Ibas: A Tool for Finding Minimal Unary NFAs

Daniel Cazalis and Geoffrey Smith

School of Computing and Information Sciences, Florida International University,
Miami, FL 33199, USA smithg@cis.fiu.edu

Abstract. In the theory of regular languages, nondeterministic finite automata (NFAs) that are minimal in the number of states are computationally challenging: there can be more than one minimal NFA recognizing a given language, and the problem of minimizing a given NFA is NP-hard, even over a unary alphabet. In this paper, we describe Ibas, a tool for finding minimal unary NFAs. The input to Ibas is a bit-vector describing an initial portion of a desired unary language; the output is a description of all minimal NFAs recognizing some extension of the given bit-vector. Searching exhaustively for such NFAs takes several hours for 6-state NFAs, and would take many trillions of year for 10-state NFAs. Ibas, in contrast, can often find minimal 10-state NFAs in about an hour. We present the theory underlying Ibas, which we call partial inductive bases, and present experimental results obtained with Ibas.

1 Introduction

In the theory of finite automata, a natural question is the minimum number of states required for a finite automaton to recognize a given regular language. With respect to *deterministic finite automata* (DFAs), the Myhill-Nerode theorem [1] shows that each regular language has a *unique* minimal DFA, up to isomorphism. Moreover, this minimal DFA can be computed efficiently: Hopcroft [2] showed that an n-state DFA can be converted into the minimal equivalent DFA in time $O(n \log n)$. When we turn our attention to the minimization of nondeterministic finite automata (NFAs), the situation is much more challenging. First, there can be more than one minimal NFA recognizing a given language. Second, the problem of converting an NFA to a minimal equivalent NFA is NP-hard, even for NFAs over a unary alphabet [3–5].

In this paper, we describe Ibas, a tool for finding minimal unary NFAs. The input to Ibas is a bit-vector describing an initial portion of a desired unary language. For example, if we are interested in minimal NFAs recognizing the language $a(a^4)^* \cup (a^6)^*$, then we might run

```
ibas 11000110010011000110100
```

This bit-vector tells whether each of the strings ϵ, a, a^2, a^3, ..., a^{23} is in the target language or not. For instance it indicates that ϵ and a are in the language, while a^2, a^3, and a^4 are not. It leaves the strings a^{24} and beyond unspecified. The output of Ibas is a description of *all* minimal NFAs recognizing some *extension*

of the given bit-vector; this means that the NFAs are free to either accept or reject each of the unspecified strings. Of course such NFAs may not necessarily recognize the language we had in mind; however we do at worst get a *lower bound* on the number of NFA states required. In this particular run, `Ibas` terminates in 37 minutes, finding a unique minimal 10-state NFA that recognizes an extension of the given bit-vector. It turns out that this NFA does recognize $a(a^4)^* \cup (a^6)^*$, so we conclude that this language has a unique minimal NFA with 10 states.

One might wonder whether `Ibas` could be implemented using exhaustive search. The trouble is that the number of n-state unary NFAs grows explosively—for instance, there are 2^{n^2} choices for the δ function, since each state may or may not have an arrow to each of the n states. Under the best enumeration we have found, the number of unary 10-state NFAs exceeds 10^{31}; hence even at a rate of 10^9 NFAs per second, exhaustive search of 10-state NFAs would require many trillions of years.

Looking ahead to the rest of the paper, Section 2 presents a few preliminary notions. Section 3 then presents `Ibas`'s underlying theory, which we call *partial inductive bases*. A *partial* is partially-specified language, as illustrated by the bit-vector in the example above. A *partial inductive basis* is a collection $\mathcal{B}$ of partials with the property that each of the *left quotients* of the the elements of $\mathcal{B}$ can be generated as a union of elements of $\mathcal{B}$. We prove a fundamental *characterization theorem* that says that a partial can be recognized by an n-state NFA iff it can be generated by an n-element partial inductive basis. Section 4 then describes the algorithm used by `Ibas` to search for partial inductive bases, and Section 5 gives some experimental results that demonstrate `Ibas`'s capabilities. Finally, Section 6 concludes.

1.1 Related Work

The complexity results mentioned above have been followed by results showing the difficulty of even *approximating* the size of a minimal NFA [5, 6]. See [7] for a recent survey of complexity results about NFA minimization. In view of these negative results, a number of studies [8–12] have explored techniques for *reducing the size* of an NFA without necessarily finding a *minimal* NFA. But we are not aware of any work that is closely similar to the goals or techniques of `Ibas`.

An early description of `Ibas` appeared in [13]; that paper lacked the theoretical foundation developed in Section 3 of this paper. Also, the results in Section 4 on efficiently extending partial inductive bases are new here; they speed up `Ibas` by roughly a factor of 50 on 9-state NFAs. Finally, we mention that more details about `Ibas` can be found in the first author's Ph.D. dissertation [14].

