TKK Reports in Information and Computer Science
Espoo 2009 TKK-ICS-R14

CHECKING BOUNDED REACHABILITY IN ASYNCHRONOUS SYSTEMS BY SYMBOLIC EVENT TRACING

Jori Dubrovin

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

TKK Reports in Information and Computer Science
Espoo 2009 TKK-ICS-R14

CHECKING BOUNDED REACHABILITY IN ASYNCHRONOUS SYSTEMS BY SYMBOLIC EVENT TRACING

Jori Dubrovin

Helsinki University of Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science

Teknillinen korkeakoulu
Informaatio- ja luonnontieteiden tiedekunta
Tietojenkäsittelytieteen laitos

Distribution:
Helsinki University of Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science
P.O.Box 5400
FI-02015 TKK
FINLAND
URL: http://ics.tkk.fi
Tel. +358 9 451 1
Fax +358 9 451 3369
E-mail: series@ics.tkk.fi

ISBN 978-951-22-9847-1 (Print)
ISBN 978-951-22-9848-8 (Online)
ISSN 1797-5034 (Print)
ISSN 1797-5042 (Online)
URL: http://lib.tkk.fi/Reports/2009/isbn9789512298488.pdf

TKK ICS
Espoo 2009

ABSTRACT: This report presents a new symbolic technique for checking reachability properties of asynchronous systems by reducing the problem to satisfiability in restrained difference logic. The analysis is bounded by fixing a finite set of potential events, each of which may occur at most once in any order. The events are specified using high-level Petri nets. The logic encoding describes the space of possible causal links between events rather than possible sequences of states as in Bounded Model Checking. Independence between events is exploited intrinsically without partial order reductions, and the handling of data is symbolic. On a family of benchmarks, the proposed approach is consistently faster than Bounded Model Checking. In addition, this report presents a compact encoding of the restrained subset of difference logic in propositional logic.

KEYWORDS: formal verification, Bounded Model Checking, partial order method, Coloured Petri Nets, difference logic

CONTENTS

1 INTRODUCTION

Design errors in concurrent software systems are notoriously difficult to find. This is due to the tremendous number of possible interleavings of events and combinations of data values. Symbolic model checking methods [10] attack the problem by expressing the actual and desired behavior of a system as formulas and using the tools of computational logic to search for a possible failure.

In this report, we develop a new symbolic technique for verifying bounded reachability properties of asynchronous discrete-event systems. Instead of manipulating executions as sequences of states, we take an event-centered viewpoint. First, one fixes a collection of transitions, each of which describes one discrete step of execution. This collection is called an unwinding of the system. We only consider finite-length executions in which each transition of the unwinding occurs at most once, in whichever order. From the unwinding, we generate automatically a formula that is satisfiable if and only if a predefined condition, e.g. division by zero, can be reached within this bounded set of executions. For satisfiability checking, any SAT or SMT solver [9] can be used as long as it can handle the data constraints of transitions. If the reachability property holds within the bound, a witness execution can be extracted from an interpretation that satisfies the formula. Otherwise, longer executions can be covered by adding more transitions to the unwinding and generating a new formula. This technique will be called Bounded Event Tracing.

The approach is similar to Bounded Model Checking (BMC) [2]. Both methods can find bugs and report no false alarms, but they cannot be used as such to prove the absence of bugs in realistic systems. Unlike BMC, the new technique directly exploits the defining aspect of asynchronous systems: each transition accesses only a fraction of the global state of the system. In controlled experiments with a family of simple concurrent systems, model checking with Bounded Event Tracing is approximately ten times faster than with BMC.

In the next section, we will go through the central concepts with an extensive example. Section 3 defines unwindings as a class of high-level Petri nets [16] that allows concise modeling of concurrency and software features. The conciseness carries on to the encoding as a formula, which is presented in Sect. 4. The encoding falls in a subset of what is known as difference logic. This subset can be further encoded in propositional logic as shown in Sect. 5. Section 6 discusses the relationship to other model checking approaches, and Sect. 7 reports the experiments.

The generation of unwindings is not yet automated in this work. We will sketch a straightforward unwinding scheme, but optimizations are left for future work.

2 BOUNDED EVENT TRACING BY EXAMPLE

Figure 1a presents a system with three concurrent processes that run indefinitely. Suppose the reachability property in question is whether the system

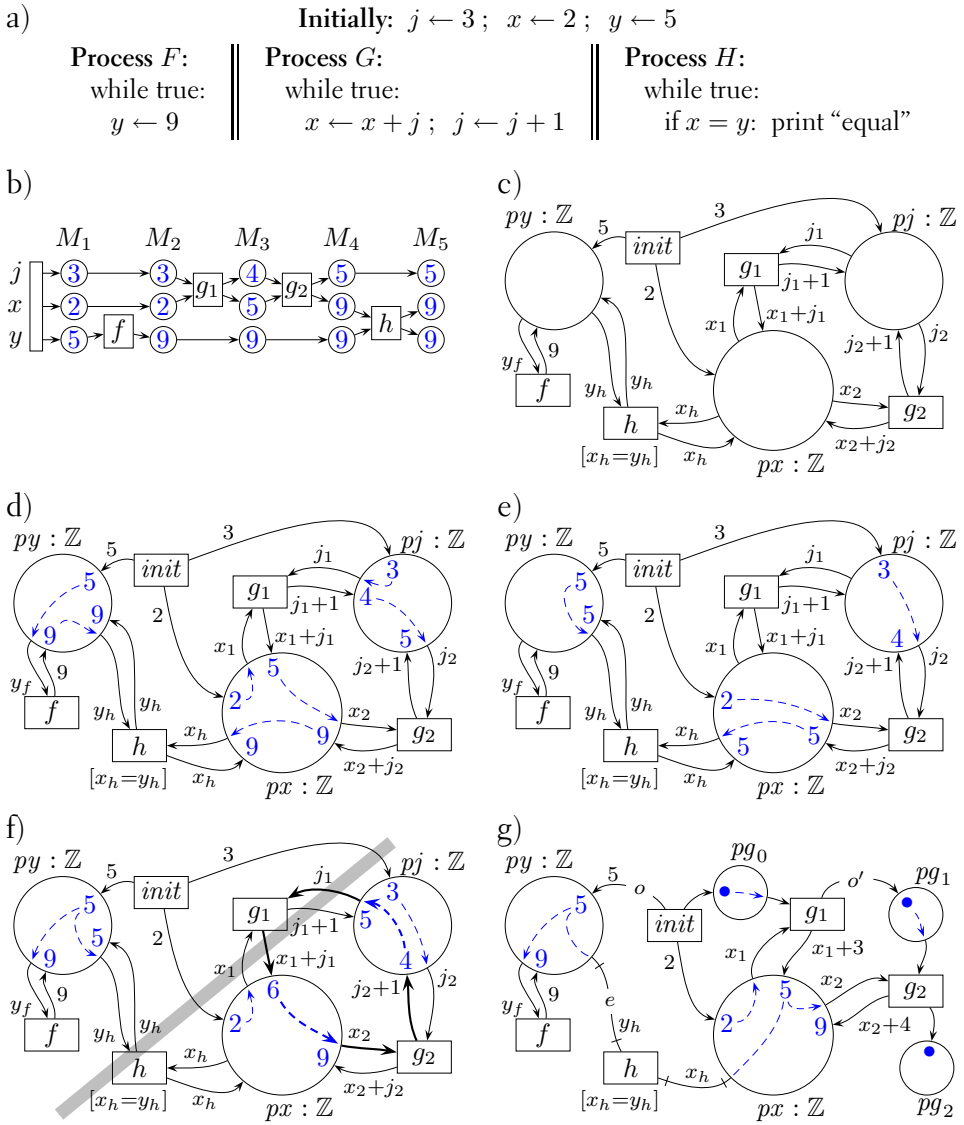

Figure 1: An example system and illustrations of its behavior.

can ever print "equal". The execution in Fig. 1b shows that the property holds: after one cycle of process F and two cycles of G, both x and y have the value 9, and process H then runs the print statement. The circles represent the values of variables in states $M_1, M_2, \ldots$, and the rectangles f, g_1, g_2, and h represent the atomic execution of one cycle of process F, G, G, and H, respectively.

Figure 1c shows a related high-level Petri net. We can interpret Fig. 1b as a finite execution of the Petri net as follows. The *transition* (rectangle) named *init* occurs first, producing a *token* in each of the *places* (circles) pj, px, and py, which correspond to the variables of the system. This leads to a state M_1, in which each place pj, px, and py contains one token that carries a value 3, 2, or 5, respectively. Transition f occurs next, consumes the token from place py and produces a new token with value 9. This results in a state M_2. Then, transition g_1 simultaneously consumes a token from each place pj and px, and uses their values to produce new tokens. Finally, the state M_5 is reached.

This is an example of a *one-off execution* of the Petri net. Generally, a

one-off execution is a finite sequence that starts with a state in which no place contains a token. Then, a transition occurs, consuming exactly one token with each input arc (an arrow from a place to the transition) and producing exactly one token with each output arc (an arrow from the transition to a place) while fulfilling the data constraints. This leads to a new state, and so on. The only distinctive requirement is that each transition occurs *at most once* in the sequence. The transitions that occur in a one-off execution are its *events*.

A Petri net whose set of one-off executions specifies a bounded set of behavior of a system is called an *unwinding* of the system. We assume that we are given an unwinding whose one-off executions map easily to finite-length executions of the original system. The unwinding of Fig. 1c has another one-off execution consisting of the sequence $init, g_2, h$ of events. This corresponds to process G running one cycle and then process H printing "equal". In total, this unwinding covers all executions of the system in which process F runs at most one cycle, process G at most two cycles, and H at most one cycle, in any possible order.

We observe that every token consumed during a one-off execution has been previously produced. Figure 1d illustrates this idea for the one-off execution of Fig. 1b. Transition g_1 consumes the token with value 2 produced by $init$, whereas the token with value 5 in place pj is not consumed at all. The numeric values and dashed arrows inside the big circles in Fig. 1d constitute an example of what we call a *token trace* of the unwinding. The token trace tells us some facts about the course of events. By following the arrows, we see that $init$ occurs before g_1, which occurs before g_2, but we cannot infer whether f occurs before or after, say, g_2. A token trace generally fixes only a *partial order* of events. Figure 1e illustrates another token trace of the same unwinding. This time, transitions f and g_1 do not occur at all. We can check that this token trace describes the second one-off execution discussed above.

It turns out that by specifying a set of rules for constructing a token trace of a fixed unwinding, we can characterize the set of *all* one-off executions of the unwinding. In other words, an unwinding induces a set of one-off executions and a set of token traces, and there is a meaningful correspondence relation between the two sets. We can thus reduce the search for a one-off execution with a certain property to finding a corresponding token trace. Given an unwinding, the rules are as follows.

1. A token trace consists of events, links (dashed arrows), and data values.
2. A subset of the transitions of the unwinding are chosen to be events.
3. Each output arc of each event is associated with a single token with a value.
4. Each input arc of each event is linked to an output arc of an event.
5. No two input arcs are linked to the same output arc.
6. The data constraints of all events are fulfilled by the values of tokens.
7. The links impose a partial order on the events.

Figure 1f contains a third attempt at a token trace of the same unwinding. However, there are several problems. First, transitions f and h are consuming the same token at place py. This breaks rule 5—an input arc denotes a destructive read operation. Second, transition h poses as an event although it gets no input from place px, breaking rule 4. Third, there is an illegal cycle, illustrated in thick arrows, that breaks rule 7. Event g_1 produces a token with value 6, then g_2 consumes it and produces a token with value 4, which in turn is consumed by g_1. No chronological ordering of the occurrences agrees with the picture. Any of these three mistakes suffices to tell that Fig. 1f does not represent a valid token trace.

A model checking procedure. The discussion above suggests the following procedure for checking reachability properties of an asynchronous system. Generate an unwinding such that one-off executions of the unwinding map to finite executions of the system, and the reachability property corresponds to the occurrence of a transition $t^\diamond$. Generate automatically a formula that encodes the rules for a token trace of the unwinding and add the constraint that $t^\diamond$ is an event. Feed the formula to an off-the-shelf satisfiability solver. If the formula is satisfiable, convert the satisfying interpretation to a token trace and further to an execution that witnesses the property. If the formula is unsatisfiable, expand the unwinding to cover more executions of the system, and start over.

Constructing unwindings. Figure 1c demonstrates a rudimentary way of obtaining unwindings: define a place for every variable of the system, add an initial transition that produces the starting values, and add a transition for each atomic action that the system can perform. To expand the unwinding, make distinct copies of some of the transitions.

This results in unwindings of a certain shape, but we will see that unwindings can be much more versatile. Namely, one can set up input and output arcs in arbitrary cyclic or acyclic configurations, decoupling the production of tokens from the consumption. In general, a place can contain any number of tokens during a one-off execution. Consequently, unwindings follow conventional Petri net semantics with the restriction that the transitions of an unwinding are treated as *potential events*: each of them occurs once or not at all.

Figure 1g shows another unwinding of our system. The labels o, o', and e do not contribute to the semantics—they only name some arcs for later reference. A token in place pg_0 denotes the fact that process G has not started. The token carries a meaningless value denoted by $\bullet$. Transition g_1 represents the first cycle of process G, and as variable j is essentially local to G, we can inline its initial value $j = 3$ in g_1. A token in pg_1 means that G has executed exactly 1 cycle, or equivalently that j equals 4, and so on. Essentially, we have unwound the while loop twice, which results in a control flow graph with three control locations represented by places pg_0, pg_1, and pg_2. The new unwinding covers exactly the same set of executions as the one in Fig. 1c. We consider the control flow based solution superior, since it breaks the symmetry of transitions g_1 and g_2 and avoids the arithmetic on the loop counter j. Development of this idea is left for future work.

