Regular Inference for State Machines with Parameters [*]

Therese Berg[1], Bengt Jonsson[1], Harald Raffelt[2]

[1] Department of Computer Systems, Uppsala University, Sweden
{thereseb, bengt}@it.uu.se
[2] Chair of Programming Systems and Compiler Construction, University of Dortmund, Germany
harald.raffelt@cs.uni-dortmund.de

Abstract. Techniques for inferring a regular language, in the form of a finite automaton, from a sufficiently large sample of accepted and nonaccepted input words, have been employed to construct models of software and hardware systems, for use, e.g., in test case generation. We intend to adapt these techniques to construct state machine models of entities of communication protocols. The alphabet of such state machines can be very large, since a symbol typically consists of a protocol data unit type with a number of parameters, each of which can assume many values. In typical algorithms for regular inference, the number of needed input words grows with the size of the alphabet and the size of the minimal DFA accepting the language. We therefore modify such an algorithm (Angluin's algorithm) so that its complexity grows not with the size of the alphabet, but only with the size of a certain symbolic representation of the DFA. The main new idea is to infer, for each state, a partitioning of input symbols into equivalence classes, under the hypothesis that all input symbols in an equivalence class have the same effect on the state machine. Whenever such a hypothesis is disproved, equivalence classes are refined. We show that our modification retains the good properties of Angluin's original algorithm, but that its complexity grows with the size of our symbolic DFA representation rather than with the size of the alphabet. We have implemented the algorithm; experiments on synthesized examples are consistent with these complexity results.

1 Introduction

Model-based techniques for verification and validation of reactive systems, such as model checking and model-based test generation [BJK$^+$04] have witnessed drastic advances in the last decades. They depend on the availability of a model, specifying the intended behavior of a system or component, which typically is developed during specification and design. However, in practice often no formal specification is available, or becomes outdated as the system evolves over time.

[*] Supported in part by the Swedish Research Council, and by the FP6 Network of Excellence ARTIST2

In, e.g., the telecommunication area, revision cycles are extremely short, and at the same time the short revision cycles necessitate extensive testing and verification. Therefore, there are many cases where the only means to attain correspondence between model and system component is to construct a model directly from the component. Such models can be constructed by static analysis techniques using its source code, as in software verification (e.g., [BR02,CDH$^+$00,HJMS02,Hol00]). However, many system components, including peripheral hardware components, library modules, or third-party components do not allow static analysis of source code, implying that models must be constructed from observations of their external behavior.

The construction of models from observations of component behavior can be performed using techniques for regular inference. Such techniques have been used, e.g., to create models of environment constraints with respect to which a component should be verified, for regression testing to create a specification and a test suite [HHNS02,HNS03], to perform model checking without access to code or to formal models [GPY02,PVY99], for program analysis [ABL02], and for formal specification and verification [CGP03]. For finite-state reactive systems, the regular inference problem means to infer a regular language (in the form of a deterministic finite automaton) from the answers to a finite set of *membership queries*, each of which asks whether a certain word is accepted by the system component under test (SUT) or not. There are several techniques (e.g., [Ang87,BDG97,Dup96,Gol67,KV94,RS93,TB73]) which use essentially the same basic principles. Given "enough" membership queries, the constructed automaton will be a correct model of the SUT. Angluin [Ang87] and others introduce *equivalence queries* which check whether the regular inference procedure is completed; if not they are answered by a counterexample on which the current hypothesis and the SUT disagree.

We intend to use regular inference to construct models of communication protocol entities. Such entities typically communicate by messages that consists of a protocol data unit (PDU) type with a number of parameters, each of which can assume several values. The alphabet of such models is thus typically very large. Since existing algorithms for regular inference use a number of queries, which grows polynomially with the size of the alphabet, they are not well suited for this situation. If some PDU parameters are irrelevant or almost never used, the algorithm should not be disturbed by their presence.

In this paper, we modify an algorithm for inferring a regular language, so that it is better adapted for inferring system components with large alphabets that are built from a small set of action types, each of which has a number of parameters. Most of these algorithms are based on similar principles: we choose Angluin's algorithm [Ang87] since it is well known, and since we have an existing implementation for this algorithm [RSB05]. The problem of inferring state machines where messages have arbitrary parameters appears to be very challenging. As a first step, we will in this paper assume that all parameters are booleans, and that a SUT can be modeled as an automaton, in which each transition is labeled by a PDU type and a guard over its parameters. We assume that guards are conjunctions over positive and negated parameter values. Furthermore, we will not consider the problem of inferring parameters of possible output data, but only how input parameters affect the state changes of a state machine. Ideas for how to extend these rather restrictive limitations are sketched in the last section of the paper.

Algorithms for regular inference must represent the inferred automaton in terms of externally observable elements. A state is represented by a set $[u]$ of input words u such that the automaton after reading u reaches this state. For each input symbol a, the transition from $[u]$ for input a is constructed by determining which state is reached after reading ua. In the parameterized case, input symbols are of the form $\alpha(d_1, \ldots, d_n)$, where α is an action type and $d_1, \ldots, d_n$ is a tuple of boolean parameter values. We could naively use Angluin's algorithm to find the state reached after each of these 2^n different input symbols. Instead, we will strive to save work by assuming that from each automaton state, many of the input symbols have the same effect on the SUT, and can be regarded as equivalent. We can then construct a symbolic automaton representation, where the effect of each set of equivalent input symbols is represented by a transition from this state, labeled by a guard, i.e., a boolean expression over the parameters, which characterizes the equivalence class. In cases where the number of equivalence classes is small, we would like to perform the inference with less work (as measured by the number of membership queries) than by a naive application of Angluin's algorithm.

Our inference algorithm maintains, for each inferred state, a partitioning of subsequent input symbols into assumed equivalence classes. Each class is represented by a small set of representative input symbols that (as far as we have observed) have the same effect on the SUT. If later, new information is obtained which contradicts this assumption, the equivalence class is split, thus also splitting transitions and generating more refined guards. The guard that labels a transition is obtained by a search procedure to identify precisely the effect of parameter values, inspired by work on learning of conjunctions, e.g., [KV94, Ch. 1.3].

