Science of Computer Programming 231 (2024) 103012

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

The complexity of evaluating nfer

Sean Kauffman *, Martin Zimmermann *

Aalborg University, Denmark

ARTICLE INFO

Article history:
Received 29 November 2022
Received in revised form 7 July 2023
Accepted 15 August 2023
Available online 22 August 2023

Keywords:
Interval logic
Complexity
Runtime verification

ABSTRACT

`Nfer` is a rule-based language for abstracting event streams into a hierarchy of intervals with data. `Nfer` has multiple implementations and has been applied in the analysis of spacecraft telemetry and autonomous vehicle logs. This work provides the first complexity analysis of `nfer` evaluation, i.e., the problem of deciding whether a given interval is generated by applying rules.

We show that the full `nfer` language is undecidable and that this depends on both recursion in the rules and an infinite data domain. By restricting either or both of those capabilities, we obtain tight decidability results. We also examine the impact on complexity of exclusive rules and minimality. For the most practical case, which is minimality with finite data, we provide a polynomial-time algorithm.

© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

`Nfer` is a rule-based language for event stream analysis, developed with scientists from the National Aeronautics and Space Administration (NASA)'s Jet Propulsion Laboratory (JPL) to analyze telemetry from spacecraft [1–3]. `Nfer` rules calculate data over periods of time called intervals. `Nfer` compares and combines these intervals to form a hierarchy of abstractions that is easier for humans and machines to comprehend than a trace of discrete events. This differs from traditional Runtime Verification (RV) which computes language inclusion and returns verdicts. The equivalent problem for `nfer`, called the evaluation problem, is to determine if an interval will be present in `nfer`'s output given a list of rules and an input trace.

The `nfer` syntax is based on Allen's Temporal Logic (ATL) [4] and is designed for simplicity and brevity in many contexts. When it was originally introduced, `nfer` was used to find false positives among warning messages from the Mars Science Laboratory (MSL), i.e., the Curiosity rover, at JPL [2]. Researchers found the language to be much more concise than the ad hoc Python scripts in common use. `Nfer` has also been deployed to capture disagreements between parallel Proportional-Integral-Derivative (PID) controllers in an embedded system ionizing radiation experiment [5,3] and to locate unstable gear shifts in an autonomous vehicle [6]. The `nfer` language is implemented in a tool available under the GPLv3 license with support for interfaces in multiple programming environments [7].

`Nfer` is expressive enough for many applications and termination of `nfer` has been conjectured to be undecidable [8]. The intuition for `nfer` undecidability is that recursion in its rules is possible and the intervals `nfer` computes may carry data from an infinite domain.

Despite this expressiveness, `nfer`'s implementations have been demonstrated to be fast in practice. Both the C [9] and Scala [10] versions have been compared against tools such as LogFire and Prolog [3], Siddhi [6], MonAmi and `DejaVu` [11],

* Corresponding authors.
E-mail addresses: seank@cs.aau.dk (S. Kauffman), mzi@cs.aau.dk (M. Zimmermann).

https://doi.org/10.1016/j.scico.2023.103012
0167-6423/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Table 1
Overview of our complexity results. The asterisk * signifies that only cycles consisting of inclusive rules are allowed.

Semantics	Cycles	Data	Minimality	Complexity	Reference
inclusive	yes	infinite	no	undecidable	Theorem 1
inclusive	no	finite	no	PSpace-complete	Theorem 2
inclusive	yes	finite	no	ExpTime-complete	Theorem 3
inclusive	no	infinite	no	NExpTime-complete	Theorem 4
full	no	finite	no	PSpace-complete	Theorem 5
full	no	infinite	no	AExpTime(poly)-complete	Theorem 6
extended	yes*	finite	no	ExpTime-complete	Theorem 7
extended	yes*	infinite	no	undecidable	Theorem 1
extended	yes*	finite	yes	PTime	Theorem 8
extended	yes*	infinite	yes	ExpTime	Theorem 9

and TeSSLa [12] and in every case found to be faster than the alternatives performing the same analysis. The question remains if `nfer` evaluation is indeed undecidable and, if so, if there are useful fragments of `nfer` with a tractable evaluation problem.

1.1. Our contribution

In this work, we determine the complexity of evaluating different fragments of `nfer`. We find that any one of several restrictions on the language permit decidable evaluation and we prove tight bounds for almost all of these fragments. This article extends the TASE 2022 [13] paper on the same topic, which was the first work to examine the complexity of any problem in `nfer`.

We begin by defining a natural syntactic fragment of `nfer` using only inclusive rules called `inc-nfer`. Full `nfer` supports a form of negation using what are called exclusive rules, but we show that these are unnecessary to obtain undecidability. The result relies, instead, on recursion between rules and on intervals carrying data from an infinite domain. This proves the conjecture mentioned above.

To regain decidability, we then examine language fragments where either or both of these capabilities are restricted. We prove that, without recursion, `inc-nfer` evaluation is NExpTime-complete, without infinite data it is ExpTime-complete, and without either it is PSpace-complete.

We then introduce exclusive rules and examine the full `nfer` language. It has been openly questioned what effect negation has on the expressiveness of `nfer` [11]. Of note is that the semantics of exclusive rules restrict the use of recursion. In particular, to ensure that an `nfer` instance with exclusive rules yields a unique output, exclusive rules may not appear within cycles. `Nfer` previously only supported exclusive rules in cycle-free settings, but we extend the semantics here (called `ext-nfer`) to support exclusive rules alongside cycles in inclusive (i.e., non-exclusive) rules.

We prove that, with finite data, adding exclusive rules has no effect and `nfer` evaluation remains PSpace-complete without cycles and ExpTime-complete with cycles. Without cycles and infinite data, however, we prove that the problem is AExpTime(poly)-complete. The remaining combination, i.e., exclusive rules with cycles and infinite data is undecidable, as the problem is already so without exclusive rules.

We go on to examine the effect of minimality on the complexity of `nfer` evaluation. Minimality is a so-called meta-constraint on the results of `nfer` that was a primary motivator of `nfer`'s development, since it was discovered existing tools like Prolog struggled with such meta-constraints [3]. We show that minimality has a substantial effect on the complexity of `nfer` evaluation. With infinite data, we prove that the problem is in ExpTime. The most common method of using `nfer` is with minimality and finite data, however, and we prove evaluation for this configuration is in PTime. These results are independent of the use of exclusive rules and cycles.

Table 1 shows an overview of our results.

This work extends the authors' previous paper presented at TASE 2022 [13]. Here, we present the complete proofs for each complexity result, including for Theorems 4, 6, and 9. We also prove a new AExpTime(poly) lower bound for full semantics with infinite data, thereby closing a gap that was left open in the conference version. Finally, we introduce the new `nfer` extended semantics to support exclusive rules in the same specification as cycles. Along with the semantics, we provide complexity results for the relevant fragments under it.

1.2. Related work

`Nfer` is closely related to other classes of declarative programming systems but it differs from them all in several ways. For example, a rule-based programming system modifies a database of facts [14,15]. Unlike these systems, however, `nfer` is monotonic and can only add intervals, not remove them. `Nfer` also resembles Complex Event Processing (CEP) systems where declarative rules are applied to compute information from a trace of events [16–18]. Zhang, Diao, and Immerman

examined the complexity of pattern queries in a number of popular CEP languages [19]. They focused on which features of pattern queries increased expressiveness and were, therefore, more expensive. CEP systems do not usually include explicit notions of time or temporal relationships, though, which are central to `nfer`. In this way, `nfer` more closely resembles stream-RV systems [20–22]. Convent et al. translated fragments of TeSSLa into timed automata to inherit decidability results [21]. Still, `nfer` is differentiated from these systems by its emphasis on temporal intervals and its ATL-based syntax.

Languages like the Planning Domain Definition Language (PDDL) [23], the Action Notation Modeling Language Light (ANMLite) [24], and the Clock Constraints Specification Language (CCSL) [25] model clock constraints in ways that resemble `nfer` and are used in planning. These languages are used to limit what inputs may satisfy a model, while `nfer` rules describe a hierarchy of abstractions of inputs. Note that the *planning* problem more closely resembles the `nfer` *satisfiability* problem which will be addressed in future work.

Some research has examined the complexity of logics based on ATL, specifically Halpern and Shoham's modal logic of intervals (HS) [26]. Montanari et al. showed that the satisfiability problem for the subset of HS consisting of only the temporal operators *begins/begun by* and *meets* is ExpSpace-complete over the natural numbers [27]. Later, they showed that adding the *met by* operator increases the complexity such that the language is only decidable over finite total orders [28]. Aceto et al. identified the expressive power of all fragments of HS over total orders as well as only dense total orders [29]. `Nfer` is not a modal logic, however, and these complexity results are not relevant to its evaluation problem.

2. The inclusive nfer language

The `nfer` language supports two types of rules: inclusive rules and exclusive rules. This section describes the `inclusive-nfer` formalism, subsequently abbreviated `inc-nfer`, that supports only inclusive rules. `Inc-nfer` is sufficiently expressive to obtain an undecidability result and we find that initially omitting exclusive rules simplifies our presentation. `Inc-nfer` is also a natural subset of `nfer` that was first introduced in [2]. It supports many use cases, including the MSL case-study described above. The implementation of `nfer` written in Scala at JPL [10,3] also supports only inclusive rules. We expand our analysis to include exclusive rules in Section 4 (without any cycles) and in Section 6 (with cycles in inclusive rules). Finally, Section 8 addresses minimality, an important extension of `nfer` semantics. Note that, to improve comprehensibility and simplify later proofs, the semantics presented in this section differs slightly from prior work but these changes do not affect the language capabilities.

2.1. Preliminary notation

We denote the set of nonnegative integers as $\mathbb{N}$. The set of Booleans is given as $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$. We fix a finite set $\mathcal{I}$ of identifiers. $\mathbb{M}$ is the set of all of maps, where a map $M \in \mathbb{M}$ is a partial function $M : \mathcal{I} \nrightarrow \mathbb{N} \cup \mathbb{B}$.

An event represents a named state change in an observed system. An event is a triple (η, t, M) where $\eta \in \mathcal{I}$ is its identifier, $t \in \mathbb{N}$ is the timestamp when it occurred, and $M \in \mathbb{M}$ is its map of data. The set of all events is $\mathbb{E} = \mathcal{I} \times \mathbb{N} \times \mathbb{M}$. A sequence of events $\tau \in \mathbb{E}^*$ is called a *trace*.

Intervals represent a named period of state in an observed system. An interval is a 4-tuple (η, s, e, M) where $\eta \in \mathcal{I}$ is its identifier, $s, e \in \mathbb{N}$ are the starting and ending timestamps where $s \leq e$, and $M \in \mathbb{M}$ is its map of data. The set of all intervals with data is $\mathbb{I} = \mathcal{I} \times \mathbb{N} \times \mathbb{N} \times \mathbb{M}$. A set of intervals is called a *pool* and the set of all pools is $\mathbb{P} = 2^{\mathbb{I}}$. We say that an interval $i = (\eta, s, e, M)$ is labeled by η. We define the functions $\mathit{id}(i) = \eta$, $\mathit{start}(i) = s$, $\mathit{end}(i) = e$, and $\mathit{map}(i) = M$.

2.2. Syntax

Inclusive rules test for the existence of two intervals matching constraints. When such a pair is found, a new interval is produced with an identifier specified by the rule. The new interval has timestamps and a map derived by applying functions, specified in the rule, to the matched pair of intervals. We define the syntax of these rules, including mathematical functions to simplify the presentation, as follows:

$$\eta \leftarrow \eta_1 \ \oplus \ \eta_2 \ \textbf{where} \ \Phi \ \textbf{map} \ \Psi$$

where, $\eta, \eta_1, \eta_2 \in \mathcal{I}$ are identifiers, $\oplus \in \{\textbf{before}, \textbf{meet}, \textbf{during}, \textbf{coincide}, \textbf{start}, \textbf{finish}, \textbf{overlap}, \textbf{slice}\}$ is a *clock predicate* on three intervals (one for each of $\eta, \eta_1,$ and η_2), $\Phi : \mathbb{M} \times \mathbb{M} \to \mathbb{B}$ is a *map predicate* taking two maps and returning a Boolean representing satisfaction of a constraint, and $\Psi : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$ is a *map update* taking two maps and returning a map.