2 Preliminaries

We generally follow the notation of Hopcroft and Ullman [15] for languages over alphabet Σ and automata. Following Rabin and Scott [16], we allow our NFAs to have multiple start states and we do not allow ϵ-transitions:

Definition 1. *An NFA M is a tuple $(Q, \Sigma, \delta, I, F)$ where*

- *Q is a finite set (the states),*
- *Σ is a finite set (the alphabet),*
- *$\delta : Q \times \Sigma \to 2^Q$ (the transition function),*
- *$I \subseteq Q$ (the set of initial states), and*
- *$F \subseteq Q$ (the set of final states).*

Typically we view an NFA as a labeled directed graph, as shown in Figure 1; note

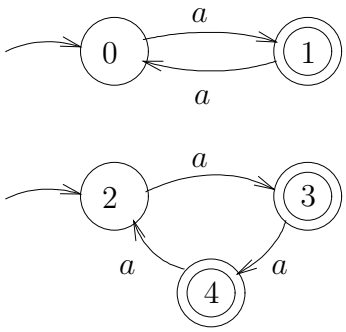

Fig. 1. A 5-state NFA for $\{a^i \mid i \bmod 6 \neq 0\}$

that this NFA has two initial states. The "guessing" ability of NFAs, sometimes called "angelic nondeterminism", is described by the following rule: NFA M accepts string w iff there exists a path labeled with w from some state in I to some state in F.

We will make heavy use of the notions of *left quotient* and *prophecy* [17]:

Definition 2. *Given language L and $a \in \Sigma$, the left quotient of L by a, written $a\backslash L$, is $\{x \in \Sigma^* \mid ax \in L\}$.*

Definition 3. *For each state q in an NFA M, the* prophecy *of q, denoted $Proph(q)$, is the set of all strings that can reach F from q.*

The name "prophecy" is appropriate, because when an NFA enters a state, it implicitly makes a prediction about what the rest of the input will be (assuming that the input is good). As an example, the states of the NFA in Figure 1 have the following prophecies:

$$\begin{aligned}
Proph(0) &= \{a, a^3, a^5, a^7, a^9, \ldots\} \\
Proph(1) &= \{\epsilon, a^2, a^4, a^6, a^8, \ldots\} \\
Proph(2) &= \{a, a^2, a^4, a^5, a^7, \ldots\} \\
Proph(3) &= \{\epsilon, a, a^3, a^4, a^6, \ldots\} \\
Proph(4) &= \{\epsilon, a^2, a^3, a^5, a^6, \ldots\}
\end{aligned}$$

We write *Prophecies*(M) to denote the collection of the prophecies of the states of NFA M; notice that this is in general a *multiset* because two states of M may have the same prophecy.

3 Partials and Partial Inductive Bases

In this section we develop the theory underlying `Ibas`. We begin with what we call *partials*, which generalize languages by allowing the status of some strings in Σ^* to be unspecified:

Definition 4. *A partial P is a function from Σ^* to $\{true, false, \bot\}$. We say that P_1 is* extended *by P_2, written $P_1 \sqsubseteq P_2$, if $P_1(w) \sqsubseteq P_2(w)$ for all $w \in \Sigma^*$, where as usual we say that $\bot \sqsubseteq true$, $\bot \sqsubseteq false$, and true and false are incomparable.*

Notice that a *language* is simply a partial that maps no strings to $\bot$. We will mostly limit ourselves subsequently to *initial* partials of some width $k \geq 0$, which specify the status of the first k strings lexicographically of Σ^*, and leave all later strings unspecified. We use the notation $Init^{(k)}(L)$ for the initial partial of width k of language L. An initial partial of width k is conveniently represented as a bit-vector of length k; for example, $Init^{(8)}((aaa)^*)$ is `10010010`.

Note that we can naturally extend union and left quotient to partials of width k. With the bit-vector representation, union is just bitwise logical *or*. Left quotient $a\backslash P$ is defined by $(a\backslash P)(w) = P(aw)$, which is rather complicated in general. But in the case of unary languages, this is just a *left shift*. For example, $a\backslash Init^{(8)}((aaa)^*)$ is `0010010`; notice that this partial has width 7 rather than 8.

We now explore the relationship between partials and NFAs.