In order to develop a consistent algorithm to do the above, we present in this paper two significant extensions of Angluin's algorithm:

1. We generalize Angluin's algorithm so that it can infer a "partially defined" automaton, which from each state defines the effect of a set of representative input symbols. The representative symbols are in general only a subset of all input symbols.
2. We define a mechanism for inferring guards of a parameterized system from the symbols in an underlying partially defined automaton, by replacing the representative symbols by guards that characterize the transitions represented by each symbol. Extra queries may need to be performed to determine guards more precisely.

Our resulting inference algorithm is intended to infer parameterized systems where guards of transitions use only a small subset of all parameters of a particular action type. We establish an upper bound on the number of posed membership queries, which is exponential in the number of parameters that appear in guards. In contrast, using Angluin's original algorithm requires a number of membership queries which is exponential in the total number of parameters of input symbols. On the other hand, the number of equivalence queries may grow in our case, since we add possibilities to construct hypothesized automata based on less information than in the original algorithm. We have performed a set of experiments on synthesized examples, which confirm this picture.

Organization. The paper is organized as follows. In the next section, we review Angluin's algorithm for inferring regular sets, and present a modification which can cope with the situation that the queries investigate different sets of suffixes for different prefixes. In Section 3, we present parameterized systems, and the technique to learn "partially defined" automata, from which guards of transitions are inferred. We prove that our algorithm retains good properties of Angluin's original algorithm, and establish upper bounds on the number of performed queries. Section 4 describes how we have implemented the ideas of the preceding section, and Section 5 presents the outcome of experiments on synthesized examples. Conclusions are presented in Section 6.

2 Inference of Finite Automata

In this section, we review the ideas underlying Angluin's algorithm, and present our generalization.

Let Σ be a finite alphabet of symbols. A *deterministic finite automaton* (*DFA*) over Σ is a structure $\mathcal{M} = (Q, \delta, q_0, F)$ where Q is a non-empty finite set of *states*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma \to Q$ is the *transition function*, and $F \subseteq Q$ is the set of *accepting states*. The transition function is extended from input symbols to words of input symbols in the standard way, by defining

$$\begin{aligned} \delta(q, \varepsilon) &= q \\ \delta(q, ua) &= \delta(\delta(q, u), a) \end{aligned}$$

An input word u is *accepted* iff $\delta(q_0, u) \in F$. The *language* accepted by $\mathcal{M}$, denoted by $\mathcal{L}(\mathcal{M})$, is the set of accepted input words.

Angluin's algorithm is designed to infer a (minimized) DFA $\mathcal{M}$ from a set of queries, each of which reveals whether a certain word is accepted or not. The algorithm is formulated in a setting, where a so called *Learner*, who initially knows nothing about $\mathcal{M}$, is trying to infer $\mathcal{M}$ by asking queries, which are of two kinds.

- A *membership query* consists in asking whether a word $w \in \Sigma^*$ is in $\mathcal{L}(\mathcal{M})$.
- An *equivalence query* consists in asking whether a hypothesized DFA $\mathcal{H}$ is correct, i.e., whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{M})$. The *Oracle* will answer *yes* if $\mathcal{H}$ is correct, or else supply a *counterexample*, which is a word u that is either in $\mathcal{L}(\mathcal{M}) \setminus \mathcal{L}(\mathcal{H})$ or in $\mathcal{L}(\mathcal{H}) \setminus \mathcal{L}(\mathcal{M})$.

The typical behavior of a *Learner* is to start by asking a sequence of membership queries, and gradually build a hypothesized DFA $\mathcal{H}$ using the obtained answers. When the *Learner* feels that she has built a "stable" hypothesis $\mathcal{H}$, she makes an equivalence query to find out whether $\mathcal{H}$ is equivalent to $\mathcal{M}$. If the result is successful, the *Learner* has succeeded, otherwise she uses the returned counterexample to revise $\mathcal{H}$ and perform subsequent membership queries until converging at a new hypothesized DFA, etc.

Let us represent the information gained by the *Learner* at any point during the learning process, as a partial mapping Obs from Σ^* to $\{+, -\}$, where $+$ stands for *accepted* and $-$ for *rejected*. The domain $Dom(Obs)$ of Obs is the set of words for which membership queries have been performed, or which the *Oracle* has given as counterexamples in equivalence queries. An inference algorithm should prescribe how to transform Obs into a DFA $\mathcal{H} = (Q, \delta, q_0, F)$, which is *conformant* with Obs, in the sense that any word $u \in Dom(Obs)$ is accepted by $\mathcal{H}$ if $Obs(u) = +$ and rejected by $\mathcal{H}$ if $Obs(u) = -$. In general, there are many such automata, and the problem to find a smallest (in number of states) such automaton is NP-complete [Gol78]. Angluin and others circumvent this problem by prescribing conditions on $Dom(Obs)$, under which it is "easy" to find a unique smallest automaton. These conditions regard each word in $Dom(Obs)$ as the concatenation of a prefix and a suffix. The idea is that prefixes are candidates for representing states of the hypothesized automaton, whereas suffixes are used to distinguish the states.

Angluin [Ang87] supports this prefix-suffix view by representing Obs in terms of an *observation table* $\mathcal{T}$, which is a partial function from a prefix-closed set $Dom(\mathcal{T}) \subseteq \Sigma^*$ of prefixes. For each $u \in Dom(\mathcal{T})$, $\mathcal{T}(u)$ is a partial function from a set $Dom(\mathcal{T}(u)) \subseteq \Sigma^*$ of suffixes to $\{+, -\}$. It is required that $\varepsilon \in Dom(\mathcal{T}(u))$ for each $u \in Dom(\mathcal{T})$. We write $Entries(\mathcal{T})$ to denote $\{(u, v) : u \in Dom(\mathcal{T})$ and $v \in Dom(\mathcal{T}(u))\}$. An observation table $\mathcal{T}$ represents the partial mapping Obs if $uv \in Dom(Obs)$ and $Obs(uv) = \mathcal{T}(u)(v)$ whenever $(u, v) \in Entries(\mathcal{T})$.