We omit the precise syntax for specifying map predicates and updates, but we require that these functions are limited to only simple arithmetic operations. This matches what is possible using the C `nfer` tool [12]. Specifically, map predicates and map updates must be expressible using the standard mathematical operations $+$, $-$, $\cdot$ (multiplication), / (division), mod, the comparisons $<, \leq, >, \geq, =$, and the Boolean operators $\wedge, \vee, \neg$. This limitation excludes exponentiation and any form of recursion in the functions. Since we do not support real numbers in the theory, division is limited to integer quotients. These decisions are discussed in Section 9.

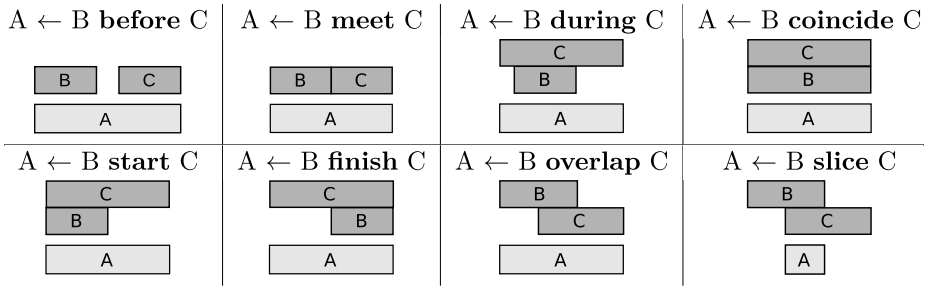

Fig. 1. `nfer` clock predicates for inclusive rules.

Table 2
Formal definition of `nfer` clock predicates for inclusive rules.

Syntax	Definition of $\oplus(i, i_1, i_2)$
before	$end(i_1) < start(i_2) \wedge start(i) = start(i_1) \wedge end(i) = end(i_2)$
meet	$end(i_1) = start(i_2) \wedge start(i) = start(i_1) \wedge end(i) = end(i_2)$
during	$start(i_1) \geq start(i_2) = start(i) \wedge end(i_1) \leq end(i_2) = end(i)$
coincide	$start(i_1) = start(i_2) = start(i) \wedge end(i_1) = end(i_2) = end(i)$
start	$start(i_1) = start(i_2) = start(i) \wedge end(i) = \max(end(i_1), end(i_2))$
finish	$start(i) = \min(start(i_1), start(i_2)) \wedge end(i) = end(i_1) = end(i_2)$
overlap	$start(i_1) < end(i_2) \wedge start(i_2) < end(i_1) \wedge$ $start(i) = \min(start(i_1), start(i_2)) \wedge$ $end(i) = \max(end(i_1), end(i_2))$
slice	$start(i_1) < end(i_2) \wedge start(i_2) < end(i_1) \wedge$ $start(i) = \max(start(i_1), start(i_2)) \wedge$ $end(i) = \min(end(i_1), end(i_2))$

2.3. Semantics

`Inc-nfer` defines how rules are interpreted to generate pools of intervals from inputs. The semantics utilizes functions, referenced by the rule syntax, that specify the temporal and data relationships between intervals. The semantics of the `nfer` language is defined in three steps: the semantics R of individual rules on pools, the semantics S of a specification (a list of rules) on pools, and finally the semantics T of a specification on traces of events.

We first define the semantics of inclusive rules with the interpretation function R. Let Δ be the set of all rules. Semantic functions are defined using the brackets $[\![_]\!]$ around syntax being given semantics.

$$
\begin{aligned}
&R\,[\![\,_\,]\!] \;:\; \Delta \to \mathbb{P} \to \mathbb{P} \\
&R\,[\![\,\eta \leftarrow \eta_1 \oplus\ \eta_2\ \textbf{where}\ \Phi\ \ \textbf{map}\ \Psi\,]\!]\ \pi\ =\ \\
&\quad \{\ i \in \mathbb{I} \;:\; i_1, i_2 \in \pi\ \ . \\
&\qquad\qquad id(i) = \eta \wedge\ id(i_1) = \eta_1 \wedge\ id(i_2) = \eta_2\ \wedge \\
&\qquad\qquad \oplus(i, i_1, i_2) \wedge \Phi(map(i_1), map(i_2)) \wedge \\
&\qquad\qquad map(i) = \Psi(map(i_1), map(i_2))\ \}
\end{aligned}
$$

In the definition, a new interval i is produced when two existing intervals in π match the identifiers η_1 and η_2, the temporal constraint $\oplus$, and the map constraint Φ. $\oplus$ defines the start and end timestamps of i and Ψ defines its map.

The possibilities referenced by $\oplus$ are shown in Fig. 1 and formally defined in Table 2. These clock predicates relate two intervals using the familiar ATL temporal operators and also specify the start and end timestamps of the produced intervals. In the figure, the two matched intervals are shown as dark-gray boxes where time flows from left to right and the light-gray box is the produced interval. Note that the generated interval labeled by C has start and end timestamps inherited from the intervals labeled A and B, i.e., no new timestamps are generated by applying a rule. For example, given intervals i, i_1, i_2 where $id(i) = A$, $id(i_1) = B$ and $id(i_2) = C$, $A \leftarrow B$ **meet** C holds when $end(i_1) = start(i_2)$, $start(i) = start(i_1)$, and $end(i) = end(i_2)$.

The following one-step interpretation function S defines the semantics of a finite list of rules, also called a specification. Given a specification $\delta_1 \cdots \delta_n \in \Delta^*$ and a pool $\pi \in \mathbb{P}$, $S[\![_]\!]$ returns a new pool obtained by recursively applying $R[\![_]\!]$ to every rule in $\delta_1 \cdots \delta_n$ in order, where each is called using the union of π with the new intervals returned thus far.

$$S\,[\![\,_\,]\!] \;:\; \Delta^* \to \mathbb{P} \to \mathbb{P}$$

$$S\,[\![\,\delta_1\cdots\delta_n\,]\!]\,\pi \;=\; \begin{cases} S\,[\![\,\delta_2\cdots\delta_n\,]\!]\,(\pi \;\cup\; R\,[\![\,\delta_1\,]\!]\,\pi\;) & \textbf{if } n>0 \\ \pi & \textbf{otherwise} \end{cases}$$

`Inc-nfer` specifications may contain recursion in the rules, so one application of the specification may not be sufficient to produce all of the intervals. The interpretation function $T_{\text{inc}}[\![_]\!]$ for *inc*lusive `nfer` defines the semantics of a specification on a pool by applying S until the inflationary fixed point is reached.

$$T_{\text{inc}}[\![_]\!] \;:\; \Delta^* \to \mathbb{P} \;\to \mathbb{P}$$
$$T_{\text{inc}}[\![\delta_1\cdots\delta_n\,]\!]\,\pi \;=\; \bigcup_{i>0} \pi_i \,.\, \pi_1 = \pi \;\wedge\; \pi_{i+1} = S\,[\![\,\delta_1\,\cdots\delta_n\,]\!]\,(\pi_i)$$

To maintain consistency with prior work and simplify our presentation, we also overload T_{inc} to operate on a trace of events $\tau \in \mathbb{E}^*$ by first converting τ to the pool $\{init(e) : e \text{ is an element of } \tau\}$ where $init(\eta, t, M) = (\eta, t, t, M)$.

Example 1. Here, we present an example of an `inc-nfer` specification with rules useful for our complexity analysis. Fix $\mathcal{I} = \{\eta_j : 0 \le j \le n\} \cup \{d\}$ and consider the specification $D_n = \delta_1 \cdots \delta_n$ where δ_j is the rule

$$\eta_{j+1} \leftarrow \eta_j \textbf{ coincide } \eta_j \textbf{ where } m_1, m_2 \mapsto m_1(d) = m_2(d) \textbf{ map } m_1, m_2 \mapsto \{d \mapsto m_1(d)^2\}.$$

Here, m_1 and m_2 denote the maps of the intervals matched by the left and right side of the coincide operator and d represents the only element in their domain.

When applying this specification to the trace $\tau = (\eta_0, 0, \{d \mapsto 2\})$ we obtain

$$T_{\text{inc}}[\![D_n]\!]\,\tau = \{(\eta_0, 0, 0, \{d \mapsto 2\}), (\eta_1, 0, 0, \{d \mapsto 4\}), \ldots, (\eta_n, 0, 0, \{d \mapsto 2^{2^n}\})\}.$$

Remark 1. In many of our lower bound proofs, the timestamps of intervals are irrelevant. For the sake of readability, we will therefore often disregard the timestamps and denote intervals by $(\eta, y_0, \ldots, y_k)$ where $\{y_0, \ldots, y_k\}$ is the image of the map function of the interval. Here, we assume a fixed order of the map domain that will be clear from context.

Also, note that the rules δ_j in Example 1 produce an interval i' labeled by η_{j+1} from an interval i such that i and i' have the same timestamps and the map value of i' is obtained by squaring the map value of i. Many of the rules we use in our lower bounds proofs have this format. Again, for the sake of readability, we will not spell out those rules but instead say that the rule produces the interval (η_{j+1}, y^2) from an interval of the form (η_j, y).

2.4. The evaluation problem

We are interested in the `nfer` evaluation problem: Given a specification D, a trace τ of events, and a target identifier η_T, is there an η_T-labeled interval in $T_{\text{inc}}[\![D]\!]\,\tau$? Here, we measure the size of a single rule in D by the sum of the length of its map predicate and map update measured in their number of arithmetic and logical operators, with numbers encoded in binary. The size of an event is the sum of the binary encodings of its timestamps and its map values. We disregard the identifiers, as their number is bounded by the number of events in the input trace and the number of rules.

Example 2. Consider an extension of the specification from Example 1 that adds rules to generate a η_T-labeled interval if any η_j-labeled interval has a map value equal to a given number $T \in \mathbb{N}$. We define the specification $D_n^T = D_n \delta_{n+1} \cdots \delta_{2n}$ where δ_j for $j > n$ is the rule

$$\eta_T \leftarrow \eta_j \textbf{ coincide } \eta_j \textbf{ where } m_1, m_2 \mapsto m_1(d) = m_2(d) = T \textbf{ map } \{\,\}.$$

If we fix $T = 16$, then

$(\eta_T, 0, 0, \{\,\}) \in T_{\text{inc}}[\![D_n^{16}]\!]\,\tau$ for $\tau = (\eta_0, 0, \{d \mapsto 2\})$, and
$(\eta_T, 0, 0, \{\,\}) \in T_{\text{inc}}[\![D_n^{16}]\!]\,\tau$ for $\tau = (\eta_0, 0, \{d \mapsto 4\})$, but
$(\eta_T, 0, 0, \{\,\}) \notin T_{\text{inc}}[\![D_n^{16}]\!]\,\tau$ for $\tau = (\eta_0, 0, \{d \mapsto 3\})$.

The `nfer` evaluation problem is to determine whether or not an η_T-labeled interval is in the fixed point given a specification D and a trace of events τ.

3. Complexity results for inclusive nfer

In this section, we determine the complexity of the `inc-nfer` evaluation problem. In its most general form it is shown to be undecidable, but we show decidability for three natural fragments.

3.1. The unrestricted `Inc-nfer` language

The undecidability result relies on the recursive nature of `inc-nfer`, i.e., an η-labeled interval can be (directly or indirectly) produced from another η-labeled interval, and on the fact that the map functions range over the natural numbers, i.e., we have access to an infinite data domain.

