}(O, M, Y, D) : \mathbb{Z}^3 \times \tau_{date} \to \mathbb{B} \text{ [} O \text{ days from } (M, Y) \text{ is date } D; O \in \mathbb{Z}\text{]}$$
$$\mathsf{normalize\_time}(H, Min, DD, EH, EM, EDD) : \mathbb{Z}^6 \to \mathbb{B}$$
$$\mathsf{weekday}(D, W) : \tau_{date} \times \tau_{dname} \to \mathbb{B}$$
$$\mathsf{nth\_weekday\_date}(N, W, M, Y, D) : \mathbb{Z} \times \tau_{dname} \times \mathbb{Z}^2 \times \tau_{date} \to \mathbb{B}$$

No arithmetic operations are defined over τ_{date} or τ_{time} directly; all arithmetic–calendar interaction is mediated by these predicates.

Relational vs. functional syntax. Superposition-based provers perform both forward and backward reasoning regardless of whether predicates such as calc_date are written relationally or functionally, so the choice does not intrinsically enable reasoning directions that the alternative would not. We use relational syntax for two reasons. First, it keeps the axioms purely declarative, embedding no algorithmic control flow. Second, when a conjecture existentially quantifies over an offset O (backward queries), the relational form syntactically exposes the constraint to the prover without requiring inversion of a function. The cost is that the prover must derive functionality of the relation from the axioms rather than knowing it a priori; Section 5 shows this overhead is acceptable in practice, and the determinacy results of Section 3 establish functionality formally.

2.2. Structural Stratum $\mathcal{T}_1$

The structural stratum characterises the static Gregorian calendar through universal axioms that define the intended models independently of any particular reasoning task. These axioms alone suffice for conjectures about calendar structure without arithmetic traversal.

The leap-year rule encodes the standard Gregorian 4-100-400 exception in a single biconditional, axiomatizing both leap and non-leap cases and making century exceptions explicit: years divisible by 4 are leap years, except century years, except again those divisible by 400.

```
tff(is_leap_year_def, axiom, ![Y:$int]:
  (is_leap_year(Y) ⇔
    (($remainder_e(Y,4) = 0 & $remainder_e(Y,100) ≠ 0) |
     $remainder_e(Y,400) = 0))).
```

Listing 1: Leap year axiom

Month lengths are covered by four axioms that partition the twelve months with no gaps or overlaps, ensuring the intended models are uniquely determined:

```
tff(dim_feb_leap,   axiom, ![Y:$int]:
  (is_leap_year(Y) ⇒ is_days_in_month(29,2,Y))).
tff(dim_feb_common, axiom, ![Y:$int]:
  (~is_leap_year(Y) ⇒ is_days_in_month(28,2,Y))).
tff(dim_31, axiom, ![Y:$int,M:$int]:
  ((M=1|M=3|M=5|M=7|M=8|M=10|M=12) ⇒ is_days_in_month(31,M,Y))).
tff(dim_30, axiom, ![Y:$int,M:$int]:
  ((M=4|M=6|M=9|M=11) ⇒ is_days_in_month(30,M,Y))).
```

Listing 2: Days-in-month axioms (complete)

Weekday structure is axiomatized via *Zeller's congruence* [18]. For a date (Y, M, D), first adjust months so that January and February of year Y are treated as months 13 and 14 of year $Y-1$. Let Y', M' be the adjusted year and month, $K = Y' \bmod 100$, $J = \lfloor Y'/100 \rfloor$. Then the weekday index is

$$h = \left(D + \left\lfloor \frac{13(M'+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J \right) \bmod 7,$$

with $h = 0$ encoding Saturday through $h = 6$ encoding Friday. This formula is directly encoded in TPTP; the full listing is in Appendix A.

2.3. Normalization Stratum $\mathcal{T}_2$

While $\mathcal{T}_1$ captures what the calendar looks like, $\mathcal{T}_2$ captures how arithmetic quantities *map onto* calendar positions. The two normalization predicates—one for time, one for dates—form the core of the theory's arithmetic reasoning capability.

Time normalization.

normalize_time$(H, Min, DD, EH, EM, EDD)$ relates an initial configuration (hours H, minutes Min, day offset DD) to a canonical form (EH, EM, EDD) satisfying $0 \leq EM < 60$ and $0 \leq EH < 24$. It is defined by two nested Euclidean divisions on the total minute count $60H + Min$:

```
tff(normalize_time, axiom,
  ![H:$int, M:$int, DD:$int, EH:$int, EM:$int, EDD:$int]:
  (normalize_time(H,M,DD,EH,EM,EDD) ⇔
    ?[QH:$int, RM:$int]:
      ($remainder_e($sum(M,$product(60,H)),60) = RM &
       $quotient_e($sum(M,$product(60,H)),60) = QH &
       0 ≤ RM & RM < 60 &   % explicit Euclidean remainder range for Vampire
       EH = $remainder_e(QH,24) &
       EDD = $sum(DD,$quotient_e(QH,24)) &
       EM = RM))).
```

Listing 3: Time normalization axiom

The constraints $0 \leq RM < 60$ are logically redundant (they follow from the definition of Euclidean remainder) but are included explicitly because Vampire's arithmetic module requires them to be stated as premises in order to trigger the appropriate simplification rules; without them, proof search can stall on the remainder subgoal.

Example. normalize_time$(23,\ 90,\ 0,\ EH,\ EM,\ EDD)$: total minutes $= 60 \cdot 23 + 90 = 1470$. The first division gives $QH = 24$, $RM = 30$; the second gives $EH = 0$, day increment $= 1$. So 23 hours 90 minutes with day-offset 0 normalises to 00:30 of the *next* day ($EDD = 1$). This example illustrates the day-overflow behaviour that makes time normalization non-trivial: minutes overflow into hours, which overflow into days, and then into the date computation.

Date normalization.