Define the *short prefixes* of an observation table $\mathcal{T}$, denoted $Sp(\mathcal{T})$, to be the set of words $u \in Dom(\mathcal{T})$ such that $ua \in Dom(\mathcal{T})$ for some $a \in \Sigma$. An observation table $\mathcal{T}$ is *complete* if $ua \in Dom(\mathcal{T})$ for all $u \in Sp(\mathcal{T})$ and $a \in \Sigma$; it is *suffix-closed* if $(u, av) \in Entries(\mathcal{T})$ where $u \in Sp(\mathcal{T})$ and $a \in \Sigma$ implies that $(ua, v) \in Entries(\mathcal{T})$. For $u, u' \in Dom(\mathcal{T})$, let $u \approx_{\mathcal{T}} u'$ denote that $\mathcal{T}(u)(v) = \mathcal{T}(u')(v)$ whenever $v \in (Dom(\mathcal{T}(u)) \cap Dom(\mathcal{T}(u')))$. The table $\mathcal{T}$ *partitioned* if $\approx_{\mathcal{T}}$ is an equivalence relation on $Dom(\mathcal{T}(u))$. A partitioned table is *closed* if whenever $(u, v) \in Entries(\mathcal{T})$ there is a $u' \in Sp(\mathcal{T})$ with $u \approx_{\mathcal{T}} u'$ and $v \in Dom(\mathcal{T}(u'))$; it is *consistent* if $ua \approx_{\mathcal{T}} u'a$ whenever $ua, u'a \in Dom(\mathcal{T})$ and $u \approx_{\mathcal{T}} u'$.

Angluin showed how to construct a unique minimal automaton from a complete, closed, and consistent observation table in the case that $Dom(\mathcal{T}(u))$ is the same for all $u \in Dom(\mathcal{T})$. Our goal in this section is to generalize this construction to the case where the set $Dom(\mathcal{T}(u))$ of suffixes may differ significantly for different prefixes $u \in Dom(\mathcal{T})$.

Definition 1. *Let $\mathcal{T}$ be a partitioned, complete, closed, and consistent observation table. Define the DFA $\mathcal{T}/\approx_{\mathcal{T}}$ as (Q, δ, q_0, F), where*

- $Q = Dom(\mathcal{T})/\approx_{\mathcal{T}}$*, i.e., Q is the set of equivalence classes of* $\approx_{\mathcal{T}}$*,*
- $\delta([u], a) = [ua]$ *for* $u \in Sp(\mathcal{T})$*,*
- $q_0 = [\varepsilon]$*,*
- $F = \{[u] \ : \ \mathcal{T}(u)(\varepsilon) = +\}$ □

Note how closedness and completeness ensures that we can define a transition for each equivalence class and symbol in Σ, and how consistency ensures that such transitions have a unique target equivalence class.

We are now ready to state a general theorem that gives constraints on any FSM that is conformant with an observation function.

Theorem 1 (Characterization Theorem). *Let $\mathcal{T}$ be a partitioned, complete, closed, and consistent observation table which represents Obs. If $\mathcal{T}$ is suffix-closed, then the DFA $\mathcal{T}/\approx_{\mathcal{T}}$ is the minimal automaton conformant with Obs.*

Angluin's algorithm uses a specialization of the conditions in Theorem 1, where $Dom(\mathcal{T}(u))$ is the same for all $u \in Dom(\mathcal{T})$.

3 Inference of Parameterized Systems

In this section, we consider how to adapt the techniques of the previous section to a setting where symbols in the alphabet are messages with parameters, e.g., as in a typical communication protocol. Since the problem of inferring state machines where messages have arbitrary parameters appears to be very challenging, we will here assume that all parameters are booleans, and that a SUT can be modeled as an automaton, in which each transition is labeled by a PDU type and a guard over its parameters. We assume that guards are conjunctions over positive and negated parameter values. Furthermore, we will not consider the problem of inferring parameters of possible output data, but only how input parameters affect the state changes of a state machine.

Let *Act* be a finite set of *actions*, each of which has a nonnegative arity. Let Σ_{Act} be the set of *symbols* of form $\alpha(d_1, \ldots, d_n)$, where α is an action of arity n, and $d_1, \ldots, d_n$ are booleans. We will use 0 and 1 to denote the boolean values *false* and *true*, respectively.

We assume a set of *formal parameters*, ranged over by $p, p_1, p_2, \ldots$. A *parameterized action* is a term of form $\alpha(p_1, \ldots, p_n)$, where α is an action α of arity n, and $p_1, \ldots, p_n$ are formal parameters. A *guard* for $\alpha(p_1, \ldots, p_n)$ is a conjunction whose conjuncts are of form p_i or $\neg p_i$ with $p_i \in \{p_1, \ldots, p_n\}$. We write $\overline{p}$ for $p_1, \ldots, p_n$ and $\overline{d}$ for $d_1, \ldots, d_n$. A *guarded action* is a pair $(\alpha(\overline{p}), g)$, where $\alpha(\overline{p})$ is a parameterized action, and g is a *guard* for $\alpha(\overline{p})$. A guarded action $(\alpha(\overline{p}), g)$ denotes the set $[\![(\alpha(\overline{p}), g)]\!] = \{\alpha(\overline{d}) \ : \ g[\overline{d}/\overline{p}]\}$ of symbols, whose parameters satisfy g.