Definition 5. *NFA M recognizes partial P iff $P \sqsubseteq L(M)$.*

If we want to find all the minimal NFAs recognizing a language L, then one approach is to search for the minimal NFAs recognizing $Init^{(k)}(L)$, for some $k \geq 0$. Such NFAs may not recognize L, of course, but this approach is useful in several ways. First, if n is the minimum number of states needed to recognize $Init^{(k)}(L)$, then n is a *lower bound* on the number of states needed to recognize L. This follows that the fact that any NFA that recognizes L also recognizes $Init^{(k)}(L)$, for every k. Second, it turns out that if k is large enough, then any minimal NFA recognizing $Init^{(k)}(L)$ must in fact recognize L. (However, it does not seem easy to know how large k must be.) Finally, even if k is not large enough, it may be that the set of all minimal NFAs recognizing $Init^{(k)}(L)$ is small enough that we can feasibly check each such NFA to see whether it accepts L. So we now develop the theory that allows `Ibas` to find NFAs recognizing a given partial.

Definition 6. *A* partial basis $\mathcal{B}$ *(of width k) is a finite multiset of partials (of width k). We say that a partial basis $\mathcal{B}$* generates *a partial P if there is some subcollection of $\mathcal{B}$ whose union extends P.*

Definition 7. *A partial basis $\mathcal{B}$ is a* partial inductive basis *if for each $B \in \mathcal{B}$ and $a \in \Sigma$, $\mathcal{B}$ generates $a\backslash B$.*

Notice that the definition of *generates* above allows the union of the subcollection to *extend* $a\backslash B$; this is needed because $a\backslash B$ is less wide than the elements of $\mathcal{B}$.

A partial basis $\mathcal{B}$ of size n and width k is conveniently represented as a bit-matrix of size $n \times k$, each row of which is the bit-vector of one of the partials. For example, over the unary alphabet $\{a\}$, the 4×8 matrix shown in Figure 2 represents a 4-element partial inductive basis of width 8. To see that this is a

	ϵ	a	a^2	a^3	a^4	a^5	a^6	a^7
1	0	0	1	1	0	1	1	1
2	0	1	0	0	1	1	0	1
3	0	1	1	0	1	1	1	1
4	1	0	0	1	1	0	1	1

Fig. 2. A matrix representation of a partial inductive basis

partial inductive basis, note for instance that the left quotient of the third row with a is the width-7 partial 1101111; this is generated by taking the union of the second and fourth rows:

$$01001101 \cup 10011011 = 11011111.$$

Notice that the union properly *extends* the left quotient.

Now we develop the relationship between NFAs and partial inductive bases. First, we observe that any NFA gives rise to partial inductive bases of every width:

Theorem 1. *If M is an NFA, then for every $k \geq 0$, $Init^{(k)}(Prophecies(M))$ is a partial inductive basis.*

Proof. For every $q \in Q$ and $a \in \Sigma$, we have $a\backslash Proph(q) = \bigcup_{r \in \delta(q,a)} Proph(r)$. The theorem then follows from straightforward properties of partials. □

More interestingly, any n-element partial inductive basis $\mathcal{B}$ gives rise to an n-state NFA M whose prophecies *extend* the elements of $\mathcal{B}$; we denote this by $\mathcal{B} \sqsubseteq Prophecies(M)$ and say that M *realizes* $\mathcal{B}$.

Theorem 2. *For every partial inductive basis $\mathcal{B}$ over Σ, there exists an NFA M that realizes $\mathcal{B}$.*

Proof. Given partial inductive basis $\mathcal{B}$ over alphabet Σ, let the elements of $\mathcal{B}$ be enumerated in some order:

$$\mathcal{B} = \{P_1, P_2, \ldots, P_n\}$$

where $n \geq 0$. (Note that the P_i's need not be distinct.)

Let $\mathbb{N}_n$ denote the set of natural numbers from 1 to n. Define NFA $M = (\mathbb{N}_n, \Sigma, \delta, I, F)$, where

– $\delta(i, a)$ is any subset of $\mathbb{N}_n$ such that $\bigcup_{j \in \delta(i,a)} P_j \sqsupseteq a\backslash P_i$,
– I is arbitrary, and
– $F = \{i \in \mathbb{N}_n \mid P_i(\epsilon) = \mathit{true}\}$.

The fact that $\mathcal{B}$ is a partial inductive basis exactly ensures that δ can be constructed.

Now we show that $\mathcal{B} \sqsubseteq \mathit{Prophecies}(M)$ by proving that for each i, P_i is extended by $\mathit{Proph}(i)$:

