Subsequence Invariants*

Klaus Dräger and Bernd Finkbeiner

Universität des Saarlandes
Fachrichtung Informatik, 66123 Saarbrücken, Germany
{draeger|finkbeiner}@cs.uni-sb.de

Abstract. We introduce subsequence invariants, which characterize the behavior of a concurrent system in terms of the occurrences of synchronization events. Unlike state invariants, which refer to the state variables of the system, subsequence invariants are defined over auxiliary counter variables that reflect how often the event sequences from a given set have occurred so far. A subsequence invariant is a linear constraint over the possible counter values. We allow every occurrence of a subsequence to be interleaved arbitrarily with other events. As a result, subsequence invariants are preserved when a given process is composed with additional processes. Subsequence invariants can therefore be computed individually for each process and then be used to reason about the full system. We present an efficient algorithm for the synthesis of subsequence invariants. Our construction can be applied incrementally to obtain a growing set of invariants given a growing set of event sequences.

1 Introduction

An invariant is an assertion that holds true in every reachable state. Since most program properties can either directly be stated as invariants or need invariants as part of their proof, considerable effort has been devoted to synthesizing invariants automatically from the program text [7, 4, 2, 1, 14, 3].

The most natural approach to invariant generation, followed in almost all previous work, is to look for constraints over the program variables that are inductive with respect to the program transitions. This approach works well for sequential programs, but often fails for concurrent systems: to be inductive, the invariants must refer to variables from all (or at least multiple) processes; working on the product state space, however, is only feasible if the number of processes is very small.

We introduce a new type of program invariants, which we call *subsequence invariants*. Instead of referring to program variables, subsequence invariants characterize the behavior of a concurrent system in terms of the occurrences of synchronization events. Subsequence invariants are defined over auxiliary counter

* This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

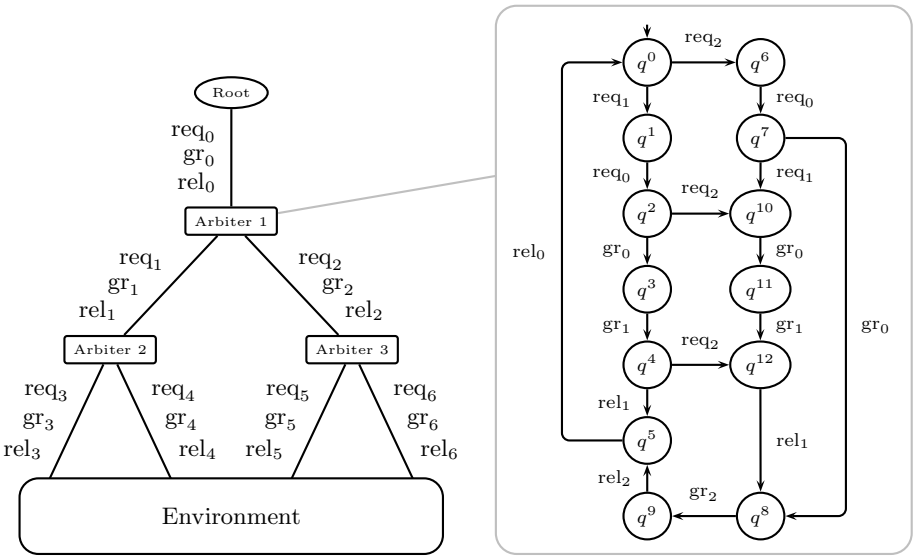

Fig. 1. Arbiter tree: Access to a shared resource is controlled by binary arbiters arranged in a tree, with a central root process.

variables that reflect how often the event sequences from a given set of subsequences have occurred so far. A subsequence invariant is a linear constraint over the possible counter values. Each occurrence of a subsequence may be *scattered* over a sequence of synchronization events: for example, the sequence *acacb* contains two occurrences (*acacb* and *acacb*) of the subsequence *ab*. This robustness with respect to arbitrary interleavings with other events ensures that subsequence invariants are preserved when a given process is composed with additional processes. Subsequence invariants can therefore be computed individually for each process and then be used to reason about the full system.

As an example, consider the arbiter tree shown in Figure 1. The environment represents the clients of the system, which may request access to a shared resource from one of the leaf nodes of the arbiter tree. The arbiter node then sends a request to its parent in the tree. This request is forwarded up to the central root process, which generates a grant as soon as the resource is available. The grant is propagated down to a requesting client, which then accesses the resource and eventually sends a release signal when it is done. Each arbiter node satisfies the following subsequence invariants:

(1) Whenever a grant is given to a child, the number of grants given to the other child so far equals the number of releases received from it. For example, for Arbiter 1, each occurrence of gr_2 in an event sequence w is preceded by an equal number of occurrences of gr_1 and rel_1:

$$|w|_{\mathrm{gr}_1\,\mathrm{gr}_2} = |w|_{\mathrm{rel}_1\,gr_2} \text{ and, symmetrically, } |w|_{\mathrm{gr}_2\,\mathrm{gr}_1} = |w|_{\mathrm{rel}_2\,\mathrm{gr}_1}.$$

(2) Whenever a grant is given to a child, the number of grants received from the parent exceeds the number of releases sent to it by exactly 1. For example, for Arbiter 1, each occurrence of gr_1 or gr_2 is preceded by one more occurrence of gr_0 than of rel_0:

$$|w|_{\mathrm{gr}_0\,\mathrm{gr}_i} = |w|_{\mathrm{rel}_0\,\mathrm{gr}_i} + |w|_{\mathrm{gr}_i}, \text{ for } i = 1, 2.$$

(3) Whenever a release is sent to, or a grant received from, the parent, the number of releases received from each child equals the number of grants given to that child. For Arbiter 1:

$$|w|_{\mathrm{gr}_i\,\mathrm{gr}_0} = |w|_{\mathrm{rel}_i\,\mathrm{gr}_0} \text{ and } |w|_{\mathrm{gr}_i\,\mathrm{rel}_0} = |w|_{\mathrm{rel}_i\,\mathrm{rel}_0}, \text{ for } i = 1, 2.$$

(4) The differences between the corresponding numbers of grants and releases only ever take values in $\{0, 1\}$. For Arbiter 1:

$$|w|_{\mathrm{gr}_i\,\mathrm{rel}_i} + |w|_{\mathrm{rel}_i\,\mathrm{gr}_i} = |w|_{\mathrm{gr}_i\,\mathrm{gr}_i} + |w|_{\mathrm{rel}_i\,\mathrm{rel}_i} + |w|_{\mathrm{rel}_i}, \text{ for } i = 0, 1, 2.$$

Combined, the subsequence invariants (1) - (4) of all arbiter nodes imply that the arbiter tree guarantees mutual exclusion among its clients.