Theorem 1. The evaluation problem for `inc-nfer` is undecidable.

Proof. We show how to simulate a two-counter Minsky machine [30] with `inc-nfer` rules so that the machine terminates iff an interval with a given target identifier can be generated by the rules.

Formally, a two-counter Minsky machine is a sequence

$$(0 : \mathtt{I}_0)(1 : \mathtt{I}_1)\cdots(k-2 : \mathtt{I}_{k-2})(k-1 : \mathtt{STOP}),$$

of pairs $(\ell : \mathtt{I}_\ell)$ where ℓ is a line number and $\mathtt{I}_\ell$ for $0 \leq \ell < k-1$ is one of `INC(`$\mathtt{X}_i$`)`, `DEC(`$\mathtt{X}_i$`)`, or `IF` $\mathtt{X}_i$`=0 GOTO` ℓ' with $i \in \{0,1\}$ and $\ell' \in \{0,\cdots,k-1\}$.

A configuration of the machine is a triple (ℓ, c_0, c_1) consisting of a line number ℓ and the contents $c_i \in \mathbb{N}$ of counter i. The semantics is defined as expected with the convention that a decrement of a zero counter has no effect. The problem of deciding whether the unique run of a given two-counter Minsky machine starting in the initial configuration $(0,0,0)$ reaches a stopping configuration (i.e., one of the form $(k-1, c_0, c_1)$) is undecidable [30].

This problem is captured with `inc-nfer` as follows: We encode a configuration (ℓ, c_0, c_1) by an interval with identifier ℓ and two map values c_0, c_1. These intervals use the same timestamps so we drop them from our notation and also write (ℓ, c_0, c_1) for the interval encoding that configuration.

For every line number $0 \leq \ell < k-1$ we have one or two rules that are defined as follows (here, we only consider instruction for the first counter, the rules for the second counter are analogous):

- $\mathtt{I}_\ell =$ `INC(`$\mathtt{X}_0$`)`: We have a rule producing the interval $(\ell+1, c_0+1, c_1)$ from an interval of the form (ℓ, c_0, c_1).
- $\mathtt{I}_\ell =$ `DEC(`$\mathtt{X}_0$`)`: We have two rules, one producing the interval $(\ell+1, c_0-1, c_1)$ from an interval of the form (ℓ, c_0, c_1) with $c_0 > 0$, and one producing the interval $(\ell+1, c_0, c_1)$ from an interval of the form (ℓ, c_0, c_1) with $c_0 = 0$.
- $\mathtt{I}_\ell =$ `IF` $\mathtt{X}_0$`=0 GOTO` ℓ': We have two rules, one producing the interval (ℓ', c_0, c_1) from an interval of the form (ℓ, c_0, c_1) with $c_0 = 0$, and one producing the interval $(\ell+1, c_0, c_1)$ from an interval of the form (ℓ, c_0, c_1) with $c_0 > 0$.

Then, we have an interval labeled by $k-1$ in the fixed point iff the machine reaches a stopping configuration. □

As already discussed, the undecidability relies both on recursion in the rules and on the map functions having an infinite range. In the following, we show that restricting one of these two aspects allows us to recover decidability. In fact, we give tight complexity bounds for all three fragments. We continue by introducing some necessary notation to formalize these two restrictions.

First, recall that a map of an interval is a partial function from $\mathcal{I}$ to $\mathbb{N} \cup \mathbb{B}$, i.e., it has an infinite range. We will consider the evaluation problem restricted to intervals with maps that are partial functions from $\mathcal{I}$ to $\{0,1,\ldots,k-1\} \cup \mathbb{B}$ with a *bound* k given in binary and all arithmetic operations performed modulo k. We denote the fixed point resulting from these semantics by $T^k_{\mathrm{inc}}[\![_]\!]$.

Second, for a rule $\eta \leftarrow \eta_1 \oplus \eta_2$ **where** Φ **map** Ψ we say that η appears on the left-hand side and the η_i appear on the right-hand side. An `inc-nfer` specification $D \in \Delta^*$ forms a directed graph $G(D)$ over the rules in D such that there is an edge from δ to δ' iff there is an identifier η that appears on the left-hand side of δ and the right-hand side of δ'. We say that D contains a cycle if $G(D)$ contains one; otherwise D is cycle-free.

Example 3. Here, we present an example of an `inc-nfer` specification that captures the natural, even, and odd numbers via intervals. There are many ways to do so with `nfer` rules, but here we choose rules that serve to demonstrate the graph construction described above. Note that, like in our proofs, we use **coincide** rules and only have intervals that begin and end at timestamp zero. As such, we omit timestamps in our notation. Also, all intervals we consider have a single map value v storing an integer.

We define the following rules to compute intervals carrying values in the sets of naturals (labeled N), evens (E), and odds (O). To simplify our presentation, we define a map predicate and two map updates. The map predicate $\Phi_{\mathrm{eq}} = m_1, m_2 \mapsto m_1(v) = m_2(v)$ tests that the map values of the two intervals are equal. The map update $\Psi_{\mathrm{zero}} = m_1, m_2 \mapsto \{v \mapsto 0\}$ sets the map value to zero and the map update $\Psi_{\mathrm{incr}} = m_1, m_2 \mapsto \{v \mapsto m_1(v)+1\}$ sets the map value to one more than the value of the (equal, assuming Φ_{eq} holds) value from the matched intervals.

Now, consider the input trace $\tau = (I, \{v \mapsto 0\})$ and the rules

$$\delta_1 = N \leftarrow I \textbf{ coincide } I \textbf{ where } \Phi_{\mathrm{eq}} \textbf{ map } \Psi_{\mathrm{zero}}$$

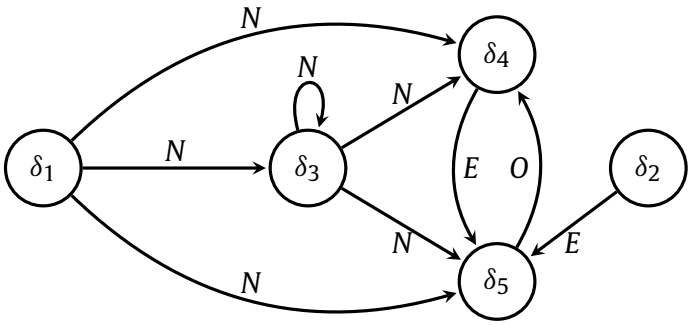

Fig. 2. The directed graph formed by the rules to compute naturals, evens, and odds.

$$\delta_2 = E \leftarrow I \textbf{ coincide } I \textbf{ where } \Phi_{\text{eq}} \textbf{ map } \Psi_{\text{zero}}$$
$$\delta_3 = N \leftarrow N \textbf{ coincide } N \textbf{ where } \Phi_{\text{eq}} \textbf{ map } \Psi_{\text{incr}}$$
$$\delta_4 = E \leftarrow O \textbf{ coincide } N \textbf{ where } \Phi_{\text{eq}} \textbf{ map } \Psi_{\text{incr}}$$
$$\delta_5 = O \leftarrow E \textbf{ coincide } N \textbf{ where } \Phi_{\text{eq}} \textbf{ map } \Psi_{\text{incr}}.$$

These rules form the directed graph shown in Fig. 2. In the graph, edges are labeled by the identifier witnessing their existence. For example, there is an edge from δ_2 to δ_5 labeled by E because E apears on the left-hand side of δ_2 and the right-hand side of δ_5. The graph contains two cycles: one between δ_4 and δ_5 and one with only δ_3, due to its self-loop.

Starting with the initial trace $\tau = (I, \{v \mapsto 0\})$ we obtain the fixed point

$$\begin{aligned} T_{\text{inc}}[\![\delta_1 \cdots \delta_5]\!](\tau) = &\{(N, \{v \mapsto n\}) : n \in \mathbb{N}\} \cup \\ &\{(E, \{v \mapsto n\}) : n \in \mathbb{N} \text{ is even}\} \cup \\ &\{(O, \{v \mapsto n\}) : n \in \mathbb{N} \text{ is odd}\} \cup \{(I, \{v \mapsto 0\})\}. \end{aligned}$$

3.2. Cycle-free, finite-data `Inc-nfer`

We begin our study of decidable fragments of `inc-nfer` by considering both the restriction to cycle-free specifications and to finite data at the same time.

Theorem 2. The cycle-free `inc-nfer` evaluation problem with finite data is PSPACE-complete.

Proof. We only prove the lower bound here, the upper bound is shown for full `nfer` in Theorem 5. We proceed by a reduction from TQBF, the problem of determining whether a formula of quantified propositional logic evaluates to true (see, e.g., [31] for a detailed definition), which is PSPACE-hard. So, fix such a formula φ. Let π_j for $j \geq 1$ denote the j-th prime number. We assume without loss of generality that $\varphi = Q_2 x_2 Q_3 x_3 \cdots Q_{\pi_n} x_{\pi_n} \bigwedge_{i=1}^{m} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ where each Q_{π_j} is in $\{\exists, \forall\}$, and each $\ell_{i,i'}$ is either x_{π_j} or $\neg x_{\pi_j}$ for some j. As we label variables by prime numbers, we can uniquely identify a variable valuation $V \subseteq \{x_{\pi_j} : 1 \leq j \leq n\}$ by the number $\prod_{x_{\pi_j} \in V} \pi_j$. As the map values we will consider only have to encode valuations, and are therefore bounded by $\prod_{j \leq n} \pi_j$, we can use the bound $1 + \prod_{j \leq n} \pi_j$ on the map values we consider.

We present three types of rules:

1. Rules to *generate* every possible variable valuation (encoded by an interval whose map contains the number representing the valuation).
2. A rule to *check* whether a valuation satisfies $\bigwedge_{i=1}^{n} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$.
3. Rules to simulate the quantifier prefix to check whether the full formula evaluates to true.

Let us explain all steps in detail. As all intervals in this proof will have the same timestamps, we will drop those to simplify our notation. Furthermore, the map of an interval will contain a single integer value. For these reasons, we denote intervals by (η, s) where η is an identifier and s is the map value.

To generate the valuations, we start with the trace containing only a single fixed event that yields the interval $(G_0, 1)$. Further, for $1 \leq j \leq n$ we have rules producing the intervals $(G_j, s \cdot \pi_j)$ and (G_j, s) from an interval of the form (G_{j-1}, s) for some s. The fixed point reached by applying these rules contains the 2^n intervals of the form (G_n, s) where s encodes a variable valuation.

In the valuation encoded by some s, a variable x_{π_j} evaluates to true if $s \bmod \pi_j = 0$ and evaluates to false if $s \bmod \pi_j \neq 0$. Hence, to check whether the valuation encoded by some s satisfies $\bigwedge_{i=1}^{m}(\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ we have a rule that produces the interval (C_n, s) from an interval of the form (G_n, s) for some s such that $\bigwedge_{i=1}^{m}(\psi_{i,1} \vee \psi_{i,2} \vee \psi_{i,3})$ evaluates to true, where $\psi_{i,i'}$ is equal to $s \bmod \pi_j = 0$ if $\ell_{i,i'} = x_{\pi_j}$, and where $\psi_{i,i'}$ is equal to $s \bmod \pi_j > 0$ if $\ell_{i,i'} = \neg x_{\pi_j}$.

We now simulate the quantifier prefix. Intuitively, we check whether partial variable valuations cause the formula to hold. We do so by the following rules: If the variable x_{π_j} is existentially quantified, we have a rule producing the interval (C_{j-1}, s) from an interval of the form (C_j, s) with $s \bmod \pi_j > 0$, and a rule producing the interval $(C_{j-1}, s/\pi_j)$ from an interval of the form (C_j, s) with $s \bmod \pi_j = 0$. So, to generate an interval labeled by C_{j-1} at least one interval labeled by C_j has to exist, and their maps must be compatible.