Lemma 1. *For all $i \in \mathbb{N}_n$, $P_i \sqsubseteq \mathit{Proph}(i)$.*

Proof. We must show that for every $w \in \Sigma^*$ and every $i \in \mathbb{N}_n$, $P_i(w) \sqsubseteq \mathit{Proph}(i)(w)$. We do this by induction on the length of w.
Basis:
If $P_i(\epsilon) = \mathit{true}$, then $P_i \in F$, so $\epsilon \in \mathit{Proph}(i)$, so $\mathit{Proph}(i)(\epsilon) = \mathit{true}$. If $P_i(\epsilon) = \mathit{false}$ or $P_i(\epsilon) = \bot$, then $i \notin F$, so $\epsilon \notin \mathit{Proph}(i)$, so $\mathit{Proph}(i)(\epsilon) = \mathit{false}$.
Induction:
Let $k \geq 0$. The induction hypothesis is that for all strings u such that $|u| \leq k$ and all $j \in \mathbb{N}_n$, $P_j(u) \sqsubseteq \mathit{Proph}(j)(u)$. Now consider the string au, where $a \in \Sigma$.
If $P_i(au) = \mathit{true}$ then, by the definition of left quotient, we have that $(a\backslash P_i)(u) = \mathit{true}$. So, by the definition of δ, we have that there exists $j \in \delta(i, a)$ such that $P_j(u) = \mathit{true}$. By the induction hypothesis, this gives $\mathit{Proph}(j)(u) = \mathit{true}$. Hence $u \in \mathit{Proph}(j)$ and (since $j \in \delta(i, a)$) therefore $au \in \mathit{Proph}(i)$, which implies that $\mathit{Proph}(i)(au) = \mathit{true}$.
If $P_i(au) = \mathit{false}$, then $(a\backslash P_i)(u) = \mathit{false}$. Hence, by the definition of δ, we have $(\bigcup_{j \in \delta(i,a)} P_j)(u) = \mathit{false}$. Hence for every $j \in \delta(i, a)$, we have $P_j(u) = \mathit{false}$, which (by the induction hypothesis) gives $u \notin \mathit{Proph}(j)$. Hence $au \notin \mathit{Proph}(i)$, which gives $\mathit{Proph}(i)(au) = \mathit{false}$.
Finally, if $P_i(au) = \bot$, then trivially $P_i(au) \sqsubseteq \mathit{Proph}(i)(au)$. □

□

As an illustration of this theorem, Figure 3 shows an NFA realizing the partial inductive basis from Figure 2. Notice that the NFA is shown with no initial states. But because I is arbitrary in the construction, any or all of its states can freely be made initial.

Now we turn our attention to the main goal of this section: to develop techniques for finding NFAs recognizing a given partial. So far we have shown that NFAs are, in some sense, equivalent to partial inductive bases. We now show that NFAs *recognizing* a partial P are, in some sense, equivalent to partial inductive bases *generating* P. This equivalence is made precise in the following two theorems.

Theorem 3. *Let P be a partial of width $k \geq 0$. If NFA M recognizes P, then $\mathit{Init}^{(k)}(\mathit{Prophecies}(M))$ is a partial inductive basis that generates P.*

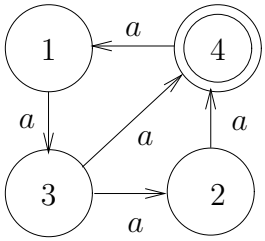

Fig. 3. An NFA that realizes a partial inductive basis

Proof. By Theorem 1, $\mathit{Init}^{(k)}(\mathit{Prophecies}(M))$ is a partial inductive basis. And if P is an partial of width k and NFA $M = (Q, \Sigma, \delta, I, F)$ recognizes P, then

$$\begin{aligned} P &= \mathit{Init}^{(k)}(L(M)) \\ &= \mathit{Init}^{(k)}(\bigcup_{q \in I} \mathit{Proph}(q)) \\ &= \bigcup_{q \in I} \mathit{Init}^{(k)}(\mathit{Proph}(q)) \end{aligned}$$

□

Theorem 4. *If $\mathcal{B}$ is an n-element partial inductive basis that generates a partial P, then there exists an n-state NFA M that recognizes P.*

Proof. If $\mathcal{B} = \{P_1, P_2, \ldots, P_n\}$, then by Theorem 2, there exists an NFA M with state set $\mathbb{N}_n$ such that for all i, $P_i \sqsubseteq \mathit{Proph}(i)$. Since $\mathcal{B}$ generates P, there is a subcollection of $\mathcal{B}$ whose union extends P. Choose I to be the set of indices of that subcollection. Then we have

$$L(M) = \bigcup_{i \in I} \mathit{Proph}(i) \sqsupseteq \bigcup_{i \in I} P_i \sqsupseteq P.$$

□

Hence the problem of finding minimal NFAs recognizing an initial partial P is essentially equivalent to the problem of finding minimal partial inductive bases generating P.

4 The `Ibas` Algorithm

Given a target language L, we let $P = \mathit{Init}^{(k)}(L)$ for some $k \geq 0$. If we wish to find all minimal partial inductive bases generating P, it is natural to try to proceed iteratively. For note that if $\mathcal{B}$ is a partial inductive basis of width k that generates P, then for any i with $0 \leq i < k$ we have that $\mathit{Init}^{(i)}(\mathcal{B})$ is a partial inductive basis of width i that generates $\mathit{Init}^{(i)}(P)$. Thus any of the bases that we seek can be built by widening narrower ones; `Ibas` does this systematically in depth-first order, starting with a trivial basis of width 0.

