Appendix

THE UPDATE GRAPH

In this section, Rivest and Schapire's *diversity representation* [73, 85] is described in detail. It is a particular encoding of an FSA where the state variables form a distributed representation of the FSA's state. The *update graph* is then presented. It is a particular machine that updates the values of the internal state variables. Rivest and Schapire's algorithm for learning update graphs is then described.

A.1 The Diversity Representation

The diversity representation concepts are now presented:

Definition 1 (Action Sequences)*: Let $S = A^*$ denote the set of all strings of zero or more actions, where A is the set of actions, and the Kleene star $*$ denotes the closure operation. Each such string of actions is called an action sequence. Let λ be the zero-length action sequence.*

The test state variables can now be defined:

Definition 2 (Test)*: A test $t = sp$ is an action sequence followed by a predicate.*

Let $T = SP = \{sp | s \in S, p \in P\}$ denote the set of all tests. The *value* of a test $t = sp$ in a state q is denoted by qt, where $qt = q(sp) = qsp = (qs)p$, qs denotes the state resulting from executing action sequence s in state q, and $(qs)p$ is the value of predicate p in state qs. A test $t = sp$ is said to *succeed at state* q if $qt =$ **true**. Otherwise test t is said to *fail at* q. Executing a test $t = sp$ in state q means executing action sequence s in state q and observing the resulting value of p. The *length* $|t|$ of a

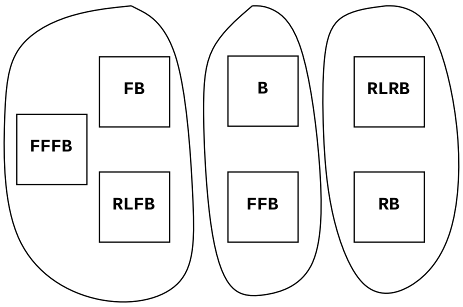

Figure A.1. Some Test Equivalence Classes for the 3-Bit Shift Register Environment.

test t is one plus the number of actions it contains. If the robot knows the values of all the tests in the current state q, then it can predict the sensations resulting from executing any action sequence. In that sense it has a *perfect model* of the environment in state q.

In the n-bit shift register environment, `B`, `FB`, `LRB`, `FFFFFB`, and `LLLLFB` are a few of the infinite number of tests. In this environment let 1 denote **true** and 0 denote **false**. For a 3-bit register in state $q = 011$, the values of qt for these tests are `B`$= 0$, `FB`$= 1$, `LRB`$= 0$, `FFFFFB`$= 1$, `LLLLFB`$= 0$.

In order to make a finite state representation, a finite number of tests must be chosen to serve as the state variables. This is done by grouping the infinite set of tests into equivalence classes. The equivalence between two tests is defined as follows:

Definition 3 (Equivalence)*:* $t_1 \equiv t_2 \Leftrightarrow (\forall q \in Q)(qt_1 = qt_2)$.

The equivalence class containing test t is denoted by $[t]$. The *canonical* member of a class is the shortest length test in that class and is denoted by $\langle t \rangle$. Figure A.1 shows some of the equivalence classes for the shift register environment. All the tests within a circle are equivalent to each other. Note that there are an infinite number of members within any class; only a few members of each class are shown in the figure.

Rivest and Schapire show that the canonical tests can serve as the internal state variables, such that the values of the canonical tests completely describe the state of the environment. An environment is assumed to be *reduced*, that is, an environment has the property that for any two states there is a test that will distinguish them:

Definition 4 (Reduced Environment)*:*

$$(\forall q_1 \in Q)(\forall q_2 \in Q)(q_1 \neq q_2 \Rightarrow (\exists t \in T)q_1 t \neq q_2 t). \tag{A.1}$$

For a reduced environment, two states are equal if and only if the values of all the canonical tests are the same in both states. For each state of an FSA there is a corresponding set of test values over the canonical tests. In this sense, these values form a distributed representation of the state.