Finally, if the variable x_{π_j} is universally quantified, we have a rule producing the interval (C_{j-1}, s) from two intervals of the form (C_j, s) and $(C_j, s \cdot \pi_j)$ (which can be done using a **coincide**-rule). Thus, to obtain an interval labeled by C_{j-1} both intervals labeled by C_j with corresponding map values have to exist.

An induction shows that a partial valuation $V \subseteq \{x_{\pi_j} : 1 \leq j \leq n'\}$ for some $0 \leq n' \leq n$ satisfies $Q_{\pi_{n'+1}} x_{\pi_{n'+1}} \cdots Q_{\pi_n} x_{\pi_n} \bigwedge_{i=1}^{m}(\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ iff the interval $(C_{n'}, \prod_{x_{\pi_j} \in V} \pi_j)$ is generated by applying these rules. So, for $n' = 0$ we obtain the correctness of our reduction: The formula φ evaluates to true iff $(C_0, 1)$ is in the fixed point induced by the rules above.

Furthermore, the rules above are cycle-free, there are linearly many rules in the number n of variables and each rule is of polynomial size in the size of φ. Finally, as $\pi_j \leq j(\ln j + \ln\ln j)$ for all $j \geq 6$ [32], all numbers appearing in the maps of the intervals are bounded by

$$\prod_{j=1}^{n} \pi_j \leq c \cdot \prod_{j=1}^{n} j(\ln j + \ln\ln j) \leq c \cdot (n(\ln n + \ln\ln n))^n$$

whose binary representation is polynomial in the size of φ. Here, c is some constant that is independent of n. □

Now, we turn our attention to the remaining two fragments obtained by considering finite-data with cycles and cycle-free specifications with infinite data. In both cases, we again prove tight complexity bounds. For both upper bounds, we rely on algorithms searching for witnesses for the existence of an interval in the fixed point. As these arguments are used in multiple proofs, we introduce them first in a general format. So, fix some specification D and some trace τ of events. If i is an interval in $T_{\text{inc}}[\![D]\!]\,\tau$, then either there is an event e in τ such that $\mathit{init}(e) = i$ (we say that i is initial in this case) or there are intervals i_1, i_2 in $T_{\text{inc}}[\![D]\!]\,\tau$ and a rule $\delta \in D$ such that i is obtained by applying δ to i_1 and i_2. So, for every interval i_0 in $T_{\text{inc}}[\![D]\!]\,\tau$ there is a binary (witness) tree whose nodes are labeled by intervals in $T_{\text{inc}}[\![D]\!]\,\tau$, whose root is labeled by i_0, whose leaves are labeled by initial intervals, and where the children of a node labeled by i are labeled by i_1 and i_2 such that there is a rule δ so that i is obtained by applying δ to i_1 and i_2. Note that the tree might contain multiple occurrences of the same interval. But, we can assume without loss of generality that each path in the tree does not contain a repetition of an interval (if it does we can just remove the part of the tree between the repetitions). Hence, the height of the tree is bounded by the number of intervals, which might be infinite in the case of infinite data. Furthermore, if D is cycle-free then the height of a witness tree is also bounded by the number of rules in D. Note that the same arguments also apply to $T^k_{\text{inc}}[\![D]\!]\,\tau$ in case we deal with finite data.

Proposition 1. An interval is in $T_{\text{inc}}[\![D]\!]\,\tau$ ($T^k_{\text{inc}}[\![D]\!]\,\tau$) iff it has a witness tree.

3.3. Finite-data `Inc-nfer`, with cycles

We continue by settling the case of specifications with cycles, but restricted to finite data.

Theorem 3. The `inc-nfer` evaluation problem with finite data is ExpTime-complete.

Proof. We first prove the lower bound by reducing from the word problem for alternating polynomial space Turing machines (see, e.g., [31] for detailed definitions). As ExpTime = APSpace, this yields the desired lower bound.

Recall that an alternating Turing machine has both existential states and universal states and a run of such a machine is a tree labeled by configurations. An accepting run is finite, labeled by the initial configuration in the root, every vertex labeled by an existential configuration (i.e., one with an existential state) has one child labeled by a successor configuration, every vertex labeled by a universal configuration has for all successor configurations a child labeled by it, and all leaves are labeled by accepting configurations. We refer to paths through such a (tree-like) run as run-branches.

So, fix an alternating polynomial space Turing machine $\mathcal{M}$, i.e., there is some polynomial p such that $\mathcal{M}$ uses at most space $p(|w|)$ when started on input w. Let us also fix some input w for $\mathcal{M}$. We construct an instance of `inc-nfer` that simulates a run of $\mathcal{M}$ on w. To simplify our construction, we make some assumptions (all without loss of generality):

- The set Q of states of $\mathcal{M}$ is of the form $\{1, 2, \ldots, |Q|\}$ and 1 is the initial state.
- The tape alphabet Γ of $\mathcal{M}$ is equal to $\{0, 1, \ldots, 9\}$ and 0 is the blank symbol.

- Every run of $\mathcal{M}$ has only finite branches, i.e., $\mathcal{M}$ terminates on every input. To this end, we assume the existence of a set of terminal states, which is split into accepting and rejecting ones.
- Every nonterminal configuration (one with a nonterminal state) has exactly two successor configurations. Such states are either existential or universal.

So, a configuration of $\mathcal{M}$ is of the form ℓqr with $q \in Q$ and $\ell, r \in \Gamma^*$ such that $|\ell| + |r| = p(|w|)$, with the convention that the head is on the first letter of r.

For $c \in \Gamma^*$, let c^R denote the reverse of c. Due to our assumption on Γ we can treat ℓ and r^R as natural numbers encoded in base ten. We uniquely identify a configuration ℓqr by the triple (ℓ, q, r^R) of natural numbers. The initial configuration of $\mathcal{M}$ on w is encoded by the triple $(0, 1, w^R)$ representing that the tape to the left of the head has only blanks, the machine is in the initial state 1, and w is to the right of the head with the remaining cells of the tape being blank.

This encoding allows us to read the tape cell the head is currently pointing to, update the tape cell the head is pointing to, and move the head by simple arithmetic operations. For example, whether the head points to a cell containing a 3 is captured by $r^R \bmod 10$ being 3, and writing a 7 to the cell pointed to by the head is captured by adding $-(r^R \bmod 10) + 7$ to r^R. Finally, moving the head to, say, the right, is captured by multiplying ℓ by 10 and then adding $r^R \bmod 10$ to it, and then dividing r^R by 10 (which is done without remainder and therefore removes the last digit of r^R). In the following, we use intervals of the form (A, ℓ, q, r^R) to encode the configuration ℓqr of $\mathcal{M}$. Here, A is some identifier and we disregard timestamps, as all intervals have the same start and end. Hence, ℓ, q, and r^R are three map values of the interval.

We now describe the rules simulating $\mathcal{M}$ on w. We start with some fixed event that yields the interval $(G, 0, 1, w^R)$ encoding the initial configuration. As described above, the computation of a successor configuration can be implemented using arithmetic operations. Thus, given the interval encoding the initial configuration, one can write rules (one for each transition of $\mathcal{M}$) that generate the set of all configurations, encoded as intervals of the form (G, ℓ, q, r^R). Furthermore, one can write a rule that produces the interval (A, ℓ, q, r^R) from every interval (G, ℓ, q, r^R) with an accepting q.

Now, we describe rules to compute the set of accepting configurations, i.e., the smallest set A of configurations that contains all those with an accepting terminal state, all existential ones that have a successor in A, and all universal ones that have both successors in A. For every transition t from an existential state q, there is a rule to produce the interval (A, ℓ, q, r^R) if the intervals (G, ℓ, q, r^R) and $(A, \ell', q', r^{R'})$ already exist, where $(\ell', q', r^{R'})$ encodes the configuration obtained by applying the transition t to the configuration encoded by (ℓ, q, r^R). Thus, to declare an existential configuration as accepting at least one of its successor configurations has to be already declared as accepting.

Now, let us consider universal configurations. Due to our assumption, for every pair of a state and a tape symbol, there are exactly two transitions t_1 and t_2 that are applicable. There are two rules for this situation. The first one produces the interval (B, ℓ, q, r^R) if the intervals (G, ℓ, q, r^R) and $(A, \ell', q', r^{R'})$ already exist, where $(\ell', q', r^{R'})$ encodes the configuration obtained by applying the transition t_1 to the configuration encoded by (ℓ, q, r^R). The second one produces the interval (A, ℓ, q, r^R) if the intervals (B, ℓ, q, r^R) and $(A, \ell', q', r^{R'})$ already exist, where $(\ell', q', r^{R'})$ encodes the configuration obtained by applying the transition t_2 to the configuration encoded by (ℓ, q, r^R). Thus, to declare a universal configuration as accepting both of its successor configurations have to be already declared as accepting.

Finally, there is a rule producing an interval with identifier η_T from the interval $(A, 0, 1, w^R)$, indicating that the initial configuration is accepting. Thus, the fixed point contains an interval labeled by η_T iff $\mathcal{M}$ accepts w.

It remains to show that the specification has the required properties. It is of polynomial size and each rule has polynomial size (both measured in $|\mathcal{M}| + |w|$). Further, all numbers used in the intervals are bounded by $\max\{|Q|, 10^{p(|w|)}\}$, whose binary representation is bounded polynomially in $|\mathcal{M}| + |w|$.

Now, we prove the upper bound. We are given a specification D, an input trace τ of events, a $k \in \mathbb{N}$ (given in binary), and a target label η_T and have to determine whether the fixed point $T^k_{\text{inc}}[\![D]\!]\,\tau$ contains an interval labeled by η_T. We describe an alternating polynomial space Turing machine solving this problem by searching for a witness tree. APSPACE = EXPTIME yields the result.

To this end, we rely on the following properties.

1. Every interval in $T^k_{\text{inc}}[\![D]\!]\,\tau$ can be stored in polynomial space, as every value in its map can be stored using $\log k$ bits, and there are only linearly many such values (measured in $|D| + |\tau|$).
2. There are only exponentially many intervals in $T^k_{\text{inc}}[\![D]\!]\,\tau$, e.g.,

$$b(D, \tau, k) = \iota \cdot t^2 \cdot k^{|D|+|\tau|} \leq \iota \cdot |\tau|^2 \cdot 2^{(\log k)(|D|+|\tau|)}$$

   is a crude upper bound. Here, ι is the number of identifiers appearing in D and τ and t is the number of timestamps in τ (recall that `inc-nfer` does not create new timestamps).
3. Given three intervals i, i_1, i_2 and a rule $\delta \in D$ one can determine in polynomial space whether i is obtained by applying δ to i_1 and i_2.

Using alternation, Algorithm 1 determines whether a witness tree exists whose root is labeled by η_T and whose height is bounded by $b(D, \tau, k)$. Due to Proposition 1, this is equivalent to an interval labeled by η_T being in $T^k_{\text{inc}}[\![D]\!]\,\tau$. Due to the above properties, one can easily implement the algorithm on an alternating polynomial space Turing machine, yielding the desired upper bound. □

Algorithm 1 Algorithm checking the existence of a witness tree.

Input: Specification D, trace τ, bound k, target identifier η_T
1: $n := 0$
2: **nondeterministically guess** interval i labeled by η_T
3: **while** $n < b(D, \tau, k)$ **and** i is not initial **do**
4: $n := n + 1$
5: **nondeterministically guess** intervals i_1, i_2 and $\delta \in D$ such that i is obtained by applying δ to i_1 and i_2
6: **universally pick** $i := i_j$ for $j \in 1, 2$
7: **if** i is initial **then return** accept
8: **else return** reject

3.4. Cycle-free, infinite data `Inc-nfer`

Finally, we consider the last fragment: cycle-free specifications with infinite data. A crucial aspect here is that cycle-free specifications imply an upper bound on the map values of intervals in the fixed point, as each interval in the fixed point can be generated by applying each rule at most once. For the lower bound, we generate *large* numbers using a set of cycle-free rules (cf. Example 1) and encode configurations using these numbers as before.