In this paper, we present algorithms for the synthesis of subsequence invariants that automatically compute all invariants of an automaton for a given set of subsequences. Since the set of invariants is in general not finite, it is represented algebraically by a finite set of generators. Based on the synthesis algorithms, we propose the following verification technique for subsequence invariants:

To prove a desired system property φ, we first choose, for each process, a set U of relevant subsequences and then synthesize a basis of the subsequence invariants over U. The invariants computed for each individual process translate to invariants of the system. If φ is a linear combination of the system invariants, we know that φ itself is a valid invariant.

The only manual step in this technique is the choice of an appropriate set of subsequences, which depends on the complexity of the interaction between the processes. A practical approach is therefore to begin with a small set of subsequences and then incrementally compute a growing set of invariants based on a growing set of subsequences until φ is proved.

In the following, due to space constraints, all proofs have been omitted. We refer the reader to the full version of this paper [6].

Related work. There is a significant body of work on the generation of invariants over program variables, ranging from heuristics (cf. [7]), to methods based on abstract interpretation (cf. [4, 2, 1, 14]) and constraint solving (cf. [3]). The key difference to our approach is that, while these approaches aim at finding a concise characterization of a complex state space, we aim at finding a concise representation of a complex process interaction. T-invariants, which relate the number of firings of different transitions in a Petri net, have a similar motivation (cf. [11]), but are not applied in the compositional manner of subsequence invariants.

Subsequence occurrences have, to the best of our knowledge, not been used in verification before. However, there has been substantial interest in subsequences in the context of formal languages, in particular in connection with Parikh matrices and their generalizations; see, for example, [8, 10, 13, 5], as well as Parikh's original paper [12], introducing Parikh images.

Subsequences are also used in machine learning, in the context of kernel-based methods for text classification [9]; here the focus is on their use as characteristic values of given pieces of text, not on the characterization of languages or systems by constraints on their possible values.

2 Preliminaries

Linear algebra. For a given finite set U, the *real vector space* $\mathbb{R}^U$ *generated by* U consists of all tuples $\phi = (\phi_u)_{u \in U}$ of real numbers indexed by the elements of U. For a given set of vectors $\phi^1, \ldots, \phi^k \in \mathbb{R}^U$, the subspace $\mathrm{span}(\phi^1, \ldots, \phi^k)$ *spanned by* $\phi^1, \ldots, \phi^k$ consists of all linear combinations $\lambda_1 \phi^1 + \ldots \lambda_k \phi^k$ for $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$. We assume that the set U is equipped with a total ordering $<$, i.e., $U = \{u^1, \ldots, u^m\}$ with $u^1 < \cdots < u^m$. We write vectors as tuples $(\phi_{u^1}, \ldots, \phi_{u^m})$ according to this order. The *pivot element* $pivot(\phi)$ is the $<$-least element u such that ϕ_u is nonzero.

A set B of linearly independent vectors is a *basis* for a subspace $H \subseteq \mathbb{R}^U$ iff $H = \mathrm{span}(B)$. When we collect the basis vectors of a subspace, we ensure the linear independence of the vectors with the following construction: To add a new vector η to a set $\{\phi^1, \ldots, \phi^k\}$ of vectors, we consider, for each vector ϕ^i, the pivot element $u^i := pivot(\phi^i)$ and reduce η to $\eta - (\eta_{u^i}/\phi^i_{u^i})\phi^i$. For the resulting vector η' we know that $\eta'_{u^i} = 0$ for all i. If $\eta' = \mathbf{0}$, then the new set of vectors is the same as the original set $\{\phi^1, \ldots, \phi^k\}$; otherwise, we reduce each vector ϕ^i from the original set to $\psi^i := \phi^i - (\phi^i_{u^i}/\eta_{u^i})\eta$, resulting in the new set $\{\psi^1, \ldots, \psi^k, \eta'\}$.

As an example, consider the set of vectors $\{\phi^1, \phi^2\} \subset \mathbb{R}^{\{1,\ldots,5\}}$, where $\phi^1 = (1, 2, 0, -1, 1)^T$ and $\phi^2 = (0, 0, 1, 2, -2)^T$, with pivot elements 1 and 3, respectively. A new vector $\eta = (1, 1, 2, 2, -1)^T$ would first be reduced to $(0, -1, 2, 3, -2)^T$ (by subtracting ϕ^1), and then to $\eta' = (0, -1, 0, -1, 2)^T$ (by subtracting $2\phi^2$). Reducing ϕ^1, we obtain $\psi^1 = (1, 0, 0, -3, 5)^T$, resulting in the new set $\{(1, 0, 0, -3, 5)^T, (0, 0, 1, 2, -2)^T, (0, -1, 0, -1, 2)^T\}$.

The *orthogonal complement* $H^\perp$ of a subspace $H \subseteq \mathbb{R}^U$ consists of the vectors that are orthogonal to those in H, i.e., all vectors ψ where the scalar product $\psi \cdot \phi = \sum_{u \in U} \psi_u \phi_u$ is zero for all $\phi \in H$. Given a basis B for H that has been reduced as described above, a basis for $H^\perp$ is obtained as follows:

Let $V \subseteq U$ be the set of all $u \in U$ which are not the pivot element of any $\phi \in B$. For each $u \in V$, define a vector ψ^u by $\psi^u_u = 1, \psi^u_v = 0$ for all $v \in V \setminus \{u\}$, and for each $\phi \in B$, $\psi^u_{pivot(\phi)} = -\phi_u/\phi_{pivot(\phi)}$. For example, given the basis $B = \{(1, 0, 0, -3, 5)^T, (0, 0, 1, 2, -2)^T, (0, -1, 0, -1, 2)^T\}$, we have that $V = \{4, 5\}$, and therefore obtain the basis vectors $\psi^4 = (3, -1, -2, 1, 0)^T$ and $\psi^5 = (-5, 2, 2, 0, 1)^T$ for $\mathrm{span}(B)^\perp$.