Diversity can now be defined:

Definition 5 (Diversity)*: The diversity of the environment $\mathcal{E}$, denoted $D(\mathcal{E})$, is the number of equivalence classes of $\mathcal{E}$: $D(\mathcal{E}) = |\{[t] | t \in T\}|$.*

Rivest and Schapire show that $\log_2 |Q| \leq D(\mathcal{E}) \leq 2^{|Q|}$. They claim that for "natural" environments, the diversity can be much smaller than the number of environmental states. For instance, in the grid environment, they show that $D(\mathcal{E}) = 3(n^2 - 1)$, but $|Q| = 3^{n^2-1}$.

Figure A.2 shows the canonical members of the six equivalence classes for the 3-bit shift register environment. In the n-bit shift register environment, only $2n$ state variables are needed to represent the 2^n states. Figure A.3 shows the values of the canonical tests for the given state of the 3-bit register. It turns out that for the shift register environment, there are two state variables for each bit in the shift register, one for the actual value and one for the negated value of that bit. Clearly, this is not the best possible state representation, but it is far superior to explicitly representing all 2^n states.

Now that the state variables are defined, a method is needed to update their values and properly initialize them. First the canonical tests can be related to each other. Figure A.4 shows the canonical tests for the shift register environment and

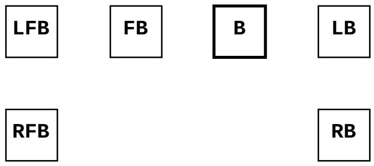

Figure A.2. The Diversity-Based State Variables for the 3-Bit Shift Register Environment.

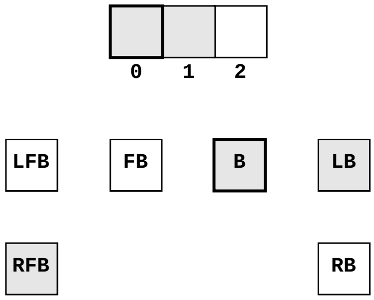

Figure A.3. The Values for the Diversity-Based State Variables for the Given State of the 3-Bit Shift Register. The 3-bit shift register is depicted above. A shaded box means the element contains a one and otherwise contains a zero.

also a few relationships between the tests as depicted by the arcs. These relationships are denoted with an arc $t_1 \xrightarrow{a} t_2$ which means that $t_1{\equiv}at_2$, where t_1 and t_2 are tests and a is an action. The simplest example is the $\texttt{LB} \xrightarrow{\texttt{L}} \texttt{B}$ arc, which means that the value of `LB` before executing `L` is the same as the value of `B` after executing `L`, or $\texttt{LB} \equiv \texttt{LB}$. The arc $\texttt{B} \xrightarrow{\texttt{R}} \texttt{LB}$ is a more complicated example and means that $\texttt{B} \equiv \texttt{RLB}$. These relationships provide a technique for updating the values of the canonical tests after executing actions. For example, if the value of test `LB` is known and the robot executes an `R`, then the value of `B` will be known.

A.2 The Update Graph

If an arc $t_1 \xrightarrow{a} t_2$ is known for each canonical test t_1 and each action a, then a machine can be built that maintains the correct values for the canonical tests. Rivest and Schapire call this machine the *update graph*:

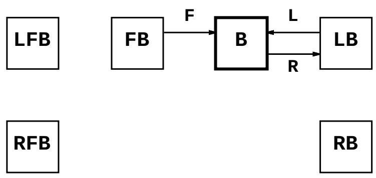

Figure A.4. Some Relationships Between Canonical Tests for the 3-Bit Shift Register Environment.

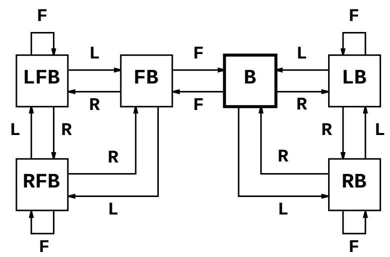

Figure A.5. The Update Graph for the 3-Bit Shift Register.

