# The Complexity of Evaluating nfer*

Sean Kauffman and Martin Zimmermann

Aalborg University, Denmark
{seank,mzi}@cs.aau.dk

**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.

**Keywords:** Interval Logic · Complexity · Runtime Verification

## 1 Introduction

`Nfer` is a rule-based language and tool 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 [20, 18, 19]. `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) [2] 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 [18]. 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 [26, 19] and to locate unstable gear shifts in an autonomous vehicle [17].

* This research was partly funded by the ERC Advanced Grant LASSO, the Villum Investigator Grant S4OS and DIREC, Digital Research Center Denmark.

Nfer is expressive enough for many applications and termination of the nfer monitoring algorithm has been conjectured to be undecidable [15]. 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 [16] and Scala [11] versions have been compared against tools such as LogFire and Prolog [19], Siddhi [17], MonAmi and DejaVu [12], and TeSSLa [14] 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.

*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 most of these fragments.

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. Thus, 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 [12]. Of note is that recursion in rules must be prohibited when exclusive rules are used. We prove that, without infinite data, adding exclusive rules has no effect and nfer evaluation remains PSpace-complete. With infinite data, however, we prove the problem is in AExpTime(poly).

Finally, we 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 [19]. We show that minimality has a substantial effect on the complexity of nfer evaluation. With infinite data, we prove 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.

All proofs omitted due to space restrictions can be found in the full version [21].

*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 [4, 10]. 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 [5, 22, 28]. 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 [8, 6, 7]. Still, `nfer` is differentiated from these systems by its emphasis on temporal intervals and its ATL-based syntax.

Some research has examined the complexity of logics based on ATL, specifically Halpern and Shoham's modal logic of intervals (HS) [9]. Montanari et al. showed that the satisfiability problem for the subset of HS consisting of only *begins/begun by* and *meets* is ExpSpace-complete over the natural numbers [25]. Later, they showed that adding the *met by* operator increases the complexity such that the language is only decidable over finite total orders [24]. Aceto et al. identified the expressive power of all fragments of HS over total orders as well as only dense total orders [1]. `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 desribes 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 [18]. It supports many use cases, including the MSL case-study described above. The implementation of `nfer` written in Scala at JPL [11, 19] also supports only inclusive rules. We expand our analysis to include exclusive rules in Section 4 while Section 5 addresses minimality, an important extension of `nfer` semantics. Note that, to improve comprehensibility and simplify later proofs, the semantics presented here differs slightly from prior work but these changes do not affect the language capabilities.

*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 type 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 $(\eta, t, M)$ 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 type of an event is given by $\mathbb{E} = \mathcal{I} \times \mathbb{N} \times \mathbb{M}$. 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 $(\eta, s, e, M)$ 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 type of 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 its type is given by $\mathbb{P} = 2^{\mathbb{I}}$. We say that an interval $i = (\eta, s, e, M)$ is labeled by $\eta$. We define the functions $\mathit{id}(i) = \eta$, $\mathit{start}(i) = s$, $\mathit{end}(i) = e$, and $\mathit{map}(i) = M$.

*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 $\eta_2$), $\Phi : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{B}$ is a *map predicate* taking two maps and returning a Boolean representing satisfaction of a constraint, and $\Psi : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{M}$ 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 [14]. Specifically, map predicates and map updates must be expressible using the standard mathematical operations: addition, subtraction, multiplication, division, modulo, and the comparisons: $<, \leq, >, \geq, =$ on natural numbers, 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 6.

*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 $\Delta$ be the type of rules. Semantic functions are defined using the brackets $[\![\_]\!]$ around syntax being given semantics.

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

In the definition, a new interval $i$ is produced when two existing intervals in $\pi$ match the identifiers $\eta_1$ and $\eta_2$, the temporal constraint $\oplus$, and the map constraint $\Phi$. $\oplus$ defines the start and end timestamps of $i$ and $\Psi$ defines its map.