4.1 Extending Partial Inductive Bases

This leads us to consider the problem of widening a partial inductive basis of width i to one of width $i+1$. We have the following corollary that says that this can always be done:

Corollary 1. *Let $\mathcal{B}$ be a partial inductive basis of width i, for some $i \geq 0$. There exists a partial inductive basis $\mathcal{B}'$ of width $i+1$ such that $\mathcal{B} \sqsubseteq \mathcal{B}'$.*

Proof. If $\mathcal{B}$ is a partial inductive basis of width i, then by Theorem 2 there exists an NFA M such that $\mathcal{B} \sqsubseteq \mathit{Prophecies}(M)$. So $\mathit{Init}^{(i+1)}(\mathit{Prophecies}(M))$ is a partial inductive basis of width $i+1$, by Theorem 1. And this basis is easily seen to extend $\mathcal{B}$. □

When $\mathcal{B}$ is a partial inductive basis of size n and width i, recall that $\mathcal{B}$ can be represented as an $n \times i$ matrix. We can then understand "widening" $\mathcal{B}$ concretely as a matter of adding a new column to this matrix. The question is whether we can do this efficiently, given that there are 2^n possible columns.

Must we try all 2^n possibilities for the new column? This was what we did in our first version of the `ibas` program (described in [13]), but fortunately we can do much better. The key idea is that we can consider the extension of each element of $\mathcal{B}$ *independently* of how we extend the other elements. Since we know that *some* new column is possible, this means that for each position in the new column there are exactly three possibilities: a fixed 0, a fixed 1, or a free choice.

Let's look at an example before developing this idea formally. Consider the following partial inductive basis over a 1-symbol alphabet:

a 0010
b 0011
c 0100
d 1001

When we try to widen to a partial inductive basis of width 5, we must be sure that the widened basis can generate the left quotient of each newly-widened row. But notice that these left quotients have *width 4*, which means that the new column is irrelevant in determining whether or not they can be generated. For example, if we extend row a with 0, its left quotient will be 0100; this is generated by row c, regardless of how we extend row c. If we extend row a with 1, its left quotient will be 0101, which cannot be generated, regardless of how the other rows are extended. Thus we see that to widen this inductive basis, we *must* extend row a with 0. Now consider row b. If we extend it with 0, its left quotient will be 0110, which is generated as $a \cup c$. And if we extend it with 1, its left quotient will be 0111, which is generated as $b \cup c$. Hence we can extend row b with *either* 0 or 1, and we will still have a partial inductive basis. The reader can similarly verify that row c must be extended with 1, and row d can be extended with either 0 or 1. We can therefore exactly characterize the possible

width-5 extensions of our partial inductive basis as follows:

a 00100
b 0011?
c 01001
d 1001?

The two "?"s can be independently replaced by either 0 or 1, giving a total of 4 width-5 extensions.

The general situation is that we have a partial inductive basis $\mathcal{B}$ of width i, over some alphabet, which we are trying to extend to a partial inductive basis $\mathcal{B}'$ of width $i+1$. For each $P \in \mathcal{B}$, we want to determine how P can be extended from width i to width $i+1$ while preserving the property that each of the left quotients can be generated. Now the key is to notice that the left quotient of an initial partial of width $i+1$ has width *at most* i. This means that $\mathcal{B}'$ will be able to generate the left quotient of the extension of P iff $\mathcal{B}$ can. Hence, as in the example above, we can determine for each $P \in \mathcal{B}$ one of three possibilities: P must be extended with 0, P must be extended with 1, or P can be extended freely with either 0 or 1.

We now make some remarks about implementing these calculations efficiently. There are two main concerns: first, how to compute the left quotients of the extensions; second, how to check whether these can be generated. We can represent our partial inductive basis as an array of bit-vectors. As discussed in Section 3, computing the left quotients is then easy in the case of a unary alphabet, since left quotient is then just a left shift. In the case of a non-unary alphabet, computing the left quotients is harder—in that case it seems better to store the left quotients of P so that the left quotients of the extension P' can be computed incrementally from them. Turning to the second concern, we can efficiently check whether $\mathcal{B}$ generates a left quotient using bit operations. First, we can see whether an element P of $\mathcal{B}$ is useful by bitwise ANDing it with the bitwise negation of the left quotient; the result is 0 iff P is useful. Then we just union together all the useful elements of $\mathcal{B}$, to see whether the quotient is generated.