Definition 6 (Update Graph)*: An update graph is a directed graph in which each vertex represents an equivalence class, and an edge labeled $a \in A$ is directed from vertex $[t]$ to $[t']$ iff $t \equiv at'$.*

Figure A.5 shows the update graph for the shift register environment. The update graph has a number of useful properties. Since there is one vertex for each equivalence class, there are $D(\mathcal{E})$ vertices. For each vertex of the update graph, there is exactly one incoming edge for each action (because $(\forall a \in A) t \equiv t' \Rightarrow at \equiv at'$).

If the value of the test in the current state q is assigned to each vertex $[t]$, qt, in a complete update graph, then the value of any test after executing any action can be found. First, it is known that an edge $t_1 \xrightarrow{a} t_2$ exists iff $t_1 \equiv at_2$. With a complete update graph, each vertex has one incoming edge per action. With each execution of an action $a \in A$ the value of each vertex $[t]$ is updated synchronously with the value of the vertex $[at]$ at the tail of $[t]$'s incoming a-edge. This is correct because after executing a, $[t]$ will have the value of $[at]$ before executing a. Rivest and Schapire formally show that this accurately computes the value of each canonical test and that the class of machines represented by update graphs, namely *simple assignment automata*, are as powerful as FSA's.

In the register environment, shown in Figure A.5, it is easy to see how the update graph is updated. When an `R` action is executed, the values in the left half of the update graph are rotated counterclockwise, and in the right half clockwise; an `L` action rotates the values in the opposite direction. An `F` action causes the `FB` and `B` tests to exchange values, whereas all the other tests keep their old values. This updating process maintains the bits of the shift register in the right half of the update graph and the negated bits in the left half.

Assuming that the structure of the update graph is known, Rivest and Schapire provide a state estimation algorithm for inferring the values of the tests. Assign x_i to each canonical test, where x_i represents the value of the i'th canonical test, t_i, at the beginning of this process. In order to solve for the values of the canonical tests it is sufficient to solve for each x_i because all tests get their values only from other canonical tests. After the first action is executed x_i will move from t_i to t_k, and in general, a given t_i will have value x_j. The value of a test t_i with unknown value x_j can be solved for by executing the test t_i. This will serve to predict the value of sensation predicate p associated with test $t_i = sp$ with the unknown value x_j. The true value of x_j can then be found by reading the value of the sensation predicate p from the actual sensations produced by the environment. The procedure for inferring the values of all the tests in an update graph is: while $\exists$ a test t with an unknown value, execute t. The initial values of the tests can be solved for in $O(D(\mathcal{E})^2)$, because there are $D(\mathcal{E})$ unknown variables and each test is no longer than $D(\mathcal{E})$.

The update graph keeps track of the state of the FSA environment through tests. The tests code this information with respect to the actions available to the robot. The number of these canonical tests is the diversity of the environment. It is useful to add to the concepts borrowed from Rivest and Schapire the concept of *depth*:

Definition 7 (Depth)*: The depth of an environment is the length of the longest canonical test,* $\mathcal{H} = \max_{t \in T} |\langle t \rangle|$.

A *low depth* environment has the property that the depth is much smaller than the diversity:

Definition 8 (Low Depth)*:* $\mathcal{H} \leq c \log_{|A|} D(\mathcal{E})$*, where* c *is a fixed constant.*