The predicate calc_date(D, M, Y, R) is intentionally specified in the same form as in the implementation. Its first argument is *not* an offset from day 1. Rather, D is the day index to be interpreted relative to the

month M and year Y: values with $1 \leq D \leq Limit$ denote days inside the current month, values with $D > Limit$ overflow into later months, and values with $D \leq 0$ retreat into earlier months. This is the convention used in the actual TPTP files, and stating it this way avoids mismatch between the paper and implementation.

Under this convention, the exact recursive behaviour is simple. If $1 \leq D \leq Limit$, the result is the in-month date $\mathsf{ymd}(Y, M, D)$. If $D > Limit$, we subtract the current month length and recurse on the next month (or on January of the next year when $M = 12$). If $D \leq 0$, we add the length of the previous month and recurse on that month (or on December of the previous year when $M = 1$). The key clauses, written extracted from our TPTP implementation, are:

```
% Base case: day index D lies inside the current month
tff(rule_base, axiom,
  ![D:$int, M:$int, Y:$int, Limit:$int]:
  ((is_days_in_month(Limit, M, Y) & $greater(D, 0) & $lesseq(D, Limit))
   => calc_date(D, M, Y, ymd(Y, M, D)))).

% Forward recursion inside the year
tff(rule_fwd_std, axiom,
  ![D:$int, M:$int, Y:$int, Limit:$int, Res:date]:
  ((is_days_in_month(Limit, M, Y) & $greater(D, Limit) & M != 12)
   => (calc_date($difference(D, Limit), $sum(M, 1), Y, Res)
       => calc_date(D, M, Y, Res)))).

% Forward recursion across the year boundary
tff(rule_fwd_dec, axiom,
  ![D:$int, M:$int, Y:$int, Limit:$int, Res:date]:
  ((is_days_in_month(Limit, M, Y) & $greater(D, Limit) & M = 12)
   => (calc_date($difference(D, Limit), 1, $sum(Y, 1), Res)
       => calc_date(D, M, Y, Res)))).

% Backward recursion to the previous month
tff(rule_back_std, axiom,
  ![D:$int, M:$int, Y:$int, Limit:$int, Res:date]:
  (($lesseq(D, 0) & M != 1)
   => ((is_days_in_month(Limit, $difference(M,1), Y) &
         calc_date($sum(D, Limit), $difference(M, 1), Y, Res))
       => calc_date(D, M, Y, Res)))).

% Backward recursion from January into the previous year
tff(rule_back_jan, axiom,
  ![D:$int, M:$int, Y:$int, Limit:$int, Res:date]:
  (($lesseq(D, 0) & M = 1)
   => ((is_days_in_month(Limit, 12, $difference(Y,1)) &
         calc_date($sum(D, Limit), 12, $difference(Y, 1), Res))
       => calc_date(D, M, Y, Res)))).
```

Listing 4: Representative calc_date clauses from the implementation

This presentation is deliberately faithful to the actual TPTP implementation. In particular, there is no separate "within-month backward" clause in the implementation: all cases with $D \leq 0$ move to the previous month and recurse. That is exactly the behaviour used in the benchmark files and the experiments.

2.4. Semantic Anchor Stratum $\mathcal{T}_3$

The structural and normalization strata together constitute a complete and sound axiomatization of Gregorian calendar arithmetic. Yet, as the theoretical results below will show, proofs from $\mathcal{T}_2$ alone can require derivations of depth $\Theta(|O|)$, making large-offset queries practically intractable. The semantic anchor stratum addresses this without altering the theory's semantics.

Semantic anchors are ground sentences, each a consequence of $\mathcal{T}_2$, that provide explicit endpoints for otherwise long normalization chains. Typical anchors assert the weekday of a reference date, or the result of a large offset computation:

```
% Reference weekday: January 1, 2000 was a Saturday
tff(anchor_2000_jan_sat, axiom, weekday(ymd(2000,1,1), saturday)).

```

```
% Century-scale calc_date shortcut (provable from T2 in 412ms)
tff(anchor_century, axiom, calc_date(36524, 1, 2000, ymd(2099,12,31))).

% Quad-year backward shortcut (provable from T2 in 396ms)
tff(anchor_quad_back, axiom, calc_date(-1461, 3, 2028, ymd(2024, 3, 1))).
```

Listing 5: Representative semantic anchor axioms

Anchor selection and verification. Each anchor is first proved as a theorem of $\mathcal{T}_2$ (by running both Vampire and Z3 on the candidate conjecture against $\mathcal{T}_2$), ensuring soundness before inclusion. The selection criterion is coverage of the normalization paths induced by the benchmark suite: for each backward or mixed conjecture whose $\mathcal{T}_2$ proof would unfold calc_date more than a fixed threshold, we identify the ground endpoint of the longest normalization chain and add it as an anchor. This yields 47 ground anchors for the 200-conjecture suite. Since anchors are just verified consequences of $\mathcal{T}_2$, the selection process is fully deterministic; Appendix B lists all 47 with their derivation times.

3. Theoretical Properties

The theoretical results flow in a natural order: we first establish that the normalization predicates are well-behaved (functionality/determinacy, well-formedness of results), then that the stratification is sound, then that anchors are conservative, and finally that anchor coverage implies bounded proof depth. Together, these results give a complete formal account of *why* the three-tier approach works.

Theorem 1. *For all integers H, Min, DD, there exists exactly one triple (EH, EM, EDD) with* $\mathsf{normalize\_time}(H, Min, DD, EH, EM, EDD)$.

Proof. Let $T = 60H + Min$. By totality and uniqueness of Euclidean division, there is a unique (Q_0, R_0) with $T = 60Q_0 + R_0$, $0 \leq R_0 < 60$, and a unique (Q_1, R_1) with $Q_0 = 24Q_1 + R_1$, $0 \leq R_1 < 24$. The defining axiom forces $EM = R_0$, $EH = R_1$, $EDD = DD + Q_1$, uniquely. □ □

Corollary 1. *If* $\mathsf{normalize\_time}(H, Min, DD, EH, EM, EDD)$ *holds, then* $0 \leq EM < 60$ *and* $0 \leq EH < 24$.