Definition 2 (Parameterized system). *Let Act be a finite set of actions. A* parameterized system *over Act is a tuple $\mathcal{P} = (Q, \longrightarrow, q_0, F)$, where*

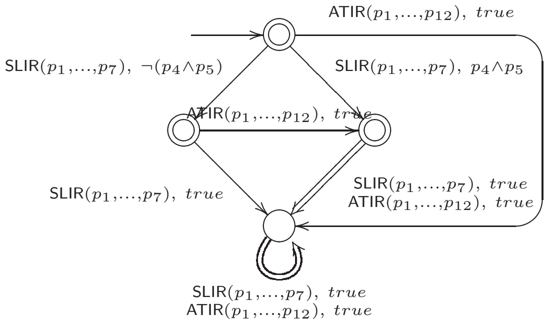
ATIR(p_1,...,p_{12}), $true$
SLIR(p_1,...,p_7), $\neg(p_4 \wedge p_5)$
SLIR(p_1,...,p_7), $p_4 \wedge p_5$
ATIR(p_1,...,p_{12}), $true$
SLIR(p_1,...,p_7), $true$
SLIR(p_1,...,p_7), $true$
ATIR(p_1,...,p_{12}), $true$
SLIR(p_1,...,p_7), $true$
ATIR(p_1,...,p_{12}), $true$

Fig. 1. Example of a parameterized system

- *Q is a finite set of states,*
- *$\longrightarrow$ is a finite set of* transitions*. Each transition is a tuple $\langle q, \alpha(\overline{p}), g, q' \rangle$, where $q, q' \in Q$ are states, and $(\alpha(\overline{p}), g)$ is a guarded action,*
- *$q_0 \in Q$ is the initial state, and*
- *$F \subseteq Q$ is a set of accepting states,*

which is completely specified and deterministic, i.e., for each state q and symbol $\alpha(\overline{d})$, there is exactly one transition $\langle q, \alpha(\overline{p}), g, q' \rangle$ from q such that $\alpha(\overline{d}) \in [\![(\alpha(\overline{p}), g)]\!]$. □

We write $q \xrightarrow{\alpha(\overline{p}),g} q'$ to denote that $\langle q, \alpha(\overline{p}), g, q' \rangle \in \longrightarrow$. A parameterized system is *expanded* if whenever $q \xrightarrow{\alpha(\overline{p}),g'} q'$ and $q \xrightarrow{\alpha(\overline{p}),g''} q''$, and in addition p_i or $\neg p_i$ is a conjunct of g', then either p_i or $\neg p_i$ must be a conjunct of g''. In other words, a parameterized system is expanded if all transitions from a state for some action test the same set of parameters. In Fig. **??** a fragment of a protocol provided by Mobile Arts AB [**?**] is given as an example of a parameterized system.

A parameterized system $\mathcal{P} = (Q, \longrightarrow, q_0, F)$ over *Act* denotes the DFA $\mathcal{M}_{\mathcal{P}} = (Q, \delta, q_0, F)$ over Σ_{Act}, where δ is defined by

$$\delta(q, \alpha(\overline{d})) = q' \qquad \text{whenever} \qquad q \xrightarrow{\alpha(\overline{p}),g} q' \text{ and } \alpha(\overline{d}) \in [\![(\alpha(\overline{p}), g)]\!].$$

Note that δ is well-defined, since $\mathcal{P}$ is completely specified and deterministic.

We will adapt Angluin's algorithm to inference of parameterized systems, in a situation where each symbol typically has many parameters, but for which the number of outgoing transitions from each state is small compared to the number of symbols in Σ_{Act}. Ideally, the effort needed to learn a parameterized system $\mathcal{P}$ should be in proportion to the size of its description as a parameterized system, and not to its number of states and $|\Sigma_{Act}|$, as is the case for Angluin's algorithm.

To accomplish this, we make two extensions to Angluin's algorithm. First, we must abandon the requirement that the constructed observation table $\mathcal{T}$ be complete, since then $Dom(\mathcal{T})$ is at least $|\Sigma_{Act}|$ times larger than the number of states of the constructed automaton. Instead of requiring that $\mathcal{T}$ be complete, $Dom(\mathcal{T})$ will for each $u \in Sp(\mathcal{T})$ contain a set of representative continuations $u\alpha(\overline{d})$, where $\alpha(\overline{d})$ is taken from a *subset* of Σ_{Act} which in general depends on u. The ambition is that for each transition of the SUT, labeled $\alpha(\overline{p}), g$, from the state represented by u, the table contains at least one continuation $u\alpha(\overline{d})$ for a representative symbol $\alpha(\overline{d})$ with $\alpha(\overline{d}) \in [\![(\alpha(\overline{p}), g)]\!]$.

Second, in order to construct a parameterized system from an incomplete observation table, we present a technique to construct guards from representative symbols. This implies asking additional queries in order to determine guards as precisely as possible. Of course, we do not know *a priori* how many transitions leave a particular state, or how the guards partition symbols into equivalence classes. Therefore we start with a coarse default partitioning into equivalence classes, which is refined "by need". Whenever two words in the same equivalence class generate different reactions by the SUT, we split the equivalence class by introducing more guards.

In order to maintain a current hypothesis about guards, we augment the observation table $\mathcal{T}$ by a *labeling function* γ, which to each prefix $ua \in Dom(\mathcal{T})$ assigns a guarded action $\gamma_u(a)$. The idea is that the constructed parameterized system, after having processed the input word u, will process the input symbol a using a transition labeled by $\gamma_u(a)$. We make the natural requirements that $a \in [\![\gamma_u(a)]\!]$, and that the labeling function should suggest guards that make the resulting automaton completely specified and deterministic, i.e., for each $u \in Sp(\mathcal{T})$, we have

- $\bigcup_{ua \in Dom(\mathcal{T})} [\![\gamma_u(a)]\!] = \Sigma_{Act}$, and
- $ua, ua' \in Dom(\mathcal{T})$ implies either $[\![\gamma_u(a)]\!] = [\![\gamma_u(a')]\!]$ or $[\![\gamma_u(a)]\!] \cap [\![\gamma_u(a')]\!] = \emptyset$.

The addition of a labeling function makes it natural to strengthen the notion of consistency, to allow a unique parameterized system to be constructed from an observation table with a labeling function.