A.3 The Rivest-Schapire Learning Algorithm

The basic idea of Rivest and Schapire's learning approach is to incrementally build the update graph one edge at a time adding new vertices as needed. The update graph starts out with the predicates as the only vertices and with their incoming edges undefined. Note that since all vertices will have exactly one incoming edge per action, the inference procedure can keep track of which edges are undefined and when the update graph is complete. The main loop of the inference procedure sequentially discovers the vertices at the tails of the undefined incoming edges. It can infer the vertex at the tail by running an equivalence procedure that compares the test defined at the tail with the current set of canonical tests. Let a test with an undefined incoming edge be called t, and let the incoming edge correspond to action a. The vertex at the tail of the incoming edge will be equivalent to at. If at is found to be equivalent to an already existing test t', then $[t'] \xrightarrow{a} [t]$ is added. Otherwise, a new vertex, $[at]$, is added, the edge $[at] \xrightarrow{a} [t]$ is added, and all $[at]$'s incoming edges are added to the undefined edge list. This continues until there are no more undefined incoming edges. If an oracle is available to decide whether two tests are equivalent, then the running time of this algorithm is $O(2|A|D(\mathcal{E})^2)$, where A is the set of actions and $D(\mathcal{E})$ is the diversity. The pseudocode for this inference procedure is given in Table A.1.

Suppose we want to learn a model for the 3-bit shift register. We start with Vertices = $\{$`B(it?)`$\}$, the only sensory predicate. Because there are three actions, `L`,

Table A.1. Diversity-Based Inference Procedure.

Vertices $\leftarrow$ Sensations
`while` $\exists$ a vertex $[t]$ missing incoming edge a
  `if` $at \equiv t'$ for some vertex $[t']$
  `then` add $[t'] \xrightarrow{a} [t]$
  `else`
    create new vertex $[at]$ and add to Vertices
    add $[at] \xrightarrow{a} [t]$
  `end else`
`end while`

`R`, and `F`, there are three undefined incoming arcs for the `B` vertex. By definition, the vertices at the tails of those arcs are `LB`, `RB`, and `FB` shown in Panel A of Figure A.6, where we depict the proposed vertices as shaded and the known vertices as white. The equivalence oracle decides that these proposed tests are inequivalent to the canonical test `B` as well as to each other, and thus these proposed canonical tests are added to the list of canonical tests. This is shown in Panel B of Figure A.6 along with all newly proposed vertices corresponding to undefined incoming arcs. At this point the set of vertices is {`B`, `LB`, `RB`, `FB`} and all the incoming arcs on `LB`, `RB`, and `FB` are undefined. Examining the proposed `FFB` node in Panel B of Figure A.6, we see that it is equivalent to a canonical test already defined, namely `B`. Therefore this proposed node, `FFB`, is redundant and when a `F` action is taken, `FB` can get its value from `B` instead (of `FFB`) as shown in Panel C of Figure A.6. Continuing in this fashion, the rest of the vertices and arcs are inferred and the final update graph is shown in Panel D of Figure A.6.

In order to make the inference procedure polynomial in diversity, one needs a polynomial time equivalence procedure. Unfortunately, in order to know whether two tests are equivalent one must compare their values when executed in all environmental