Many other optimizations are possible. For example, the set of elements of $\mathcal{B}$ that are useful in making a left quotient can only *shrink* as we widen the partial inductive basis. It is therefore beneficial to remember and update this set of useful elements as we iteratively widen $\mathcal{B}$.

Also, as will be discussed below, the iterative search for partial inductive bases involves both widening and narrowing. It is therefore necessary to maintain a stack-like representation of our knowledge of the 0's, 1's, and ?'s in each new column.

4.2 Generating the Target Partial

As we widen our partial inductive basis, we are not only concerned with keeping it inductive; we also want it to be able to generate the target partial $Init^{(k)}(L)$. We

deal with this concern iteratively. During our search, we have a partial inductive basis $\mathcal{B}$ of width i, where $0 \leq i < k$, which generates $Init^{(i)}(L)$. When we try to widen $\mathcal{B}$ to width $i+1$, we also widen the target partial to $Init^{(i+1)}(L)$ and see whether the widened $\mathcal{B}$ can generate the widened target.

If not, then no extension of $\mathcal{B}$ can generate the desired language L, and $\mathcal{B}$ need not be considered further. We then proceed to look at the "next" partial inductive basis by the use of the *increment* operation, which advances to the next possible last column of $\mathcal{B}$.

4.3 Incrementing a Partial Inductive Basis

When we increment the last column of $\mathcal{B}$, we must not modify the fixed 0's and fixed 1's; we can only change the "?"s. In our implementation, we maintain a bit mask to indicate the positions of the "?"s. Initially we set all the "?"s to 0, and than increment those bits as a binary counter.

This approach guarantees that we will only get partial inductive bases, but there is a further issue. Because we want to to avoid considering the same basis more than once (in different permutations of its rows), `Ibas` always maintains the rows of the basis in nondecreasing order. But when we increment the "?" bits of the last column, we might break this property. (Note that this can arise only if the width-i basis has repeated elements, which is rare unless i is less than 10.) If incrementing gives a basis that is not in nondecreasing order, we skip to the next; this guarantees that the same partial inductive basis is not considered more than once.

4.4 Shrinking a Partial Inductive Basis

Once we have exhausted all possibilities for the new column (as indicated by an *overflow* from the "?"s), we need to see whether there are any more possibilities for the previous column. We decrease the width of the basis and reinstate the previous fixed values bit mask (they are kept in a stack) then continue the increment operation at the place it was left off. When we shrink back to width 0, we know that all partial inductive bases of that size have been considered. If any were found that generate the target partial, then we terminate the search, since we only want minimal partial inductive bases. If none were found, then we increment the number of elements in the basis and begin the search again.

In conclusion, `Ibas` finds minimal partial inductive bases generating $Init^{(k)}(L)$ iteratively, by widening the partial inductive bases generating $Init^{(i)}(L)$, for $0 \leq i < k$. The actual implementation of `Ibas` is written in C; for maximum efficiency we use low-level operations such as bitwise "or" (`|`), bitwise "and" (`&`), and left shift (`<<`).

5 Experimental Results

In this section, we present some experimental results to show the effectiveness of `Ibas` in finding minimal NFAs for unary languages.

We begin by briefly mentioning some results about the limits of searching exhaustively for NFAs recognizing a given partial of a unary language. The key problem is the explosive growth in the number of n-state NFAs, even in the unary case, together with the difficulty of efficiently enumerating the non-isomorphic automata. The best enumeration that we have found enumerates a total of

$$\binom{n+7}{n} 2^{n(n-1)}$$

n-state unary NFAs. We found experimentally that our fastest C programs could search exhaustively for 6-state NFAs in about 4 hours, meaning that we can check about 120 million NFAs per second. This implies that a search of 7-state NFAs would require several years, while searching for NFAs of size 10 would require many trillions of years.

We now report on some experimental results obtained with a carefully optimized C implementation of the unary version of `Ibas`. The implementation limits the width of all partials to at most 31; as a result, operations like union and left quotient can be done in just one or two machine instructions. Our experiments were run on a 2.66 GHz Pentium 4 processor. Note that memory should not be an issue at all, as our implementation maintains only a few thousand bytes of data.

Suppose we wish to find the minimal NFAs recognizing the language $a(a^4)^* \cup (a^6)^*$. To do this, we run `Ibas` with a suitable partial:

```
ibas 1100011001001100011001000
```

In about 37 minutes, the run terminates, finding that there is a unique minimal partial inductive basis of size 10:

```
0000010000010000010000001
0000100000100000100000010
0001000001000001000000100
0001000100010001000100001
0010000010000010000010000
0010001000100010001000010
0100000100000100000100000
0100010001000100010001000
1000001000001000001000000
1000100010001000100010000
```

`Ibas` also displays a representation of the NFAs that realize this basis, using `-->` to indicate initial states and `*` to indicate final states:

```
      0 | 1
      1 | 2
      2 | 4
      3 | 5
      4 | 6
      5 | 7
      6 | 8
-->   7 | 9
--> *8 | 0
    *9 | 3
NFA completely determined.
```

In this case, `Ibas` notes that there is a unique NFA realizing the basis.