Another change in Fig. 1g is that transition h is incident to two *test arcs* (lines with a cross bar close to each end). A test arc represents an ordinary, non-destructive read operation. It is like an input arc but does not consume the token, and it is usually behaviorally equivalent to a pair of input and output arcs. The use of test arcs is optional, but they may result in a more efficient encoding. The following rules need to be added for token traces. Each test arc is linked to an output arc, and multiple test arcs plus at most one input arc can be linked to the same output arc. The partial order must be such that a transition that tests a token occurs after the transition that produces the token. A third transition can consume the token, but it must occur after the testing transition. The token trace in Fig. 1g obeys these rules. In particular, transition h occurs after g_1 and before g_2.

3 SEMANTICS OF UNWINDINGS

We will use the following notations for formalizing unwindings and token traces.

For a function $f : X \to Y$, sets $A \subseteq X$, $B \subseteq Y$, and an element $y \in Y$, we call X the *domain* of f and Y the *codomain* of f, and we adopt the usual notation $f(A) := \{f(x) \mid x \in A\}$, $f^{-1}(B) := \{x \in X \mid f(x) \in B\}$, and $f^{-1}(y) := f^{-1}(\{y\})$. The function is *injective* iff $|f^{-1}(y)| \leq 1$ for all $y \in Y$. The function $f|_A : A \to Y$ defined as $f|_A(x) := f(x)$ for all $x \in A$ is called the *restriction of f to A*.

We will use *types*, *variables*, and *expressions* to model data manipulation in systems. Each type is identified with the set of elements of the type; in particular, the Boolean type is $\mathbb{B} = \{\text{false}, \text{true}\}$. Every variable v and expression ϕ has a type $\mathit{type}(v)$ or $\mathit{type}(\phi)$. The set of variables in an expression or a set of expressions ϕ is denoted by $\mathit{vars}(\phi)$. A *binding* of a set V of variables maps each variable $v \in V$ to a value $d \in \mathit{type}(v)$. If ϕ is an expression and b is a binding of (a superset of) $\mathit{vars}(\phi)$, the *value of ϕ in b*, denoted by ϕ^b, is obtained by substituting $b(v)$ for each occurrence of a variable $v \in \mathit{vars}(\phi)$ in the expression and evaluating the result. We will not fix a concrete language for expressions—the choice of a proper language depends on the problem domain and on the capabilities of the solver used.

A *multiset* M over a set U is a function $U \to \mathbb{N}$, interpreted as a collection that contains $M(u)$ indistinguishable copies of each element $u \in U$. A multiset M is *finite* iff the sum $\sum_{u \in U} M(u)$ is finite. When the base set U is clear from the context, we will identify an ordinary set $A \subseteq U$ with the multiset χ_A over U, defined as $\chi_A(u) = 1$ if $u \in A$ and $\chi_A(u) = 0$ otherwise. If M_1 and M_2 are multisets over U, then M_1 is a *subset* of M_2, denoted $M_1 \leq M_2$, iff $M_1(u) \leq M_2(u)$ for all $u \in U$. Multiset M *contains* an element $u \in U$, denoted $u \in M$, iff $M(u) \geq 1$. If $M_1, M_2, \ldots$ are multisets over U, we will use $M_1 + M_2$ and $M_2 - M_1$ with their usual meanings (as functions) to denote multiset union and multiset difference, respectively. The latter is defined only if $M_1 \leq M_2$.

A binary relation $\prec$ over a set X is a *strict partial order* iff it is irreflexive, asymmetric, and transitive, that is, iff for all $x, y, z \in X$ (i) $x \prec y$ implies not $y \prec x$ and (ii) $x \prec y$ and $y \prec z$ together imply $x \prec z$. A *strict total order*

is a strict partial order that additionally fulfills for all $x, y \in X$ (iii) $x \prec y$ or $y \prec x$.

3.1 Colored Contextual Unweighted Petri Nets

Colored Petri Nets [16] are a powerful language for the design and analysis of distributed systems. In this work however, we use Petri nets with restricted semantics to specify a bounded portion of the behavior of a system. Our variant is called *Colored Contextual Unweighted Petri Nets*, or "nets" for short. The word *contextual* means that nets can contain test arcs [5], allowing compact modeling of non-destructive read operations. By *unweighted* we mean that each arc is associated with a single token instead of a multiset of tokens as in Colored Petri Nets. This restriction is crucial for the encoding, but does not seriously weaken the formalism. Places can still contain multisets of tokens, and multiple arcs can be placed in parallel to move several tokens at the same time.

Definition 1. *A* net *is a tuple* $N = \langle \Sigma, P, T, A_{in}, A_{test}, A_{out}, place, trans, colors, guard, expr \rangle$, *where*

1. Σ *is a set of non-empty* types *(sometimes called color sets),*
2. P *is a set of* places*,*
3. T *is a set of* transitions*,*
4. A_{in} *is a set of* input arcs*,*
5. A_{test} *is a set of* test arcs*,*
6. A_{out} *is a set of* output arcs*,*
7. P*,* T*,* A_{in}*,* A_{test}*, and* A_{out} *are all pairwise disjoint,*
8. $place$ *is a* place incidence function $A_{in} \cup A_{test} \cup A_{out} \to P$*,*
9. $trans$ *is a* transition incidence function $A_{in} \cup A_{test} \cup A_{out} \to T$*,*
10. *the set* $trans^{-1}(t)$ *is finite for all* $t \in T$*,*
11. $colors$ *is a* color function $P \to \Sigma$*,*
12. $guard$ *is a* guard function *over* T *such that for all* $t \in T$*,* $guard(t)$ *is an expression with* $type(guard(t)) = \mathbb{B}$ *and* $type(vars(guard(t))) \subseteq \Sigma$*,*
13. $expr$ *is an* arc expression function *over* $A_{in} \cup A_{test} \cup A_{out}$ *such that for all arcs* a*,* $expr(a)$ *is an expression with* $type(expr(a)) = colors(place(a))$ *and* $type(vars(expr(a))) \subseteq \Sigma$*,*

A net is *finite* iff P and T are finite sets. For a transition or a set of transitions t and a place or a set of places p, we use the shorthand notations

$$\begin{aligned} in(t) &:= A_{in} \cap trans^{-1}(t) \ , & in(p) &:= A_{in} \cap place^{-1}(p) \ , \\ test(t) &:= A_{test} \cap trans^{-1}(t) \ , & test(p) &:= A_{test} \cap place^{-1}(p) \ , \\ out(t) &:= A_{out} \cap trans^{-1}(t) \ , & out(p) &:= A_{out} \cap place^{-1}(p) \ , \\ vars(t) &:= vars(guard(t)) \cup \bigcup_{a \in trans^{-1}(t)} vars(expr(a)) \ . \end{aligned}$$

In particular, the variables of a transition are the variables that appear either in its guard or in the arc expression of any incident arc.

Figure 1g shows a net where $place(o) = py$, $trans(o) = init$, $test(py) = \{e\}$, $out(pg_1) = \{o'\}$, $place(in(g_2)) = \{px, pg_1\}$, and $colors(py) = \mathbb{Z}$. Some of the data constraints are $expr(e) = y_h$ and $guard(h) = (x_h{=}y_h)$, with $vars(h) = \{x_h, y_h\}$ and $vars(init) = \emptyset$. We omit vacuously true guards, so $guard(f) = \text{true}$ implicitly. Also, $colors(pg_1)$ is implicitly the type $\{\bullet\}$ with only one meaningless value, and $expr(o')$ is the constant expression $\bullet$.

A *token element* is a pair $\langle p, d \rangle$, where $p \in P$ is a place and $d \in colors(p)$ is a value. A *marking* M is a finite multiset over the set of token elements. Markings represent states of the system. The interpretation of the multiset M is that if $M(\langle p, d \rangle) = n$, then in state M, place p contains n tokens of value d.

A *binding element* is a pair $\langle t, b \rangle$, where $t \in T$ is a transition and b is a binding of $vars(t)$. Let us use the following shorthand notations representing the multisets of token elements consumed and produced by a binding element.

$$\begin{aligned} consumed_{\langle t,b \rangle} &:= \sum_{c \in in(t)} \left\{ \langle place(c), expr(c)^b \rangle \right\} \ , \\ produced_{\langle t,b \rangle} &:= \sum_{o \in out(t)} \left\{ \langle place(o), expr(o)^b \rangle \right\} \ . \end{aligned}$$

Definition 2. *A binding element $\langle t, b \rangle$ is* enabled *in a marking M iff the following conditions hold.*

1. $consumed_{\langle t,b \rangle} \leq M$,
2. $\langle place(e), expr(e)^b \rangle \in \left(M - consumed_{\langle t,b \rangle} \right)$ *for all* $e \in test(t)$, *and*
3. $guard(t)^b = \text{true}$.

The binding element can occur *in the marking iff it is enabled in the marking, leading to a new marking $M' = M - consumed_{\langle t,b \rangle} + produced_{\langle t,b \rangle}$.*

We denote by $M \ [t, b\rangle \ M'$ the fact that the binding element is enabled in M and leads from M to M' if it occurs. A *finite occurrence sequence* of a net is a finite sequence $M_0 \ [t_1, b_1\rangle \ M_1 \ \cdots \ [t_k, b_k\rangle \ M_k$ such that $k \geq 0$ and $M_{i-1} \ [t_i, b_i\rangle \ M_i$ holds for each $1 \leq i \leq k$. The *length* of the finite occurrence sequence is k, and the *final marking* is M_k.

3.2 Unwindings and One-Off Executions

We define an *unwinding* to be any net $N = \langle \Sigma, P, T, \ldots, expr \rangle$ that fulfills the two constraints below. Those constraints are just technicalities—the true restriction is that we allow each transition of an unwinding to occur at most once. Thus, a *one-off execution* of an unwinding is a finite occurrence sequence $M_0 \, [t_1, b_1\rangle \, M_1 \, \cdots \, [t_k, b_k\rangle \, M_k$ such that $M_0 = \emptyset$ and $t_i \neq t_j$ for all $1 \leq i < j \leq k$. The set $\{t_1, \ldots, t_k\}$ is the *event set* of the one-off execution. A transition $t \in T$ is *one-off reachable* iff it is an event in some one-off execution.

The informal requirement for constructing an unwinding is that there is an easily computable projection from the one-off executions of an unwinding to some finite executions of the system under verification. Furthermore, the reachability property in question should hold if a specific transition $t^\diamond \in T$ is one-off reachable. If the original property is expressed in terms of the state of the system, the property must be somehow mapped to the enabledness condition of $t^\diamond$. The technical requirements for an unwinding N are as follows.

1. Transitions do not share variables: when $t \in T$ and $u \in T$ are distinct, $vars(t) \cap vars(u) = \emptyset$. We can always achieve this by renaming variables if necessary, as done in Fig. 1c by using subscripts.

2. Every place is incident to an output arc: $out(p) \neq \emptyset$ for all $p \in P$. As places with no incident output arcs are useless, they can be eliminated.

One possible one-off execution of the unwinding of Fig. 1c is the sequence $M_0 \, [init, b_{init}\rangle \, M_1 \, [f, b_f\rangle \, M_2$, where $M_2 = \{\langle pj, 3\rangle, \langle px, 2\rangle, \langle py, 9\rangle\}$, the binding b_{init} is empty, and ${y_f}^{b_f} = 5$. The initial marking M_0 is fixed to be empty, but we work around this by specifying the starting conditions with a transition *init* that necessarily occurs once in the beginning of any non-trivial one-off execution.

3.3 Token Traces

Let us formalize the rules presented in Sect. 2 for a token trace.

Definition 3. *A* token trace *of an unwinding $N = \langle \Sigma, P, T, \ldots, expr \rangle$ is a tuple $R = \langle E, src, b \rangle$, where*

1. *$E \subseteq T$ is a finite set of* events*,*

2. *src is a* source function *$in(E) \cup test(E) \to out(E)$ such that*

   (a) *$place(a) = place(src(a))$ for all arcs $a \in in(E) \cup test(E)$ and*

   (b) *$src(c_1) \neq src(c_2)$ for all input arcs $c_1, c_2 \in in(E)$ such that $c_1 \neq c_2$,*

3. *b is a binding of $vars(E)$, called the* total binding*, such that $expr(a)^b = expr(src(a))^b$ for all arcs $a \in in(E) \cup test(E)$,*

4. *$guard(t)^b = \text{true}$ for all events $t \in E$,*

5. *there exists a strict partial order* $\prec$ *over the set* E *such that*

   (a) $trans(src(a)) \prec trans(a)$ *for all arcs* $a \in in(E) \cup test(E)$ *and*

   (b) $trans(e) \prec trans(c)$ *for all test arcs* $e \in test(E)$ *and input arcs* $c \in in(E)$ *such that* $src(e) = src(c)$.

Relating to Sect. 2, the source function forms the links between the arcs, while the total binding takes care of the data constraints. According to item 3, the arc expression at each end of a link must evaluate to the same value, i.e. the value of the token. As $vars(E)$ is a *disjoint* union of the variables of each event, b can bind the variables of each event independently. Item 5 above says that the events can be ordered in such a way that each token is produced before any event consumes or tests it, and a token is not tested during or after its consumption. Any strict partial order over (a superset of) E that fulfills item 5 will be called a *chronological partial order* of the token trace.

Figure 1g portrays a token trace where $E = T$, ${y_f}^b = {x_h}^b = 5$, $src(e) = o$, $src(in(E)) \cap out(g_2) = \emptyset$, and necessarily $h \prec g_2$ and $init \prec g_2$. One of $f \prec g_2$ and $g_2 \prec f$ can be true, or both can be false, but not both true.