Theorem 4. The cycle-free `inc-nfer` evaluation problem with infinite data is NExpTime-complete.

Proof. We begin with the lower bound and reduce from the word problem for nondeterministic exponential-time Turing machines, which is NExpTime-hard. Thus, fix such a machine $\mathcal{M}$, i.e., there is some polynomial p such that $\mathcal{M}$ uses at most time $2^{p(|w|)}$ when started on input w. Let us also fix some input w for $\mathcal{M}$ and define $n = p(|w|)$. We construct an instance of `inc-nfer` that simulates the run of $\mathcal{M}$ on w. To simplify our construction, we make some assumptions (all without loss of generality):

- The set Q of states of $\mathcal{M}$ is of the form $\{1, 2, \ldots, |Q|\}$ and 1 is the initial state.
- The tape alphabet Γ of $\mathcal{M}$ is equal to $\{0, 1, \ldots, 9\}$ and 0 is the blank symbol.
- Accepting states are equipped with a self-loop. This allows us to restrict our attention to runs of length 2^n.

A configuration of $\mathcal{M}$ is of the form $\ell q r$ with $q \in Q$ and $\ell, r \in \Gamma^*$ such that $|\ell| + |r| \leq 2^n$, with the convention that the head of $\mathcal{M}$ is on the first letter of r. So, all the numbers encoding configurations are bounded by $b = 2^{2^{n+2}} \geq 10^{2^n}$.

First we describe how to generate all natural numbers up to b using cycle-free rules (as before, we drop the timestamps and only write the identifier and map values). This construction is based on the fact that squaring two k times yields 2^{2^k} (cf. Example 1): We start with an initial interval $(N_0, 1)$ (coming from the input trace) and have, for every $1 \leq j \leq n+2$, a rule producing the interval $(N_j, s^2 + s')$ from the intervals (N_{j-1}, s) and (N_{j-1}, s'). An induction shows that applying these rules until a fixed point is reached generates all intervals (N_{n+2}, s) with $1 \leq s \leq b$. Thus, we can also write a rule generating all intervals of the form (C, ℓ, q, r^R) with $\ell + r \leq b$. Thus, each C-labeled interval encodes a configuration of $\mathcal{M}$.

Now, we piece together a run out of configurations as follows: We have a rule producing the interval $(C_0, \ell, q, r^R, \ell, q, r^R)$ from an interval of the form (C, ℓ, q, r^R). The resulting interval represents a run infix of length $1 = 2^0$.

For all $0 < j \leq n$ we have a rule producing the interval

$$(C_j, \ell_0, q_0, r_0^R, \ell_3, q_3, r_3^R)$$

from two intervals of the form

$$(C_{j-1}, \ell_0, q, r_0^R, \ell_1, q_1, r_1^R)$$

and

$$(C_{j-1}, \ell_2, q_2, r_2^R, \ell_3, q_3, r_3^R)$$

such that $\ell_2 q_2 r_2$ encodes a successor configuration of the configuration encoded by $\ell_1 q_1 r_1$. These rules allow to combine two run infixes of length 2^{j-1} into a one of length 2^j, provided the second one starts with a successor configuration of the first one. As we can double the length of the run infix with each rule application, a linear number of rules suffices to construct a full run of exponential length.

Thus, we have a rule producing the interval (η_T) from an interval of the form

$$(C_0, \ell_0, q_0, r_0^R, \ell_1, q_1, r_1^R)$$

such that (ℓ_0, q_0, r_0^R) encodes the initial configuration of $\mathcal{M}$ on w and q_1 is an accepting state. An induction shows that an interval labeled with η_T is generated iff $\mathcal{M}$ has an accepting run on w.

For the matching upper bound, we show that a witness tree for an interval can be found in nondeterministic exponential time. This is the based on the following properties for a given specification D and a given trace τ of events. As D is cycle-free, every interval in $T_{\text{inc}}[\![D]\!]\,\tau$ has a witness tree whose height is bounded by $|D|$. Now, if an interval i is obtained from intervals i_1, i_2 by the application of some rule $\delta \in D$, the maximal map value in i is only polynomial larger than the maximal map value in i_1 and i_2, with the polynomial only depending on δ. This is because map updates are limited to simple arithmetic operations including multiplication but excluding exponentiation.

Hence, there is some polynomial p (dependent on D and τ) such that all map values of intervals in $T_{\text{inc}}[\![D]\!]\,\tau$ are bounded by $2^{2^{p(|D|+|\tau|)}}$. Thus, every interval in $T_{\text{inc}}[\![D]\!]\,\tau$ can be stored (with map values encoded in binary) in exponential space (measured in the size of D and τ).

Furthermore, as the height of a witness tree (which is a binary tree) can be bounded by $|D|$, its size can be bounded by $2^{|D|}$. Combining both bounds shows that each interval in $T_{\text{inc}}[\![D]\!]\,\tau$ has a witness tree that can be encoded using an exponential number of bits. Hence, given D, τ, and a target label η_T we can guess a tree and verify that it is indeed a witness tree for some η_T-labeled interval in exponential time. □

4. The full nfer language

This section introduces the second type of `nfer` rules, called *exclusive rules*, that test for the existence of one interval and the absence of another interval matching constraints. These rules were introduced in [3] and they, together with inclusive rules, complete the `nfer` language. We define the syntax of these rules, including mathematical functions to simplify the presentation, as follows:

$$\eta \leftarrow \eta_1 \textbf{ unless } \ominus\ \eta_2 \textbf{ where } \Phi \textbf{ map } \Psi$$

where $\eta, \eta_1, \eta_2 \in \mathcal{I}$ are identifiers, $\ominus \in \{\textbf{after}, \textbf{follow}, \textbf{contain}\}$ is a *clock predicate* on two intervals (one for each of η_1 and η_2), and Φ and Ψ are the same as in inclusive rules. We say that an exclusive rule *includes* η_1 and *excludes* η_2.

4.1. Semantics

Exclusive rules share many features with inclusive rules but they require additions to the `inc-nfer` semantics that were omitted in Section 2 for brevity. Notably, these changes to the semantics produce equivalent results when evaluating inclusive rules. The following definition gives semantics to exclusive rules:

$$\begin{aligned}
&R\,[\![\,\eta \leftarrow \eta_1 \textbf{ unless } \ominus \eta_2 \textbf{ where } \Phi \textbf{ map } \Psi\,]\!]\,\pi \;=\\
&\quad \{\; i \in \mathbb{I} \;:\; i_1 \in \pi \;\;.\;\; id(i) = \eta \wedge\; id(i_1) = \eta_1 \wedge\\
&\qquad\qquad start(i) = start(i_1) \wedge end(i) = end(i_1) \wedge\\
&\qquad\qquad map(i) = \Psi(map(i_1), \{\,\}) \wedge\\
&\qquad\qquad \neg\ (\,\exists\ i_2 \in \pi \;\;.\;\; i_2 \neq i_1 \wedge\; id(i_2) = \eta_2 \;\wedge\\
&\qquad\qquad\quad \ominus(i_1, i_2) \wedge \Phi(map(i_1), map(i_2))\ )\ \}
\end{aligned}$$

Like with inclusive rules, exclusive rules match intervals in the input pool π to produce a pool of new intervals. The difference is that exclusive rules produce new intervals where one existing interval in π matches the identifier η_1 and no intervals exist that match the identifier η_2 such that the clock predicate $\ominus$ and the map predicate Φ hold for the η_1-labeled and the η_2-labeled interval.

Remark 2. This definition is more general than the exclusive rule semantics in [3] where intervals with excluded identifiers were limited to those with end timestamps strictly less than the extant interval ($end(i_2) < end(i_1)$). Here, we instead enforce only that an interval does not exclude itself ($i_2 \neq i_1$). The restriction on end timestamps was originally introduced to facilitate completeness in an online monitoring algorithm but is more restrictive than necessary for the offline semantics we present here.

The three possibilities referenced by $\ominus$ are shown in Fig. 3 and formally defined in Table 3. These clock predicates relate two intervals using familiar ATL temporal operators while the timestamps of the produced interval are copied from the included interval rather than being defined by the clock predicate. In the figure, the excluded interval labeled C is shown as a rectangle with a dotted outline and the produced interval labeled A is always the same as the included interval labeled B. For example, given intervals i, i_1, i_2 where $id(i) = A$, $id(i_1) = B$ and $id(i_2) = C$, $A \leftarrow B$ **unless follow** C holds when $end(i_2) = start(i_1)$, $start(i) = start(i_1)$, and $end(i) = end(i_1)$.

To ensure that evaluating an `nfer` specification yields a unique fixed point, exclusive rules may not appear in cycles because the intervals they produce depend on the persistent non-existence of other intervals. When cycles exist in an `inc-nfer` specification, rules are evaluated multiple times and each evaluation may add intervals. Exclusive rules may have non-deterministic behavior in a cycle because the intervals they exclude may be produced either before or after the exclusive rule is evaluated. Note that it is possible to evaluate specifications with both cycles *among inclusive rules*

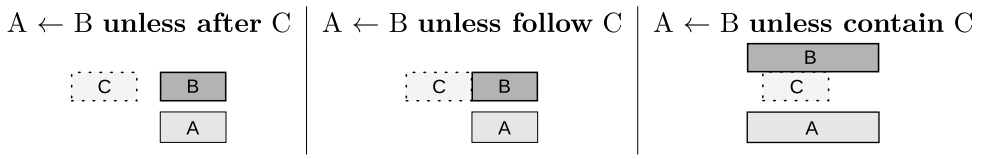

Fig. 3. `nfer` clock predicates for exclusive rules.

Table 3
Formal definition of `nfer` clock predicates for exclusive rules.

Syntax	Definition of $\ominus(i_1, i_2)$
after	$start(i_1) > end(i_2)$
follow	$start(i_1) = end(i_2)$
contain	$start(i_2) \geq start(i_1) \wedge end(i_2) \leq end(i_1)$

and exclusive rules (not appearing in the cycles) with deterministic behavior and in Section 6 we introduce an extended semantics that supports this case. Here, we first consider the case of cycle-free specifications with exclusive rules.

The order in which rules are evaluated may also affect the result of applying exclusive rules for this reason, which motivates a generalization of the $T_{\text{inc}}[\![_]\!]$ (resp. $T^k_{\text{inc}}[\![_]\!]$) function.

$$T_{\text{full}}[\![_]\!] : \Delta^* \to \mathbb{P} \to \mathbb{P}$$

$$T_{\text{full}}[\![\ \delta_1 \cdots \delta_n\ ]\!]\,\pi = \begin{cases} S[\![\ topsort(\delta_1 \cdots \delta_n)\ ]\!]\,(\pi) & \textbf{if } \exists\, topsort(\delta_1 \cdots \delta_n) \\ T_{\text{inc}}[\![\ \delta_1 \cdots \delta_n\ ]\!]\,(\pi) & \textbf{otherwise} \end{cases}$$

where *topsort* is a topological sort of the directed graph $G(D)$ described in Section 3 and $T_{\text{inc}}[\![_]\!]$ is the interpretation function defined in Section 2.3. A topological sort, which can be computed in linear time [33], only exists in a cycle-free specification. In that case, *topsort* orders the rules such that the fixed-point computation of $T_{\text{inc}}[\![_]\!]$ can be short-circuited, since one application of $S[\![_]\!]$ is sufficient to produce the final pool. The results of $T_{\text{full}}[\![_]\!]$ are independent of the topological sort, as any such ordering will guarantee that all intervals matched by a rule exist before it is applied using $R[\![_]\!]$.