The possibilities referenced by $\oplus$ are shown in Figure 1. These clock predicates are based on ATL and described formally in previous definitions of `nfer` [18, 19]. They 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. For example, given inter-

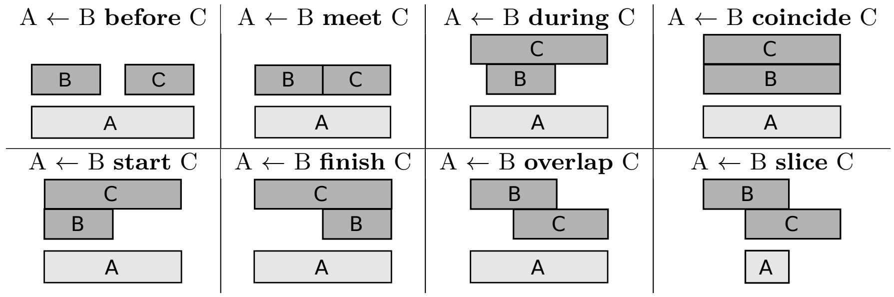


**Fig. 1.** `nfer` clock predicates for inclusive rules

vals $i, i_1, i_2$ 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 $\pi$ with the new intevals 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 $T_{\mathrm{inc}}[\![\_]\!]$ for *inc*lusive `nfer` defines the semantics of a specification on a pool by applying $S$ until the inflationary fixed point is reached.

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

To maintain consistency with prior work and simplify our presentation, we also overload $T_{\mathrm{inc}}[\![\_]\!]$ to operate on a trace of events $\tau \in \mathbb{E}^*$ by first converting $\tau$ 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 $\delta_j$ is the rule

$\eta_{j+1} \leftarrow \eta_j$ **coincide** $\eta_j$ **where** $m_1, m_2 \mapsto m_1 = m_2$ **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

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

*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 $\delta_j$ in Example 1 produce an interval $i'$ labeled by $\eta_{j+1}$ 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 $(\eta_{j+1}, y^2)$ from an interval of the form $(\eta_j, y)$.

We are interested in the `nfer` evaluation problem: Given a specification $D$, a trace $\tau$ of events, and a target identifier $\eta_T$, is there an $\eta_T$-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.

## 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.

The undecidability result relies on the recursive nature of `inc-nfer`, i.e., an $\eta$-labeled interval can be (directly or indirectly) produced from an another $\eta$-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 [23] 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 $\ell$ is a line number and $\mathtt{I}_\ell$ for $0 \leq \ell < k-1$ is one of $\mathtt{INC(X}_i\mathtt{)}$, $\mathtt{DEC(X}_i\mathtt{)}$, or $\mathtt{IF\ X}_i\mathtt{=0\ GOTO}\ \ell'$ with $i \in \{0, 1\}$ and $\ell' \in \{0, \cdots, k-1\}$.

A configuration of the machine is a triple $(\ell, c_0, c_1)$ consisting of a line number $\ell$ 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 [23].

This problem is captured with `inc-nfer` as follows: We encode a configuration $(\ell, c_0, c_1)$ by an interval with identifier $\ell$ and two map values $c_0, c_1$. These intervals use the same timestamps so we drop them from our notation and also write $(\ell, c_0, c_1)$ 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 $i = 0$, the rules for $i = 1$ are analogous):

- $\mathtt{I}_\ell = \mathtt{INC(X_0)}$: We have a rule producing the interval $(\ell+1, c_0+1, c_1)$ from an interval of the form $(\ell, c_0, c_1)$.
- $\mathtt{I}_\ell = \mathtt{DEC(X_0)}$: We have two rules, one producing the interval $(\ell+1, c_0-1, c_1)$ from an interval of the form $(\ell, c_0, c_1)$ with $c_0 > 0$, and one producing the interval $(\ell+1, c_0, c_1)$ from an interval of the form $(\ell, c_0, c_1)$ with $c_0 = 0$.
- $\mathtt{I}_\ell = \mathtt{IF\ X_0{=}0\ GOTO}\ \ell'$: We have two rules, one producing the interval $(\ell', c_0, c_1)$ from an interval of the form $(\ell, c_0, c_1)$ with $c_0 = 0$, and one producing the interval $(\ell+1, c_0, c_1)$ from an interval of the form $(\ell, c_0, c_1)$ 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** $\Phi$ **map** $\Psi$ we say that $\eta$ appears on the left-hand side and the $\eta_i$ 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 $\delta$ to $\delta'$ iff there is an identifier $\eta$ that appears on the left-hand side of $\delta$ and the right-hand side of $\delta'$. We say that $D$ contains a cycle if $G(D)$ contains one; otherwise $D$ is cycle-free.