3.4 Mappings Between Token Traces and One-Off Executions

From a one-off execution $M_0\ [t_1, b_1\rangle\ M_1\ \cdots\ [t_k, b_k\rangle\ M_k$, we can construct a token trace by conjoining $b_1, \ldots, b_k$ to a total binding and tracing each consumed or tested token to its source. The interleaving $t_1 \prec t_2 \prec \cdots \prec t_k$ then gives a chronological partial order. Conversely, we can take a token trace and linearize its chronological partial order to obtain a one-off execution. These constructions constitute the proof of the following theorem.

Theorem 1. *Given an unwinding* N *and a finite subset* E *of transitions, there is a one-off execution of* N *with event set* E *if and only if there is a token trace of* N *with event set* E.

Before going to the proof, we need to define a few more concepts. The set of *residual producers* of a token trace is the set

$$out_{res}(R) := out(E) \setminus src(in(E)) \ ,$$

that is the set of output arcs that produce a token that is not consumed by any input arc. The *residual marking* of a token trace is the corresponding multiset of token elements

$$\sum_{o \in out_{res}(R)} \left\{ \langle place(o), expr(o)^b \rangle \right\} \ .$$

The residual marking of the token trace of Fig. 1d is the same as marking M_5 in Fig. 1b.

A subset E' of events E is *downward closed* in $\prec$ iff $t \prec t'$ implies $t \in E'$ for all $t \in E$ and $t' \in E'$. A downward closed subset can be used to construct a "prefix" of a token trace as shown by the following lemma.

Lemma 2. *Let $R = \langle E, src, b\rangle$ be a token trace of an unwinding N and let $\prec$ be a chronological partial order of R. If E' is a subset of E that is downward closed in $\prec$, then the tuple $R' := \langle E', src', b'\rangle$ is a token trace of N, where src' is the restriction of src to $in(E') \cup test(E')$ and b' is the restriction of b to $vars(E')$. Furthermore, $\prec$ is a chronological partial order of R'.*

Proof. Assume E' is a downward closed subset of E, and let src', b', and R' be defined as stated in the lemma. For src' to be a valid source function, it must map $in(E') \cup test(E')$ onto $out(E')$. Let a be an arbitrary arc in $in(E') \cup test(E')$. Then $trans(src(a)) \prec trans(a)$ holds because $\prec$ is a chronological partial order of R. Because $trans(a) \in E'$ and E' is downward closed, also $trans(src(a))$ is in E', in other words $src(a)$ is in $out(E')$. Thus src' is a mapping from $in(E') \cup test(E')$ onto $out(E')$.

All other requirements of Definition 3 follow directly for R' and $\prec$ because R is a token trace. □

The following two lemmas state that we can get a token trace from any one-off execution, and vice versa. Theorem 1 then follows immediately.

Lemma 3. *If $M_0\ [t_1, b_1\rangle\ M_1\ \cdots\ [t_k, b_k\rangle\ M_k$ is a one-off execution of an unwinding N, then there is a token trace of N with event set $E = \{t_1, \ldots, t_k\}$ and residual marking M_k.*

Proof. Proof by induction over k. If $k = 0$, then $M_k = M_0 = \emptyset$, and we can pick the empty token trace, which has event set $\emptyset$ and residual marking $\emptyset$.

Let $k > 0$. Assume $X := M_0\ [t_1, b_1\rangle\ M_1\ \cdots\ [t_k, b_k\rangle\ M_k$ is a one-off execution and let $E := \{t_1, \ldots, t_k\}$. Let

$$consumed_k := \sum_{c \in in(t_k)} \left\{\langle place(c), expr(c)^{b_k}\rangle\right\} , \tag{1}$$

$$produced_k := \sum_{o \in out(t_k)} \left\{\langle place(o), expr(o)^{b_k}\rangle\right\} \tag{2}$$

be the token elements consumed and produced by $\langle t_k, b_k\rangle$.

It follows immediately from the definition of one-off execution that $X' := M_0\ [t_1, b_1\rangle\ M_1\ \cdots\ [t_{k-1}, b_{k-1}\rangle\ M_{k-1}$ is also a one-off execution. By the inductive hypothesis, there is a token trace $R' = \langle E', src', b'\rangle$ with event set $E' := E \setminus \{t_k\}$ and residual marking M_{k-1}. By the definition of residual marking, we have

$$M_{k-1} = \sum_{o \in out_{res}(R')} \left\{\langle place(o), expr(o)^{b'}\rangle\right\} . \tag{3}$$

Because the binding element $\langle t_k, b_k\rangle$ is enabled in marking M_{k-1}, we have $consumed_k \leq M_{k-1}$. Replace $consumed_k$ with its definition and M_{k-1} with the right-hand side of (3) to obtain

$$\sum_{c \in in(t_k)} \left\{\langle place(c), expr(c)^{b_k}\rangle\right\} \leq \sum_{o \in out_{res}(R')} \left\{\langle place(o), expr(o)^{b'}\rangle\right\} .$$

Because $in(t_k)$ is a finite set, we can by this multiset inclusion pick for every input arc $c \in in(t_k)$ an output arc in $out_{res}(R')$, call it $src_{in}(c)$,

such that $\langle place(c), expr(c)^{b_k} \rangle = \langle place(src_{in}(c)), expr(src_{in}(c))^{b'} \rangle$. Furthermore, we can construct this mapping in such a way that no two input arcs are mapped to the same output arc. This way, we obtain an injective function $src_{in} : in(t_k) \to out_{res}(R')$ that satisfies

$$place(c) = place(src_{in}(c)) \quad \text{for all } c \in in(t_k), \text{ and} \tag{4}$$
$$expr(c)^{b_k} = expr(src_{in}(c))^{b'} \quad \text{for all } c \in in(t_k). \tag{5}$$

Now, $consumed_k$ can be expressed as

$$\begin{aligned} consumed_k = & \sum_{c \in in(t_k)} \left\{ \langle place(c), expr(c)^{b_k} \rangle \right\} = \\ & \sum_{c \in in(t_k)} \left\{ \langle place(src_{in}(c)), expr(src_{in}(c))^{b'} \rangle \right\} = \\ & \sum_{o \in src_{in}(in(t_k))} \left\{ \langle place(o), expr(o)^{b'} \rangle \right\} \ . \end{aligned} \tag{6}$$

Let U be the set $out_{res}(R') \setminus src_{in}(in(t_k))$, i.e. the set of output arcs that are residual producers of R' but are not matched to any input arc of t_k. Then it can be seen from (3) and (6) that

$$M_{k-1} - consumed_k = \sum_{o \in U} \left\{ \langle place(o), expr(o)^{b'} \rangle \right\} \ . \tag{7}$$

Definition 2, when applied to $\langle t_k, b_k \rangle$ which is enabled in M_{k-1}, says that for all test arcs $e \in test(t_k)$, the token element $\langle place(e), expr(e)^{b_k} \rangle$ is in the multiset $M_{k-1} - consumed_k$. By (7), we can thus map each test arc $e \in test(t_k)$ to an output arc $src_{test}(e)$ in U such that $\langle place(e), expr(e)^{b_k} \rangle = \langle place(src_{test}(e)), expr(src_{test}(e))^{b'} \rangle$. This way, we define another new function $src_{test} : test(t_k) \to U$ that satisfies

$$place(e) = place(src_{test}(e)) \quad \text{for all } e \in test(t_k), \text{ and} \tag{8}$$
$$expr(e)^{b_k} = expr(src_{test}(e))^{b'} \quad \text{for all } e \in test(t_k). \tag{9}$$

The domains of the functions src_{in}, src_{test}, and src' are pairwise disjoint and they cover the set $in(E) \cup test(E)$. The codomains of these functions are contained in $out(E)$. Thus, we can conjoin the functions as a new source function $src : in(E) \cup test(E) \to out(E)$ defined by

$$src(a) := \begin{cases} src_{in}(a) & \text{if } a \in in(t_k), \\ src_{test}(a) & \text{if } a \in test(t_k), \\ src'(a) & \text{if } a \in in(E') \cup test(E'). \end{cases}$$

Similarly, we define a new total binding b of the variables $vars(E)$ by

$$b(v) := \begin{cases} b_k(v) & \text{if } v \in vars(t_k), \\ b'(v) & \text{otherwise.} \end{cases}$$

The claim is now that $R := \langle E, src, b \rangle$ is a token trace of N. We will prove the claim step by step by going through the items of Definition 3.

1. E is a finite subset of transitions of N as it is the event set of a one-off execution of N.

2. As stated above, src is a function from $in(E) \cup test(E)$ to $out(E)$.

   (a) By (4) and (8), we have $place(a) = place(src(a))$ for all arcs $a \in in(t_k) \cup test(t_k)$. For $a \in in(E') \cup test(E')$, the equality $place(a) = place(src(a))$ holds because src' is the source function of a token trace R'.

   (b) Assume $c_1, c_2 \in in(E)$ and $c_1 \neq c_2$. If c_1 and c_2 are both in $in(E')$, then $src(c_1) \neq src(c_2)$ because src' is a valid source function. If c_1 and c_2 are both in $in(t_k)$, then $src(c_1) \neq src(c_2)$ because src_{in} is injective by construction. The remaining case is that $c_1 \in in(E')$ and $c_2 \in in(t_k)$. Then $src(c_1)$ is in $src'(in(E'))$ and $src(c_2)$ is in $out_{res}(R') = out(E') \setminus src'(in(E'))$, and hence $src(c_1) \neq src(c_2)$. Thus, src restricted to $in(E)$ is injective.

3. Binding b gives values to $vars(t)$ for all transitions $t \in E$, and the equality $expr(a)^b = expr(src(a))^b$ follows for all $a \in in(E) \cup test(E)$ from (5), (9), and the fact that b' is the total binding of a token trace R'.

4. We have $guard(t_k)^b = guard(t_k)^{b_k} = \text{true}$ because $\langle t_k, b_k \rangle$ is enabled in a marking M_{k-1}. For $t \in E$, $t \neq t_k$, the guard value $guard(t)^b = guard(t)^{b'}$ is true because b' is the total binding of R'.

5. Let $\prec'$ be a strict partial order over E' that is a chronological partial order of R'. Define $\prec$ as the relation $\prec' \cup \{(t, t_k) \mid t \in E'\}$. It is straightforward to check that $\prec$ is a strict partial order over E. Let us verify that $\prec$ is a chronological partial order of R.

   (a) Assume $a \in in(E) \cup test(E)$. If $trans(a)$ is in E', then we have $trans(src'(a)) \prec' trans(a)$ and thus $trans(src(a)) \prec trans(a)$ holds. If, on the other hand, $trans(a) = t_k$, then $src(a)$ is in $out_{res}(R') \subseteq out(E')$ and thus $trans(src(a))$ is in E'. Therefore $trans(src(a)) \prec trans(a)$.

   (b) Assume arcs $e \in test(E)$ and $c \in in(E)$ such that $src(e) = src(c)$. We cannot have $trans(e) = t_k$, because this would imply $src(c) = src(e) = src_{test}(e) \in U = out_{res}(R') \setminus src_{in}(in(t_k)) = out(E') \setminus src(in(E')) \setminus src(in(t_k)) = out(E') \setminus src(in(E))$, which entails the contradiction $src(c) \notin src(in(E))$. Therefore $trans(e)$ is in E'. Now if $trans(c)$ is in E', then $trans(e) \prec' trans(c)$ holds because $\prec'$ is a chronological partial order of R', and consequently $trans(e) \prec trans(c)$ is true. On the other hand, if $trans(c) = t_k$, then $trans(e) \prec trans(c)$ follows directly from $trans(e) \in E'$ and the definition of $\prec$. In either case, we have $trans(e) \prec trans(c)$.

It is now established that R is a token trace of N. We still have to show

that the residual marking of R is M_k. First, the residual producers of R are

$$\begin{aligned} out_{res}(R) = out(E) \setminus src(in(E)) = \\ out(t_k) \cup out(E') \setminus src'(in(E')) \setminus src_{in}(in(t_k)) = \\ out(t_k) \cup out_{res}(R') \setminus src_{in}(in(t_k)) = out(t_k) \cup U. \end{aligned}$$

The union is disjoint because t_k does not share output arcs with E'. By Definition 2, the final marking M_k is $M_{k-1} - consumed_k + produced_k$. Applying (7) and (2), we get

$$\begin{aligned} M_k = \sum_{o \in U} \left\{ \langle place(o), expr(o)^{b'} \rangle \right\} + \sum_{o \in out(t_k)} \left\{ \langle place(o), expr(o)^{b_k} \rangle \right\} = \\ \sum_{o \in out_{res}(R)} \left\{ \langle place(o), expr(o)^{b} \rangle \right\}. \end{aligned}$$

The last sum is exactly the residual marking of R. □

Lemma 4. *If $R = \langle E, src, b \rangle$ is a token trace of an unwinding N, then there is a one-off execution $M_0\ [t_1, b_1\rangle\ M_1\ \cdots\ [t_k, b_k\rangle\ M_k$ of N such that $E = \{t_1, \ldots, t_k\}$ and the residual marking of R is M_k.*

Proof. Proof by induction over $k = |E|$. If $k = 0$, then we have the empty token trace whose residual marking is $\emptyset$, and we can pick the zero-length one-off execution M_0 where $M_0 = \emptyset$.

Let $k > 0$ and assume $R = \langle E, src, b \rangle$ is a token trace with $k = |E|$. Let $\prec$ be a chronological partial order of R. Because E is finite and nonempty, it has a maximal element, call it t_k, with respect to $\prec$.