Example 4. From the trace $(N, \{v \mapsto 0\}) \cdots (N, \{v \mapsto 100\})$, the rule

$$E \leftarrow N \textbf{ coincide } N \textbf{ where } m_1, m_2 \mapsto m_1(v) \bmod 2 = 0 \textbf{ map } m_1, m_2 \mapsto \{v \mapsto m_1(v)\}$$

produces the set $\{(E, \{v \mapsto n\}) : n \leq 100 \text{ is even}\}$. Here, m_i for $i \in \{1, 2\}$ is the map of the interval matched with the i-th N on the right-hand side of the rule. Then, the (exclusive) rule

$$O \leftarrow N \textbf{ unless contain } E \textbf{ where } m_1, m_2 \mapsto m_1(v) = m_2(v) \textbf{ map } m_1 \mapsto \{v \mapsto m_1(v)\}$$

yields the set $\{(N, \{v \mapsto n\}) : n \leq 100 \text{ is odd}\}$. Here, m_1 is the map of the interval matched with N while m_2 is the interval matched with E. So, the exclusive rule intuitively describes the complement of the E-labeled intervals with respect to the N-labeled intervals (which are all identifiable by numbers $0 \leq n \leq 100$).

5. Complexity results for full nfer

In the following, we study the complexity of the cycle-free `nfer` evaluation problem with finite and infinite data, starting with the former.

Theorem 5. The cycle-free `nfer` evaluation problem with finite data is PSPACE-complete.

Proof. The lower bound already holds for the special case of `inc-nfer` (see Theorem 2), so we only need to prove the upper bound. To this end, we show how to witness in alternating polynomial time that a given interval is in the fixed point, which yields the desired bound due to APTIME = PSPACE. Note that we cannot just search for a witness tree as for `inc-nfer`, as we also have to handle exclusive rules.

Intuitively, an exclusive rule requires the existence of one interval in the fixed point and the non-existence of other intervals in the fixed point. We have seen how to capture existence of an interval via the existence of a witness tree. Hence, we can capture the non-existence of an interval via the non-existence of a witness tree. As we construct an alternating algorithm, we use duality to capture the non-existence of a witness tree and switch between an existential and a universal mode every time the non-existence of an interval is to be checked.

Algorithm 2 Solving the cycle-free `nfer` evaluation problem with finite data.

Input: Specification D, trace τ, target identifier η_T
1: $n := 0$, $f := 0$
2: **nondeterministically guess** interval i labeled by η_T
3: **while** $n < |D|$ **and** i is not initial **do**
4: $n := n + 1$
5: **if** $f = 0$ **then**
6: **nondeterministically guess** rule $\delta \in D$
7: **if** δ is inclusive **then**
8: **nondeterministically guess** intervals i_1, i_2 such that i is obtained by applying δ to i_1 and i_2
9: **universally pick** $i := i_j$ for $j \in 1, 2$
10: **else** // δ is exclusive
11: **nondeterministically guess** interval i_1 such that i is obtained by applying δ to i_1
12: **universally pick** an interval $i_2 \in \{i' : \delta$ includes i_1 and excludes $i'\} \cup \{\bot\}$
13: **if** $i_2 = \bot$ **then**
14: $i := i_1$
15: **else**
16: **universally pick** $i := i_j$ for $j \in \{1, 2\}$
17: **if** $i = i_2$ **then**
18: $f := 1 - f$
19: **else** // $f = 1$
20: **universally pick** rule $\delta \in D$
21: **if** δ is inclusive **then**
22: **universally pick** intervals i_1, i_2 such that i is obtained by applying δ to i_1 and i_2
23: **nondeterministically guess** $i := i_j$ for some $j \in 1, 2$
24: **else** // δ is exclusive
25: **universally pick** interval i_1 such that i is obtained by applying δ to i_1
26: **nondeterministically guess** an interval $i_2 \in \{i' : \delta$ includes i_1 and excludes $i'\} \cup \{\bot\}$
27: **if** $i_2 = \bot$ **then**
28: $i := i_1$
29: **else**
30: **nondeterministically guess** $i := i_j$ for $j \in \{1, 2\}$
31: **if** $i = i_2$ **then**
32: $f := 1 - f$
33: **if** i is initial **and** $f = 0$ **then**
34: **return** accept
35: **else**
36: **return** reject

Algorithm 2 keeps track of a single interval and applies rules in a backwards fashion. Using alternation, it guesses and verifies a tree structure witnessing the (non-)existence of intervals in the fixed point. To simulate exclusive rules, it uses a Boolean flag f to keep track of the parity of the number of exclusive rules that have been simulated, initialized with zero. If f is zero, then a rule δ is guessed nondeterministically. If this rule is inclusive, two intervals i_1 and i_2 are guessed nondeterministically such that the current interval i is obtained from i_1 and i_2 by applying δ. Then, the current interval is updated by universally picking $i := i_1$ or $i := i_2$, so that both choices are checked. This case is similar to Algorithm 1.

On the other hand, if the rule is exclusive, then a single interval i_1 is guessed nondeterministically and another interval i_2 is picked universally so that δ includes i_1, excludes i_2, and i is the result of applying δ to i_1. Now, the current interval is updated by universally picking $i := i_1$ or $i := i_2$, so that both choices are checked. In the second case, the flag is toggled to signify that another exclusive rule is simulated.

In the case where f is equal to one, the approach is just dual, i.e., we switch existential and universal choices. As the input specification is cycle-free, we need to simulate at most $|D|$ applications of a rule. Finally, acceptance depends on the value of the flag, i.e., while the flag is zero the last interval has to be initial (i.e., in the input trace) while it has to be non-initial if the flag is one.

The algorithm runs in alternating polynomial time as each run simulates at most $|D|$ rule applications and each application can be implemented in deterministic polynomial time due to the encodings of the map values and time stamps being bounded by $|D| + |\tau| + \log(k)$, where k is the bound on the map values. □

5.1. Infinite data full `nfer`

We now consider the case of infinite data. Here, the bound we obtain is AExpTime(poly), the class of problems decided by alternating exponential-time Turing machines with a polynomial number of alternations between existential and universal states.

Theorem 6. The cycle-free `nfer` evaluation problem with infinite data is AExpTime(poly)-complete.

Proof. We begin with the lower bound, showing a reduction from the word problem for alternating exponential-time Turing machines with a polynomial number of alternations. To this end, fix such a machine $\mathcal{M}$, i.e., there are two polynomials p, p' such that $\mathcal{M}$ uses at most time $2^{p(|w|)}$ and at most $p'(|w|)$ alternations when started on input w. We fix an input w and define $n = p(|w|)$ and $m = p'(|w|)$. We construct a cycle-free `nfer` instance with infinite data that simulates a run of $\mathcal{M}$ on w.

Recall that in the proof of Theorem 4, we have shown that a *nondeterministic* exponential-time Turing machine can be simulated by a cycle-free `inc-nfer` instance (assuming infinite data). Here, we show that exclusive rules can simulate alternation.

As before, we make some simplifying assumptions on $\mathcal{M}$ (all without loss of generality):

- The set Q of states of $\mathcal{M}$ is of the form $\{1, 2, \dots, |Q|\}$ and 1 is the initial state.
- The tape alphabet Γ of $\mathcal{M}$ is equal to $\{0, 1, \dots, 9\}$ and 0 is the blank symbol.
- All accepting states are existential and every branch of a run of $\mathcal{M}$ on w has exactly m alternations.

As in the proof of Theorem 4, we encode a configuration of $\mathcal{M}$ as $\ell q r^R$ where q is the state and $\ell, r^R \in \Gamma^*$ are treated as natural numbers bounded by $2^{2^{n+2}}$ encoding the tape to the left of the reading head (ℓ) and the reverse of the tape to the right of the reading head (r^R).

As shown in the proof of Theorem 4, we can construct a set D of cycle-free inclusive rules and a trace τ such that the fixed point $T_{\text{inc}}[\![D]\!]\,\tau$ encodes all those pairs (c, c') of configurations of a nondeterministic Turing machine such that c' is reachable from c via a run of length at most 2^n. As usual, here, all intervals have the same time stamps, which is the reason we drop them from the notation. In this proof, we work with tree-like runs, as $\mathcal{M}$ is an alternating Turing machine. In the following, we need to distinguish between branches of such runs using only existential and only universal states. To this end, we use the identifiers $\exists$ and $\forall$. We adapt the rules from the proof of Theorem 2 such that

- the interval $(\exists, \ell_0, q_0, r_0^R, \ell_1, q_1, r_1^R)$ is in the fixed point iff the configuration $\ell_1 q_1 r_1$ is reachable from the configuration $\ell_0 q_0 r_0$ via a branch of length at most 2^n such that all states (including q_0 and q_1) are existential, and
- the interval $(\forall, \ell_0, q_0, r_0^R, \ell_1, q_1, r_1^R)$ is in the fixed point iff the configuration $\ell_1 q_1 r_1$ is reachable from the configuration $\ell_0 q_0 r_0$ via a branch of length at most 2^n such that all states (including q_0 and q_1) are universal.

Recall that m is the number of alternations during every branch of a run on our fixed input w and that all accepting states are existential. In the following, we show how to inductively *compute* the set of configurations from which an accepting subrun of length at most 2^n and with $m' \leq m$ alternations starts, using cycle-free rules only.

For $m' = 0$, this is the set of existential configurations that can reach an accepting one via existential configurations only. Hence, we have a rule that turns an interval of the form

$$(\exists, \ell_0, q_0, r_0^R, \ell_1, q_1, r_1^R)$$

with accepting q_1 into the interval $(L_0, \ell_0, q_0, r_0^R)$.

Now, using an exclusive rule we can obtain the intervals of the form $(\overline{L_0}, \ell_0, q_0, r_0^R)$ such that $(L_0, \ell_0, q_0, r_0^R)$ has not been generated so far, i.e., we generate the complement (cf. Example 4). This is the set of existential configurations that cannot reach an accepting one via existential configurations only. Here, we first need to generate intervals encoding all (exponentially-sized) configurations of $\mathcal{M}$, which can easily be achieved with cycle-free rules as described in the proof of Theorem 4.

Next, we have a rule that turns an interval

$$(\forall, \ell_0, q_0, r_0^R, \ell_1, q_1, r_1^R)$$

and an interval $(\overline{L_0}, \ell_2, q_2, r_2^R)$ such that $\ell_2 q_2 r_2$ is a successor of $\ell_1 q_1 r_1$ into the interval $(\overline{L_1}, \ell_0, q_0, r_0^R)$. Thus, the intervals with the label $\overline{L_1}$ encode those universal configurations that have a branch that ends in an existential configuration from which no accepting configuration is reachable. These are exactly the configurations where no accepting subrun with one alternation starts.

Hence, another exclusive rule yields the intervals $(L_1, \ell_0, q_0, r_0^R)$ such that $(\overline{L_1}, \ell_0, q_0, r_0^R)$ has not been not generated so far, i.e., exactly the configurations where an accepting subrun with one alternation starts.

Now, we are in a dual situation: We have a rule that turns an interval

$$(\exists, \ell_0, q_0, r_0^R, \ell_1, q_1, r_1^R)$$

and an interval $(L_1, \ell_2, q_2, r_2^R)$ such that ℓ_2, q_2, r_2 is a successor of $\ell_1 q_1, r_1$ into the interval $(L_2, \ell_0, q_0, r_0^R)$. Thus, the intervals with the label L_2 encode those existential configurations that have a branch that ends in an universal configuration from

which an accepting subrun with one alternation starts. These are exactly the configurations where an accepting subrun with two alternation starts.

We repeat this construction with labels $L_{m'}$ for every $m' \leq m$. Finally, there is a rule producing the target interval η_T from the interval (L_m, ℓ, q, r^R) if ℓqr is the initial configuration.

Thus, the fixed point contains the interval (η_T) iff $\mathcal{M}$ accepts w. Note that all the rules we have constructed are indeed cycle-free (just consider the order of appearance in the description above). To conclude the proof, it remains to note that there are only polynomially many rules, which are all of polynomial size.