We begin our study of decidable fragments of `inc-nfer` by considering both restrictions 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., [3] for a detailed definition), which is PSpace-hard. So, fix such a formula $\varphi$. Let $\pi_j$ 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 $Q_{\pi_j}$ is in $\{\exists, \forall\}$, and each $\ell_{i,i'}$ is either $x_{\pi_j}$ 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 $(\eta, s)$ where $\eta$ 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 $x_{\pi_j}$ 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 $(C_k, s)$ from an interval of the form $(G_k, s)$ 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 $x_{\pi_j}$ 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 $x_{\pi_j}$ 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 $\varphi$ 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 $\varphi$. Finally, as $\pi_j \leq j(\ln j + \ln \ln j)$ for all $j \geq 6$ [27], 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 $\varphi$. Here, $c$ is some constant that is independent of $n$. $\square$

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 $\tau$ of events. If $i$ is an interval in $T_{\text{inc}}[\![D]\!]\,\tau$, then either there is an event $e$ in $\tau$ such that $init(e) = i$ (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 $\delta$ 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 $\delta$ so that $i$ is obtained by applying $\delta$ to $i_1$ and $i_2$. Further, 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. 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.*

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., [3] for detailed definitions). As ExpTime = APSpace, this yields the desired lower bound. Thus, 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 $\{1, 2, \ldots, q\}$ for some $q \in \mathbb{N}$ and 1 is the initial state.
- The tape alphabet $\Gamma$ of $\mathcal{M}$ is equal to $\{0, 1, \ldots, 9\}$ and 0 is the blank symbol.
- Every run tree 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 $\ell q r$ with $q \in Q$ and $\ell, r \in \mathit{\Gamma}^*$ such that $|\ell| + |r| = p(|w|)$, with the convention that the head is on the first letter of $r$.

For $c \in \mathit{\Gamma}^*$, let $c^R$ denote the reverse of $c$. Due to our assumption on $\mathit{\Gamma}$ we can treat $\ell$ and $r^R$ as natural numbers encoded in base ten. We uniquely identify a configuration $\ell q r$ by the triple $(\ell, q, r^R)$ 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 $\ell$ 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 $(A, \ell, q, r^R)$ to encode the configuration $\ell q r$ of $\mathcal{M}$. Here, $A$ is some identifier and we disregard timestamps, as all intervals have the same start and end. Hence, $\ell$, $q$, 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 $(G, \ell, q, r^R)$. Furthermore, one can write a rule that produces the interval $(A, \ell, q, r^R)$ from every interval $(G, \ell, q, r^R)$ 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 $(A, \ell, q, r^R)$ if the intervals $(G, \ell, q, r^R)$ 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 $(\ell, q, r^R)$. 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 $(B, \ell, q, r^R)$ if the intervals $(G, \ell, q, r^R)$ 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 $(\ell, q, r^R)$. The second one produces the interval $(A, \ell, q, r^R)$ if the intervals $(B, \ell, q, r^R)$ and $(A, \ell', q', r^{R'})$ already exist, where $(\ell', q', r^{R'})$ encodes the configuration obtained by applying

**Algorithm 1** Algorithm checking the existence of a witness tree

**Input:** Specification $D$, trace $\tau$, bound $k$, target identifier $\eta_T$

$n := 0$
**nondeterministically guess** interval $i$ labeled by $\eta_T$
**while** $n < b(D, \tau, k)$ **and** $i$ is not initial **do**
  $n := n + 1$
  **nondeterministically guess** intervals $i_1, i_2$ and $\delta \in D$ such that $i$ is obtained by applying $\delta$ to $i_1$ and $i_2$
  **universally pick** $i := i_j$ for $j \in 1, 2$
**if** $i$ is initial **then return** accept
**else return** reject

the transition $t_2$ to the configuration encoded by $(\ell, q, r^R)$. 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 $\eta_T$ from the interval $(A, 0, 1, w^R)$, indicating that the initial configuration is accepting. Thus, the fixed point contains an interval labeled by $\eta_T$ 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 $\tau$ of events, a $k \in \mathbb{N}$ (given in binary), and a target label $\eta_T$ and have to determine whether the fixed point $T^k_{\text{inc}}[\![D]\!]\,\tau$ contains an interval labeled by $\eta_T$. 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, $\iota$ is the number of identifiers appearing in $D$ and $\tau$ and $t$ is the number of timestamps in $\tau$ (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 $\delta$ to $i_1$ and $i_2$.