Let $E' := E \setminus \{t_k\}$. As E' is downward closed in $\prec$, we can apply Lemma 2 to obtain a new token trace $R' = \langle E', src', b' \rangle$. By the inductive hypothesis, there is a one-off execution $X' = M_0\ [t_1, b_1\rangle\ M_1\ \cdots\ [t_{k-1}, b_{k-1}\rangle\ M_{k-1}$ such that $E' = \{t_1, \ldots, t_{k-1}\}$ and the residual marking of R' is M_{k-1}. Let b_k be the restriction of b to $vars(t_k)$, and define

$$consumed_k := \sum_{c \in in(t_k)} \left\{ \langle place(c), expr(c)^{b_k} \rangle \right\} \ , \tag{10}$$

$$produced_k := \sum_{o \in out(t_k)} \left\{ \langle place(o), expr(o)^{b_k} \rangle \right\} \ . \tag{11}$$

We claim that X' can be extended to a one-off execution X of length k by appending an occurrence of the binding element $\langle t_k, b_k \rangle$ and that the final marking of X is the residual marking of R. For this, we need to show that $M_{k-1}\ [t_k, b_k\rangle\ M_k$ holds, when M_k is the residual marking of R. According to Definition 2, we have to prove the following four items.

P1. $consumed_k \leq M_{k-1}$,

P2. $\langle place(e), expr(e)^{b_k} \rangle \in (M_{k-1} - consumed_k)$ for all $e \in test(t_k)$,

P3. $guard(t_k)^{b_k} = \text{true}$, and

P4. the residual marking of R is $M_{k-1} - consumed_k + produced_k$.

By items 2 and 3 of Definition 3, we know that for all $c \in in(t_k)$, we have $place(c) = place(src(c))$ and $expr(c)^{b_k} = expr(c)^b = expr(src(c))^b$, and that src restricted to $in(t_k)$ is injective. We can thus rewrite (10) as

$$consumed_k = \sum_{o \in src(in(t_k))} \left\{ \langle place(o), expr(o)^b \rangle \right\} \ . \tag{12}$$

Furthermore, M_{k-1} is the residual marking of R':

$$M_{k-1} = \sum_{o \in out_{res}(R')} \left\{ \langle place(o), expr(o)^b \rangle \right\} \ . \tag{13}$$

To prove item P1, i.e. that $consumed_k \leq M_{k-1}$, it suffices by (12) and (13) to show that $src(in(t_k)) \subseteq out_{res}(R')$. Recall that $out_{res}(R') = out(E') \setminus src(in(E'))$. The set $src(in(t_k))$ must be a subset of $out(E')$ because otherwise there would be an input arc $c \in in(t_k)$ such that $src(c) \in out(t_k)$, which by item 5a of Definition 3 would imply $t_k \prec t_k$, a contradiction. Also, $src(in(t_k))$ is disjoint from $src(in(E'))$ because src restricted to input arcs is injective. Thus $src(in(t_k)) \subseteq out(E') \setminus src(in(E'))$, and item P1 is proved.

Let U be the set of output arcs $U := out_{res}(R') \setminus src(in(t_k)) = out(E') \setminus src(in(E')) \setminus src(in(t_k)) = out(E') \setminus src(in(E))$, and let

$$M_U := \sum_{o \in U} \left\{ \langle place(o), expr(o)^b \rangle \right\} \ . \tag{14}$$

By (12) and (13), $M_U = M_{k-1} - consumed_k$.

To prove item P2, we fix an arbitrary test arc $e \in test(t_k)$ and show that the token element $\langle place(e), expr(e)^{b_k} \rangle$ is in $M_{k-1} - consumed_k$, or equivalently, in M_U. Because R is a token trace, that token element is the same as $\langle place(src(e)), expr(src(e))^b \rangle$, so by (14) suffices to show that $src(e) \in U$. First, $src(e)$ is in $out(E')$ because otherwise $src(e)$ would be in $out(t_k)$, and this, together with item 5a of Definition 3, would imply the contradiction $t_k \prec t_k$. Second, $src(e)$ is not in $src(in(E))$. Namely, if this were the case, there would be an input arc $c \in in(E)$ such that $src(e) = src(c)$, and by item 5b of Definition 3, this would mean that $t_k = trans(e) \prec trans(c)$, contradicting the choice of t_k as a maximal element with respect to $\prec$. Therefore, $src(e) \in out(E') \setminus src(in(E)) = U$, and item P2 is established.

Item P3 follows directly by Definition 3, item 4 from the fact that t_k is an event of the token trace R.

Because the set of residual producers of R is $out_{res}(R) = out(E) \setminus src(in(E)) = out(t_k) \cup out(E') \setminus src(in(E)) = out(t_k) \cup U$, where the union is disjoint, the residual marking of R is

$$\sum_{o \in out(t_k) \cup U} \left\{ \langle place(o), expr(o)^b \rangle \right\} =$$

$$M_U + \sum_{o \in out(t_k)} \left\{ \langle place(o), expr(o)^{b_k} \rangle \right\} =$$

$$(M_{k-1} - consumed_k) + produced_k,$$

and item P4 holds. This completes the proof. □

4 ENCODING TOKEN TRACES

Let $N = \langle \Sigma, P, T, \ldots, expr \rangle$ be a finite unwinding. We are interested in whether a transition $t^\diamond \in T$ is one-off reachable, or equivalently, whether there is a token trace of N whose event set contains $t^\diamond$. In this section, we will construct a formula that is satisfiable if and only if such a token trace exists. In Sect. 4.2, we will revise the encoding to potentially speed up satisfiability checking.

A formula ϕ is satisfiable iff there is an interpretation I such that ϕ^I is true. In this context, an interpretation is a binding of the symbols in the formula. In propositional satisfiability (SAT), the formula only contains propositional (Boolean) symbols and Boolean connectives. Extensions known as SMT [9] also allow non-Boolean constraints. For example, an interpretation I satisfies the formula $\bigwedge_{j \in J}(\mathsf{X}_j < \mathsf{Y}_j)$, where the X_j and Y_j are symbols of real type, if and only if ${\mathsf{X}_j}^I$ is less than ${\mathsf{Y}_j}^I$ for all $j \in J$.

The formula will be built using the following set of symbols:

- for each $t \in T$, a propositional symbol Occur_t ("transition t occurs"),
- for each $t \in T$, a symbol Time_t of type $\mathbb{R}$ ("when transition t occurs"),
- for each pair $o \in A_{out}, a \in A_{in} \cup A_{test}$ such that $place(o) = place(a)$, a propositional symbol $\mathsf{Link}_{o,a}$ ("arc a is linked to arc o"), and
- for each $v \in vars(T)$, a symbol Val_v of type $type(v)$ ("the value of v").

We get an interpretation from a token trace $\langle E, src, b \rangle$ by setting ${\mathsf{Occur}_t}^I$ to true iff $t \in E$, setting ${\mathsf{Link}_{o,a}}^I$ to true iff $o = src(a)$, letting ${\mathsf{Val}_v}^I := v^b$, and assigning the values ${\mathsf{Time}_t}^I$ according to some chronological partial order $\prec$. The detailed construction and its reverse are presented in Sect. 4.1.

For a guard or arc expression ϕ, we will use the special notation ϕ^{vals} to denote the substitution of each variable $v \in vars(T)$ with the symbol Val_v.

The formula $\epsilon_\emptyset$ below encodes the rules for a token trace in terms of the symbols presented above. Checking the existence of a token trace containing the event $t^\diamond$ then reduces to checking the satisfiability of the formula $\epsilon_\emptyset \wedge \mathsf{Occur}_{t^\diamond}$.

$$\epsilon_\emptyset := \bigwedge_{t \in T} \gamma_t \;\wedge \bigwedge_{a \in A_{in} \cup A_{test}} \Big(\beta_a \wedge \bigwedge_{o \in out(place(a))} \psi_{o,a} \Big) \wedge \bigwedge_{p \in P} \delta_p \;. \tag{15}$$

The subformulas γ_t and β_a encode items 4 and 2a of Definition 3.

$$\gamma_t := \mathsf{Occur}_t \rightarrow guard(t)^{vals} \;, \tag{16}$$

$$\beta_a := \mathsf{Occur}_{trans(a)} \rightarrow \bigvee_{o \in out(place(a))} \mathsf{Link}_{o,a} \;. \tag{17}$$

The subformula $\psi_{o,a}$ places constraints on linking arc a to output arc o, namely that $trans(o)$ must be an event, and items 5a and 3 of Definition 3 must hold.

$$\begin{aligned} \psi_{o,a} := \big(\mathsf{Link}_{o,a} \rightarrow \mathsf{Occur}_{trans(o)}\big) \wedge \\ \big(\mathsf{Link}_{o,a} \rightarrow (\mathsf{Time}_{trans(o)} < \mathsf{Time}_{trans(a)})\big) \wedge \\ \big(\mathsf{Link}_{o,a} \rightarrow (expr(o)^{vals} = expr(a)^{vals})\big) \;. \end{aligned} \tag{18}$$

The symbol $<$ above has a fixed interpretation as the less-than relation of real numbers. The symbol $=$ denotes equality, interpreted as the identity relation in the domain of the type $colors(place(o))$. If the type happens to be the singleton type $\{\bullet\}$, then the formula $(expr(o)^{vals} = expr(a)^{vals})$ is vacuously true, and the third conjunct of (18) is omitted in practice.

The constraints in δ_p are required to make sure that tokens consumed from a place p are indeed removed. We encode items 2b and 5b of Definition 3 as

$$\begin{aligned}\delta_p := & \bigwedge_{o \in out(p)} AtMostOne\left(\left\{\mathsf{Link}_{o,c} \;\middle|\; c \in in(p)\right\}\right) \wedge \\ & \bigwedge_{o \in out(p)} \bigwedge_{e \in test(p)} \bigwedge_{c \in in(p)} \rho_{o,e,c} \;,\end{aligned} \tag{19}$$

$$\rho_{o,e,c} := \mathsf{Link}_{o,e} \wedge \mathsf{Link}_{o,c} \rightarrow (\mathsf{Time}_{trans(e)} < \mathsf{Time}_{trans(c)}) \;, \tag{20}$$

where $AtMostOne\,(\Phi)$ denotes a formula that is true iff exactly zero or one formulas in the finite set Φ are true. This can be expressed in size linear in $|\Phi|$ as follows. Letting $\Phi = \{\phi_1, \ldots, \phi_n\}$, write the formula as

$$\begin{aligned}AtMostOne\,(\Phi) = & (\phi_1 \rightarrow \neg\phi_2) \wedge \\ & ((\phi_1 \vee \phi_2) \rightarrow \neg\phi_3) \wedge \\ & \quad \vdots \\ & ((\phi_1 \vee \cdots \vee \phi_{n-1}) \rightarrow \neg\phi_n).\end{aligned}$$

With maximal sharing of the subformulas ϕ_k and $\phi_1 \vee \cdots \vee \phi_k$ for each $1 \leq k \leq n$, the size of $AtMostOne\,(\Phi)$ is $O(n)$ plus the total size of the formulas in Φ.

4.1 Interpreting the Encoding

There is a correspondence between interpretations satisfying $\epsilon_\emptyset$ and token traces, although it is not strictly one-to-one. Below, we will show explicitly how to construct a token trace from a satisfying interpretation and vice versa.

Construction 1. *Let $N = \langle \Sigma, P, T, \ldots, expr\rangle$ be a finite unwinding and let I be an interpretation that satisfies the formula $\epsilon_\emptyset$. A tuple $R = \langle E, src, b\rangle$ and a binary relation $\prec$ over E are constructed as follows.*

1. *Define $E := \{t \in T \mid \mathsf{Occur}_t{}^I = \text{true}\}$.*
2. *For each input or test arc $a \in in(E) \cup test(E)$, define $src(a) :=$ any output arc $o \in out(place(a))$ such that $\mathsf{Link}_{o,a}{}^I = \text{true}$.*
3. *For each variable $v \in vars(E)$, define $v^b := \mathsf{Val}_v{}^I$.*
4. *For all elements $t, u \in E$, define $t \prec u$ iff $(\mathsf{Time}_t < \mathsf{Time}_u)^I$.*

We justify item 2 by noting that for any input or test arc $a \in in(E) \cup test(E)$, the formulas β_a and $\mathsf{Occur}_{trans(a)}$ are both true in I, and by (17), there is at least one output arc $o \in out(place(a))$ such that $\mathsf{Link}_{o,a}$ is true in I. Thus, a suitable arc $src(a)$ can always be selected. Moreover, by (18),

$\mathsf{Occur}_{trans(src(a))}$ is true in I, and thus $src(a)$ is in $out(E)$. Therefore, Construction 1 defines a function src from $in(E) \cup test(E)$ to $out(E)$.

Item 3 defines a total binding b such that the values ϕ^b and $(\phi^{vals})^I$ are equal for any guard or arc expression ϕ. The relation $\prec$ from item 4 is a strict partial order over E since $<^I$ is the less-than relation of real numbers. Relation $\prec$ is not necessarily a strict total order, because Time_t and Time_u may be equal in I even if $t \neq u$.

Let us present the construction in the opposite direction.

Construction 2. *Let $N = \langle \Sigma, P, T, \ldots, expr \rangle$ be a finite unwinding and let $R = \langle E, src, b \rangle$ be its token trace with a chronological partial order $\prec$. Construct an interpretation I of the symbols Occur_t, Time_t, $\mathsf{Link}_{o,a}$, Val_v as follows.*

First, if E is nonempty, pick an element of E that is minimal in $\prec$ and call it t_1. If $E \setminus \{t_1\}$ is nonempty, pick a minimal element t_2 of $E \setminus \{t_1\}$, and so on. Finally, there will be a sequence $t_1, \ldots, t_{|E|}$ consisting of the events in E such that $t_i \prec t_j$ implies $i < j$. Extend the sequence to cover T entirely by adding the elements of $T \setminus E$ in arbitrary order. Then define I as follows.