Alphabets and Sequences. An alphabet is a finite set of symbols. For an alphabet A, A^* is the set of finite sequences over A. The empty sequence is denoted by ϵ, the composition of two sequences $v, w \in \mathrm{A}^*$ by $v.w$, and the length of a sequence w by $|w|$.

For alphabets $\mathrm{A}_1 \subseteq \mathrm{A}_2$, the *projection* $w \downarrow_{\mathrm{A}_1}$ of a sequence $w \in \mathrm{A}_2^*$ onto A_1 is defined recursively by

$$\epsilon \downarrow_{\mathrm{A}_1} = \epsilon,\ (w.a) \downarrow_{\mathrm{A}_1} = \begin{cases} (w \downarrow_{\mathrm{A}_1}).a & \text{if } a \in \mathrm{A}_1, \\ w \downarrow_{\mathrm{A}_1} & \text{otherwise.} \end{cases}$$

We equip A with a total order $<$, and A^* with the corresponding length-lexicographical ordering given by $u <_{llex} v$ iff either

- $|u| < |v|$ or
- $|u| = |v|$, and there are $x, y, z \in \mathrm{A}^*, a, b \in \mathrm{A}$ with $a < b, u = xay, v = xbz$.

In particular, elements ϕ of the vector space $\mathbb{R}^U$, generated by a finite subset $U \subset \mathrm{A}^*$, are written according to this order, i.e., $\phi = (\phi_{u^1}, \ldots, \phi_{u^n})$ for $U = \{u^1, \ldots, u^n\}, u^1 <_{llex} \cdots <_{llex} u^n$.

Communicating automata. We consider concurrent systems that are given as a set of communicating finite-state automata. A (nondeterministic) *finite automaton* $P = (Q_P, \mathrm{A}_P, q_P^0, T_P)$ consists of

- a finite set Q_P of locations,
- a finite alphabet A_P of synchronization events,
- an initial location $q_P^0 \in Q_P$, and
- a transition relation $T_P \subseteq Q_P \times \mathrm{A}_P \times Q_P$.

When dealing with automata $P_1, \ldots, P_n$, we use i as the subscript instead of P_i.

We denote $(q, a, r) \in T_P$ by $q \xrightarrow{a}_P r$. For a sequence $w = w_1 \ldots w_n \in \mathrm{A}_P^*$, $q \xrightarrow{w}_P r$ iff $q \xrightarrow{w_1}_P \cdots \xrightarrow{w_n}_P r$. The *language* of a location $q \in Q_P$ is the set $\mathrm{L}(q) := \{w \in \mathrm{A}_P^* : q^0 \xrightarrow{w}_P q\}$; q is *reachable* iff $\mathrm{L}(q) \neq \emptyset$. We assume in the following that our automata only contain reachable locations. For a subset $Q' \subseteq Q_P$ of the locations, the language of Q' is the union of all languages of the locations in Q'. The language of an automaton P is the language of its locations, $\mathrm{L}(P) := \mathrm{L}(Q_P)$.

A set $\{P_1, \ldots, P_n\}$ of finite automata defines a system automaton $S = (Q, \mathrm{A}, q^0, \rightarrow)$, where $Q = Q_1 \times \cdots \times Q_n$, $\mathrm{A} = \mathrm{A}_1 \cup \cdots \cup \mathrm{A}_n$, and $(q_1, \ldots, q_n) \xrightarrow{a} (r_1, \ldots, r_n)$ iff for all $i \in \{1, \ldots, n\}$ either

- $a \in \mathrm{A}_i$ and $q_i \xrightarrow{a}_i r_i$, or
- $a \notin \mathrm{A}_i$ and $q_i = r_i$.

The language $\mathrm{L}(S)$ of S thus consists of all sequences w over A, such that, for each automaton P_i, the projection $w \downarrow_{\mathrm{A}_i}$ to the alphabet A_i is in the language $\mathrm{L}(P_i)$.

3 Subsequence Invariants

Let $P = (Q, \mathrm{A}, q^0, T)$ be a finite automaton. We define the subsequence invariants of P relative to a given finite, prefix-closed set of sequences $U = \{u^1, \ldots, u^n\} \subset \mathrm{A}^*$, which we call the *set of subsequences*.

Given two sequences $u = u_1 \ldots u_k$ and $w = w_1 \ldots w_n \in \mathrm{A}^*$, the *set of occurrences of u as a subsequence* in w is $[w]_u := \{(i_1, \ldots, i_k) : 1 \leq i_1 < \cdots < i_k \leq n, w_{i_j} = u_j \text{ for all } j\}$. For example, $[aababb]_{ab} = \{(1,3), (1,5), (1,6), (2,3), (2,5), (2,6), (4,5), (4,6)\}$. The sizes of these sets define the *numbers of occurrences* $|w|_u := \|[w]_u\|$. These numbers can be computed recursively, using the recurrence [8]

$$|w|_\epsilon = 1, \qquad |\epsilon|_{u.b} = 0, \qquad |w.a|_{u.b} = \begin{cases} |w|_{u.b} + |w|_u & \text{if } a = b, \\ |w|_{u.b} & \text{otherwise,} \end{cases}$$

for all $u, w \in \mathrm{A}^*, a, b \in \mathrm{A}$. This gives rise to a mapping $|.|_U$ from A^* into $\mathbb{R}^U$ defined by $|w|_U = (|w|_{u^1}, \ldots, |w|_{u^n})$.

For any subset $Q' \subseteq Q$, the *subsequence hull* of Q' is the subspace $H_{Q'}$ of $\mathbb{R}^U$ spanned by the subsequence occurrences $\{|w|_U : w \in \mathrm{L}(Q')\}$.

Definition 1. *A* subsequence invariant *for $Q' \subseteq Q$ over U is a vector $\phi \in \mathbb{R}^U$ such that for all $w \in \mathrm{L}(Q')$, $\sum_{u \in U} \phi_u |w|_u = 0$.*

The subsequence invariants for Q' define a linear subspace $I_{Q'} \subseteq \mathbb{R}^U$, which is the orthogonal complement of $H_{Q'}$ in $\mathbb{R}^U$. Special cases are the *local subsequence invariants* $I_q = I_{\{q\}}$ at $q \in Q$ and the *global invariants* of P, $I_P = I_Q$. The spaces of the invariants satisfy the relation $I_{Q'} = \bigcap_{q \in Q'} I_q$.