Definition 3. *A labeling function γ for an observation table $\mathcal{T}$ is* guard-consistent *if for any $ua, u'a' \in Dom(\mathcal{T})$ such that $u \approx_{\mathcal{T}} u'$ and $[\![\gamma_u(a)]\!] \cap [\![\gamma_{u'}(a')]\!] \neq \emptyset$, we have $ua \approx_{\mathcal{T}} u'a'$.*

Intuitively, whereas consistency states that extensions ua and $u'a$ in $Dom(\mathcal{T})$ of equivalent prefixes u and u' with the *same symbol* a should also be equivalent, guard-consistency requires that two symbols a, a', whose labeling functions *overlap* should have equivalent extensions in $Dom(\mathcal{T})$. Note that guard-consistency as a special case implies that $ua \approx_{\mathcal{T}} ua'$ whenever $ua, ua' \in Dom(\mathcal{T})$ and $[\![\gamma_u(a)]\!] = [\![\gamma_u(a')]\!]$.

We now have defined enough concepts to be able to define how to construct a parameterized system from an observation table with a labeling function.

Definition 4. *Let Act be a finite set of actions. Let $\mathcal{T}$ be a partitioned, closed, and consistent observation table, and let γ be a guard-consistent labeling function for $\mathcal{T}$. Define the parameterized system $\langle \mathcal{T}, \gamma \rangle / \approx_{\mathcal{T}}$ as $(Q, \longrightarrow, q_0, F)$, where*

- $Q = Dom(\mathcal{T})/ \approx_{\mathcal{T}}$,
- $[u] \overset{\alpha(\overline{p}),g}{\longrightarrow} [ua]$ *whenever* $ua \in Dom(\mathcal{T})$ *and* $\gamma_u(a) = (\alpha(\overline{p}), g)$, *and* u *is the principal prefix in* $[u]$,
- $q_0 = [\varepsilon]$, *and*
- $F = \{[u] \ : \ \mathcal{T}(u)(\varepsilon) = +\}$.

where for each equivalence class $[u]$ *we have designated a unique principal prefix* $u' \in [u]$ *with* $u' \in Sp(\mathcal{T})$. □

Note that guard-consistency guarantees that different choices of principal prefixes result in equivalent parameterized systems.

In general, there are many different guard-consistent labeling functions for a given observation table. We therefore define an additional criterion which constrains how conjuncts may occur in guards of a labeling function. In a table $\mathcal{T}$, define a *witnessing pair* for a prefix $u \in Sp(\mathcal{T})$, action α, and index i, to be a pair of prefixes $u\alpha(\overline{d}), u\alpha(\overline{d}') \in Dom(\mathcal{T})$ such that

- $u\alpha(\overline{d}) \not\approx_{\mathcal{T}} u\alpha(\overline{d}')$, and
- $\overline{d} = (d_1, \ldots, d_i, \ldots, d_n)$ and $\overline{d}' = (d_1, \ldots, d_i', \ldots, d_n)$ differ only in the ith parameter.

Definition 5. *A labeling function* γ *for* $\mathcal{T}$ *is* well-witnessed *if whenever* $\gamma_u(a) = (\alpha(\overline{p}), g)$ *then*

- *whenever* p_i *or* $\neg p_i$ *is a conjunct in* g, *then* $\mathcal{T}$ *contains a witnessing pair for* u, α, *and* i.
- *there is a conjunct* p_j *or* $\neg p_j$ *of* g *such that* $\mathcal{T}$ *contains a witnessing pair* $u\alpha(\overline{d}), u\alpha(\overline{d}')$ *for* u, α, *and* j, *such that* $\alpha(\overline{d}) \in [\![(\alpha(\overline{p}), g)]\!]$. □

Intuitively, the first requirement states that each conjunct of a guard g should be motivated by a witnessing pair in $\mathcal{T}$, which however need not contain a prefix that satisfies g. The second requirement states that g should be satisfied by the last symbol of at least one prefix in a witnessing pair.

We are now ready to state a theorem which relates a parameterized system $\langle \mathcal{T}, \gamma \rangle / \approx_{\mathcal{T}}$ constructed from an observation table $\mathcal{T}$, and the internal structure of the SUT.

We first adapt Theorem 1 to be sure that $\langle \mathcal{T}, \gamma \rangle / \approx_{\mathcal{T}}$ agrees with the observations.

Theorem 2. *Let* $\mathcal{T}$ *be a partitioned, closed, and consistent observation table, and let* γ *be an* $\approx_{\mathcal{T}}$*-compatible and guard-consistent labeling function for* $\mathcal{T}$. *If* $\mathcal{T}$ *is suffix-closed, then the parameterized system* $\langle \mathcal{T}, \gamma \rangle / \approx_{\mathcal{T}}$ *is conformant with* $\mathcal{T}$.

Proof. The theorem follows by adapting Theorem 1 to incomplete observation tables, and the requirement that $a \in [\![\gamma_u(a)]\!]$ for all $ua \in Dom(\mathcal{T})$. □

A more informative theorem, which can be seen as an analogue of Theorem 1 in [Ang87], is as follows,.

Theorem 3. *Let $\mathcal{T}$ be a partitioned, closed, consistent, and suffix-closed observation table, and let γ be a guard-consistent and well-witnessed labeling function for $\mathcal{T}$. Let $\langle \mathcal{T}, \gamma \rangle / \approx_{\mathcal{T}}$ be $(Q, \longrightarrow, q_0, F)$ with n states. Let $\mathcal{P} = (R, \longrightarrow', r_0, G)$ be any expanded parameterized system which is conformant with $\mathcal{T}$. Then $\mathcal{P}$ has at least n states and there is a surjective mapping h from R to Q such that*

- *$h(r_0) = q_0$,*
- *$r \in G$ iff $h(r) \in F$,*
- *if $\mathcal{P}$ has exactly n states then, whenever $h(r) = q$ and $q \xrightarrow{\alpha(\overline{p}),g} q'$, there are g', r' such that $r \xrightarrow{\alpha(\overline{p}),g'} r'$ with $h(r') = q'$ and $g' \Longrightarrow g$.*

This implies that if $\mathcal{P}$ has n states then $\langle \mathcal{T}, \gamma \rangle / \approx_{\mathcal{T}}$ has at most as many transitions as $\mathcal{P}$.