1. $\mathsf{Occur}_t{}^I :=$ true *if* $t \in E$ *and* $\mathsf{Occur}_t{}^I :=$ false *if* $t \in T \setminus E$.
2. $\mathsf{Time}_{t_i}{}^I := i$ *for each transition* t_i *in the sequence* $t_1, \ldots, t_{|T|}$.
3. $\mathsf{Link}_{o,a}{}^I :=$ true *if* $a \in in(E) \cup test(E)$ *and* $src(a) = o$, *otherwise* $\mathsf{Link}_{o,a}{}^I :=$ false.
4. $\mathsf{Val}_v{}^I := v^b$ *if* $v \in vars(E)$, *otherwise* $\mathsf{Val}_v{}^I$ *is fixed to an arbitrary value in* $type(v)$.

The following lemmas state that the formula $\epsilon_\emptyset$ encodes the set of all token traces of a finite unwinding. From the constructions, we see that the correspondence between token traces and satisfying interpretations of $\epsilon_\emptyset$ is meaningful. In particular, because $\mathsf{Occur}_t{}^I$ corresponds to $t \in E$, we can use the encoding for finding a token trace with a specified set of events.

Lemma 5. *If R and $\prec$ are built by Construction 1 from a finite unwinding N and a satisfying interpretation of its encoding $\epsilon_\emptyset$, then R is a token trace of N and $\prec$ is a chronological partial order of R.*

Lemma 6. *If I is an interpretation built by Construction 2 from a finite unwinding N, a token trace R of N, and a chronological partial order $\prec$ of R, then I satisfies the encoding $\epsilon_\emptyset$ of N.*

Proving Lemma 5 or 6 involves matching the items of Definition 3 to the conjuncts of $\epsilon_\emptyset$. The process in both cases is solely mechanical, so the details are omitted.

4.2 Specialized Encoding for Safe Places

A place p of an unwinding is *one-off safe* iff there is no one-off execution $M_0\ [t_1, b_1\rangle\ M_1\ \cdots\ [t_k, b_k\rangle\ M_k$, an index $0 \leq i \leq k$, and a pair of values $d_1, d_2 \in colors(p)$ such that $\{\langle p, d_1\rangle\} + \{\langle p, d_2\rangle\} \leq M_i$. In other words, a one-off safe place is one that cannot contain more than one token in any one-off execution.

If we know beforehand that a place $\hat{p}$ of an unwinding is one-off safe "by construction", we can modify the encoding formula for the part related to $\hat{p}$. The resulting formula encodes the same set of token traces but uses a more specialized set of constraints in an attempt to better guide the solver. Let $\hat{P} \subseteq P$ be a set of places that are known to be one-off safe, and define the encoding by

$$\epsilon_{\hat{P}} := \bigwedge_{t \in T} \gamma_t \wedge \bigwedge_{a \in A_{in} \cup A_{test}} \Big(\beta_a \wedge \bigwedge_{o \in out(place(a))} \psi_{o,a} \Big) \wedge \bigwedge_{p \in P \setminus \hat{P}} \delta_p \wedge \bigwedge_{\hat{p} \in \hat{P}} \hat{\delta}_{\hat{p}} \ . \tag{21}$$

We can see that (21) is a generalization of (15), with the constraint $\delta_{\hat{p}}$ replaced by $\hat{\delta}_{\hat{p}}$ for all assuredly one-off safe places $\hat{p}$. The new subformula $\hat{\delta}_{\hat{p}}$ is defined by

$$\begin{aligned} \hat{\delta}_{\hat{p}} := \bigwedge_{o \in out(\hat{p})} \bigwedge_{c \in G_{\hat{p}}} \neg \mathsf{Link}_{o,c} \ \wedge \\ \bigwedge_{o \in out(\hat{p})} \bigwedge_{\substack{c \in in(\hat{p}) \\ trans(c) \neq trans(o)}} \bigwedge_{\substack{a \in in(\hat{p}) \cup test(\hat{p}) \\ trans(a) \neq trans(c)}} \hat{\rho}_{o,c,a} \ , \end{aligned} \tag{22}$$

where the set

$$G_{\hat{p}} := \big\{ c \in in(\hat{p}) \ \big| \ trans(c) \in trans(test(\hat{p}) \cup in(\hat{p}) \setminus \{c\}) \big\} \tag{23}$$

is the set of input arcs c that are incident to place $\hat{p}$ such that there is another input or test arc also incident to both $\hat{p}$ and $trans(c)$. This means that the transition $trans(c)$ tries to read two distinct tokens in place $\hat{p}$ at the same time. As $\hat{p}$ is one-off safe, such a transition cannot occur, and therefore $\mathsf{Link}_{o,c}$ must be false for any output arc o.

The subformula $\hat{\rho}_{o,c,a}$ is defined for triplets consisting of an output arc o, an input arc c, and an input or test arc a, such that all three are incident to the same place, and c is not incident to the same transition as o or a (see Fig. 2). The meaning of the formula is roughly that if o produces a token and later a reads it, then the input arc c cannot consume any token between the occurrences of $trans(o)$ and $trans(a)$.

$$\begin{aligned} \hat{\rho}_{o,c,a} := \mathsf{Link}_{o,a} \rightarrow \\ \big(\mathsf{Time}_{trans(c)} < \mathsf{Time}_{trans(o)} \big) \vee \big(\mathsf{Time}_{trans(a)} < \mathsf{Time}_{trans(c)} \big) \ . \end{aligned} \tag{24}$$

All places in Figs. 1c and 1g are one-off safe, and $G_{\hat{p}}$ is an empty set for all $\hat{p}$.

The correspondence between satisfying interpretations and token traces applies to $\epsilon_{\hat{P}}$ as well as $\epsilon_{\emptyset}$. The same Constructions 1 and 2 still apply like before, but the correctness argument is more involved and requires three more lemmas. We will first see an example and a related lemma that states a set of conditions under which a place cannot be one-off safe.

Example 1. Figure 2 presents an unwinding and its token trace R. A chronological order of R must satisfy $t_o \prec t_c \prec t_a$ and $t_z \prec t_c$. The two one-off executions corresponding to R are

$$M_0 \, [t_z, b_z\rangle \, M_1 \, [t_o, b_o\rangle \, M_2 \, [t_c, b_c\rangle \, M_3 \, [t_a, b_a\rangle \, M_4 \quad \text{and}$$
$$M_0 \, [t_o, b_o\rangle \, M_1' \, [t_z, b_z\rangle \, M_2 \, [t_c, b_c\rangle \, M_3 \, [t_a, b_a\rangle \, M_4.$$

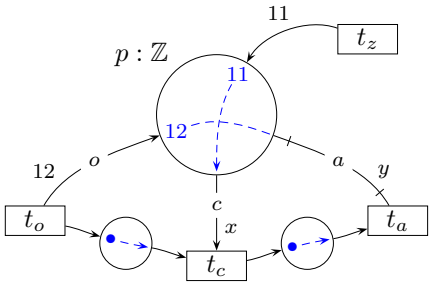

Figure 2: A token trace showing that place p is not one-off safe.

In both cases, marking M_2 has the multiset $\{\langle p, 11\rangle, \langle p, 12\rangle\}$ as a subset, so either of these executions reveals that place p is not one-off safe.

We could also infer this fact directly from the structure of R with respect to the arcs o, c, and a. Namely, since $\mathit{trans}(\mathit{src}(a)) \prec \mathit{trans}(c) \prec \mathit{trans}(a)$, there must be two distinct tokens present in place p right before $\mathit{trans}(c)$ occurs: one token that c is about to consume, and another that $\mathit{src}(a)$ has produced but a has not yet tested. Thus, any execution corresponding to R has an intermediate marking (the one preceding the occurrence of $\mathit{trans}(c)$) that shows p not to be one-off safe.

The following lemma generalizes the example and gives a sufficient (but not necessary) condition for a place not to be one-off safe.

Lemma 7. *Let p be a place of an unwinding N and let $R = \langle E, \mathit{src}, b\rangle$ be a token trace of N with a chronological partial order $\prec$. If there is an input arc $c \in \mathit{in}(E)$ and an input or test arc $a \in \mathit{in}(E) \cup \mathit{test}(E)$ such that $\mathit{place}(c) = \mathit{place}(a) = p$ and $\mathit{trans}(\mathit{src}(a)) \prec \mathit{trans}(c) \prec \mathit{trans}(a)$, then p is not one-off safe.*

Proof. Assume $\mathit{place}(c) = \mathit{place}(a) = p$ and $\mathit{trans}(\mathit{src}(a)) \prec \mathit{trans}(c) \prec \mathit{trans}(a)$. Let $E' := \{t \in E \mid t \prec \mathit{trans}(c)\}$. Then $\mathit{trans}(\mathit{src}(c))$ is in E' because $\prec$ is a chronological partial order. Also $\mathit{trans}(\mathit{src}(a))$ is in E' by assumption, but $\mathit{trans}(c)$ and $\mathit{trans}(a)$ are not.

Because $\prec$ is transitive, E' is downward closed in $\prec$, and Lemma 2 applies. Let $R' = \langle E', \mathit{src}', b'\rangle$ be the token trace of Lemma 2.

Let us verify that there is no input arc $c' \in \mathit{in}(E)$ such that $\mathit{trans}(c') \prec \mathit{trans}(a)$ and $\mathit{src}(c') = \mathit{src}(a)$. In other words, no input arc that precedes a consumes the token that a tests. Namely, if a is an input arc, src maps the two distinct input arcs c' and a to distinct output arcs, i.e. $\mathit{src}(c') \neq \mathit{src}(a)$. On the other hand, if a is a test arc, the equality $\mathit{src}(c') = \mathit{src}(a)$ would imply $\mathit{trans}(a) \prec \mathit{trans}(c')$ by item 5b of Definition 3, in violation of the assumption $\mathit{trans}(c') \prec \mathit{trans}(a)$. Therefore, $\mathit{src}(a)$ is distinct from $\mathit{src}(c')$ for every input arc c' such that $\mathit{trans}(c') \prec \mathit{trans}(a)$. In particular, $\mathit{src}(a)$ is distinct from $\mathit{src}(c)$, and $\mathit{src}(a)$ is also distinct from $\mathit{src}(c')$ for every input arc $c' \in \mathit{in}(E')$.

Furthermore, $\mathit{src}(c)$ is not in $\mathit{src}(\mathit{in}(E'))$ because $\mathit{trans}(c) \notin E'$ and no two input arcs share a source.

Since $\mathit{trans}(\mathit{src}(a))$ and $\mathit{trans}(\mathit{src}(c))$ are both in E' as stated above, and neither $\mathit{src}(a)$ nor $\mathit{src}(c)$ is in $\mathit{src}(\mathit{in}(E'))$, it follows that both $\mathit{src}(a)$ and $\mathit{src}(c)$ are residual producers of R'. These output arcs are distinct, and they

produce the token element $\langle place(src(a)), d_a\rangle = \langle p, d_a\rangle$ and the token element $\langle place(src(c)), d_c\rangle = \langle p, d_c\rangle$, respectively, where $d_a = expr(src(a))^b$ and $d_c = expr(src(c))^b$. Therefore, the residual marking M of R' has the multiset $\{\langle p, d_a\rangle\} + \{\langle p, d_c\rangle\}$ as a subset. By applying Lemma 4 to R', there is a one-off execution of N whose final marking is M. Because the final marking satisfies $\{\langle p, d_a\rangle\} + \{\langle p, d_c\rangle\} \leq M$, place p is not one-off safe. □

The next lemma says that the subformula $\hat{\delta}_{\hat{p}}$ introduced for a one-off safe place $\hat{p}$ is not too restricting. The proof is partly based on the previous lemma.

Lemma 8. *Let $N = \langle \Sigma, P, T, \ldots, expr\rangle$ be a finite unwinding, let R be a token trace of N, let $\prec$ be a chronological partial order of R, and let $\hat{p} \in P$ be a one-off safe place. Then, any interpretation I built by Construction 2 from R and $\prec$ satisfies $\hat{\delta}_{\hat{p}}$.*

Proof. Assume that the claim does not hold: there is a one-off safe place $\hat{p}$, a token trace $R = \langle E, src, b\rangle$ of N with a chronological partial order $\prec$, and an interpretation I built from R and $\prec$ by Construction 2 such that $\hat{\delta}_{\hat{p}}$ is false in I. Then, according to (22), at least one of the following is true:

C1. there is an output arc $o \in out(\hat{p})$ and an input arc $c \in G_{\hat{p}}$ such that $\mathsf{Link}_{o,c}$ is true in I, or

C2. there is an output arc $o \in out(\hat{p})$, an input arc $c \in in(\hat{p})$, and an input or test arc $a \in in(\hat{p}) \cup test(\hat{p})$ such that $trans(c) \neq trans(o)$ and $trans(a) \neq trans(c)$ and $\hat{\rho}_{o,c,a}$ is false in I.

We will show that C1 leads to contradiction, and so does C2. We will abbreviate the symbol $\mathsf{Time}_{trans(a)}$, where a denotes an arc, to the shorthand notation $\underline{a}$.