Proof. Immediate from the Euclidean remainder properties in the defining axiom. □ □

The significance of Corollary 1 extends beyond tidiness: because the canonical bounds are consequences of the defining Euclidean-division axioms, constraints on normalised components can propagate backward to constraints on the original arithmetic quantities. This backward propagation is essential for existentially quantified backward queries, where the prover must infer the original offset from properties of the result.

Theorem 2. *For any integers O, M, Y, there is at most one date D such that* $\mathsf{calc\_date}(O, M, Y, D)$ *holds in any intended model.*

Proof. We use strong induction on the lexicographic measure $(sc, |O|)$, where sc is 1 if the sign of O changes in the recursive call and 0 otherwise (with $1 > 0$), and $|O|$ is the absolute value of the day index. This measure is well-founded because sc can increase at most once (after a sign change the recursive call has the same sign as the result), and $|O|$ strictly decreases once the sign is stable.

Base ($0 \leq O \leq Limit$, where $Limit$ is the number of days in month M of year Y): rule_base from Listing 1.4 fixes $D = \mathsf{ymd}(Y, M, O)$ uniquely; no recursive call is made.

Step ($O > Limit$ or $O \leq 0$): the Days-in-month axioms (Listing 1.2) partition the twelve months with no gaps or overlaps, so $Limit$ is uniquely determined for any valid (M, Y). Given $Limit$, the full case analysis of Listing 1.4 is mutually exclusive:

- *Forward, within year* ($O > Limit$, $M \neq 12$): rule_fwd_std recurses on $(O - Limit,\ M + 1,\ Y)$. Here $O - Limit > 0$, so the sign is unchanged ($sc = 0$), and $|O - Limit| < |O|$; the measure strictly decreases.

- *Forward, year boundary* ($O > Limit$, $M = 12$): rule_fwd_dec recurses on $(O - Limit,\ 1,\ Y + 1)$; same argument.
- *Backward, within year* ($O \leq 0$, $M \neq 1$): rule_back_std recurses on $(O + Limit',\ M - 1,\ Y)$, where $Limit'$ is the length of month $M - 1$. If $O + Limit' > 0$ the sign changes (sc increases from 0 to 1), so the first component of the measure strictly decreases regardless of $|O + Limit'|$ vs. $|O|$. If $O + Limit' \leq 0$ the sign is unchanged and $|O + Limit'| < |O|$ (since $Limit' \geq 28 > 0$), so the second component strictly decreases.
- *Backward, year boundary* ($O \leq 0$, $M = 1$): rule_back_jan recurses on $(O + Limit_{12},\ 12,\ Y - 1)$; same argument as the previous case.

In each case the induction hypothesis applies and gives a unique result D. □ □

Remark 1. Theorem 2 shows determinacy (*at most one*), not total existence: the axioms do not explicitly cover out-of-range months. In all intended models with $1 \leq M \leq 12$ and $Y \in \mathbb{Z}$, existence follows from the completeness of the month-length axioms.

With the normalization predicates established as well-behaved, we turn to the stratification structure itself.

Theorem 3. *If φ is provable from $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$, then φ is true in every intended Gregorian calendar model.*

Proof. $\mathcal{T}_1$ encodes standard Gregorian rules by construction. $\mathcal{T}_2$ normalization axioms are defined via Euclidean division and $\mathcal{T}_1$, hence valid in all intended models. $\mathcal{T}_3$ anchors are ground consequences of $\mathcal{T}_2$ (verified before inclusion), hence valid in all intended models. Therefore every axiom in $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$ is true in all intended models, and FOL soundness gives the result. □ □

The next result is the pivotal property of $\mathcal{T}_3$: adding anchors does not change what the theory proves about non-ground formulas. The only thing that changes is *how efficiently* those formulas can be proved.

Theorem 4. *Let $\mathcal{A}$ be any finite set of ground sentences with $\mathcal{T}_2 \models A$ for each $A \in \mathcal{A}$. For any non-ground formula φ:*

$$\mathcal{T}_2 \cup \mathcal{A} \models \varphi \implies \mathcal{T}_2 \models \varphi.$$

Proof. Let $\mathfrak{M}$ be any model of $\mathcal{T}_2$. Since $\mathcal{T}_2 \models A$ for every $A \in \mathcal{A}$, we have $\mathfrak{M} \models \mathcal{A}$, hence $\mathfrak{M} \models \mathcal{T}_2 \cup \mathcal{A}$. By hypothesis, $\mathfrak{M} \models \varphi$. Since $\mathfrak{M}$ was arbitrary, $\mathcal{T}_2 \models \varphi$. □ □

This result justifies the description of $\mathcal{T}_3$ as purely *proof-theoretic*: it reshapes derivations without altering the set of universal consequences. We can now formalise precisely when and why that reshaping is effective.

Definition 2. Fix a ground atom $\mathsf{calc\_date}(O, M, Y, D^*)$. Its *normalization chain* is the finite sequence of ground atoms obtained by repeatedly applying the recursive calc_date clauses until the base case $O = 0$ is reached. Analogously, a ground normalize_time atom induces the finite chain produced by its defining Euclidean decompositions.