The sequences that satisfy a given set of subsequence invariants form a context-sensitive language [13]. The expressiveness of subsequence invariants is, however, incomparable to the regular and context-free languages. For example, subsequence invariants can characterize the context-sensitive language $\{a^n b^n c^n : n \in \mathbb{N}\} = \{w \in \mathrm{A}^* : |w|_a = |w|_b, |w|_b = |w|_c, |w|_{ba} = 0, |w|_{cb} = 0\}$, but not the regular language $\{a.w : w \in \mathrm{A}^*\}$ for some $a \in \mathrm{A}$.

Requiring invariants to be linear equalities may appear restrictive. In the remainder of this section we illustrate the expressive power of subsequence invariants by translating two useful types of invariants, *conditional* and *disjunctive* invariants, to equivalent linear subsequence invariants.

Resolving conditions. Properties (1)–(3) of the arbiter tree discussed in the introduction are examples of *conditional invariants*, stating that a linear equality over the numbers $|w|_u$ should hold whenever some event $a \in \mathrm{A}$ occurs. Obviously, the equality must be in I_{E_a}, where E_a is the set of locations in which an a-transition can occur. We can resolve the event condition to obtain a global statement, using subsequences no more than one symbol longer than those in U, as follows:

Theorem 1. *Let* $a \in \mathrm{A}$ *and* $E_a := \{q \in Q : q \xrightarrow{a} r \text{ for some } r \in Q\}$. *Then* $\sum_{u \in U} \phi_u |w|_u = 0$ *for all* $w \in \mathrm{L}(E_a)$ *if and only if* $\sum_{u \in U} \phi_u |w|_{u.a} = 0$ *for all* $w \in \mathrm{L}(P)$.

Thus, for example, the condition that the number of releases received from the left child must equal the number of grants given to it whenever a grant is given to the right child, i.e., $|w|_{\mathrm{gr}_1} = |w|_{\mathrm{rel}_1}$ for all $w \in E_{\mathrm{gr}_2}$, is equivalent to $|w|_{gr_1.gr_2} = |w|_{rel_1.gr_2}$ for all $w \in \mathrm{L}(P)$.

Resolving disjunctions. Consider now the fourth statement in the introductory example: The differences between the corresponding numbers of grants and releases only ever take values in $\{0, 1\}$. Such a *disjunctive condition* can be translated in two steps into an equivalent linear equation: The condition is first transformed into a polynomial equation (Step 1), and then reduced, using algebraic dependencies, to an equivalent linear equation (Step 2).

Step 1 is simple: The condition $\sum_{u \in U} \phi_u |w|_u \in \{c_1, \ldots, c_k\}$ is equivalent to $(\sum_{u \in U} \phi_u |w|_u - c_1) \cdots (\sum_{u \in U} \phi_u |w|_u - c_k) = 0$.

For the transformation of the resulting polynomial equation into a linear equation, we define, as an auxiliary notion, the set of *coverings* of $x \in \mathrm{A}^*$ by u and v to be

$$\begin{aligned}[x]_{u,v} := \{((i_1, \ldots, i_k), (j_1, \ldots, j_m)) : {} & i_1 < \cdots < i_k, j_1 < \cdots < j_m, \\ & u = x_{i_1} \ldots x_{i_k}, v = x_{j_l} \ldots x_{j_m}, \\ & \{i_1, \ldots, i_k, j_1, \ldots, j_m\} = \{1, \ldots, |x|\}\},\end{aligned}$$

i.e., the set of pairs of occurrences of u and v as subsequences of x such that every index in $1, \ldots, |x|$ is used in at least one of them. For example,

$$\begin{aligned}[aabaa]_{aaa,aba} = {} & \{((1,2,4),(1,3,5)), ((1,2,4),(2,3,5)), ((1,2,5),(1,3,4)), ((1,2,5),(2,3,4)), \\ & ((1,4,5),(2,3,4)), ((1,4,5),(2,3,5)), ((2,4,5),(1,3,4)), ((2,4,5),(1,3,5))\}.\end{aligned}$$

Let $|w|_{u,v} = \|[w]_{u,v}\|$ denote the number of coverings, which can be computed recursively as follows:

$$|w|_{u,\epsilon} = |w|_{\epsilon,u} = \begin{cases} 1 & \text{if } u = w, \\ 0 & \text{otherwise,} \end{cases} \qquad |\epsilon|_{u,v} = \begin{cases} 1 & \text{if } u = v = \epsilon, \\ 0 & \text{otherwise,} \end{cases}$$

$$|w.a|_{u.b,v.c} = \begin{cases} |w|_{u,v} + |w|_{u.b,v} + |w|_{u,v.c} & \text{if } b = a = c, \\ |w|_{u,v.c} & \text{if } b = a \neq c, \\ |w|_{u.b,v} & \text{if } b \neq a = c, \\ 0 & \text{if } b \neq a \neq c. \end{cases}$$

It is easy to see that for every $u, v \in \mathrm{A}^*$, the set $C(u, v) := \{x \in \mathrm{A}^* : [x]_{u,v} \neq \emptyset\}$ of sequences coverable by u and v is finite, since it cannot contain sequences longer than $|u| + |v|$.

Theorem 2. *(See Theorem 6.3.18 in [8] for an equivalent statement to (2))*

1. *For all* $u, v, w \in \mathrm{A}^*$, *there is a bijection between* $[w]_u \times [w]_v$ *and* $\biguplus_{x \in C(u,v)}([x]_{u,v} \times [w]_x)$, *and therefore,*
2. *For all* $u, v, w \in \mathrm{A}^*$, $|w|_u |w|_v = \sum_{x \in C(u,v)} |x|_{u,v} |w|_x$.

Simple examples for Theorem 2 are the equalities $|w|_a^2 = 2|w|_{aa} + |w|_a$ and $|w|_a|w|_b = |w|_{ab} + |w|_{ba}$. For $u = ab$ and $v = ba$, we obtain the equality $|w|_{ab}|w|_{ba} = |w|_{aba} + |w|_{bab} + |w|_{abab} + 2|w|_{abba} + 2|w|_{baab} + |w|_{baba}$.

The degree k polynomial equation $p(|w|_{u^1}, \ldots, |w|_{u^n}) = 0$ resulting from Step 1 can then be transformed into a linear equation using the equalities from Theorem 2. This linear equation involves subsequences of length up to kl, where l is the maximum length of any $u \in U$.