First, assume that C1 holds. Then, by (23), there is an input or test arc $a \in test(\hat{p}) \cup in(\hat{p}) \setminus \{c\}$ such that $trans(a) = trans(c)$. Because I is the result of Construction 2 and $\mathsf{Link}_{o,c}{}^I$ is true, $trans(c)$ is in E. As $\langle E, src, b\rangle$ is a token trace, there is a one-off execution X in which $trans(c)$ occurs (Theorem 1). We can thus write X as a sequence

$$X = M_0\ [t_1, b_1\rangle\ \cdots\ M_{i-1}\ [t_i, b_i\rangle\ M_i\ \cdots\ [t_k, b_k\rangle\ M_k\ \ ,$$

where i is the index such that $t_i = trans(c)$. As the binding element $\langle t_i, b_i\rangle$ is enabled in the marking M_{i-1}, and t_i has two incident arcs $c \in A_{in}$ and $a \in A_{in} \cup A_{test}$ such that $place(c) = place(a) = \hat{p}$, by Definition 2 of enabledness, place $\hat{p}$ must contain two distinct tokens in the marking M_{i-1}. Thus, $\hat{p}$ is not a one-off safe place, in contradiction with the assumptions. This rules out the possibility of C1.

Then, assume that C2 holds. As $\hat{\rho}_{o,c,a}$ is false in I, by (24) we have $\mathsf{Link}_{o,a}{}^I = \text{true}$, $(\underline{c} < \underline{o})^I = \text{false}$, and $(\underline{a} < \underline{c})^I = \text{false}$. By Construction 2, interpretation I assigns distinct time values to distinct transitions. Therefore, as $trans(c) \neq trans(o)$ and $(\underline{c} < \underline{o})^I = \text{false}$, we must have $(\underline{o} < \underline{c})^I = \text{true}$. Similarly, $(\underline{c} < \underline{a})$ is true in I. By item 3 of Construction 2, it follows from $\mathsf{Link}_{o,a}{}^I$ that $a \in in(E) \cup test(E)$ and $src(a) = o$. Also by the construction, since $trans(a) \in E$, the value $\underline{a}^I$ is less than $\mathsf{Time}_t{}^I$ for any $t \in T \setminus E$. Therefore, as $\underline{a}^I$ is not less than $\underline{c}^I$, we must have $trans(c) \in E$.

Let us define a strict partial order $\prec'$ over E by setting for each $t, u \in E$, $t \prec' u$ if and only if $(\mathsf{Time}_t < \mathsf{Time}_u)^I$. Then, $t \prec u$ implies $t \prec' u$, so it follows immediately from Definition 3 that $\prec'$ is another chronological partial order of R.

We have shown that c is in $in(E)$, a is in $in(E) \cup test(E)$, and the relation $trans(src(a)) \prec' trans(c) \prec' trans(a)$ holds. We can then apply Lemma 7 to R, $\prec'$, c, a, and $\hat{p}$, leading to the conclusion that place $\hat{p}$ is not one-off safe. This contradicts the assumption on $\hat{p}$. Case C2 is thus refuted, so the claim must hold. □

The final lemma of this section shows that the encoding $\epsilon_{\hat{P}}$ is at least as strict as $\epsilon_\emptyset$. Thus, every interpretation that satisfies $\epsilon_{\hat{P}}$, also satisfies $\epsilon_\emptyset$, and it translatable to a token trace. We will collect this and the previous intermediate results in the correctness proof in Sect. 4.3.

Lemma 9. *Let $N = \langle \Sigma, P, T, \ldots, expr \rangle$ be a finite unwinding and let $\hat{P}$ be any subset of the places of N. Then, $\epsilon_\emptyset$ is a logical consequence of $\epsilon_{\hat{P}}$.*

Proof. Assume that the claim does not hold: there is an interpretation I such that ${\epsilon_{\hat{P}}}^I$ is true and ${\epsilon_\emptyset}^I$ is false. From ${\epsilon_{\hat{P}}}^I = \text{true}$, it follows by (21) that γ_t, β_a, and $\psi_{o,a}$ are all true in I for all choices of t, a, and o, and δ_p is true in I when p is in $P \setminus \hat{P}$. Therefore, the only way $\epsilon_\emptyset$ can be false is by ${\delta_{\hat{p}}}^I$ being false for some place $\hat{p} \in \hat{P}$. Let us pick one place $\hat{p} \in \hat{P}$ such that ${\delta_{\hat{p}}}^I = \text{false}$.

According to (19), at least one of the following holds:

C1. $AtMostOne\left(\left\{\mathsf{Link}_{o,c} \;\middle|\; c \in in(\hat{p})\right\}\right)$ is false in I for some output arc $o \in out(\hat{p})$, or

C2. $\rho_{o,e,c}$ is false in I for some arcs $o \in out(\hat{p})$, $e \in test(\hat{p})$, $c \in in(\hat{p})$.

We will show that C1 leads to contradiction, and so does C2. For brevity, we will abbreviate the symbol $\mathsf{Time}_{trans(a)}$, where a denotes an arc, to the shorthand notation $\underline{a}$.

First, assume that C1 holds. That is, there is an output arc $o \in out(\hat{p})$ and a pair of distinct input arcs $c, c' \in in(\hat{p})$ such that $\mathsf{Link}_{o,c}$ and $\mathsf{Link}_{o,c'}$ are both true in I. As ${\psi_{o,c}}^I$ and ${\psi_{o,c'}}^I$ are true (because ${\epsilon_{\hat{P}}}^I$ is true), equation (18) gives us

$$(\underline{o} < \underline{c})^I \text{ and } (\underline{o} < \underline{c'})^I. \tag{25}$$

In particular, this means that $trans(o) \neq trans(c)$ and $trans(o) \neq trans(c')$. If c and c' were incident to the same transition, then c would be in the set $G_{\hat{p}}$, and because $\hat{\delta}^I_{\hat{p}}$ is true, we would deduce from (22) that $\mathsf{Link}_{o,c}$ is false in I, a contradiction. Therefore, $trans(c) \neq trans(c')$. As $trans(o)$, $trans(c)$, and $trans(c')$ are three distinct transitions, the formulas $\hat{\rho}_{o,c,c'}$ and $\hat{\rho}_{o,c',c}$ are both defined and true in I by (22). Since $\mathsf{Link}_{o,c}$ and $\mathsf{Link}_{o,c'}$ are also true in I, the definition (24) yields

$$\left(\underline{c} < \underline{o} \;\vee\; \underline{c'} < \underline{c}\right)^I \text{ and } \left(\underline{c'} < \underline{o} \;\vee\; \underline{c} < \underline{c'}\right)^I. \tag{26}$$

Contradiction then follows from (25) and (26), and case C1 is thus disproven.

Let us then assume that C2 holds. According to the definition (20), there is an output arc $o \in out(\hat{p})$, a test arc $e \in test(\hat{p})$, and an input arc

$c \in in(\hat{p})$ such that $\mathsf{Link}_{o,e}$ and $\mathsf{Link}_{o,c}$ are true in I and $(\underline{e} < \underline{c})$ is false in I. If e and c were incident to the same transition, then c would be in the set $G_{\hat{p}}$, and (22) would imply ${\mathsf{Link}_{o,c}}^I = \text{false}$, a contradiction. Thus, $trans(e) \neq trans(c)$. From ${\psi_{o,c}}^I$ and ${\mathsf{Link}_{o,c}}^I$, we get $(\underline{o} < \underline{c})^I$, which also implies $trans(c) \neq trans(o)$. This means that $\hat{\delta}_{\hat{p}}$ has the subformula $\hat{\rho}_{o,c,e}$, which is then true in I. Together with ${\mathsf{Link}_{o,e}}^I$, this implies by (24) that $\big(\underline{c} < \underline{o} \vee \underline{e} < \underline{c}\big)$ is true in I, contradicting the previous assertions that $(\underline{e} < \underline{c})$ is false and $(\underline{o} < \underline{c})$ is true in I. Case C2 is thus also refuted, which concludes the proof. □

4.3 Properties of the Encodings

The principal motivation for formula (21) and its special case (15) is that they can be used for model checking reachability properties.

Theorem 10. *Let $N = \langle \Sigma, P, T, \ldots, expr \rangle$ be a finite unwinding, let $t^\diamond \in T$ be a transition, and let $\hat{P} \subseteq P$ be any subset of the one-off safe places of N. Then, $t^\diamond$ is one-off reachable if and only if the formula $\epsilon_{\hat{P}} \wedge \mathsf{Occur}_{t^\diamond}$ is satisfiable.*

Proof. First, we will show that if the formula is satisfiable, then a one-off execution with event $t^\diamond$ exists. Assume that $\epsilon_{\hat{P}} \wedge \mathsf{Occur}_{t^\diamond}$ has a satisfying interpretation I. By Lemma 9, I also satisfies $\epsilon_\emptyset \wedge \mathsf{Occur}_{t^\diamond}$. Therefore, we can apply Construction 1 to I to build a tuple $R = \langle E, src, b \rangle$. By Lemma 5, R is a token trace, and by item 1 of Construction 1, E contains $t^\diamond$. Theorem 1 then gives us a one-off execution whose event set contains $t^\diamond$.

For the other direction, assume that there is a one-off execution in which $t^\diamond$ occurs. According to Theorem 1, there is a token trace R of N whose event set E contains $t^\diamond$. Build an interpretation I from R using Construction 2. By Lemma 6, I satisfies $\epsilon_\emptyset$, and by Lemma 8, I satisfies $\hat{\delta}_{\hat{p}}$ for all $\hat{p} \in \hat{P}$. By the structure of (21), I satisfies $\epsilon_{\hat{P}}$. By construction, $\mathsf{Occur}_{t^\diamond}$ is true in I. Thus, the formula $\epsilon_{\hat{P}} \wedge \mathsf{Occur}_{t^\diamond}$ has a satisfying interpretation I. □

The proof is constructive and can be used to extract witness executions. We point out that knowing which places are one-off safe is not a prerequisite for applying the result. For example, setting $\hat{P} = \emptyset$ is always a valid choice.

Concerning compactness, the formula $\epsilon_{\hat{P}}$ contains *one* instance of each guard and arc expression of the unwinding, so there is no duplication of transitions involved in the encoding phase. The rest of the encoding adds a term $O(|out(p)|\,(1 + |in(p)|)(1 + |test(p)|))$ to the size for each place p, plus $O(|out(\hat{p})|\,|in(\hat{p})|^2)$ for each $\hat{p} \in \hat{P}$. The encoding is thus *locally* cubic in the number of arcs incident to a place, or quadratic if the place is not in $\hat{P}$ and there are no test arcs.

Apart from the inner parts of guards and arc expressions, our encodings are examples of *difference logic* formulas. Section 5 discusses the satisfiability of such formulas.

5 ENCODING RESTRICTED DIFFERENCE LOGIC IN SAT

In the encoding of Sect. 4, we are interested in deciding the satisfiability of formulas that contain ordinary propositional symbols, additional symbols u, w of real type from a set V of symbols of type $\mathbb{R}$, and *strict inequalities* of the form $u < w$. We will show a way to solve the decision problem by reducing it to propositional satisfiability (SAT) using a compact encoding.

The encoding formulas also contain guard expressions and equalities between arc expressions, but we assume that they are either separately converted to propositional logic or handled with another decision procedure.

We are trying to decide satisfiability in a restricted subset of *difference logic*. In full difference logic, predicates have the form $u - w \leq c$, where u and w are uninterpreted real or integer symbols, c is an interpreted constant (a real or integer number), and the symbols $-$ and $\leq$ have their usual mathematical meaning. In our case, c is always zero and the inequality predicates are expressed in a complemented form ($u < w$ is equivalent to $\neg(w - u \leq 0)$).

The satisfiability problem of difference logic formulas is well studied. Below, three common approaches based on SAT solver techniques are briefly presented.

1. **Eager small-domain encoding.** In an *eager* approach, the difference logic formula is translated to a purely propositional formula such that the two formulas are equisatisfiable (one is satisfiable if and only if the other is satisfiable). The resulting formula is then sent to a SAT solver. In the small-domain encoding, one defines a bounded integer symbol ord_v for each symbol $v \in V$. Each inequality of the form $u - w \leq c$ is then replaced by a bounded integer comparator $ord_u \leq ord_w + c$. The bounded integers and comparators are then binary encoded using only propositional symbols. If the constant term c is always zero, a sufficient number of bits for representing the bounded integers is $\lceil \log_2 |V| \rceil$, so that every symbol in V can potentially be assigned a unique integer value. The small-domain encoding is studied in [4, 25] for full difference logic over integers.

2. **Eager per-constraint encoding.** In another eager approach, the idea is to treat each inequality as an individual proposition. As the dependencies between inequalities are lost, the SAT solver may end up with an illegal interpretation, for example assigning the value true to both $u - w \leq -1$ and $w - u \leq -1$. Additional *transitivity constraints* are placed to rule out interpretations that contain illegal cycles like this. A per-constraint encoding for difference logic is presented in [24], and [23] combines this with the small-domain approach.

3. **Lazy approach.** In Jussila's thesis [17], a set of similar transitivity constraints is used to prevent cyclic dependencies of executed actions in the context of SAT-based BMC of labeled transition systems. It turns out that in some analyzed test cases, the end result is the same even if the transitivity constraints are dropped from the formula. This can speed up the SAT solving process, since sometimes the transitivity constrains dominate the total formula size. If the solver returns an interpre-

tation that entails an illegal cycle, new transitivity constraints are added on demand to narrow the search iteratively. This method is taken further in many SMT solvers that implement a decision procedure for difference logic, with various techniques for coupling the Boolean search with the discovery of illegal cycles of inequalities [8, 21, 1, 3, 26].