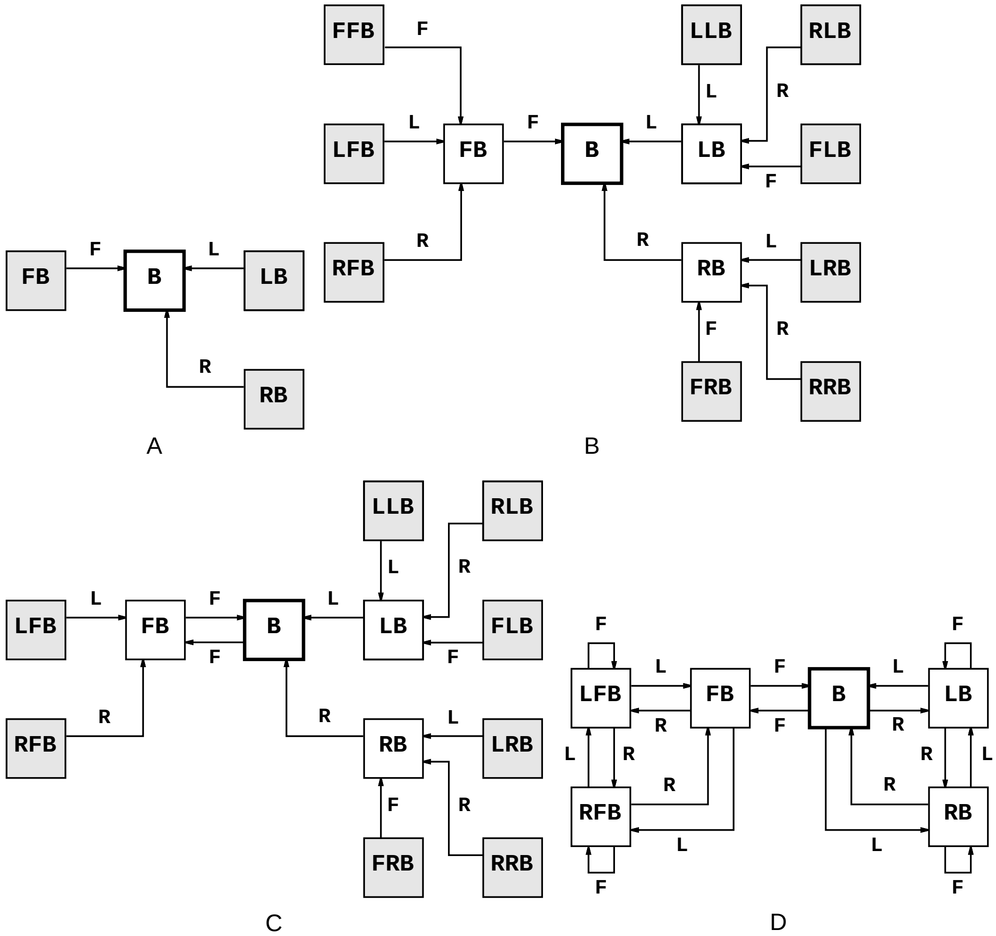

Figure A.6. A Series of Steps in the Diversity-Based Inference Process of the Update Graph for the 3-Bit Shift Register Environment.

Table A.2. Probabilistic Procedure for Determining $t \equiv t'$ Given Tolerable Probability of Error ϵ.

```
Repeat k times, where k is chosen so as to make
        the probability of an error less than ε
    Get into a random environmental state q
    Perform test t
    Get back into same environmental state q
    Perform test t'
    If qt ≠ qt' then return false
return true
```

states. This would be acceptable if one were interested in an equivalence procedure that is polynomial in the number of FSA states. The best that can be hoped for is an equivalence procedure that samples the global states and determines either: (1) that two tests are equivalent with a given confidence or (2) are inequivalent if a counterexample is found (since if we can find just one counterexample where they disagree, then the two tests are inequivalent). Rivest and Schapire's probabilistic equivalence procedure for asking $t \equiv t'$ is summarized in Table A.2. This is one of their suggested heuristic methods; they showed that other more complicated techniques are provably polynomial time.

Rivest and Schapire cast the equivalence problem in terms of finding counterexamples and state that there are two problems to face. The first is the problem of *accessibility of counterexamples*: it might be very difficult to get into an environmental state where the two tests disagree. The second is the problem of *irreversibility of actions*: once the first test is executed in state q, it is not always possible to go back to state q (i.e., undo the effects of an action) in order to execute the second test.

It turns out that for permutation environments, it can be proved that these problems can be overcome. In a *permutation environment* there exists a fixed inverse for any test (see Schapire [85] for more details). In permutation environments a counterexample (if one exists) has a high probability of occurring during a long enough random walk (where the actions are chosen from a uniform distribution), and two tests can always be compared without undoing actions. Rivest and Schapire describe an equivalence procedure that they prove to be polynomial time on permutation environments. The result requires very long random walks that require running time on the order of $|A|D^8 \log(|A|\frac{D^2}{\epsilon})$, where the parameter ϵ is the tolerable probability of error. In practice, however, they have found that much shorter random walks suffice. They also describe a heuristic algorithm that is reasonably effective for nonpermutation environments [73, 85].