Example: For property (4) from the introduction, we obtain

$$\begin{aligned}
&|w|_{\mathrm{gr}_i} - |w|_{\mathrm{rel}_i} \in \{0, 1\} \\
&\quad \Leftrightarrow (|w|_{\mathrm{gr}_i} - |w|_{\mathrm{rel}_i})(|w|_{\mathrm{gr}_i} - |w|_{\mathrm{rel}_i} - 1) = 0 \\
&\quad \Leftrightarrow |w|^2_{\mathrm{gr}_i} - 2|w|_{\mathrm{gr}_i}|w|_{\mathrm{rel}_i} + |w|^2_{\mathrm{rel}_i} - |w|_{\mathrm{gr}_i} + |w|_{\mathrm{rel}_i} = 0 \\
&\quad \Leftrightarrow |w|_{\mathrm{gr}_i \,.\, \mathrm{gr}_i} + |w|_{rel_i . rel_i} + |w|_{rel_i} = |w|_{\mathrm{gr}_i \,.\, \mathrm{rel}_i} + |w|_{\mathrm{rel}_i \,.\, \mathrm{gr}_i}.
\end{aligned}$$

This technique can also handle more complicated constraints: An alternative characterization of Arbiter 1 is given by the requirement that for all $w \in \mathrm{L}(P)$,

$$\begin{pmatrix} |w|_{\mathrm{gr}_0} - |w|_{\mathrm{rel}_0} \\ |w|_{\mathrm{gr}_1} - |w|_{\mathrm{rel}_1} \\ |w|_{\mathrm{gr}_2} - |w|_{\mathrm{rel}_2} \end{pmatrix} \in \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \right\}.$$

Note that the possible values for the linear expressions are mutually dependent. The set of vectors on the right-hand side can be characterized as the set of all $(x, y, z)^T$ for which $x^2 - x, y^2 - y, z^2 - z, xy - y, xz - z$ and yz are all zero. Using Theorem 2, we again obtain a set of linear subsequence constraints. In general, we have:

Theorem 3. *Let* $\|U\| = n, \max\{|u| : u \in U\} = l, M \in \mathbb{R}^{k \times n}$, *and* $\phi^1, \ldots, \phi^m \in \mathbb{R}^k$. *Then the constraint given by* $M|w|_U \in \{\phi^1, \ldots, \phi^m\}$ *is equivalent to a finite set of linear subsequence constraints involving subsequences of length* $\leq ml$.

4 Computing Subsequence Invariants

In this section, we present two algorithms for computing the subsequence invariants of a given finite automaton $P = (Q, \mathrm{A}, q^0, T)$ with respect to a finite, prefix-closed set $U \subset \mathrm{A}^*$ of subsequences. The first algorithm is generally applicable. The second algorithm is a more efficient solution that is applicable if the state graph is strongly connected.

Data: Automaton $P = (Q, \mathrm{A}, q^0, T)$, finite prefix-closed $U \subset \mathrm{A}^*$
Result: Bases B_q for the subspaces $H_q = \mathrm{span}(|w|_U : w \in \mathrm{L}(q))$
// Initialization:
foreach $q \in Q$ **do** $B_q := \emptyset$;
// B_{q^0} initially contains $\{|\epsilon|_U\}$
$B_{q^0} := \{(1, 0, \ldots, 0)^T\}$;
// The open list, containing pairs (q, ϕ) to be explored
$O := \{(q^0, (1, 0, \ldots, 0)^T)\}$;
// Basis construction:
while $O \neq \emptyset$ **do**
  take (q, ϕ) from O;
  foreach $q \xrightarrow{a} r$ **do**
    $\psi := F_a \phi$;
    begin reduce ψ with B_r:
      foreach $\eta \in B_r$ **do**
        $v := \min\{u \in U : \eta_u \neq 0\}$;
        $\psi := \psi - (\psi_v / \eta_v)\eta$;
    end
    if $\psi \neq \mathbf{0}$ **then**
      $B_r := B_r \cup \{\psi\}$;
      $O := O \cup \{(r, \psi)\}$;

Fig. 2. Fixpoint iteration computing the subspaces H_q.

4.1 The general algorithm

The subsequence invariants are computed using matrices F_a representing the effect of appending $a \in \mathrm{A}$, which are defined by

$$F_a = (f_{u,v})_{u,v \in U} : f_{u,v} = \begin{cases} 1 & \text{if } u \in \{v, v.a\}, \\ 0 & \text{otherwise.} \end{cases}$$

For example, for $U = \{\epsilon, a, b, aa, ab, ba, bb\}$,

$$|w.a|_U = \begin{pmatrix} 1&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&1&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&1&0&0&1&0 \\ 0&0&0&0&0&0&1 \end{pmatrix} |w|_U, |w.b|_U = \begin{pmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 1&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&1&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&1&0&0&0&1 \end{pmatrix} |w|_U.$$

These matrices are easily seen to be unit lower triangular matrices (recall that U is ordered by $<_{llex}$) and thus have determinant 1; their inverses are

$$F_a^{-1} = (b_{u,v})_{u,v \in U} : b_{u,v} = \begin{cases} (-1)^k & \text{if } u = v.a^k, k \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

To compute the invariants, we determine, for all $q \in Q$, a basis of the subspace $H_q = \mathrm{span}(\{|w|_U : w \in \mathrm{L}(q)\})$, using the fixpoint iteration shown in Figure 2.

```
Data: Automaton P = (Q, A, q^0, T), finite prefix-closed U ⊂ A*
Result: Bases B_q for the subspaces H_q = span(|w|_U : w ∈ L(q))
// Initialization:
M_{q^0} := IC := ∅;
O := {q^0};
// Exploration:
while O ≠ ∅ do
    take q from O;
    foreach q -a-> r do
        N := F_a M_q;
        if M_r not yet defined then
            define M_r := N;
            O := O ∪ {r};
        else if M_r ≠ N then
            C := C ∪ {M_r^{-1} N};
// Basis construction:
B_{q^0} := {(1, 0, ..., 0)};
O := {(1, 0, ..., 0)};
while O ≠ ∅ do
    take φ from O;  foreach M ∈ C do
        ψ := Mφ;
        begin reduce ψ with B_{q^0}:
            foreach η ∈ B_{q^0} do
                v := min{u ∈ U : η_u ≠ 0};
                ψ := ψ − (ψ_v/η_v)η;
        end
        if ψ ≠ 0 then
            B_{q^0} := B_{q^0} ∪ {ψ};
            O := O ∪ {ψ};
foreach q ∈ Q \ {q^0} do
    B_q := {M_q φ : φ ∈ B_{q^0}}
```

Fig. 3. Local fixpoint iteration computing the subspaces H_q.