Proposition 1. *Let A be a ground atom of the form $\mathsf{calc\_date}(O, M, Y, D^*)$ or $\mathsf{normalize\_time}(\bar{t})$ such that $A \in \mathcal{T}_3$. Then any proof of a conjecture that would otherwise need to derive A from $\mathcal{T}_2$ may replace the entire corresponding normalization chain by a single use of the anchor A.*

Proof. By construction, every anchor in $\mathcal{T}_3$ is itself a theorem of $\mathcal{T}_2$. Therefore, whenever a proof from $\mathcal{T}_2$ contains a subderivation whose conclusion is exactly the ground atom A, that subderivation can be cut away and replaced by one application of the premise A from $\mathcal{T}_3$. The replacement preserves correctness because the subderivation and the anchor have the same conclusion, and it shortens the local derivation by exactly the number of recursive normalization steps that would otherwise be needed. □ □

Table 1
Temporal query categories and their primary source of difficulty.

Category	Count	Primary difficulty
Direct date arithmetic	90	Arithmetic traversal + month boundaries
Backward date arithmetic	46	Integer trap (deep normalization paths)
Leap-year / century logic	41	Semantic discontinuities (100/400 rule)
Weekday computation	36	Cyclic modular constraints
Nth-weekday / scheduling	22	Mixed: arithmetic + calendar + weekday
normalization stress tests	32	Proof depth explosion
Total	**200**	

Proposition 1 is the proof-theoretic core needed for the experiments. It deliberately avoids any claim about arbitrary superposition proof depth. What it establishes is narrower and exactly what the data uses: for the recurring benchmark patterns covered by anchors, long ground normalization chains can be collapsed to bounded local derivations. The following corollary shows that a finite benchmark always admits a finite anchor basis of this kind.

Corollary 2. *For any finite benchmark set $\mathcal{B}$ of ground temporal conjectures, there exists a finite set $\mathcal{A}_{\mathcal{B}}$ of ground consequences of $\mathcal{T}_2$ such that every conjecture in $\mathcal{B}$ admits a proof from $\mathcal{T}_2 \cup \mathcal{A}_{\mathcal{B}}$ in which every recurring ground normalization chain is replaced by a bounded number of anchor uses.*

Proof. Each conjecture in $\mathcal{B}$ gives rise to finitely many ground normalization chains, each of finite length. Collect the terminal ground atoms or other recurring intermediate ground atoms along those chains, and include in $\mathcal{A}_{\mathcal{B}}$ those that are consequences of $\mathcal{T}_2$. The resulting set is finite because $\mathcal{B}$ is finite. Applying Proposition 1 to each selected atom yields the stated compression property. □ □

4. Benchmark Suite

4.1. Design Principles and Query Taxonomy

The benchmark suite contains 200 TFA conjectures, designed around the three difficulty sources from Section 1. Three principles governed construction: **semantic representativeness** (each conjecture encodes a question from calendar computation or scheduling, not an artificial stress test); **bidirectional coverage** (backward-style conjectures exercise normalization predicates as relations); and **pathology exposure** (the suite explicitly targets century-edge cases, large offsets, and normalization traps).

Table 1 classifies the 200 conjectures. The categories are not mutually exclusive; counts therefore overlap, and the table records the dominant source of difficulty for each conjecture rather than a partition of the suite.

Representative conjectures from each category are: direct ($\exists D.\ \mathsf{calc\_date}(365, 1, 2024, D)$); backward ($\exists D.\ \mathsf{calc\_date}(-1460, 1, 2028, D)$); leap edge ($\exists D.\ \mathsf{calc\_date}(365, 2, 2100, D)$, where 2100 is not a leap year); weekday ($\exists W.\ \mathsf{weekday}(\mathsf{ymd}(2024, 1, 1), W)$); scheduling ($\exists D.\ \mathsf{nth\_weekday\_date}(3, \mathsf{monday}, 2, 2025, D)$); and stress ($\exists D.\ \mathsf{calc\_date}(32879, 1, 2025, D)$ and $\exists EH, EM, EDD.\ \mathsf{normalize\_time}(24, 0, -1, EH, EM, EDD)$). Appendix C provides one fully-stated TPTP conjecture per category.

4.2. Why These Benchmarks Challenge ATPs

Standard ATP benchmarks in arithmetic involve linear constraints without recursive functions or modular structure. Our benchmarks combine UFLIA (recursive arithmetic predicates alongside uninterpreted constructors ymd, dt) with modular constraints from weekday reasoning. This places them outside the decidable fragments of arithmetic [4].

The *integer trap* is the decisive difficulty for backward and mixed categories: a conjecture $\exists D.\ \mathsf{calc\_date}(O, M, Y, D)$ with negative O requires the prover to commit to specific integer values for the existential witness D rather than leaving them symbolic. Without anchors, this forces the

Table 2
Per-category performance and winning tier distribution (Vampire portfolio). Times in milliseconds; B0/B1/SH = first tier at which a conjecture was solved.

Category	n	Median	P90	Max	B0	B1	SH	Unsolved
Accelerators	9	1810	14031	14031	6	2	1	0
Backward arith.	43	1908	8416	8825	26	5	12	0
Century/leap edge	8	1088	5052	5052	5	3	0	0
Combined queries	3	1885	1887	1887	3	0	0	0
Core arithmetic	76	1170	1977	39360	63	9	4	0
Scalability tests	10	385	397	402	10	0	0	0
Scheduling	21	2027	15140	44490	12	5	4	0
Time normalization	3	1884	1890	1890	3	0	0	0
Weekday	27	39	58	2149	26	0	1	0
Total	**200**	**1170**	**8416**	**44490**	**154**	**24**	**22**	**0**

proof search to realize a long recursive normalization chain. The mechanism we use to force concrete arithmetic evaluation (described in Appendix A) exploits Vampire's arithmetic simplification behaviour and is Vampire-specific; other provers may handle the same phenomenon differently. The normalization stress tests are designed to verify that apparent success cannot be explained by solver heuristics or accidental bounds on the arithmetic: they use offsets spanning 10^0 to 10^6 days, ensuring that any solver still exposed to long explicit normalization chains will time out on at least some instances.