Using alternation, Algorithm 1 determines whether a witness tree exists whose root is labeled by $\eta_T$ and whose height is bounded by $b$. Due to Proposition 1, this is equivalent to an interval labeled by $\eta_T$ 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. □

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 and encode configurations using these numbers as before.

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

## 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 [19] 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 $\eta_1$ and $\eta_2$), and $\Phi$ and $\Psi$ are the same as in inclusive rules. We say that an exclusive rule *includes* $\eta_1$ and *excludes* $\eta_2$.

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:

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

Like with inclusive rules, exclusive rules match intervals in the input pool $\pi$ to produce a pool of new intervals. The difference is that exclusive rules produce new intervals where one existing interval in $\pi$ matches the identifier $\eta_1$ and no intervals exist that match the identifier $\eta_2$ such that the clock predicate $\ominus$ and the map predicate $\Phi$ hold for the $\eta_1$-labeled and the $\eta_2$-labeled interval.

The three possibilities referenced by $\ominus$ are shown in Figure 2. These clock predicates are based on ATL and described formally in a previous definition of `nfer` [19]. They 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 $id(i_2) = C$,

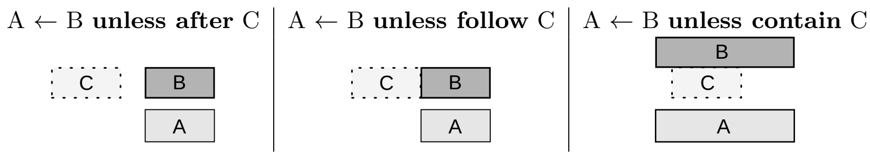


**Fig. 2.** `nfer` clock predicates for exclusive rules

$A \leftarrow B$ **unless follow** $C$ holds when $end(i_2) = start(i_1)$, $start(i) = start(i_1)$, and $end(i) = end(i_1)$.

Exclusive rules are forbidden in specifications with cycles because the intervals they produce depend on the persistent non-existence of other intervals. When cycles exist in a specification, rules are evaluated multiple times and each evaluation may add intervals. Exclusive rules may have non-deterministic behavior in a specification with cycles because the intervals they exclude may be produced either before or after the exclusive rule is evaluated. 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. A topological sort, which can be computed in linear time [13], 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 $R[\![\_]\!]$ .

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.

As in Algorithm 1, the algorithm 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 $\delta$ 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 $\delta$. 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 $\delta$ includes $i_1$, excludes $i_2$, and $i$ is the result of applying $\delta$ 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 whether 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 $|D| + |\tau|$. □

Finally, we consider the case of infinite data. Here, the upper 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 NExpTime-hard and in AExpTime(poly).*

## 5 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 $R[\![\_]\!]$ [19]. 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 [18], it has positive implications for evaluation complexity as well.

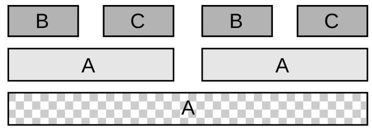


**Fig. 3.** Minimality discards the checkered interval produced by $A \leftarrow B$ **before** $C$

Figure 3 shows the effect of minimality on the evaluation of a single rule. In the figure, time moves from left to right and the dark-gray intervals are the inputs to $R[\![A \leftarrow B \textbf{ before } C \textbf{ where } \textit{true} \textbf{ map } \{\ \}]\!]$. This evaluation produces the three intervals labeled $A$ but minimality discards the longer interval with a checkerboard pattern because there are shorter $A$ intervals in the same period.