Theorem 4. *1. The sets B_q computed by the fixpoint iteration shown in Figure 2 are bases for the vector spaces H_q spanned by $\{|w|_U : w \in \mathrm{L}(q)\}$.*

2. When called for an automaton $P = (Q, \mathrm{A}, q^0, T)$ with $\|T\| = m$ and $U \subset \mathrm{A}^*$ with $\|U\| = n$, the fixpoint iteration terminates in time $O(mn^3)$.

4.2 An optimized algorithm for strongly-connected automata

If P is strongly connected, i.e., there is a path from q to r for all locations $q, r \in Q$, we can improve the construction of the invariants. For $w = w_1 \dots w_n$ such that $q \xrightarrow{w} r$, the composition $F_w = F_{w_n} \dots F_{w_1}$ is an isomorphism from H_q to its image $F_w(H_q) \subseteq H_r$, implying in particular $\dim(H_q) \leq \dim(H_r)$. In

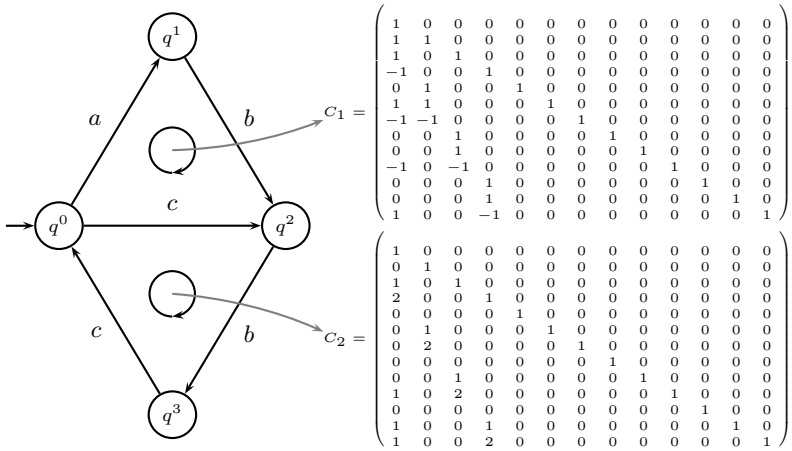

Fig. 4. Example for the local fixpoint construction.

the strongly connected case, this implies $\dim(H_q) = \dim(H_r)$ for all q, r, and $H_r = F_w(H_q)$, i.e., F_w is an isomorphism from H_q to H_r when $q \xrightarrow{w} r$.

The local fixpoint iteration shown in Figure 3 exploits this observation by finding isomorphisms $M_q : H_{q^0} \to H_q$ for all $q \in Q$ as well as a set C of automorphisms of H_{q^0} corresponding to a cycle basis of the automaton. The matrices $C_i \in C$ are then used to compute a basis of H_{q^0}. For all other $q \in Q$, H_q is obtained via M_q. The main advantage of this algorithm is the lower number of reduction steps if the cycle degree $\|T\| - \|Q\| + 1$ of P is small:

Theorem 5. *1. The set B_{q^0} computed by the local fixpoint iteration shown in Figure 3 forms a basis of H_{q^0}.*

2. When called for an automaton $P = (Q, \mathrm{A}, q^0, T)$ with $\|T\| = m$ and cycle degree $\gamma := \|T\| - \|Q\| + 1$, and $U \subset \mathrm{A}^*$ with $\|U\| = n$, the local fixpoint iteration terminates in time $O(mn^2 + \gamma n^3)$.

Example: Consider the automaton in Figure 4. Using $U = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc\}$, we compute B_{q^0} as follows:

1. Initialization: $M_{q^0} = I, C = \emptyset, B = \{\phi^0\}$, where $\phi^0 = (1, 0, \ldots, 0)$;
2. Exploration:

$$
\begin{aligned}
&q^0 \xrightarrow{a} q^1 : M_{q^1} = F_a;\\
&q^0 \xrightarrow{c} q^2 : M_{q^2} = F_c;\\
&q^1 \xrightarrow{b} q^2 : \text{add } C_1 = F_c^{-1} F_b F_a \text{ to } C;\\
&q^2 \xrightarrow{b} q^3 : M_{q^3} = F_b F_c;\\
&q^3 \xrightarrow{c} q^0 : \text{add } C_2 = F_c F_b F_c \text{ to } C;
\end{aligned}
$$

C_1 and C_2 correspond to the two basic undirected cycles $q^0 \xrightarrow{a} q^1 \xrightarrow{b} q^2 \xleftarrow{c} q^0$ and $q^0 \xrightarrow{c} q^2 \xrightarrow{b} q^3 \xrightarrow{c} q^0$.

3. Basis construction: Successively extending B by reducing and adding $L_1\phi^0$, $L_2\phi^0$, $L_1^2\phi^0$, $L_2L_1\phi^0$, $L_1L_2\phi^0$, $L_2^2\phi^0$, we obtain basis vectors

$$\begin{aligned}
\phi^0 &= (1, 0, 0,\ 0,\ 0, 0,\ 0,\ 0, 0,\ 0,\ \ 0,\ \ 0,\ \ 0)^T,\\
\phi^1 &= (0, 1, 0, -3, 0, 0, -3, 0, 0, -2,\ 0,\ \ 2,\ \ 6)^T,\\
\phi^2 &= (0, 0, 1,\ 2,\ 0, 0,\ 0,\ 0, 0,\ 1,\ \ 0,\ \ 1,\ \ 1)^T,\\
\phi^3 &= (0, 0, 0,\ 0,\ 1, 0, -3, 0, 0,\ 0,\ -3,\ \ 0,\ \ 9)^T,\\
\phi^4 &= (0, 0, 0,\ 0,\ 0, 1,\ 2,\ 0, 0,\ 0,\ \ 0,\ -3, -6)^T,\\
\phi^5 &= (0, 0, 0,\ 0,\ 0, 0,\ 0,\ 1, 0, -3,\ 2,\ \ 0,\ -6)^T,\\
\phi^6 &= (0, 0, 0,\ 0,\ 0, 0,\ 0,\ 0, 1,\ 2,\ \ 0,\ \ 2,\ \ 4)^T.
\end{aligned}$$

   All further products $L_i\phi^j$ reduce to $\mathbf{0}$.