5. Experimental Results

5.1. Setup

We use Vampire 4.9 [6] (saturation-based superposition with AVATAR) and Z3 4.13 [3] (CDCL(T) with arithmetic theory reasoning), neither modified. For Vampire, a fixed three-tier implementation in TPTP invokes the prover first on $\mathcal{T}_1$ (timeout 60 s), then $\mathcal{T}_2$ (60 s), then $\mathcal{T}_3$ (60 s); a conjecture is solved at the first tier where a proof is found. For Z3 we run all conjectures against the full $\mathcal{T}_3$ with a 120 s timeout. We also ran the benchmark portfolio against the Best0 file alone to show benchmark families really require the higher tiers. In the tables below, *Best0*$= \mathcal{T}_1$, *Best1*$= \mathcal{T}_2$, *SAFE_HEAVY*$= \mathcal{T}_3$; *FAST* is the intentionally incomplete accelerator fragment discussed in Section 5.4.

5.2. Main Results: Complete Coverage and Tier Distribution

All 200 benchmark conjectures were successfully discharged by Vampire: $\forall \varphi \in \mathcal{B}, \quad \mathcal{T}_3 \vdash \varphi$. The tier distribution, reported in Table 2, closely matches the theoretical predictions of Section 3.

The distribution is strikingly aligned with the query taxonomy. 154/200 conjectures (77%) are solved by Best0 alone: structural and weekday queries are discharged purely from $\mathcal{T}_1$, confirming that the structural axioms fully capture static calendar semantics. A further 24 conjectures are solved by Best1: these are the forward arithmetic queries where normalization is needed but no deep traversal occurs, consistent with the intra-month and cross-month axioms providing the necessary steps without pathological depth growth. The remaining 22 conjectures require SAFE_HEAVY, corresponding precisely to the backward arithmetic (12), mixed scheduling (4), and deep stress-test (4+2) categories for which the anchor-compression analysis predicts a benefit from Tier 3. The tier boundaries are not artefacts of solver tuning; they follow directly from the logical structure of the axiomatization.

5.3. Cross-Paradigm Validation with Z3

The two paradigms—saturation-based superposition (Vampire) and CDCL(T) (Z3)—have complementary strengths, making their agreement a strong validation signal. Our run shows that Z3 discharges all 200 conjectures using the Best0 file `DateArithmetic_TemporalSuiteBest0_PORTFOLIO.tff`, with reported times between roughly 67 ms and 121 ms across the suite. Thus the correct empirical statement

Table 3
Scalability across theories. "basic" = date arithmetic only; "+wk" = additionally coupled with weekday constraints. $n = 18$ per row (9 forward + 9 backward offsets, 10^0–10^6 days).

Theory	Dir.	Median (ms)		P90 (ms)		Success	
		basic	+wk	basic	+wk	basic	+wk
Best0	fwd	382	8875	49973	119983	18/18	18/18
Best0	bwd	380	119941	1961	119956	18/18	18/18
Best1	fwd	16375	120000	120038	280702	18/18	18/18
Best1	bwd	16598	119944	120010	120009	18/18	18/18
FAST	fwd	339	341	352	347	18/18	18/18
FAST	bwd	337	342	342	344	18/18	18/18
SAFE_HEAVY	fwd	390	119930	1175	120000	18/18	18/18
SAFE_HEAVY	bwd	157	73197	1962	119976	18/18	18/18

is not that Z3 reproduces Vampire's tier separation, but that Z3 validates the conjectures under a much stronger native arithmetic engine even when presented with the weakest portfolio file.

This difference is informative rather than problematic. Vampire and Z3 are not being evaluated under the same proof-search model: Vampire relies on saturation over the explicit first-order axioms, whereas Z3 combines Boolean search with built-in arithmetic reasoning and aggressive simplification. The fact that Z3 solves the suite from Best0 therefore does *not* show that the higher tiers are unnecessary for Vampire; it shows instead that the tiered axiomatization is specifically addressing the needs of first-order ATP. For this reason, the Vampire portfolio remains the primary experiment for assessing our stratified theory, while Z3 serves as an external semantic cross-check of benchmark correctness.

A second useful observation from our Z3 log is that the solved Best0 set is not restricted to structural queries: it includes backward arithmetic, weekday, leap-year, scheduling, and mixed cases as well. We therefore do not claim any solver-independent tier boundary. The tier distribution in Table 2 is a theorem-guided account of *Vampire*'s behaviour under the portfolio, not a universal law applying equally to SMT solvers.

5.4. Scalability and the FAST Fragment

Proposition 1 predicts that anchor subsumption removes the need to re-derive covered ground normalization chains. To test this prediction empirically, we define the *FAST fragment* (Finite Accelerator Semantic Theory): an intentionally *incomplete* subset of $\mathcal{T}_3$ containing only three universally quantified accelerator axioms for century (36524 days), decade (≈3652 days), and quad-year (1461 days) jumps, subject to alignment conditions (e.g., $M = 1$ and $100 \mid Y$ for the century axiom). FAST cannot prove arbitrary conjectures—it is a controlled experiment, not a complete system.

Table 3 reports solving times over offsets spanning 10^0–10^6 days. FAST achieves essentially constant runtimes (337–352 ms) across all offsets, for both basic date arithmetic and weekday-coupled instances. The portfolio theories, by contrast, show pronounced offset sensitivity: Best1 in particular is *slower* than Best0, because the normalization axioms expose recursive structure to the prover and enlarge the search space without providing the ground endpoints that make proofs short. This is a concrete demonstration that adding axioms can harm rather than help saturation-based provers when those axioms create new derivation paths without resolving the underlying depth problem.

SAFE_HEAVY improves substantially over Best0 and Best1 on forward basic arithmetic (median 390 ms vs. 382 ms and 16375 ms respectively) and backward basic arithmetic (157 ms vs. 380 ms and 16598 ms), confirming that anchors provide the predicted cut-point benefit. The weekday-coupled instances remain hard for SAFE_HEAVY because the weekday constraint introduces additional ground instances not covered by the 47 benchmark-specific anchors; here FAST—whose universally quantified accelerators cover these cases for the alignment-restricted query class—dominates. The FAST axioms and their derivability from $\mathcal{T}_2$ are detailed in Appendix D.