For the upper bound, we again use the algorithm described in the proof of Theorem 5, with the exponential upper bound on the runtime relying on the upper bound on the map values as in Theorem 4, which holds here as well. Hence, we obtain an alternating algorithm with exponential running time (due to the fact that we need to perform calculations on exponentially-sized binary numbers), but with only linearly many (at most $|D|$) alternations. □

6. The extended nfer language

There is a further generalization of the semantics of `nfer` that permits cycles in the same specification as exclusive rules so long as the exclusive rules are not part of a cycle. We differentiate this extension of the semantics by the name `extended-nfer`, and abbreviate it as `ext-nfer`. This relaxation of the restrictions on the use of exclusive rules is desirable to support more use-cases for the language. Since it is possible to obtain deterministic semantics for such specifications, an extended version of `nfer` could support this. This change to the semantics is also interesting for our purposes, since the evaluation complexity of `inc-nfer` with cycles is higher than the cycle-free evaluation complexity of `nfer` with exclusive rules.

6.1. Semantics

To support cycles and exclusive rules in the same specification, we modify $T_{\text{full}}[\![_]\!]$ to compute the strongly-connected components of the directed graph $G(D)$ described in Section 3. Recall that this graph has a vertex for each rule and edges from rules with an identifier on their left-hand side to rules with the same identifier on their right-hand side. The strongly connected components of the graph represent rules included in cycles (*non-trivial* components) or individual rules outside of cycles (*trivial* components). We interpret the rules from each component as a specification with cycles and ensure deterministic behavior from exclusive rules by evaluating the components in topological sort order.

$$
\begin{aligned}
&T_{\text{ext}}[\![_]\!] : \Delta^* \to \mathbb{P} \to \mathbb{P} \\
&T_{\text{ext}}[\![\delta_1 \cdots \delta_n]\!]\pi = \pi_{\ell+1} \centerdot\ \pi_1 = \pi \ \wedge\ \mathcal{D} = \text{SCC}(\delta_1 \cdots \delta_n) \wedge \\
&\qquad\qquad (D_1, \ldots, D_\ell) = \mathit{topsort}(\mathcal{D}) \centerdot \pi_{i+1} = T_{\text{full}}[\![D_i]\!](\pi_i)
\end{aligned}
$$

where $\text{SCC}(\delta_1 \cdots \delta_n)$ is the set $\mathcal{D}$ of strongly connected components of the directed graph $G(\delta_1 \cdots \delta_n)$ described in Section 3 and $\mathit{topsort}(\mathcal{D})$ is a topological sort of these components. Here, $T_{\text{full}}[\![_]\!]$ is the interpretation function for full `nfer` defined in Section 4. The results of $T_{\text{ext}}[\![_]\!]$ are independent of the topological sort, as any such ordering will guarantee that all intervals matched by a sub-specification exist before it is evaluated using $T_{\text{full}}[\![_]\!]$. Both the strongly-connected components and their topological sort can be computed in linear time [34].

Example 5. We can now extend Example 3 with inclusive and exclusive rules to compute the composite and prime numbers. Recall the five `inc-nfer` rules from that example to compute the natural (labeled with the identifier N), even (E), and odd (O) numbers from the input trace $\tau = (I, \{v \mapsto 0\})$. We introduce two new rules that first compute the composite (C) numbers and then use them to find the primes (P):

$$\delta_6 = C \leftarrow N \textbf{ coincide } N \textbf{ where } m_1, m_2 \mapsto m_1(v) > 1 \wedge m_2(v) > 1 \textbf{ map } m_1, m_2 \mapsto \{v \mapsto m_1(v) \cdot m_2(v)\}$$

$$\delta_7 = P \leftarrow N \textbf{ unless contain } C \textbf{ where } m_1, m_2 \mapsto m_1(v) = m_2(v) \textbf{ map } m_1 \mapsto \{v \mapsto m_1(v)\}.$$

Rule δ_6 matches two N-labeled intervals with map values greater than one and creates a C-labeled interval with its map value set to the product of the values from the two matched intervals. Rule δ_7 then finds the complement of the composite numbers defined in δ_6 in a similar way to how an exclusive rule was used in Example 4 to define odd numbers as the complement of even numbers.

Fig. 4 shows the directed graph formed by `nfer` rules $\delta_1 \cdots \delta_5$ from Example 3 with δ_6 and δ_7 added. In the graph, there are two non-trivial strongly-connected components: the cycles formed by δ_4 and δ_5 and the cycle formed by δ_3 with itself. The exclusive rule δ_7 (shown as a square shape in the figure) is not included in these components and must only fall after δ_1, δ_3, and δ_6 in any topological sort.

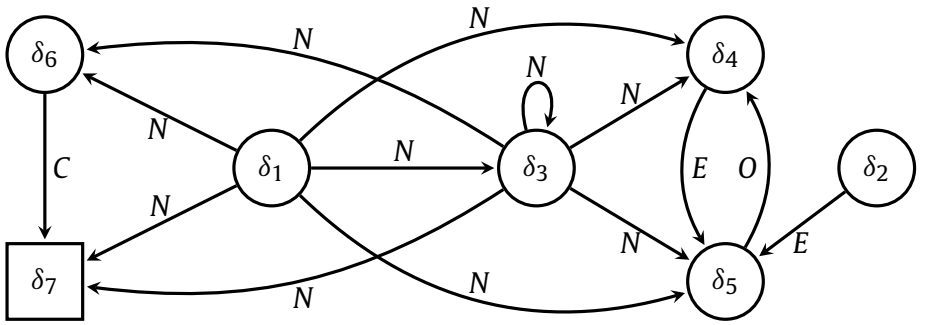

Fig. 4. The directed graph formed by the rules to compute composite and prime numbers.

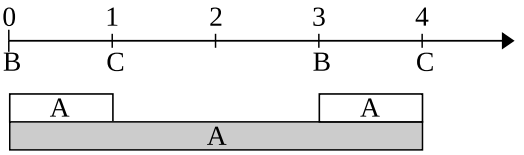

Fig. 5. Minimality discards the shaded interval produced by $A \leftarrow B$ **before** C.

7. Complexity results for extended nfer

The `ext-nfer` evaluation problem with infinite data is trivially undecidable, as already the `inc-nfer` evaluation problem with infinite data is undecidable (see Theorem 1). So, we only consider the case of finite data.

Theorem 7. The `ext-nfer` evaluation problem with finite data is ExpTime-complete.

Proof. The lower bound already holds for the `inc-nfer` evaluation problem with finite data (see Theorem 3), so we only need to prove the upper bound.

To this end, we again use Algorithm 2, but replace the bound $|D|$ in Line 3 by $(|D|+1) \cdot b(D, \tau, k)$, where $b(D, \tau, k)$ is the upper bound on the number of intervals defined in the proof of Theorem 3. Recall that this bound is also an upper bound on the height of a witness tree for an interval.

The correctness of the algorithm follows by combining the proofs of Theorem 3 and Theorem 5: As exclusive rules do not appear on cycles, each one of them can be applied at most once when producing a fixed interval. Furthermore, between each such application no interval needs to be repeated in a witness tree. Altogether, the bound $(|D|+1) \cdot b(D, \tau, k)$ is sufficient to determine whether a given interval is in the fixed point or not.

Finally, it is clear that the algorithm uses only polynomial space, as it only needs to store at most two intervals (and the resulting map values) and a counter bounded by $(|D|+1) \cdot b(D, \tau, k))$. All these objects can be encoded using polynomially many bits. Thus, the upper bound on the complexity follows due to APSpace = ExpTime. □

Finally, let us consider `ext-nfer` combined with minimality. With finite data, the resulting evaluation problem is in PTime while it is in ExpTime with infinite data. Both results can be proven using the same arguments as for `nfer` (i.e., Theorem 8 and Theorem 9), as these rely on the number of possible intervals in the fixed point and the time required to compute their maps, but not the types of rules considered.

8. Minimality

This section discusses the *minimality* restriction and its implications on the complexity of the `nfer` evaluation problem. Traditionally, `nfer` supports the concept of a *selection function* that may modify the results of $R[\![_]\!]$ [3]. The reason is to support minimality, which filters any intervals that are not minimal in their timestamps. Although minimality was originally introduced for its utility [2], it has positive implications for evaluation complexity as well. All results in this section hold for our extended semantics, i.e., with cycles and exclusive rules.

Fig. 5 shows the effect of minimality on the evaluation of a single rule. In the figure, time moves from left to right and events B and C on the timeline are the inputs to $R[\![A \leftarrow B$ **before** C **where** *true* **map** $\{\}]\!]$. This evaluation produces the three intervals labeled A but minimality discards the longer, shaded interval because there are shorter A intervals in the same period.

Given a pool π of existing intervals and a pool π' of intervals to add, the **minimality** function returns only the minimal intervals in π' that do not subsume any interval in π. That is, the intervals where there is not another interval with the

same identifier with a shorter duration during the same time. No new intervals will be produced with the same identifier and timestamps when one already exists in π. If there are multiple intervals with the same identifier and the same timestamps in π', the one with the least map is retained (with respect to some fixed ordering of maps). We define minimality as the following:

$$
\begin{array}{l}
\textbf{minimality} : \mathbb{P} \times \mathbb{P} \to \mathbb{P} \\
\textbf{minimality}\,(\pi', \pi) = \\
\{i \in \pi' : \nexists i_1 \in \pi \;.\; id(i) = id(i_1) \wedge start(i) \leq start(i_1) \wedge end(i_1) \leq end(i)\} \cap \\
\{i \in \pi' : \nexists i_2 \in \pi' \;.\; id(i) = id(i_2) \wedge (start(i) \leq start(i_2) \wedge end(i_2) < end(i)) \vee (start(i) < start(i_2) \wedge end(i_2) \leq end(i))\} \cap \\
\{i \in \pi' : \nexists i_3 \in \pi' \;.\; id(i) = id(i_3) \wedge start(i) = start(i_3) \wedge end(i) = end(i_3) \wedge map(i_3) \prec map(i)\}
\end{array}
$$

where $\prec$ is a total order over $\mathbb{M}$ used as a tiebreaker when more than one new intervals exist in π' with equal identifiers and timestamps.

For the `nfer` evaluation problem under minimality we replace $R[\![_]\!]$ in the semantics with an interpretation function that applies **minimality** to the result of $R[\![_]\!]$.

$$
\begin{array}{l}
R_{min}[\![_]\!] : \Delta \to \mathbb{P} \to \mathbb{P} \\
R_{min}[\![\delta]\!]\,\pi \;=\; \textbf{minimality}\,(R[\![\delta]\!]\pi, \pi)
\end{array}
$$

Example 6. Assume an input trace of alternating B-labeled and C-labeled intervals with empty maps, say $\tau = (B, 0, \{\,\}), (C, 1, \{\,\}), (B, 3, \{\,\}), (C, 4, \{\,\})$ and the `nfer` rule $\delta = A \leftarrow B$ **before** C **where** *true* **map** $\{\,\}$. These are the same conditions as shown in Fig. 5, where the four input events are shown on the top line. Note that this example differs from those shown throughout the rest of the paper because we need to include timestamps to show the effect of minimality. We project τ to an initial pool (we omit writing the maps as they are not used) $\pi_1 = \{(B, 0, 0), (C, 1, 1), (B, 3, 3), (C, 4, 4)\}$.

Without minimality, $R[\![\delta]\!]\,(\pi_1) = \pi_2 = \{(A, 0, 1), (A, 0, 4), (A, 3, 4)\}$, since there are three pairings of (the interval projection of) the events in τ that satisfy the clock predicate **before**, and the map predicate is always true. Note that the start and end timestamps of the intervals in π are taken from the input events that satisfy δ.

We now examine the component parts of **minimality** (π_2, π_1). There are three sets constructed in **minimality**, and an interval must be in all three sets to be in the output.