Optimization The process of obtaining a well-witnessed labeling function may need a number of additional queries, which cause $Dom(\mathcal{T})$ to be extended. The requirement that $\approx_{\mathcal{T}}$ be an equivalence relation on $Dom(\mathcal{T})$ may then necessitate even more queries, which are not necessary for making γ well-witnessed. To allow to save queries, we allow prefixes in $Dom(\mathcal{T})$ to be classified as either *essential* or *auxiliary*. We now say that an observation table is *partitioned* if

- For each $ua \in Dom(\mathcal{T})$, there is an essential $ua' \in Dom(\mathcal{T})$ with $\gamma_u(a') = \gamma_u(a)$,
- ε is an essential prefix, and
- $\approx_{\mathcal{T}}$ is an equivalence relation on essential prefixes in $Dom(\mathcal{T})$.

4 An Algorithm for Inference of Parameterized Systems

In this section, we present an algorithm for inferring parameterized systems, based on the concepts introduced in Section 3.

The basic idea of our algorithm is to perform membership queries until we have a suffix-closed, partitioned, closed, and consistent observation table with a guard-consistent and well-witnessed labeling function. We can then construct a conjecture and pose an equivalence query. As long as the table does not satisfy some condition mentioned in Theorem 3, this is handled as follows.

- If $\mathcal{T}$ is not suffix-closed, i.e., there is a $(u, av) \in Entries(\mathcal{T})$ where $u \in Sp(\mathcal{T})$, such that $(ua, v) \notin Entries(\mathcal{T})$, then add (ua, v) to $Entries(\mathcal{T})$ (letting $\mathcal{T}(ua)(v) = \mathcal{T}(u)(av)$).
- If $\mathcal{T}$ is not partitioned, i.e., $\approx_{\mathcal{T}}$ is not an equivalence relation, then there are $u, u', u'' \in Dom(\mathcal{T})$ such that $u \approx_{\mathcal{T}} u'$, $u \approx_{\mathcal{T}} u''$ but $\mathcal{T}(u')(v) \neq \mathcal{T}(u'')(v)$ for some v. In this case, ask a membership query for uv, whose result is entered as $\mathcal{T}(u)(v)$ to determine whether u should be equivalent to u' or u''.
- If $\mathcal{T}$ is not closed, then for some $ua \in Dom(\mathcal{T})$ we have $ua \not\approx_{\mathcal{T}} u'$ for all $u' \in Sp(\mathcal{T})$. We then add ua to $Sp(\mathcal{T})$ by adding, for each $\alpha \in Act$, some word of form $ua\alpha(\overline{d})$ to $Dom(\mathcal{T})$, and let $\gamma_{ua}(\alpha(\overline{d})) = (\alpha, true)$. Priority given to parameters $\overline{d}'$ for which $\alpha(\overline{d}')v \in Dom(\mathcal{T}(ua))$ for some v, since suffix-closedness then requires that $ua\alpha(\overline{d}') \in Dom(\mathcal{T})$.

– If $\mathcal{T}$ is not consistent, then we have two entries (ua, v) and $(u'a, v)$ in *Entries*$(\mathcal{T})$ with $\mathcal{T}(ua)(v) \neq \mathcal{T}(u'a)(v)$ but $u \approx_{\mathcal{T}} u'$. Then add (u, av) and (u', av) to *Entries*$(\mathcal{T})$ and enter the results from $\mathcal{T}(ua)(v)$ and $\mathcal{T}(u'a)(v)$, respectively.

The table must also be equipped with a labeling function γ, which is maintained during the algorithm. Initially, for each $u \in Sp(\mathcal{T})$ and each action α, we choose some values $\overline{d}$ for the parameters of α, and let $u\alpha(\overline{d}) \in Dom(\mathcal{T})$ with $\gamma_u(\alpha(\overline{d})) = (\alpha, true)$. Whenever we add a prefix $u\alpha(\overline{d})$ to $Dom(\mathcal{T})$ the labeling function is updated in one of two ways. If there is not yet a prefix $u\alpha(\overline{d}') \in Dom(\mathcal{T})$ for any $\overline{d}'$ we let $\gamma_u(\alpha(\overline{d})) = (\alpha, true)$, otherwise we let $\gamma_u(\alpha(\overline{d})) = \gamma_u(\alpha(\overline{d}'))$, where $\alpha(\overline{d}')$ is the existing symbol such that $\alpha(\overline{d}) \in \left\llbracket \gamma_u(\alpha(\overline{d}')) \right\rrbracket$.

If $Dom(\mathcal{T})$ contains only one prefix $u\alpha(\overline{d})$ for each u and α, then γ is well-witnessed. However, if another prefix $u\alpha(\overline{d}')$ is entered, for which $u\alpha(\overline{d}) \not\approx_{\mathcal{T}} u\alpha(\overline{d}')$, this destroys the guard-consistency. We then have to refine the labeling function γ, and possibly also the partitioning into equivalence classes.

If γ is not guard-consistent, this may be because there are u, a, and a' such that $\gamma_u(a) = \gamma_u(a')$ but $ua \not\approx_{\mathcal{T}} ua'$. Let $\gamma_u(a)$ be $(\alpha(\overline{p}), g)$. In this case, we must split the guard g so that a and a' are assigned disjoint guards. In order to find an appropriate parameter for the splitting, and to keep γ well-witnessed, we find (e.g., by binary search) two tuples, $\overline{d} = (d_1, \ldots, 1, \ldots, d_n)$ and $\overline{d}' = (d_1, \ldots, 0, \ldots, d_n)$, of parameter values of α, with $\alpha(\overline{d}), \alpha(\overline{d}') \in \llbracket \gamma_u(a) \rrbracket$, which differ only in some parameter (with index, say, i), such that $\mathcal{T}(u\alpha(\overline{d}))(v) \neq \mathcal{T}(u\alpha(\overline{d}'))(v)$ for some v. We then add $(u\alpha(\overline{d}), v)$ and $(u\alpha(\overline{d}'), v)$ to *Entries*$(\mathcal{T})$, and update the labeling function so that all $ua'' \in \llbracket \gamma_u(a) \rrbracket$ now labeled by the guard $g \wedge p_i$ or $g \wedge \neg p_i$.