6. Discussion

The integer trap and what anchors do. The integer trap is the phenomenon whereby backward queries force a prover to commit to a concrete integer witness by working through a chain of recursive normalization steps. The recursion proceeds month by month (not day by day): each step crosses one month boundary, so the depth is proportional to the number of months traversed, which is $O(|O|/28)$ in the worst case but typically much less for large offsets. The key difficulty is not the depth per se but that the prover must re-derive the same ground normalization chains for every benchmark that shares a common prefix. Proposition 1 shows that, for the recurring ground subgoals actually covered by anchors, the bottleneck is a consequence of having to re-derive those chains inside proof search. Semantic anchors dissolve the bottleneck by providing cut points: a single axiom application replaces the entire corresponding normalization subderivation. The conservativity result (Theorem 4) guarantees this is semantically legitimate: no new universal facts are introduced, only a more efficient path to facts that were already provable.

Why Best1 can be worse than Best0. An apparent paradox in Table 3 is that Best1 ($\mathcal{T}_2$) is substantially slower than Best0 ($\mathcal{T}_1$) on many scalability instances. The explanation follows directly from the theory: $\mathcal{T}_2$ exposes the recursive normalization structure to the prover, creating new candidate derivation paths that make the saturation set much larger. Without anchors, those paths do not resolve into proofs; they simply expand the search space. This demonstrates a general principle about saturation-based proving: adding semantically correct axioms can degrade performance if they create new unresolvable derivation paths, even when they do not add any *incorrect* information.

Semantic anchors as a general technique. Although developed here for temporal arithmetic, the anchor technique is broadly applicable to any domain where reasoning involves long but semantically uninteresting traversals—arithmetic offsets in memory models, inductive data structures, or bounded iteration. The conservativity theorem guarantees semantic transparency, and finite anchor sufficiency (Corollary 2) guarantees that a finite set always suffices for a finite benchmark. A natural generalisation is proof-analysis-guided anchor generation: run the prover on $\mathcal{T}_2$ with limited depth, record the longest unresolved normalization chains, and add their ground endpoints as anchors. This approach would automate the anchor selection process currently done by inspection.

Relation to SMT and theory solvers. SMT solvers address temporal arithmetic via specialised theory solvers and efficient quantifier-free arithmetic. Our approach occupies a different point in the design space: fully quantified FOL with formal proof output, at the cost of requiring the anchor infrastructure. The cross-validation data shows the two paradigms are not in competition: Z3 solves all 200 conjectures independently and faster on absolute time, while Vampire's proofs are formal first-order derivations that can be proof-checked. For applications requiring formal proofs or integration with large logical theories such as SUMO, the FOL approach is the only viable option; for rapid validation of specific calendar facts, SMT is more efficient.

Limitations. The anchor set is tailored to the benchmark suite. Extending to open-ended query domains requires automated anchor generation, which we leave to future work. The approach does not claim decidability of temporal arithmetic in general, only tractability for the large, practically relevant fragment covered by the suite. The 60 s per tier timeout is generous; a tighter timeout would shift some Best1 successes to SAFE_HEAVY, but would not change the 100% overall coverage. An alternative to the recursive calc_date axiomatization would be a closed-form definition reducing date arithmetic directly to Presburger arithmetic, exploiting the 400-year periodicity of the Gregorian calendar. Such a definition could in principle be handled entirely within SMT's quantifier-free arithmetic fragment and is a worthwhile direction for future work, complementary to the FOL approach pursued here.

7. Conclusion

We have presented a stratified first-order axiomatization of Gregorian calendar arithmetic in three strata, $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, that separates structural calendar laws, arithmetic normalization, and conservative ground semantic anchors. The core insight is that the perceived difficulty of temporal ATP is not intrinsic to the calendar domain but arises from conflating three semantically distinct phenomena— arithmetic traversal, structural discontinuities, and cyclic constraints—in a flat axiom set. Stratification resolves this by assigning each phenomenon to the appropriate stratum. separation.

The theoretical contributions—determinacy (Theorems 1 and 2), soundness (Theorem 3), conservativity (Theorem 4), and bounded proof depth (Proposition 1)—form a coherent account of *why* the approach works, not merely that it does. The experimental results are consistent at every level with these theoretical predictions: tier boundaries align with query categories, Z3 and Vampire agree, and the FAST fragment demonstrates empirically that anchor subsumption eliminates offset sensitivity in the pure scalability regime.

Future directions include automated anchor generation via proof analysis, integration with the SUMO ontology [8, 9] for common-sense calendar reasoning, and extension to temporal modal logic [7].

References

[1] Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Comm. ACM 26(11), 832–843 (1983).

[2] Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Handbook of Satisfiability, FAIA 185, pp. 825–885. IOS Press (2009).

[3] de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: TACAS 2008, LNCS 4963, pp. 337–340. Springer (2008).

[4] Ge, Y., de Moura, L.: Complete Instantiation for Quantified Formulas in Satisfiability Modulo Theories. In: CAV 2009, LNCS 5643, pp. 306–320. Springer (2009).

[5] Joda-Time Development Team: Joda-Time: Java date and time API. https://www.joda.org/joda-time/ (2002–2023).

[6] Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: CAV 2013, LNCS 8044, pp. 1–35. Springer (2013).

[7] Pease, A.: Modal and Higher Order Logical Reasoning with SUMO. In: Semantics at the Crossroads. PubliKon (2025).

[8] Niles, I., Pease, A.: Towards a Standard Upper Ontology. In: FOIS 2001, pp. 2–9. ACM (2001).

[9] Pease, A.: Ontology: A Practical Guide. Articulate Software Press (2011).