In the following, a per-constraint encoding is presented for the formulas of Sect. 4. We apply an eager encoding to SAT because that allows us to treat the SAT solver as a black box and to make a balanced comparison between our approach to the bounded reachability problem and SAT-based bounded model checking in Sect. 7. Experimentation with alternative encodings and the lazy approach are left for future work.

Our encoding is similar to the one presented by Strichman et al. in Sect. 4 of [24] based on chordal graphs. We will call that approach the CAV'02 encoding. However, our formulas are restricted to the case where the constant term c is zero in each inequality $u < w + c$. With this restriction, the number of additional transitivity constraints using CAV'02 is $O(|V|^3)$ in the worst case, and not superpolynomial in $|V|$ as in the general case [24]. Furthermore, in our difference logic formulas there is another syntactic restriction that we will exploit. The predicates $u < w$ occur only *positively* in the formulas. A positive occurrence of a subformula is one that is nested only in an *even* number of negations. This means that the formulas can be expressed in conjunctive normal form (CNF) according to the following definition.

Definition 4. *A* CNF-formula with positive strict inequalities *over symbol sets P and V, or a* $<$-formula *for short, is a conjunction of clauses, where each clause is a disjunction of literals, and each literal is either a propositional symbol $p \in P$, a negated propositional symbol $\neg p$, or a positive strict inequality $u < w$, where u and w are distinct symbols in V.*

The distinctness requirement of u and w is immaterial, since an inequality of the form $v < v$ would be trivially equivalent to false.

An interpretation I in this context maps each propositional symbol $p \in P$ to $p^I \in \{\text{false}, \text{true}\}$, and each real symbol $v \in V$ to a real number v^I. The symbol $<$ has a fixed interpretation as the less-than relation over reals. An interpretation I is a model of a formula f iff f^I is true, and f is satisfiable iff it has a model.

The expressive power of $<$-formulas is limited. In particular, a $<$-formula cannot express the *equality* of two reals, since this would require using negations in front of inequalities ($u = w$ is expressible as $\neg(u < w) \wedge \neg(w < u)$). Therefore we only need to distinguish between two cases ($u < w$ or $w < u$) instead of the usual three ($u < w$ or $u = w$ or $w < u$). The CAV'02 algorithm of [24] uses two propositional symbols p, q to encode the inequalities $u < w$ and $w < u$. The negations $\neg p$ and $\neg q$ then represent non-strict inequalities $w \leq u$ and $u \leq w$, respectively. We will use only one symbol r_{uw} to encode the inequality $u < w$, and its negation $\neg r_{uw}$ to encode $w < u$. The number of propositional symbols is thus reduced by exploiting the one-sided expressibility of $<$-formulas. We will see that also the number of transitivity constraints can be significantly reduced on similar grounds.

5.1 The Encoding Algorithm

The encoding of a $<$-formula as an equisatisfiable propositional formula is based on adding a sufficient number of *triangular* transitivity constraints of the form $(u < v) \wedge (v < w) \rightarrow (u < w)$. This is done by executing a series of *elimination* steps. In each step, a symbol $s \in V$ is chosen for elimination. For each pair $a, b \in V$ such that the inequalities $a < s$ and $s < b$ occur in the formula, the transitivity constraint $(a < s) \wedge (s < b) \rightarrow (a < b)$ is added. Then, all inequalities $a < s$ and $s < b$ are replaced by propositional symbols as discussed above. The result is a $<$-formula with no occurrences of the symbol s. This elimination process is repeated for all symbols in V, and in the end we will have a propositional CNF-formula that is equisatisfiable with the original formula and can be sent to a SAT solver.

A more rigorous formulation of the algorithm is given below.

Procedure ELIM(f : a $<$-formula)

1. $\langle P, V \rangle :=$ the set of propositional and real symbols in f
2. **if** $V = \emptyset$ **then return** f
3. $s :=$ a symbol in V
4. $A := \{a \in V \mid a < s \text{ occurs in } f\}$,
   $B := \{b \in V \mid s < b \text{ occurs in } f\}$
5. $R :=$ a set of fresh propositional symbols $\{r_{us} \mid u \in A \cup B\}$
6. $g :=$ a copy of f with the substitutions $(a < s) \mapsto r_{as}$ and
   $(s < b) \mapsto \neg r_{bs}$ for all $a \in A$, $b \in B$
7. $h := \bigwedge_{a \in A} \bigwedge_{b \in B \setminus \{a\}} \big(\neg r_{as} \vee r_{bs} \vee (a < b)\big)$
8. $\tilde{f} := g \wedge h$
9. **return** ELIM($\tilde{f}$)

The formula h on line 7 contains all triangular transitivity constraints involving s. The number of constraints is in the worst case quadratic in the number of inequalities with s on either side. Furthermore, the transitivity constraints may introduce new inequalities $a < b$ that did not occur in f before, making later elimination steps more expensive. The size of the final formula is thus sensitive to the order of elimination. In practice, a heuristic is used on line 3 to choose the next symbol to be eliminated. In the experiments of Sect. 7, a greedy heuristic is used that locally minimizes the number of constraints in h.

It is straightforward to check that the formula $\tilde{f}$ on line 8 is a $<$-formula without occurrences of the symbol s. Thus, the procedure terminates and returns a propositional formula. The most interesting property is that f and ELIM(f) are equisatisfiable, as shown by Theorem 11 below. The proof is constructive in nature and therefore, if a SAT solver returns a model of ELIM(f), we can use the proof to build a concrete interpretation that satisfies f.

Theorem 11. *If f is a $<$-formula, then f and* ELIM(f) *are equisatisfiable.*

Proof. We will show that each single elimination step preserves satisfiability: on line 8 in procedure ELIM, the formula $\tilde{f}$ is equisatisfiable with the input formula f. The claim then follows directly by an inductive argument. Let us show that the satisfiability of f implies the satisfiability of $\tilde{f}$ and vice versa.

First, let us assume that f has a model I and construct a model of $\tilde{f}$. As before, e^I denotes the valuation of expression e under interpretation I. Let $\tilde{I}$ be an interpretation that coincides with I and extends it to the symbols R by defining ${r_{us}}^{\tilde{I}} := (u < s)^I$ for all $u \in A \cup B$. Let us show that $\tilde{I}$ is a model of $\tilde{f}$, that is, $\tilde{I}$ satisfies every clause of g and h. First, let $\tilde{C}$ be an arbitrary clause of h, i.e., $\tilde{C} = \big(\neg r_{as} \vee r_{bs} \vee (a < b)\big)$. By the definition of $\tilde{I}$, the clause evaluates to $\tilde{C}^{\tilde{I}} = \big(\neg(a < s) \vee (b < s) \vee (a < b)\big)^I$, which is true since $<^I$ is the less-than relation over reals. Then, let $\tilde{D}$ be an arbitrary clause in g and let D be the corresponding clause in f. Because I satisfies f, there is a literal l in D such that l^I is true. There are three cases. (i) If that literal is a propositional symbol $p \in P$ or a negated propositional symbol $\neg p$ or an inequality $u < w$ such that u and w are distinct from s, then $\tilde{D}$ contains the same literal l, and thus $\tilde{D}^{\tilde{I}}$ is true because $l^{\tilde{I}}$ is true. (ii) If the satisfied literal l is $a < s$ for some $a \in A$, then $\tilde{D}$ contains the literal r_{as}, and $\tilde{D}$ is satisfied by $\tilde{I}$ because ${r_{as}}^{\tilde{I}}$ is true. (iii) If the satisfied literal l is $s < b$ for some $b \in B$, then $(s < b)^I$ is true, hence $(b < s)^I = {r_{bs}}^{\tilde{I}}$ is false, and $\tilde{D}$ contains the literal $\neg r_{bs}$ satisfied by $\tilde{I}$. Thus, $\tilde{I}$ satisfies every clause of g and h, and $\tilde{f}$ is satisfiable.

For the other direction, assume that $\tilde{I}$ is a model of $\tilde{f}$. Define the auxiliary sets $A' := \{a \in A \mid {r_{as}}^{\tilde{I}} \text{ is true}\}$ and $B' := \{b \in B \mid {r_{bs}}^{\tilde{I}} \text{ is false}\}$. Intuitively, A' is the set of symbols that should be interpreted as numbers less than s, and B' is the set of symbols that should evaluate to numbers greater than s. We note that for any pair $a \in A'$, $b \in B'$, the interpretation $\tilde{I}$ satisfies the clause $\neg r_{as} \vee r_{bs} \vee (a < b)$ in h, and because ${r_{as}}^{\tilde{I}}$ is true and ${r_{bs}}^{\tilde{I}}$ is false, $(a < b)^{\tilde{I}}$ must be true. In other words, if A' and B' are both nonempty, then $a_{\max} := \max_{a \in A'}(a^{\tilde{I}})$ is less than $b_{\min} := \min_{b \in B'}(b^{\tilde{I}})$. Let us define a new interpretation I that coincides with $\tilde{I}$ and interprets s as

$$s^I := \begin{cases} (a_{\max} + b_{\min})/2 & \text{if } A' \neq \emptyset \text{ and } B' \neq \emptyset, \\ a_{\max} + 1 & \text{if } A' \neq \emptyset \text{ and } B' = \emptyset, \\ b_{\min} - 1 & \text{if } A' = \emptyset \text{ and } B' \neq \emptyset, \\ 0 & \text{if } A' = \emptyset \text{ and } B' = \emptyset. \end{cases}$$

This ensures that $(a < s)^I$ is true for all $a \in A'$ and $(s < b)^I$ is true for all $b \in B'$. It remains to show that I satisfies every clause in f. Let C be an arbitrary clause in f, and let $\tilde{C}$ be the corresponding clause in g. As $\tilde{I}$ satisfies g, there is a literal $\tilde{l}$ in $\tilde{C}$ such that $\tilde{l}^{\tilde{I}}$ is true. Again, there are three cases. (i) If that literal is a propositional symbol $p \in P$ or a negated propositional symbol $\neg p$ or an inequality $u < w$, then C contains the same literal and thus C^I is true. (ii) If the satisfied literal $\tilde{l}$ is r_{as} for some $a \in A$, then ${r_{as}}^{\tilde{I}}$ is true and thus $a \in A'$. The corresponding literal in C is $a < s$, which evaluates to true in I as noted above. (iii) If the satisfied literal $\tilde{l}$ is $\neg r_{bs}$ for some $b \in B$, then ${r_{bs}}^{\tilde{I}}$ is false and thus $b \in B'$. The corresponding literal in C is $s < b$, which evaluates to true in I. Thus, f is satisfiable as it has a model I. □

The following examples show that by exploiting the fact that only positive occurrences of inequalities are allowed, the size of the resulting propositional formula can be substantially reduced. We will compare to the CAV'02 encoding [24] that uses a similar elimination procedure as presented above, resulting in transitivity constraints over 2 or 3 symbols each.

Example 2. Let f be a $<$-formula with subformulas $u < v$, $v < w$, and $u < w$, and with no other inequalities. With the elimination order u, v, w, procedure ELIM adds no transitivity constraints. Thus, ELIM(f) is the same as f with the inequalities replaced by fresh propositional symbols. With the CAV'02 algorithm, we would get two transitivity constraints: one that is equivalent to $(u < v) \wedge (v < w) \rightarrow (u < w)$ and another equivalent to $(u < w) \rightarrow (u < v) \vee (v < w)$.

More generally, if $V = \{v_1, \ldots, v_n\}$, and f only contains inequalities $v_i < v_j$ such that $i < j$, then the greedy heuristic will find an elimination order such that no transitivity constraints are produced in ELIM(f).

The next example shows that in the worst case, the procedure adds a cubic number of constant-size clauses as transitivity constraints. However, the number of added clauses is less than one seventh of the clauses that would be added by the CAV'02 algorithm.

Example 3. Let V be a finite set of real symbols with $|V| = n \geq 3$, and let f be a $<$-formula with at least one (positive) occurrence of every possible inequality $u < w$, where $u, w \in V$ and u is not w. There are $n(n-1)$ different inequalities in total. In ELIM(f), these are encoded using $n(n-1)/2$ propositional symbols, and there are $n(n-1)(n-2)/3$ triangular transitivity constraints. In contrast, the CAV'02 encoding uses $n(n-1)$ propositional symbols and produces $7n(n-1)(n-2)/3+n(n-1)/2$ transitivity constraints.

More specifically, we can see the set of transitivity constraints in ELIM(f) as consisting of all possible instantiations of the constraint $\neg\big((u < v) \wedge (v < w) \wedge (w < u)\big)$. The transitivity constraints produced by CAV'02 include all these, and in addition all constraints of the form $\neg\big((u < v) \wedge (v < w) \wedge (w \leq u)\big)$, all constraints of the form $\neg\big((u < v) \wedge (v \leq w) \wedge (w \leq u)\big)$, and all constraints of the form $\neg\big((u < w) \wedge (w < u)\big)$. Procedure ELIM avoids these extra constraints because, essentially, a $<$-formula cannot distinguish between strict inequalities $u < w$ and non-strict inequalities $u \leq w$.

6 COMPARISON TO RELATED WORK

A straightforward way to apply Bounded Model Checking [2] to an asynchronous system is to unroll its interleaving transition relation k times to cover all executions of k steps [18]. Consider a system that performs one of n possible atomic actions in each execution step. The BMC view of executions corresponds to Fig. 1b. The long horizontal lines in Fig. 1b represent the realizations of frame conditions, which are parts of the formula that say when a variable must maintain its value. Because of unrolling, the BMC formula describes kn potential events, and only k of them are scheduled to occur. Furthermore, the notion of fixed time points adds spurious synchronization between the potential events. Consider changing the order of the

events of Fig. 1b into g_1, g_2, f, h. The reordering is insignificant with respect to our reachability property, but from the BMC point of view, results in a completely different execution.