A second source of guard-inconsistency is that we can have two equivalent prefixes in $Sp(\mathcal{T})$ which have different partitionings of the next symbols, induced by the labeling function. It must then always be the case that there exist u, u', a, and a' such that $ua, u'a' \in Dom(\mathcal{T})$, $u \approx_{\mathcal{T}} u'$, and $a' \in \llbracket \gamma_u(a) \rrbracket$ but $\mathcal{T}(ua)(v) \neq \mathcal{T}(u'a')(v)$ for some v. A membership query for $ua'v$ should clarify the situation, either giving rise to a guard-inconsistency, or causing $u \not\approx_{\mathcal{T}} u'$ (and continuing processing it as an inconsistency).

When we have a partitioned, suffix-closed, consistent, and closed table with a well-witnessed and guard-consistent labeling function, we can construct a conjecture as described in Definition 4. The conjecture is provided to the oracle in an equivalence query and the oracle in turn either gives an affirmative answer or a counter-example. In the first case, the algorithm terminates and outputs the correct model. In the second case, the oracle returns a counter-example, i.e., a word u such that $Obs(u) = +$ but the provided automaton does not accept u (or vice versa). As in the standard algorithm of Angluin, we enter all prefixes of u into $Dom(\mathcal{T})$. This will subsequently cause either an inconsistency and hence a "new" state, or a guard-inconsistency and hence a “new” transition.

Algorithm Query complexity We estimate the complexity of our algorithm in terms of a minimal expanded parameterized system which accepts exactly the language as the SUT. Let n be its number of states, let m be the number of transitions, let $|Act|$ be the number of actions in Act, let $|\Sigma_{Act}|$ be the number of symbols in Σ_{Act}, and let c be the length of the longest counter-example received from the oracle.

The expected bottle-neck in practice for an inference algorithm is the number of membership and equivalence queries, since queries often involve comparatively slow communication with an external device. Let us first estimate the number of equivalence queries. An equivalence query can either give rise to

- an inconsistency which results in a new state; this can occur at most n times, or
- a guard-inconsistency which results in splitting a guard; this can occur at most $m - n|Act|$ times.

Hence the algorithm performs at most $n + m - n|Act|$ equivalence queries.

Let us then estimate the number of membership queries. The number of membership queries required are dependent on the number of prefixes in $Dom(\mathcal{T})$ and the maximum number of suffixes in any $Dom(\mathcal{T}(u))$. Each $Dom(\mathcal{T}(u))$ contains at most n suffixes, since each time we add a new suffix to $Dom(\mathcal{T}(u))$ we separate at least a pair of prefixes into different equivalence classes. The number of prefixes in $Dom(\mathcal{T})$ is at most

- one for each equivalence class; totally n, plus
- one for each state and action, plus an extra essential pair of prefixes as witness for each transition, in total $n|Act| + 2m$, plus
- prefixes of counterexamples, in total $c(n + m - n|Act|)$.

Hence the number of membership queries performed by the algorithm is $O(cmn)$ (since $n|Act| \leq m$). We can contrast this with a naive application of Angluins algorithm, which in the worst case requires $O(cn^2|\Sigma_{Act}|)$ membership queries. Thus, whereas a naive use of Angluins algorithm uses a number of membership queries which grows linearly with $|\Sigma_{Act}|$, i.e., exponentially in the arity of actions, our algorithm grows exponentially only with the number of parameters of an action that is used in guards of transitions. It should be remarked that Angluin's treatment of counterexamples is poorly optimized, resulting in the factor c in the worst-case bound. Rivest and Shapire [RS93] have presented techniques for replacing the factor c by $\log c$, which should apply also to our algorithm.

5 Experimental Results

We are interested in examining how the performance of the inference algorithm for parameterized systems depends on the number of parameters that occur in guards in the transitions of the system and how it compares with a naive application of Angluin's algorithm. Let us first define a measure for this. Let the *parameter complexity* for a state q and action α of a parameterized system, be the

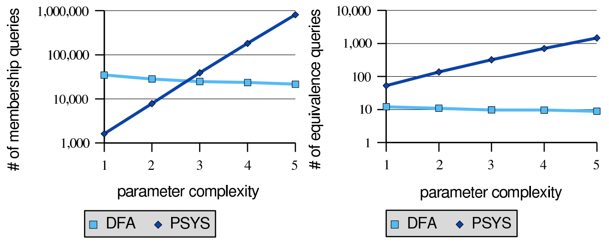

Fig. 2. Experimental results on random generated parameterized systems with 50 states and 5 parameters

total number of different parameters used in guards on transitions from q labeled $\alpha(\overline{p})$. We want to investigate how the parameter complexity effects the number of membership and equivalence queries required by the algorithm. For this purpose, we have implemented our inference algorithm for parameterized systems. The implementation is in C++ as an extension of the LearnLib tool [RSB05], developed at Dortmund University.

We measure performance on randomly generated parameterized systems and a small model of an instance of a protocol provided by Mobile Arts AB (see Fig. **??**). The protocol was first modeled in LOTOS and then transformed into a DFA by the CAESAR/ALDEBARAN Development Package [**?**]. The protocol is a small fragment of the Network Presence Center (NPC) product of the company. The NPC is a middle-ware product to allow Mobile Network Operators to provide various presence information from the GSM network. The parameterized system model of the protocol has 4 states, one action with arity 12 and another with arity 7. The first action has on average parameter complexity 0 and the second 0.5. In the randomly generated systems we have used actions with arity 5, and generated automata in which each state-action pair has the same parameter complexity. We have varied the parameter complexity between 1 and 5. The systems has then been inferred both by our algorithm and by Angluin's algorithm. The results of the experiments are summarized in Figure 1, where the left diagram shows the number of membership queries, and the right diagram shows the number of equivalence queries.

The left diagram shows that the number of membership queries for our algorithm grows exponentially with the parameter complexity of the system, whereas it is independent of parameter complexity for Angluin's original algorithm. For a parameter complexity of less than 3, our algorithm performs better, but when parameter complexity increases, the overhead of our algorithm makes it clearly worse than Angluin's. The right diagram shows that our algorithm always performs more equivalence queries than Angluin's.