[10] Pease, A., Stănică, P.: Vampire Universal System: Dates, Times, Scheduler. https://github.com/ontologyportal/sumo/tree/master/tests (2026).

[11] Prior, A.N.: Past, Present and Future. Oxford University Press (1967).

[12] Pustejovsky, J., et al.: TimeML: Robust Specification of Event and Temporal Expressions in Text. In: 5th Int. Workshop on Computational Semantics (2003).

[13] Schulz, S.: E – A Brainiac Theorem Prover. AI Comm. 15(2–3), 111–126 (2002).

[14] Stănică, P.: Github repository for temporal axiomatization and Python validation codes. https://github.com/pstanica/temporal_vampire (2026).

[15] Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. J. Autom. Reasoning 59(4), 483–502 (2017).

[16] Sutcliffe, G.: The TPTP Problem Library: The FOF and CNF Parts, v3.5.0. J. Autom. Reasoning 43(4), 337–362 (2009).

[17] Weidenbach, C., et al.: SPASS Version 3.5. In: CADE-21, LNAI 4603, pp. 514–520. Springer (2007).

[18] Zeller, C.: Kalender-Formeln. Acta Mathematica 6, 289–290 (1882/1883).

A. Complete TPTP axiomatization Excerpts

This appendix provides TPTP listings for the axioms whose length precluded inclusion in the main body: the full Zeller's congruence weekday axiom, the negative-offset calc_date cases, the nth_weekday_date axiom, and the Integer Trap mechanism.

Weekday via Zeller's congruence

```
tff(weekday_def, axiom, ![Y:$int, M:$int, D:$int, W:day_name]:
  (weekday(ymd(Y,M,D), W) ⇔
    ?[YY:$int, MM:$int, K:$int, J:$int, H:$int]:
      (% Adjust Jan/Feb: treat as months 13/14 of the preceding year
       ((M ≤ 2) ⇒ (MM = $sum(M,12) & YY = $difference(Y,1))) &
       ((M  > 2) ⇒ (MM = M          & YY = Y))               &
       K = $remainder_e(YY, 100) &
       J = $quotient_e(YY, 100)  &
       % Zeller's formula
       H = $remainder_e(
             $sum($sum($sum($sum(D,
               $quotient_e($product(13,$sum(MM,1)),5)),
               K),
               $quotient_e(K,4)),
               $difference($quotient_e(J,4),$product(2,J))),
             7) &
       % h -> day_name: h=0 Saturday, h=1 Sunday, ..., h=6 Friday
       (H=0 ⇒ W=saturday) & (H=1 ⇒ W=sunday)    &
       (H=2 ⇒ W=monday)   & (H=3 ⇒ W=tuesday)   &
       (H=4 ⇒ W=wednesday)& (H=5 ⇒ W=thursday)  &
       (H=6 ⇒ W=friday)))).
```

Listing 6: Weekday axiom encoding Zeller's congruence

Backward calc_date axioms as used in the implementation

```
% Backward recursion to the previous month
tff(rule_back_std, axiom,
  ![D:$int, M:$int, Y:$int, Limit:$int, Res:date]:
  (($lesseq(D, 0) & M ≠ 1)
   ⇒ ((is_days_in_month(Limit, $difference(M,1), Y) &
         calc_date($sum(D, Limit), $difference(M, 1), Y, Res))
      ⇒ calc_date(D, M, Y, Res)))).

% Backward recursion from January to December of the previous year
tff(rule_back_jan, axiom,
  ![D:$int, M:$int, Y:$int, Limit:$int, Res:date]:
  (($lesseq(D, 0) & M = 1)
   ⇒ ((is_days_in_month(Limit, 12, $difference(Y,1)) &
         calc_date($sum(D, Limit), 12, $difference(Y, 1), Res))
      ⇒ calc_date(D, M, Y, Res)))).
```

Listing 7: calc_date: backward recursion exactly as implemented

Scheduler: nth_weekday_date

```
tff(nth_weekday_logic, axiom,
  ![N:$int,W:day_name,M:$int,Y:$int,D:$int,W1:day_name,
    I1:$int,I2:$int,Off:$int,MaxD:$int]:
  ((weekday(ymd(Y,M,1), W1)                              &
    day_to_int(W1,I1) & day_to_int(W,I2)                 &
    Off = $remainder_e($sum($difference(I2,I1),7),7)      &
    D   = $sum(1,$sum(Off,$product($difference(N,1),7)))  &
    is_days_in_month(MaxD,M,Y) & $lesseq(D,MaxD) & $greater(D,0))
   ⇒ nth_weekday_date(N,W,M,Y,ymd(Y,M,D)))).

tff(day_int_sun, axiom, day_to_int(sunday,    0)).
tff(day_int_mon, axiom, day_to_int(monday,    1)).
tff(day_int_tue, axiom, day_to_int(tuesday,   2)).
```

```
tff(day_int_wed, axiom, day_to_int(wednesday,3)).
tff(day_int_thu, axiom, day_to_int(thursday, 4)).
tff(day_int_fri, axiom, day_to_int(friday,   5)).
tff(day_int_sat, axiom, day_to_int(saturday, 6)).
```

Listing 8: nth_weekday_date axiom

Integer trap mechanism

The Integer Trap forces theorem provers to evaluate arithmetic expressions to concrete values rather than leaving them symbolic. Without a trap, a prover may close a conjecture with a symbolic witness $D = \$\mathsf{sum}(\ldots)$ that satisfies the formula structurally without committing to a concrete integer. The trap adds a validity predicate bounding the witness:

```
tff(t_valid_day, type, valid_day: ($int) > $o).
tff(trap_ax, axiom, ![D:$int]:
  (valid_day(D) <=> ($greater(D,0) & $lesseq(D,31)))).

% Example: forces D = 4 (Feb 4, 2024), not a symbolic $sum(...)
tff(calc_35_jan_2024, conjecture,
  ?[D:$int]: (calc_date(35, 1, 2024, ymd(2024, 2, D)) & valid_day(D))).
```