In general, of course, there could be more than one NFA realizing the partial inductive bases that `Ibas` finds. In such cases, it displays "optional" arrows with parentheses. For example, on the inductive basis

```
0111111
1111111
```

`Ibas` displays

```
-->  0 |(0) 1
    *1 |(0) 1
Arrows not completely determined.
```

Notice that the left quotient of `0111111` is `111111`. To generate this, row 1 is *necessary*, while row 0 is *optional*. For this reason, `Ibas` prints "`0 |(0) 1`", indicating that state 0 *must* have an arrow to state 1 and *can* have an arrow to state 0. Note however that "optional" arrows may not be completely optional, in that it may not be possible to omit more than *one* of them; in general the partial inductive basis must be studied to determine which combinations of arrows will work.

One family of languages that we have explored rather thoroughly is

$$L_r = \{a^i \mid i \bmod r \neq 0\},$$

the set of strings of a's whose length is not a multiple of r, for $r \geq 1$. (Note that $L_1 = \emptyset$.) L_r can of course be recognized using a ring of r states, but fewer states are sufficient if r has at least 2 distinct prime factors; for example, Figure 1 shows a 5-state NFA for L_6. In general, if $r = pq$ where p and q are relatively prime, then L_r can be recognized by an NFA with $p + q$ states. We ran `Ibas` using the width-30 partial of L_r, for various values of r. For example, the run for L_9 was

```
ibas 011111111011111111011111111011
```

The results of our experiments are shown in Table 1. (The run when $r = 11$ did not finish—after 24 hours, `Ibas` had determined that there are no inductive

r	Minimum Basis Size	Number of Bases	Run Time
1	0	1	0 sec
2	2	1	0 sec
3	3	2	0 sec
4	4	3	0 sec
5	5	6	0.002 sec
6	5	2	0.003 sec
7	7	18	0.5 sec
8	8	30	13 sec
9	9	56	6.5 min
10	7	6	1.1 sec
11	11	≥ 1	$>$ 24 hrs
12	7	6	1.5 sec
14	9	18	16 min
15	8	12	42 sec
21	10	36	15.5 hrs

Table 1. Results of `Ibas` on $L_r = \{a^i \mid i \bmod r \neq 0\}$

bases of size 10 or smaller, and it had found 1 inductive basis of size 11.) We suspect that these languages are relatively difficult cases for `Ibas`, because they contain so many more 1's than 0's, which may make `Ibas`'s pruning less effective.

On $(a^3 \cup a^5)(a^{10})^*$, `Ibas` finds 2 minimal inductive bases of size 10, taking 16 minutes.

Another interesting language to try is $(a^5 \cup a^9)^*$. The minimal DFA for this language has 33 states, because the longest string of a's that is *not* in the language is a^{31}. On the width-30 partial for this language, `Ibas` finds 27 minimal inductive bases of size 9, taking 58 seconds. The first of these is

```
000000001000010001100011001110 0
000000010000100011000110011100 1
000001000010001100011001110011
000010000100011000110011100111
000100001000110001100111001110
001000000001000010001100011001
010000000010000100011000110011
100000000100001000110001100111
```

The unique NFA realizing it is shown in Figure 4.

On $(a^3 \cup a^{11})^*$, `Ibas` finds 2,225 minimal inductive bases of size 11, taking about 19 hours. So `Ibas` can sometimes find minimal 11-state NFAs, if we are patient.

Curiously, on the apparently similar language $(a^2 \cup a^{11})^*$, `Ibas` finds that 6 states are enough! One of the NFAs that it finds is shown in Figure 5.

On the finite language $\{a, a^2, a^6, a^8, a^9\}$, `Ibas` finds 443 minimal inductive bases of size 10, taking 22 minutes. Notice that the minimal DFA for this lan-