4. Local invariant generation: Computing the orthogonal complement, we obtain the following basis for I_{q^0}:

$$\begin{aligned}
\psi^1 &= (0,\ 3,\ -2, 1,\ 0,\ \ 0,\ 0,\ 0,\ \ 0,\ 0, 0, 0, 0)^T,\\
\psi^2 &= (0,\ 3,\ \ 0,\ 0,\ 3,\ -2, 1,\ 0,\ \ 0,\ 0, 0, 0, 0)^T,\\
\psi^3 &= (0,\ 2,\ -1, 0,\ 0,\ \ 0,\ 0,\ 3,\ -2, 1, 0, 0, 0)^T,\\
\psi^4 &= (0,\ 0,\ \ 0,\ 0,\ 3,\ \ 0,\ 0, -2,\ 0,\ 0, 1, 0, 0)^T,\\
\psi^5 &= (0, -2, -1, 0,\ 0,\ \ 3,\ 0,\ 0,\ -2, 0, 0, 1, 0)^T,\\
\psi^6 &= (0, -6, -1, 0, -9,\ 6,\ 0,\ 6,\ -4, 0, 0, 0, 1)^T.
\end{aligned}$$

   For example, ψ^1 represents the invariant $3|w|_a - 2|w|_b + |w|_c = 0$, ψ^2 the invariant $3|w|_a + 3|w|_{aa} - 2|w|_{ab} + |w|_{ac} = 0$.

5. Global invariant generation: Adding the vectors $M_q\phi^i$ to B and computing the orthogonal complement, we obtain the single global invariant $3|w|_{aa} - 2|w|_{ba} + |w|_{ca} = 0$.

5 From Process to System Invariants

A key advantage of subsequence invariants is that invariants that have been computed for an individual automaton are immediately inherited by the full system and can therefore by composed by simple conjunction.

Theorem 6. *Let $S = \{P_1, \ldots, P_n\}$ be a system of communicating finite automata. If $\sum_{u\in U} \phi_u|w|_u = 0$ is a global subsequence invariant for P_i over $U \subset \mathrm{A}_i^*$, then $\sum_{u\in U} \phi_u|w|_u = 0$ also holds for all $w \in \mathrm{L}(S)$.*

The system S may satisfy additional invariants, not covered by Theorem 6, that refer to interleavings of sequences from A_i^* with sequences from a different A_j^*. In the following, we present two methods for obtaining such additional invariants.

5.1 System invariants obtained by projection

The first approach works similarly to the resolution of conditions in the Section 3. It uses the fact that given any subsequence invariant for S, we can obtain a new subsequence invariant by appending the same symbol to all involved subsequences:

Theorem 7. *Let* $\sum_{u\in U}\phi_u|w|_u = 0$ *for all* $w \in \mathrm{L}(S)$*, and* $a \in \mathrm{A}$*. Then we also have* $\sum_{u\in U}\phi_u|w|_{u.a} = 0$ *for all* $w \in \mathrm{L}(S)$*.*

Example: Consider a system containing the automaton from Figure 4. From the invariant $(3|w|_{aa} - 2|w|_{ba} + |w|_{ca})|w|_{ad} = 0$ we obtain the new invariants $3|w|_{aad} - 2|w|_{bad} + |w|_{cad} = 0$, $3|w|_{aaad} - 2|w|_{baad} + |w|_{caad} = 0$, and $3|w|_{aada} - 2|w|_{bada} + |w|_{cada} = 0$ by appending d, ad, and da, respectively.

5.2 System invariants obtained by algebraic dependencies

The equalities in Theorem 2 can be used to derive new invariants from a given set of subsequence invariants:

Let $\sum_{u\in U}\phi_u|w|_u = 0$ for all $w \in \mathrm{L}(S)$, and $v \in \mathrm{A}^*$. Then obviously, $\sum_{u\in U}\phi_u|w|_u|w|_v$ is also zero; Using the equalities $|w|_u|w|_v = \sum_{x\in C(u,v)}|x|_{u,v}|w|_x$, this can be transformed into new linear subsequence invariants $\sum_{u\in U}\sum_{x\in C(u,v)}\phi_u|x|_{u,v}|w|_x = 0$.

Example: Consider a system containing the automaton from Figure 4. It contributes the invariant $3|w|_{aa} - 2|w|_{ba} + |w|_{ca} = 0$ for all $w \in \mathrm{L}(S)$. For $v = ad$, Theorem 2 provides the algebraic dependencies

$$\begin{aligned}
|w|_{aa}|w|_{ad} &= 2|w|_{aad} + |w|_{ada} + 3|w|_{aaad} + 2|w|_{aada} + |w|_{adaa},\\
|w|_{ba}|w|_{ad} &= |w|_{bad} + |w|_{abad} + |w|_{abda} + |w|_{adba} + 2|w|_{baad} + |w|_{bada},\\
|w|_{ca}|w|_{ad} &= |w|_{cad} + |w|_{acad} + |w|_{acda} + |w|_{adca} + 2|w|_{caad} + |w|_{cada},
\end{aligned}$$

which can be used to obtain from $(3|w|_{aa} - 2|w|_{ba} + |w|_{ca})|w|_{ad} = 0$ the new subsequence invariant $6|w|_{aad}+3|w|_{ada}+9|w|_{aaad}+6|w|_{aada}+3|w|_{adaa}-2|w|_{bad}-2|w|_{abad}-2|w|_{abda}-2|w|_{adba}-4|w|_{baad}-2|w|_{bada}+|w|_{cad}+|w|_{acad}+|w|_{acda}+|w|_{adca}+2|w|_{caad}+|w|_{cada} = 0$.

Using the invariants from the previous example, the new invariant reduces to $3|w|_{aad}+3|w|_{ada}+3|w|_{aaad}+3|w|_{aada}+3|w|_{adaa}-2|w|_{abad}-2|w|_{abda}-2|w|_{adba}+|w|_{acad}+|w|_{acda}+|w|_{adca} = 0$.

6 Incremental Invariant Generation

For the invariant generation algorithms of Section 4, we considered the set U of subsequences as given and fixed. In practice, however, the set of subsequences depends on the complexity of the interaction between the processes, and is therefore not necessarily known in advance. In this section, we present an incremental method that allows for growing sets of subsequences.