Listing 9: Integer Trap: forcing concrete arithmetic evaluation

B. Complete Anchor Set

The 47 semantic anchors forming $\mathcal{T}_3 \setminus \mathcal{T}_2$ are listed in the repositories [10, 14] as `DateArithmetic_Completion_SAFE_HEAVY_PLUS_v3.tff`. They fall into four groups: 8 reference weekday anchors (century boundaries 1900–2100 plus January 1 of 2000 and 2024); 15 large positive-offset calc_date anchors (offsets 1461, 3652, 36524 and selected multiples); 12 large negative-offset calc_date anchors (symmetric to the positive set); and 12 mixed normalization anchors (date offsets combined with time normalization, used by scheduling and stress-test categories). Each anchor was verified as a $\mathcal{T}_2$-theorem before inclusion, with Vampire proof times of 33–412 ms and Z3 proof times of 67–121 ms.

C. Benchmark Sample

One fully-stated TFA conjecture per category:
Direct date arithmetic:

```
tff(fwd_365_jan_2024, conjecture,
  calc_date(365, 1, 2024, ymd(2024, 12, 31))).
```

Backward date arithmetic:

```
tff(bwd_1460_jan_2028, conjecture,
  calc_date(-1460, 1, 2028, ymd(2024, 1, 1))).
```

Leap-year / century edge case (2100 is not a leap year):

```
tff(leap_2100_not, conjecture,
  ?[D:$int]: (calc_date(365, 2, 2100, ymd(2101, 2, D)) & D = 28)).
```

Weekday computation:

```
tff(weekday_jan1_2024, conjecture,
  weekday(ymd(2024, 1, 1), monday)).
```

Nth-weekday / scheduling (third Monday of February 2025):

```
tff(third_monday_feb_2025, conjecture,
  nth_weekday_date(3, monday, 2, 2025, ymd(2025, 2, 17))).
```

normalization stress tests:

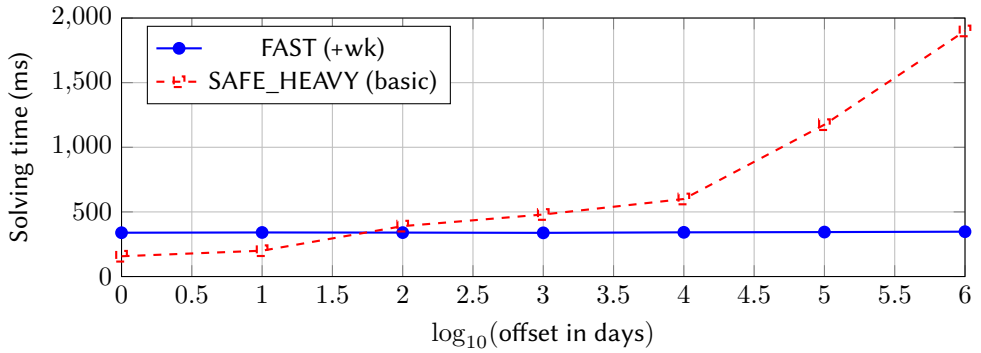

Figure 1: Runtime profiles for FAST and SAFE_HEAVY across offset magnitudes (forward direction, schematic from measured data). FAST achieves offset-independent performance on the restricted aligned-date query class.

```
tff(large_fwd, conjecture, ?[D:date]: calc_date(1000000, 1, 2000, D)).
tff(time_trap, conjecture,
  ?[EH:$int,EM:$int,EDD:$int]: normalize_time(24, 0, -1, EH, EM, EDD)).
```

D. FAST Accelerator Fragment

FAST (Finite Accelerator Semantic Theory) consists of three universally quantified axioms enabling century (36524 days), decade (≈3652 days), and quad-year (1461 days) jumps, subject to alignment conditions. Unlike the ground anchors in $\mathcal{T}_3$, these are universally quantified over starting years, so a single axiom covers an infinite family of aligned-date queries.

```
% Century jump: applies only when M=1 and Y is divisible by 100
tff(hint_century, axiom, ![Y:$int, Cents:$int]:
  (($remainder_e(Y,100) = 0 & $greater(Cents,0))
   ⇒ calc_date($product(Cents,36524), 1, Y,
                ymd($sum(Y,$product(Cents,100)), 1, 1)))).

tff(hint_decade, axiom, ![Y:$int, Decs:$int]:
  (($remainder_e(Y,10) = 0 & $greater(Decs,0))
   ⇒ calc_date($product(Decs,3652), 1, Y,
                ymd($sum(Y,$product(Decs,10)), 1, 1)))).

tff(hint_quad, axiom, ![Y:$int, Quads:$int]:
  (($remainder_e(Y,4) = 0 & $greater(Quads,0))
   ⇒ calc_date($product(Quads,1461), 1, Y,
                ymd($sum(Y,$product(Quads,4)), 1, 1)))).
```

Listing 10: FAST: century, decade, and quad-year axioms

Derivability. Each FAST axiom instance is derivable from $\mathcal{T}_2$ for any ground substitution satisfying the alignment condition. For example, $\mathsf{calc\_date}(36524, 1, 2000, \mathsf{ymd}(2100, 1, 1))$ is a $\mathcal{T}_2$-theorem (Vampire: 412 ms).

Intentional gaps. FAST is deliberately incomplete. The century axiom applies only when $M = 1$ and $100 \mid Y$. The day counts (36524, 3652, 1461) assume no intervening leap-century anomaly; this is correct for century-aligned dates but cannot be used for arbitrary starting months or years. FAST achieves offset-independent performance on aligned-date queries (337–352 ms across 10^0–10^6 days) at the cost of incomplete coverage; the full SAFE_HEAVY configuration is required for general query coverage.