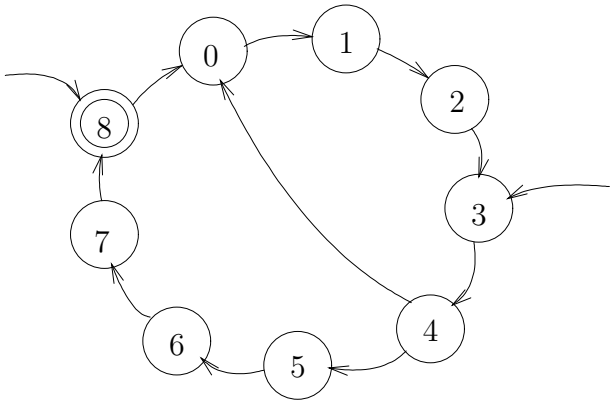

Fig. 4. A minimal NFA for $(a^5 \cup a^9)^*$

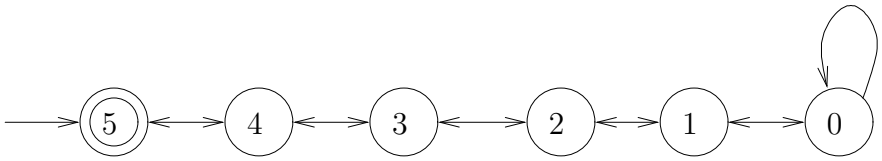

Fig. 5. A minimal NFA for $(a^2 \cup a^{11})^*$

guage contains 10 states (if we omit the "dead" state); hence nondeterminism does not let us construct a smaller machine for this language. (In fact, this is a generally-known fact—the minimal DFA for a finite unary language is also a minimal NFA.)

As a final topic, it is interesting to compare the size of `Ibas`'s search space with that of the exhaustive search considered above. On the language $L_8 = \{a^i \mid i \bmod 8 \neq 0\}$, profiling reveals that in 13 seconds `Ibas` considers about 50 million partial inductive bases of size 8, checking about 4 million bases per second. In contrast, exhaustive search can check about 120 million NFAs per second. But the search space of 8-state NFAs (using the best enumeration we have found) has size about $4.6 \cdot 10^{20}$, which is 9 trillion times as large.

6 Conclusion

Our experiments show that `Ibas` allows us to find minimal unary NFAs of up to about 10 states. Of course the NFAs found might not actually recognize the desired language, but in practice the number of NFAs found seems to be small enough that they could be inspected by hand, if necessary.

In future work, one might want to extend `Ibas` to non-unary languages. This is possible in principle, because the theory of partial inductive bases presented

in Section 3 holds over arbitrary-sized alphabets. We have experimented with a version of `Ibas` for languages over a binary alphabet. Our results have not been very encouraging, however—such languages seem to require extremely wide partials, limiting the effectiveness of `Ibas`'s search.

Finally, we mention that our `Ibas` implementation `ibas.c` will be made freely available.

References

1. Nerode, A.: Linear automaton transformations. Proc. AMS **9** (1958) 541–544
2. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing the states in a finite automaton. In Kohavi, Z., ed.: The Theory of Machines and Computations. Academic Press, New York (1971) 189–196
3. Stockmeyer, L., Meyer, A.: Word problems requiring exponential time. In: Proceedings 5th ACM Symposium on Theory of Computing. (1973) 1–9
4. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM Journal on Computing **22**(6) (December 1993) 1117–1141
5. Gramlich, G.: Probabilistic and nondeterministic unary automata. In: Proceedings of Mathematical Foundations of Computer Science. Volume 2747 of Lecture Notes in Computer Science., Springer-Verlag (August 2003) 460–469
6. Gramlich, G., Schnitger, G.: Minimizing NFA's and regular expressions. In: Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science. Volume 3404 of Lecture Notes in Computer Science., Springer-Verlag (February 2005) 399–411
7. Holzer, M., Kutrib, M.: Nondeterminstic finite automata—recent results on descriptional and computational complexity. In: Proc. CIAA 2008. Volume 5148 of Lecture Notes in Computer Science. (2008) 1–16
8. Matz, O., Potthoff, A.: Computing small nondeterministic finita automata. In: Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems. (1995) 74–88
9. Kozen, D.C.: Automata and Computability. Springer-Verlag (1997)
10. Ilie, L., Yu, S.: Algorithms for computing small NFAs. In: MFCS '02: Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science. (2002) 328–340
11. Champarnaud, J.M., Coulon, F.: NFA reduction algorithms by means of regular inequalities. Theoretical Computer Science **327**(3) (2004) 241–253
12. Ilie, L., Solis-Oba, R., Yu, S.: Reducting the size of NFAs by using equivalences and preorders. In: 16th Annual Symposium on Combinatorial Pattern Matching, Jeju Island, Korea (2005) 310–321
13. Smith, G.: Inductive bases and their application to searches for minimal unary NFAs. In: Proceedings ACMSE 2006: 44th ACM Southeast Conference, Melbourne, FL (March 2006) 470–475
14. Cazalis, D.: Algebraic Theory of Minimal Nondeterministic Finite Automata with Applications. PhD thesis, Florida International University (December 2007)
15. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
16. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development **3**(2) (1959) 114–125
17. Arnold, A., Dicky, A., Nivat, M.: A note about minimal non-deterministic automata. Bulletin of the EATCS **47** (1992) 166–169