1. $\{i \in \pi_2 : \nexists i_1 \in \pi_1 \;.\; id(i) = id(i_1) \wedge start(i) \leq start(i_1) \wedge end(i_1) \leq end(i)\}$
   This set includes all of π_2 since no A-labeled intervals appear in π_1.
2. $\{i \in \pi_2 : \nexists i_2 \in \pi_2 \;.\; id(i) = id(i_2) \wedge (start(i) \leq start(i_2) \wedge end(i_2) < end(i)) \vee (start(i) < start(i_2) \wedge end(i_2) \leq end(i))\}$
   This set includes only $\{(A, 0, 1), (A, 3, 4)\}$ because for $(A, 0, 4)$ the interval $(A, 0, 1)$ has the same identifier, the same start timestamp, and a strictly lower end timestamp. It would also be excluded by $(A, 3, 4)$.
3. $\{i \in \pi_2 : \nexists i_3 \in \pi_2 \;.\; id(i) = id(i_3) \wedge start(i) = start(i_3) \wedge end(i) = end(i_3) \wedge map(i_3) \prec map(i)\}$
   This set includes all of π_2 since there are no intervals in π_2 that vary only by their maps.

As such, $R_{\min}[\![\delta]\!]\,(\pi_1) = \{(A, 0, 1), (A, 3, 4)\}$. Thus, the minimality restriction filters out the non-minimal interval $(A, 0, 4)$.

8.1. Complexity results for `nfer` with minimality

Theorem 8. The `nfer` evaluation problem with finite data and minimality is in PTime.

Proof. Consider an instance with specification D, trace τ, and bound k on the map values. Due to minimality, the size of $T^k_{\text{full}}[\![D]\!]\,\tau$ is bounded by $(\iota \cdot t^2) + |\tau|$, where ι is the number of identifiers in D and τ and t is the number of timestamps in τ. Note that this bound is independent of k.

Also, map values and timestamps can be represented with polynomially many bits in the size of D and τ. Hence, we can compute $T^k_{\text{full}}[\![D]\!]\,\tau$ and check whether it contains an interval labeled by the target identifier in polynomial time. □

A similar approach works for infinite data.

Theorem 9. The `nfer` evaluation problem with infinite data and minimality is in ExpTime.

Proof. Again, we can compute $T_{\text{full}}[\![D]\!]\,\tau$ to determine whether it contains an interval labeled by the target identifier. Due to minimality, the size of $T_{\text{full}}[\![D]\!]\,\tau$ is still bounded by $(\iota \cdot t^2) + |\tau|$. However, map values can only be bounded doubly-exponentially (cf. Theorem 4), as one can apply a polynomial $\iota \cdot t^2$ times to the map values of the initial intervals. Hence, intervals in $T_{\text{full}}[\![D]\!]\,\tau$ can be represented with exponentially many bits in the size of the specification and the trace, which yields an exponential running time of computing $T_{\text{full}}[\![D]\!]\,\tau$. □

9. Discussion and conclusion

We have studied the complexity of the `nfer` evaluation problem. It is undecidable in the presence of recursion and infinite data, even without exclusive rules. In contrast, regardless of the presence of exclusive rules, the evaluation problem is decidable for cycle-free specifications or with respect to finite data. Most importantly for applications, the problem is in PTime if we impose the minimality constraint and restrict to finite data. While we only allow natural numbers and Booleans as map values, our upper bounds also hold for more complex data types, i.e., signed numbers, (fixed-precision) floating point numbers, and strings, which were included in the original definitions [2,3].

Most of our complexity bounds are tight, but we leave one gap. The `nfer` evaluation problem with infinite data and minimality is in ExpTime while no nontrivial lower bounds are known. The upper bound follows from the fact that the map values may be of doubly-exponential size, i.e., they require exponential time to be computed. However, minimality is a very restrictive constraint that in particular severely limits the ability to simulate nondeterministic computations. Coupled with the fact that minimality implies a polynomial upper bound on the number of intervals in the fixed point, this explains the lack of a nontrivial lower bound.

All our lower bound proofs only use intervals with the same timestamps, i.e., the complexity stems from the manipulation of data instead of temporal reasoning. Similarly, the upper bound proofs are mostly concerned with encoding of data and the temporal reasoning is trivial. One of the reasons is that `nfer` rules do not create new timestamps for intervals; newly created intervals can only use timestamps that already appear in the input trace. This leaves only a polynomial number of combinations of start points and end points, which is (at least) exponentially smaller than the number of data values. For this reason, we propose to investigate data-free `nfer` to analyze the complexity of the evaluation problem with respect to the choice of temporal operators. In this case, there are only polynomially many possible intervals in the fixed point. So, a trivial upper bound on the complexity is PTime, but we expect better results for fragments.

Finally, we are currently studying the `nfer` satisfiability problem, i.e., given a specification D and a target identifier η, is there a trace τ such that $T_{\text{ext}}[\![D]\!]\,\tau$ contains an η-labeled interval? This has a practical application in supporting `nfer` users in checking the specifications for sanity: if no η-labeled interval can be produced from any input, then the specification probably does not capture the user's intent.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research was partly funded by the ERC Advanced Grant LASSO, the Villum Foundation Investigator Grant S4OS and DIREC, Digital Research Center Denmark (Secure Internet of things (SIoT) - Risk Analysis in Design and Operation).

References

[1] S. Kauffman, R. Joshi, K. Havelund, Towards a logic for inferring properties of event streams, in: International Symposium on Leveraging Applications of Formal Methods (ISoLA'16), in: LNCS, vol. 9953, Springer, 2016, pp. 394–399.
[2] S. Kauffman, K. Havelund, R. Joshi, Nfer–a notation and system for inferring event stream abstractions, in: International Conference on Runtime Verification (RV'16), in: LNCS, vol. 10012, Springer, 2016, pp. 235–250.
[3] S. Kauffman, K. Havelund, R. Joshi, S. Fischmeister, Inferring event stream abstractions, Form. Methods Syst. Des. 53 (2018) 54–82, https://doi.org/10.1007/s10703-018-0317-z.
[4] J.F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843.
[5] A. Narayan, S. Kauffman, J. Morgan, G.M. Tchamgoue, Y. Joshi, C. Hobbs, S. Fischmeister, System call logs with natural random faults: experimental design and application, in: International Workshop on Silicon Errors in Logic – System Effects (SELSE'17), SELSE-13, IEEE, 2017.
[6] S. Kauffman, M. Dunne, G. Gracioli, W. Khan, N. Benann, S. Fischmeister, Palisade: a framework for anomaly detection in embedded systems, J. Syst. Archit. 113 (2021) 101876, https://doi.org/10.1016/j.sysarc.2020.101876.
[7] S. Kauffman, Log analysis and system monitoring with nfer, Sci. Comput. Program. 225 (2023), https://doi.org/10.1016/j.scico.2022.102909.
[8] S. Kauffman, Runtime monitoring for uncertain times, Ph.D. thesis, University of Waterloo, Department of Electrical and Computer Engineering, Waterloo, ON, Canada, 2021, http://hdl.handle.net/10012/16853.
[9] S. Kauffman, Website. http://nfer.io/, 2022. (Accessed January 2022).
[10] K. Havelund, Git repository, git@github.com:rv-tools/nfer.git, accessed: Jan 2022 (2022).
[11] K. Havelund, M. Omer, D. Pelescd, Monitoring first-order interval logic, in: Software Engineering and Formal Methods (SEFM), Springer, 2021, pp. 66–83.
[12] S. Kauffman, Nfer – a tool for event stream abstraction, in: International Conference on Software Engineering and Formal Methods (SEFM'21), in: LNCS, vol. 13085, Springer, 2021, pp. 103–109.
[13] S. Kauffman, M. Zimmermann, The complexity of evaluating nfer, in: International Symposium on Theoretical Aspects of Software Engineering (TASE'22), in: LNCS, vol. 13299, Springer, 2022, pp. 388–405.
[14] H. Barringer, K. Havelund, TraceContract: a Scala DSL for trace analysis, in: Formal Methods (FM'11), in: LNCS, vol. 6664, Springer, 2011, pp. 57–72.
[15] K. Havelund, Rule-based runtime verification revisited, Int. J. Softw. Tools Technol. Transf. 17 (2) (2015) 143–170, https://doi.org/10.1007/s10009-014-0309-2.
[16] J. Chen, D.J. DeWitt, F. Tian, Y. Wang, NiagaraCQ: a scalable continuous query system for Internet databases, in: International Conference on Management of Data (ACM SIGMOD'00), ACM, 2000, pp. 379–390.

[17] D. Luckham, The power of events: an introduction to complex event processing in distributed enterprise systems, in: Rule Representation, Interchange and Reasoning on the Web, in: LNCS, vol. 5321, Springer, 2008.
[18] S. Suhothayan, K. Gajasinghe, I. Loku Narangoda, S. Chaturanga, S. Perera, V. Nanayakkara Siddhi, A second look at complex event processing architectures, in: Workshop on Gateway Computing Environments (GCE'11), ACM, 2011, pp. 43–50.
[19] H. Zhang, Y. Diao, N. Immerman, On complexity and optimization of expensive queries in complex event processing, in: International Conference on Management of Data, SIGMOD'14, ACM, 2014, pp. 217–228.
[20] S. Hallé, When RV meets CEP, in: Runtime Verification (RV'16), in: LNCS, vol. 10012, Springer, 2016, pp. 68–91.
[21] L. Convent, S. Hungerecker, M. Leucker, T. Scheffel, M. Schmitz, D. Thoma, TeSSLa: temporal stream-based specification language, in: Formal Methods: Foundations and Applications, in: LNCS, vol. 11254, Springer, 2018, pp. 144–162.
[22] P. Faymonville, B. Finkbeiner, M. Schwenger, H. Torfah, Real-time stream-based monitoring, arXiv:1711.03829, 2019.
[23] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL - the planning domain definition language, Tech. Rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
[24] R.W. Butler, R.I. Siminiceanu, C. Muno, The ANMLite language and logic for specifying planning problems, Tech. Rep, NASA Langley Research Center, Hampton, VA, United States, 2007.
[25] F. Mallet, Clock constraint specification language: specifying clock constraints with uml/marte, Innov. Syst. Softw. Eng. 4 (3) (2008) 309–314, https://doi.org/10.1007/s11334-008-0055-2.
[26] J.Y. Halpern, Y. Shoham, A propositional modal logic of time intervals, J. ACM 38 (4) (1991) 935–962, https://doi.org/10.1145/115234.115351.
[27] A. Montanari, G. Puppis, P. Sala, G. Sciavicco, Decidability of the interval temporal logic ABB over the natural numbers, in: Proceedings of STACS 2010, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2010, pp. 597–608, https://hal.archives-ouvertes.fr/hal-00717798.
[28] A. Montanari, G. Puppis, P. Sala, Maximal decidable fragments of Halpern and shoham's modal logic of intervals, in: Automata, Languages and Programming, Springer, 2010, pp. 345–356.
[29] L. Aceto, D. Della Monica, V. Goranko, A. Ingólfsdóttir, A. Montanari, G. Sciavicco, A complete classification of the expressiveness of interval logics of Allen's relations: the general and the dense cases, Acta Inform. 53 (3) (2016) 207–246, https://doi.org/10.1007/s00236-015-0231-4.
[30] M.L. Minsky, Computation, Prentice-Hall, Englewood Cliffs, 1967.
[31] S. Arora, B. Barak, Computational Complexity: A Modern Approach, 1st edition, Cambridge University Press, USA, 2009.
[32] B. Rosser, Explicit bounds for some functions of prime numbers, Am. J. Math. 63 (1) (1941) 211–232, http://www.jstor.org/stable/2371291.
[33] A.B. Kahn, Topological sorting of large networks, Commun. ACM 5 (11) (1962) 558–562, https://doi.org/10.1145/368996.369025.
[34] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160, https://doi.org/10.1137/0201010.