In contrast, the encoding of token traces contains no frame conditions for conveying data over time steps. Instead, the inputs and outputs of transitions are directly linked to each other. The selection of links is nondeterministic, which incurs some encoding overhead. Spurious synchronization is not generated between transitions that are not connected to the same place. However, there are additional, potentially costly constraints for ordering the transitions. Using kn potential events, we can cover executions up to length kn instead of k, but this depends on the unwinding. Generally, unwindings allow finer tuning of the bound than adjusting the parameter k in BMC.

There have been several proposals for making BMC better suited to asynchronous systems. Using alternative execution semantics [18, 11], several independent actions can occur in a single step of BMC, allowing longer executions to be analyzed without considerably growing the encoding. In [27], partial order reductions are implemented on top of BMC by adding a constraint that each pair of independent actions can occur at consecutive time steps only in one predefined order. An opposite approach [15] is to start BMC with some particular interleaving and then allow more behavior by iteratively removing constraints. As Bounded Event Tracing is inherently a partial order method, there is no need for retrofitted reductions.

Ganai and Gupta present a concurrent BMC technique [14] based on a similar kind of intuition as this report. Processes are unrolled independently of each other, and all globally visible operation of all processes are potentially linked pairwise, with constraints that prevent cyclic dependencies. In the encodings of single processes, various BMC techniques are needed to avoid blowup. Bounded Event Tracing uses places to localize interprocess communication, but [14] does not allow this. Instead, there is a single token that carries all global data.

The CBMC approach [7] unwinds (up to a bound) the loops of a sequential C program, converts it to static single assignment form, and encodes the constraints on the resulting set of variables. A version for threaded programs [22] is based on bounding the number of context switches and synchronizing globally visible operations by conditioning them on the number of context switches that have occurred so far.

A completely different symbolic technique for concurrent systems is based on unfoldings [13], which are partial-order representations of state spaces as (infinite) low-level Petri nets of a fixed form. Model checking is performed by taking a suitable finite prefix of an unfolding and encoding its behavior and the desired property in SAT. As unfoldings are acyclic, the encoding is simple. Although an unfolding represents interleavings implicitly, every possible control path and every nondeterministic choice of data is explicitly present, and in practice, the generation of the unfolding prefix is the most expensive part. We could obtain unwindings directly from unfoldings, but this would mean to abandon the flexibility of symbolic data and arbitrary connections between places and transitions.

Merged processes [19] diminish the problem of diverging paths in unfoldings. A merged process is obtained from an unfolding prefix by conjoining its

places and transitions using a deterministic procedure. The result may contain cycles, but unlike the semantics of our unwindings, merged processes also restrict each *place* to be used at most once in an execution. The approach is thus fundamentally different from the one presented here, despite the apparent similarities in the encodings.

7 EXPERIMENTS

As a proof of concept, we test an implementation of Bounded Event Tracing against Bounded Model Checking using the system in Figure 3. In one atomic action, either process moves from one label, e.g. *lf1*, to the next. The property is whether the value of x equals D when both processes have terminated. By varying the integer parameter C between 4 and 9, and D between 0 and $C^3 + 2C^2 - C$ (the highest possible value of x), we get a family of 2498 instances. The last line is reached in a fixed number of steps in each instance, so we can perform complete reachability analysis using either approach.

A parameterized unwinding and interleaving transition relation for BMC were constructed by hand. The BMC bound was fixed to $4C$ and the problems were given to the NuSMV 2.4.3 model checker [6] that uses the MiniSat v1.14 SAT solver. The unwinding has $4C + 3$ places and $4C + 2$ transitions, and the values of i and j are inlined like in Fig. 1g. The encoding (21) was applied with $\hat{P} = P$, and the inequality constraints were translated to propositional logic using the per-constraint encoding of Sect. 5. The formulas were then checked for satisfiability using NuSMV by technically disguising them as BMC instances. This way, the same machinery for Boolean circuit simplification and bit-vector arithmetic encoding was employed for both BMC and Bounded Event Tracing.

Each marker in Fig. 4 represents the time spent by MiniSat when solving an instance once with BMC and once with Bounded Event Tracing, running in one core of an Intel Xeon 5130 processor. With BMC, 26 instances hit the timeout of 2 hours. We can see that Bounded Event Tracing works out in practice, and gives roughly a tenfold speedup over interleaving BMC on this problem family. Using the encoding $\epsilon_\emptyset$ instead of $\epsilon_{\hat{P}}$ would increase the run time closer to that of BMC.

Initially: $x \leftarrow 0$

Process F:		**Process** G:	
$i \leftarrow 0$		$j \leftarrow 0$	
while $i < C$:		while $j < C$:	
lf1:	$t \leftarrow x$	*lg1*:	$u \leftarrow x$
lf2:	$x \leftarrow t + C + (C+1) \times i$	*lg2*:	$x \leftarrow u + C + (C+1) \times j$
	$i \leftarrow i + 1$		$j \leftarrow j + 1$

Finally:
if $x = D$: print "match"

Figure 3: A test system.

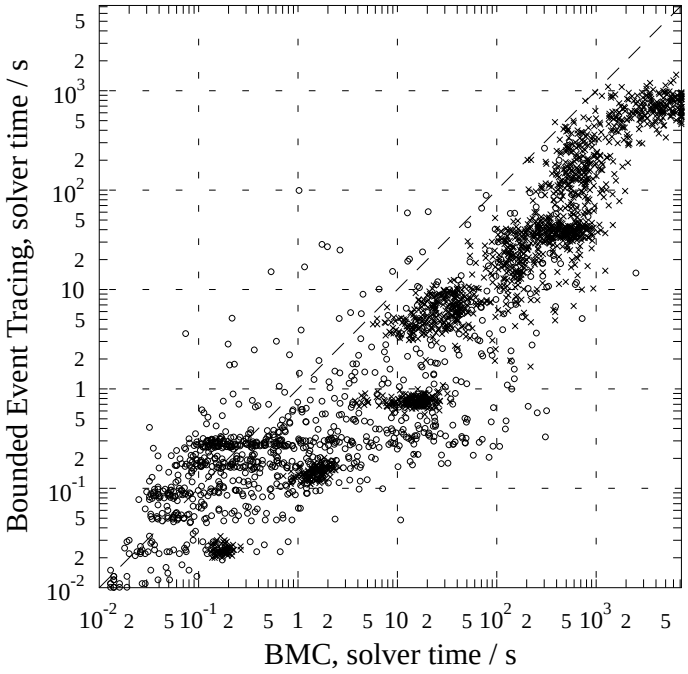

Figure 4: Results for satisfiable (○) and unsatisfiable (×) cases.

8 CONCLUSIONS AND FUTURE WORK

Bounded Event Tracing is a new symbolic technique for checking reachability properties of asynchronous systems. The analysis is bounded by a finite unwinding that fixes a collection of potential events that may occur but leaves the order of occurrences open. Unwindings are formalized as high-level Petri nets because the semantics of Petri nets rises naturally from the used concepts. The reachability problem is translated to a fragment of difference logic, which in turn can be reduced to propositional logic. The hard work is done by a SAT or SMT solver.

The technique incorporates ideas from Bounded Model Checking and unfoldings. Like in BMC, data handling is symbolic, but we avoid many pitfalls of BMC caused by viewing an execution of an asynchronous system as a sequence synchronized by fixed time steps. Like unfolding methods, Bounded Event Tracing has partial order reductions built in, but without the advance cost of explicit branching at every choice point.

The method is unfinished in that we do not define a procedure for constructing or expanding unwindings. This is the most crucial topic for further research. The expansion presumably enables incremental SAT solving [12], and it might be guided using e.g. unsatisfiable cores [15]. Successful model checking of software systems is likely to require some form of abstraction techniques [20].

It is remarkable that although the analysis covers executions in which places contain multisets of values, the encoding only talks about single values. This opens up the possibility of encoding collections such as arrays or message queues efficiently using a place that contains index-value pairs.

Acknowledgements

The author gives many thanks to Dr. Tommi Junttila for discussions and inspiration.

Financial support from Hecse (Helsinki Graduate School in Computer Science and Engineering) is gratefully acknowledged.

REFERENCES

[1] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, and Marco Maratea. The SAT-based approach to separation logic. *J. Autom. Reasoning*, 35(1-3):237–263, 2005.

[2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS 1999*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[3] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. An incremental and layered procedure for the satisfiability of linear arithmetic logic. In *TACAS 2005*, volume 3440 of *Lecture Notes in Computer Science*, pages 317–333. Springer, 2005.

[4] Randal E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2002.

[5] Søren Christensen and Niels Damgaard Hansen. Coloured Petri Nets extended with place capacities, test arcs and inhibitor arcs. In *ATPN 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 186–205. Springer, 1993.

[6] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.

[7] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In *TACAS 2004*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

[8] Scott Cotton and Oded Maler. Fast and flexible difference constraint propagation for DPLL(T). In *SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2006.

[9] Leonardo Mendonça de Moura, Bruno Dutertre, and Natarajan Shankar. A tutorial on satisfiability modulo theories. In *CAV 2007*,

volume 4590 of *Lecture Notes in Computer Science*, pages 20–36. Springer, 2007.

[10] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.

[11] Jori Dubrovin, Tommi Junttila, and Keijo Heljanko. Symbolic step encodings for object based communicating state machines. In *FMOODS 2008*, volume 5051 of *Lecture Notes in Computer Science*, pages 96–112. Springer, 2008.

[12] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, 89(4), 2003.

[13] Javier Esparza and Keijo Heljanko. *Unfoldings — A Partial-Order Approach to Model Checking*. Springer, 2008.

[14] Malay K. Ganai and Aarti Gupta. Efficient modeling of concurrent systems in BMC. In *SPIN 2008*, volume 5156 of *Lecture Notes in Computer Science*, pages 114–133. Springer, 2008.

[15] Orna Grumberg, Flavio Lerda, Ofer Strichman, and Michael Theobald. Proof-guided underapproximation-widening for multi-process systems. In *POPL 2005*, pages 122–131. ACM, 2005.

[16] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods, and Practical Use*, volume 1. Springer, 1997.

[17] Toni Jussila. On bounded model checking of asynchronous systems. Research Report A97, Helsinki University of Technology, Laboratory for Theoretical Computer Science, 2005. Doctoral dissertation.

[18] Toni Jussila, Keijo Heljanko, and Ilkka Niemelä. BMC via on-the-fly determinization. *STTT*, 7(2):89–101, 2005.

[19] Victor Khomenko, Alex Kondratyev, Maciej Koutny, and Walter Vogler. Merged processes: a new condensed representation of Petri net behaviour. *Acta Inf.*, 43(5):307–330, 2006.

[20] Kenneth L. McMillan. Lazy abstraction with interpolants. In *CAV 2006*, volume 4144 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

[21] Robert Nieuwenhuis and Albert Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *CAV 2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 321–334. Springer, 2005.

[22] Ishai Rabinovitz and Orna Grumberg. Bounded model checking of concurrent programs. In *CAV 2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2005.

[23] Sanjit A. Seshia, Shuvendu K. Lahiri, and Randal E. Bryant. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *DAC 2003*, pages 425–430. ACM, 2003.

[24] Ofer Strichman, Sanjit A. Seshia, and Randal E. Bryant. Deciding separation formulas with SAT. In *CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2002.

[25] Muralidhar Talupur, Nishant Sinha, Ofer Strichman, and Amir Pnueli. Range allocation for separation logic. In *CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 148–161. Springer, 2004.

[26] Chao Wang, Franjo Ivancic, Malay K. Ganai, and Aarti Gupta. Deciding separation logic formulae by SAT and incremental negative cycle elimination. In *LPAR 2005*, volume 3835 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2005.

[27] Chao Wang, Zijiang Yang, Vineet Kahlon, and Aarti Gupta. Peephole partial order reduction. In *TACAS 2008*, volume 4963 of *Lecture Notes in Computer Science*, pages 382–396. Springer, 2008.

TKK REPORTS IN INFORMATION AND COMPUTER SCIENCE

TKK-ICS-R4 Jani Lampinen
Interface Specification Methods for Software Components. June 2008.

TKK-ICS-R5 Matti Koskimies
Applying Model Checking to Analysing Safety Instrumented Systems. June 2008.

TKK-ICS-R6 Alexander Ilin, Tapani Raiko
Practical Approaches to Principal Component Analysis in the Presence of Missing Values. June 2008.

TKK-ICS-R7 Kai Puolamäki, Samuel Kaski
Bayesian Solutions to the Label Switching Problem. June 2008.

TKK-ICS-R8 Abhishek Tripathi, Arto Klami, Samuel Kaski
Using Dependencies to Pair Samples for Multi-View Learning. October 2008.

TKK-ICS-R9 Elia Liitiäinen, Francesco Corona, Amaury Lendasse
A Boundary Corrected Expansion of the Moments of Nearest Neighbor Distributions. October 2008.

TKK-ICS-R10 He Zhang, Markus Koskela, Jorma Laaksonen
Report on forms of enriched relevance feedback. November 2008.

TKK-ICS-R11 Ville Viitaniemi, Jorma Laaksonen
Evaluation of pointer click relevance feedback in PicSOM. November 2008.

TKK-ICS-R12 Markus Koskela, Jorma Laaksonen
Specification of information interfaces in PinView. November 2008.

TKK-ICS-R13 Jorma Laaksonen
Definition of enriched relevance feedback in PicSOM. November 2008.

ISBN 978-951-22-9847-1 (Print)
ISBN 978-951-22-9848-8 (Online)
ISSN 1797-5034 (Print)
ISSN 1797-5042 (Online)