Let $P = (Q_P, \mathrm{A}_P, q_P^0, T_P)$ be an automaton and $U \subset \mathrm{A}^*$ be finite and prefix-closed. Let $V = U \uplus \{v\}$ again be prefix-closed, i.e. $v = u.a$ for some $u \in U, a \in \mathrm{A}$.

Theorem 8. *Assume that for $q \in Q_P$ and the set of subsequences U, a basis of the space $H_{q,U} = \mathrm{span}(|w|_U : w \in \mathrm{L}(q))$ has already been computed, consisting of the vectors $\phi^1, \ldots, \phi^k$. Then either*

1. *$H_{q,V}$ is spanned by vectors $\psi^1, \ldots, \psi^k$ such that $\psi^j_u = \phi^j_u$ for all $u \in U$, or*
2. *$H_{q,V}$ is spanned by the vectors $\psi^1, \ldots, \psi^k, \eta$ given by:*
   - *$\psi^j_u = \phi^j_u$ for all $u \in U$, and $\psi^j_v = 0$;*
   - *$\eta_u = 0$ for all $u \in U$, and $\eta_v = 1$.*

All invariants obtained for U remain valid; in the first case, we additionally obtain a new invariant $|w|_v - \sum_{i=1}^{k} (\psi^i_v / \psi^i_{u^i}) |w|_{u^i} = 0$, where $u^i = \mathit{pivot}(\psi^i)$ for all i, while in the second case, the set of invariants is unchanged.

Example: Consider again the automaton in Figure 4. Starting with the smaller set of subsequences $U = \{\epsilon, a, b, c\}$, we obtain the basis $\{(1,0,0,0)^T, (0,1,0,-3)^T, (0,0,1,2)^T\}$ for $H_{q^0,U}$, along with the single local invariant $3|w|_a - 2|w|_b + |w|_c = 0$ for q^0. When U is extended to $V = U \cup \{aa, ab\}$ by first adding aa and then ab, case (2) of Theorem 8 holds each time. $H_{q^0,V}$ has the basis $\{(1,0,0,0,0,0)^T, (0,1,0,-3,0,0)^T, (0,0,1,2,0,0)^T, (0,0,0,0,1,0)^T, (0,0,0,0,0,1)^T\}$. Extending V to $W = V \cup \{ac\}$, case (1) holds: $H_{q^0,W}$ has the basis $\{(1,0,0,0,0,0,0)^T, (0,1,0,-3,0,0,-3)^T, (0,0,1,2,0,0,0)^T, (0,0,0,0,1,0,-3)^T, (0,0,0,0,0,1,2)^T\}$, and we obtain a new invariant, $3|w|_a + 3|w|_{aa} - 2|w|_{ab} + |w|_{ac} = 0$.

We compute $H_{q,V}$ incrementally from $H_{q,U}$ as follows:

- for each basis vector ϕ, except for the initial unit vector $|\epsilon|_U$, we remember by which multiplication $F_a\psi$ it was obtained and how it was reduced; these steps are repeated for the new index v.
- we also remember which successors $F_a\psi$ are reduced to zero; when extending U by $v = u.a$, where $u \in U$, we check for all such ψ whether the reductions result in a nonzero vector, indicating that case (2) of Theorem 8 holds.

If case (2) holds for some location q, then the new basis vector η of H_q is invariant under all $F_{a,V}$, because, by choice, v cannot be a prefix of another sequence in V. Therefore, η is also contained in the subspace H_r for all locations r reachable from q. The check for case (2) therefore only needs to be performed in one location of each strongly connected component.

7 Conclusions and Future Work

We have introduced a new class of invariants, subsequence invariants, which are linear equalities over the occurrences of sequences of synchronization events. Subsequence invariants are a natural specification language for the description

of the flow of synchronization events between different processes; basic equations over the number of occurrences of events as well their conditional and disjunctive combinations can easily be expressed.

The key advantage of subsequence invariants is that they can be computed individually for each process and compose by simple conjunction to invariants over the full system. The synthesis algorithms in this paper provide efficient means to obtain subsequence automatically from the process automata; they thus provide the foundation for a verification method that proves global system properties from locally obtained invariants.

A promising direction of future work is to extend the incremental invariant generation method from Section 6 into a refinement loop that automatically computes an appropriate set of subsequences. Also interesting is the idea of expanding the class of invariants by considering linear inequalities over the variables $|w|_U$. Such an approach could make use of established techniques for linear transition systems, combined with special properties of subsequence occurrences: for example, Theorem 2 can be used to derive general, system-independent inequalities like $|w|_{aa} - |w|_a + |w|_\epsilon \geq 0$.

References

1. S. Bensalem and Y. Lakhnech. Automatic generation of invariants. *Formal Methods in System Design*, 15(1):75–92, July 1999.
2. N. S. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Comput. Sci.*, 173(1):49–87, Feb. 1997.
3. M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *Proc. CAV*, LNCS 2725, pages 420–432. Springer-Verlag, 2003.
4. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *Proc. POPL*, pages 84–97, Jan. 1978.
5. V. N. Şerbănuţă and T. F. Şerbănuţă. Injectivity of the Parikh matrix mappings revisited. *Fundam. Inf.*, 73(1,2):265–283, 2006.
6. K. Dräger and B. Finkbeiner. Subsequence invariants. Technical Report 42, SFB/TR 14 AVACS, June 2008. ISSN: 1860-9821, http://www.avacs.org.
7. S. M. German and B. Wegbreit. A Synthesizer of Inductive Assertions. *IEEE transactions on Software Engineering*, 1(1):68–75, Mar. 1975.
8. J.Sakarovitch and I.Simon. Subwords. In M. Lothaire, editor, *Combinatorics on Words*, pages 105–144. Addison-Wesley, 1983.
9. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, 2002.
10. A. Mateescu, A. Salomaa, and S. Yu. Subword histories and Parikh matrices. *J. Comput. Syst. Sci.*, 68(1):1–21, 2004.
11. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
12. R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
13. A. Salomaa and S. Yu. Subword conditions and subword histories. *Inf. Comput.*, 204(12):1741–1755, 2006.
14. A. Tiwari, H. Rueß, H. Saïdi, and N. Shankar. A technique for invariant generation. In *Proc. TACAS*, LNCS 2031, pages 113–127. Springer-Verlag, Apr. 2001.