Given a pool $\pi$ of existing intervals and a pool $\pi'$ of intervals to add, the **minimality** function returns only the minimal intervals in $\pi'$ that do not subsume any interval in $\pi$. 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 $\pi$. If there are multiple intervals with the same identifier and the same timestamps in $\pi'$, the one with the least map is retained (with respect to some fixed ordering of maps). We define minimality as the following:

$$\textbf{minimality} : \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}$$

$$\textbf{minimality}\,(\pi', \pi) =$$

$$\{(\eta, s, e, M) \in \pi' : \nexists (\eta, s_1, e_1, M_1) \in \pi \,\boldsymbol{.}\; s \leq s_1 \wedge e_1 \leq e\} \cap$$

$$\{(\eta, s, e, M) \in \pi' : \nexists (\eta, s_2, e_2, M_2) \in \pi' \boldsymbol{.} (s \leq s_2 \wedge e_2 < e) \vee (s < s_2 \wedge e_2 \leq e)\} \cap$$

$$\{(\eta, s, e, M) \in \pi' : \nexists (\eta, s_3, e_3, M_3) \in \pi' \boldsymbol{.}\; s = s_3 \wedge e = e_3 \wedge M_3 \prec M\}$$

where $\prec$ is a total order over $\mathbb{M}$ used as a tiebreaker when more than one new intervals exist in $\pi'$ 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 $R[\![\_]\!]$.

$$R_{min}[\![\_]\!] : \Delta \rightarrow \mathbb{P} \rightarrow \mathbb{P}$$

$$R_{min}[\![\delta]\!]\,\pi = \textbf{minimality}\,(R[\![\delta]\!]\pi, \pi)$$

**Theorem 7.** *The* `nfer` *evaluation problem with finite data and minimality is in* PTIME.

*Proof.* Consider an instance with specification $D$, trace $\tau$, 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 $\iota$ is the number of identifiers in $D$ and $\tau$ and $t$ is the number of timestamps in $\tau$. 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 $\tau$. 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. $\square$

A similar approach works for infinite data.

**Theorem 8.** *The* `nfer` *evaluation problem with infinite data and minimality is in* ExpTime.

## 6 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 [18, 19].

Most of our complexity bounds are tight, but we leave two gaps. First, the cycle-free `nfer` evaluation problem with infinite data is NExpTime-hard and in AExpTime(poly). Recall that the lower bound already holds for `inc-nfer`, i.e., without exclusive rules, while the polynomial number of alternations in the upper bound are used to simulate exclusive rules (our algorithm requires one alternation per exclusive rule). One approach to close this gap is to capture alternations of a Turing machine using exclusive rules.

Secondly, 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.

Another interesting fragment is the combination of cycles and exclusive rules. As long as exclusive rules lie outside cycles the deterministic semantics can be defined. In the full version of this paper [21] we show that this fragment has an ExpTime-complete evaluation problem when restricted to finite data.