Applying Angluin's algorithm to Mobile Arts' protocol fragment gives rise to 76000 membership queries and 3 equivalence queries, while our algorithm only requires 21 membership queries and 4 equivalence queries. The reason for this difference is the relatively low parameter complexity in the overall system in comparison to the high arity of its actions.

The higher number of equivalence queries for our algorithm is an expected consequence of the observation that our algorithm allows to construct equivalence queries that are based on less complete information than Angluin's algorithm. In particular, we allow equivalence queries even if the refinement of equivalence classes of symbols is not completed. For higher parameter complexity (4 or 5), the difference in number of equivalence queries is significant. We believe that this explains the sharp growth of membership queries for parameter complexities 4 and 5, since a large number of equivalence queries gives rise to an explosion in membership queries that are caused by prefixes of counterexamples.

6 Conclusions

In this paper, we have adapted techniques for inference of finite automata from sets of observations, in order that they perform better for state machines whose symbols are generated from a small set of actions, each of which has a set of parameters. Our algorithm tries to find representative observations, from which we infer guards of transitions by techniques for inferring boolean expressions. Thus, our work indicates a way to combine techniques for inferring properties of data types with regular inference techniques for inferring reactive behavior. Our algorithm requires less observations in the case that only a subset of parameters are used to determine the behavior of the machine at each transition. Future work includes to improve the handling of counterexamples in our tool, and to evaluate our techniques on a realistic communication protocol module.

Since our framwork limits us to only handeling inputs but not outputs, we suggest possible solutions. One approach is to infer a Mealy machine like Steffen et al. [**?**] but with our framework of handeling parameterized input actions. The other approach is to use our framework but encode the input and output into parameterized actions of a parameterized system. This will of course blow up the alphabet, but the dependences between input and output will be recorded in boolean formulas which may lead to very compact models.

Acknowledgement At last we would like to thank Bernhard Steffen for helpful hints and discussion.

References

1. Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A., eds.: Model-Based Testing of Reactive Systems. Volume 3472 of LNCS. Springer Verlag (2004)
2. Ball, T., Rajamani, S.: The SLAM project: Debugging system software via static analysis. In: Proc. 29^{th} ACM Symp. on Principles of Programming Languages. (2002) 1–3

3. Corbett, J., Dwyer, M., Hatcliff, J., Laubach, S., Pasareanu, C., Robby, Zheng, H.: Bandera: Extracting finite-state models from java source code. In: Proc. 22nd Int. Conf. on Software Engineering. (2000)
4. Henzinger, T., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. 29^{th} ACM Symp. on Principles of Programming Languages. (2002) 58–70
5. Holzmann, G.: Logic verification of ANSI-C code with SPIN. In Havelund, K., Penix, J., Visser, W., eds.: SPIN Model Checking and Software Verification: Proc. 7^{th} Int. SPIN Workshop. Volume 1885 of Lecture Notes in Computer Science., Stanford, CA, Springer Verlag (2000) 131–147
6. Hagerer, A., Hungar, H., Niese, O., Steffen, B.: Model generation by moderated regular extrapolation. In Kutsche, R.D., Weber, H., eds.: Proc. FASE '02, 5^{th} Int. Conf. on Fundamental Approaches to Software Engineering. Volume 2306 of Lecture Notes in Computer Science., Springer Verlag (2002) 80–95
7. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: Proc. 15^{th} Int. Conf. on Computer Aided Verification. (2003)
8. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. In Katoen, J.P., Stevens, P., eds.: Proc. TACAS '02, 8^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Volume 2280 of Lecture Notes in Computer Science., Springer Verlag (2002) 357–370
9. Peled, D., Vardi, M.Y., Yannakakis, M.: Black box checking. In Wu, J., Chanson, S.T., Gao, Q., eds.: Formal Methods for Protocol Engineering and Distributed Systems, FORTE/PSTV, Beijing, China, Kluwer (1999) 225–240
10. Ammons, G., Bodik, R., Larus, J.: Mining specificatoins. In: Proc. 29^{th} ACM Symp. on Principles of Programming Languages. (2002) 4–16
11. Cobleigh, J., Giannakopoulou, D., Pasareanu, C.: Learning assumptions for compositional verification. In: Proc. TACAS '03, 9^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Volume 2619 of Lecture Notes in Computer Science., Springer Verlag (2003) 331–346
12. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75** (1987) 87–106
13. Balcázar, J., Daz, J., Gavaldá, R.: Algorithms for learning finite automata from queries: A unified view. In: Advances in Algorithms, Languages, and Complexity. Kluwer (1997) 53–72
14. Dupont, P.: Incremental regular inference. In Miclet, L., de la Higuera, C., eds.: ICGI. Volume 1147 of Lecture Notes in Computer Science., Springer (1996) 222–237
15. Gold, E.M.: Language identification in the limit. Information and Control **10** (1967) 447–474
16. Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press (1994)
17. Rivest, R., Schapire, R.: Inference of finite automata using homing sequences. Information and Computation **103** (1993) 299–347
18. Trakhtenbrot, B., Barzdin, J.: Finite automata: behaviour and synthesis. North-Holland (1973)
19. Raffelt, H., Steffen, B., Berg, T.: Learnlib: a library for automata learning and experimentation. In: FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems, New York, NY, USA, ACM Press (2005) 62–71
20. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37** (1978) 302–320

21. Blom, J., Jonsson, B.: Automated test generation for industrial erlang applications. In: Proc. 2003 ACM SIGPLAN workshop on Erlang, Uppsala, Sweden (2003) 8–14
22. Garavel, H., Lang, F., Mateescu, R.: An overview of cadp 2001 (2002) Newsletter.
23. Steffen, B., Margaria, T., Raffelt, H., Niese, O.: Efficient test-based model generation of legacy systems. In: HLDVT'04: Proc. of the 9th IEEE Int. Workshop on High Level Design Validation and Test, Sonoma (CA), USA, IEEE Computer Society Press (2004) 95–100