## References


1. Aceto, L., Della Monica, D., Goranko, V., Ingólfsdóttir, A., Montanari, A., Sciavicco, G.: A complete classification of the expressiveness of interval logics of Allen's relations: the general and the dense cases. Acta Informatica **53**(3), 207–246 (Apr 2016). https://doi.org/10.1007/s00236-015-0231-4
2. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983)
3. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, USA, 1st edn. (2009)
4. Barringer, H., Havelund, K.: TraceContract: A Scala DSL for trace analysis. In: Formal Methods (FM'11). LNCS, vol. 6664, pp. 57–72. Springer (2011). https://doi.org/10.1007/978-3-642-21437-0_7
5. Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: A scalable continuous query system for internet databases. In: International Conference on Management of Data (ACM SIGMOD'00). pp. 379–390. ACM (2000). https://doi.org/10.1145/342009.335432
6. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: Temporal stream-based specification language. In: Formal Methods: Foundations and Applications. LNCS, vol. 11254, pp. 144–162. Springer (2018). https://doi.org/10.1007/978-3-030-03044-5_10
7. Faymonville, P., Finkbeiner, B., Schwenger, M., Torfah, H.: Real-time stream-based monitoring (2019)
8. Hallé, S.: When RV meets CEP. In: Runtime Verification (RV'16). LNCS, vol. 10012, pp. 68–91. Springer (2016). https://doi.org/10.1007/978-3-319-46982-9_6
9. Halpern, J.Y., Shoham, Y.: A propositional modal logic of time intervals. J. ACM **38**(4), 935–962 (1991). https://doi.org/10.1145/115234.115351
10. Havelund, K.: Rule-based runtime verification revisited. International Journal on Software Tools for Technology Transfer **17**(2), 143–170 (2015). https://doi.org/10.1007/s10009-014-0309-2
11. Havelund, K.: Git repository. git@github.com:rv-tools/nfer.git (2022), accessed: Jan 2022
12. Havelund, K., Omer, M., Pelescd, D.: Monitoring first-order interval logic. In: Software Engineering and Formal Methods (SEFM). pp. 66–83. Springer (2021). https://doi.org/10.1007/978-3-030-92124-8_4
13. Kahn, A.B.: Topological sorting of large networks. Communications of the ACM **5**(11), 558–562 (1962). https://doi.org/10.1145/368996.369025
14. Kauffman, S.: nfer – a tool for event stream abstraction. In: International Conference on Software Engineering and Formal Methods (SEFM'21). LNCS, vol. 13085, pp. 103–109. Springer (2021). https://doi.org/10.1007/978-3-030-92124-8_6
15. Kauffman, S.: 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
16. Kauffman, S.: Website. http://nfer.io/ (2022), accessed: Jan 2022
17. Kauffman, S., Dunne, M., Gracioli, G., Khan, W., Benann, N., Fischmeister, S.: Palisade: A framework for anomaly detection in embedded systems. Journal of Systems Architecture **113**, 101876 (2021). https://doi.org/10.1016/j.sysarc.2020.101876

18. Kauffman, S., Havelund, K., Joshi, R.: nfer–a notation and system for inferring event stream abstractions. In: International Conference on Runtime Verification (RV'16). LNCS, vol. 10012, pp. 235–250. Springer (2016). https://doi.org/10.1007/978-3-319-46982-9_15
19. Kauffman, S., Havelund, K., Joshi, R., Fischmeister, S.: Inferring event stream abstractions. Formal Methods in System Design **53**, 54–82 (2018). https://doi.org/10.1007/s10703-018-0317-z
20. Kauffman, S., Joshi, R., Havelund, K.: Towards a logic for inferring properties of event streams. In: International Symposium on Leveraging Applications of Formal Methods (ISoLA'16). LNCS, vol. 9953, pp. 394–399. Springer (2016). https://doi.org/10.1007/978-3-319-47169-3_31
21. Kauffman, S., Zimmermann, M.: The complexity of evaluating nfer. arXiv **2202.13677** (2022), https://arxiv.org/abs/2202.13677
22. Luckham, D.: The power of events: An introduction to complex event processing in distributed enterprise systems. In: Rule Representation, Interchange and Reasoning on the Web. LNCS, vol. 5321. Springer (2008). https://doi.org/10.1007/978-3-540-88808-6_2
23. Minsky, M.L.: Computation. Prentice-Hall Englewood Cliffs (1967)
24. Montanari, A., Puppis, G., Sala, P.: Maximal decidable fragments of halpern and shoham's modal logic of intervals. In: Automata, Languages and Programming. pp. 345–356. Springer (2010). https://doi.org/10.1007/978-3-642-14162-1_29
25. Montanari, A., Puppis, G., Sala, P., Sciavicco, G.: Decidability of the interval temporal logic ABB over the natural numbers. In: Proceedings of STACS 2010. pp. 597–608. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2010), https://hal.archives-ouvertes.fr/hal-00717798
26. Narayan, A., Kauffman, S., Morgan, J., Tchamgoue, G.M., Joshi, Y., Hobbs, C., Fischmeister, S.: 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)
27. Rosser, B.: Explicit bounds for some functions of prime numbers. American Journal of Mathematics **63**(1), 211–232 (1941), http://www.jstor.org/stable/2371291
28. Suhothayan, S., Gajasinghe, K., Loku Narangoda, I., Chaturanga, S., Perera, S., Nanayakkara, V.: Siddhi: A second look at complex event processing architectures. In: Workshop on Gateway Computing Environments (GCE'11). pp. 43–50. ACM (2011). https://doi.org/10.1145/2110486.2110493