Appendix B

Prefix based representation of computation sets

This appendix gives formal definition of prefix based representation of computation sets, discusses its properties, and gives algorithms for computing set intersection and backtracing of computation sets represented as lists of prefixes.

B.1 Prefix based representation of computation sets

A prefix based representation of a computation set is a list

$$L_X = (x_0, \ldots, x_{|L_X|-1}) \tag{B.1}$$

whose every element is a non-empty computation: $x_i \in C_T$, and $|x_i| > 0$.
The elements of L_X are called *prefixes*.
The meaning of L_X is the set of all computations whose prefixes are

contained in L_X:

$$X = \bigcup_{i=0}^{|L_X|-1} \{\, c \,|\, c \in C_T,\ |x_i| \leq |c|,\ \text{and for all integer } 0 \leq j < |x_i| \ :\ c_j = (x_i)_j \,\} \tag{B.2}$$

B.2 Basic properties of prefix lists

The following properties follow directly form the equation B.2 and properties of lists

- $c \in X$ if and only if there exists x_i such that for all integer $0 \leq j < |x_i|$, $c_j = (x_i)_j$
- $X = \emptyset$ if and only if $L_X = \epsilon$. As a result, set emptiness of X can be checked in constant time by checking emptiness of L_X.
- Let X and Y be computation sets represented by lists L_X and L_Y respectively, then concatenation $L_X \cdot L_Y$ represents $X \cup Y$.
- Let $a \in C_T$ and $b \in C_T$ be two prefixes, and let A and B be the sets of computations represented by (a) and (b) respectively. Observe that
  - $A \subseteq B$ if b is a prefix of a
  - $b \subseteq A$ if a is a prefix of b
  - A and B have no common elements if neither b is a prefix of a, nor a is a prefix of b.

As a result,

$$A \cap B = \begin{cases} A & \text{if } b \text{ is a prefix of } a \\ B & \text{if } a \text{ is a prefix of } b \\ \emptyset & \text{otherwise} \end{cases}$$

Representation of C_T With prefix-based representation, the set C_T can be represented by the list

$$L_{C_T} = (((q_0, \iota_0)) \ldots ((q_{|Q\times I|}, \iota_{|Q\times I|})))$$

where $q_i \in Q$ is a state, $\iota_i \in I$ is an event, and there is an element $((q_i, \iota_i))$ for every possible combination of state and event. Since every computation begins with some state–event pair, it has to be represented by one of the elements in L_{C_T}.

Clearly, C_T has more than one possible representation. Instead of using the list of all singleton prefixes $((q, \iota))$ it is possible to use a list that contains all prefixes of length 2, 3, or any fixed length. The number of elements in such lists will increase exponentially with the length of prefixes. Observe that the list of all possible computation prefixes of length m consists of $|Q||I|^m$ elements, because transition function identifies $|I|$ possible next states for every possible state of the machine.

Representation of a set of computations with restricted prefixes Consider a set of computations P that restricts only the first m elements of its member computations:

$$P_m = \{\, c \,|\, c \in C_T \text{ where } c_0, \ldots, c_{m-1} \text{ satisfy condition } f(c_0, \ldots, c_{m-1})\}$$

A prefix-based representation L_{P_m} of the set P_m can be constructed by listing all prefixes $(c_0, \ldots, c_{m-1})$ that satisfy $f(c_0, \ldots, c_{m-1})$. Since $P_m \subseteq C_T$, the number of elements in L_{P_m} is less or equal than the number of elements in the representation of C_T with prefixes of length m:

$$|L_{P_m}| \leq |Q||I|^m$$

```
1: function IntersectPrefixes(x, y)
2:     for i ← 0 to min(|x|, |y|) step 1 do
3:         if x_i ≠ y_i then
4:             return ϵ
5:         end if
6:     end for
7:     if |x| > |y| then
8:         return x                                  ▷ because x ⊂ y
9:     else
10:        return y                                  ▷ because y ⊆ x
11:    end if
12: end function
```

Figure B.1: Algorithm for computing *IntersectPrefixes*(x, y)

Set intersection. Observe that, by distributivity of $\cap$ over $\cup$ and by the equation B.2, the intersection of two sets X and Y represented by L_X and L_Y can be computed as a union of pair-wise intersections of sets represented by the elements of L_X and L_Y.

Function *IntersectPrefixes*(x, y) computes a prefix that represents the intersection of sets represented by its argument prefixes x and y. The algorithm for computing it is given in Figure B.1. Assuming that all operations in *IntersectPrefixes*(x, y) are constant time operations, the worst case running time of *IntersectPrefixes*(x, y) algorithm is $O(min(|x|, |y|)$, because the loop in Figure B.1 iterates at most $min(|x|, |y|)$ times.

The algorithm for computing intersection of sets represented by lists L_X and L_Y is given in Figure B.2. The running time of this algorithm is proportional to the lengths of both lists L_X and L_Y and the running time of *IntersectPrefixes*(x, y). Since the running time of *IntersectPrefixes*(x, y) is bounded by $O(min(x, y))$, the running time of $\bigcap(L_X, L_Y)$ is bounded by $O(|L_X||L_Y|m)$, where $m = max(min(x_i, y_j)))$ for all $0 \leq i < |L_X|$, and $0 \leq j < |L_Y|$. Clearly, m is less or equal than the length of the longest prefix in both L_X and L_Y.

```
 1: function ⋂(L_X, L_Y)
 2:     result ← ∅
 3:     for each prefix x in L_X do
 4:         for each prefix y in L_Y do
 5:             z ← IntersectPrefixes(x, y)
 6:             if z is not empty then
 7:                 result ← result · (z)
 8:             end if
 9:         end for
10:     end for
11: end function
```

Figure B.2: Algorithm for computing $X \cap Y$

```
 1: function Ψ^{-1}(L_X)
 2:     result ← ∅
 3:     for each pattern x in L_X do
 4:         q ← the first state in x
 5:         for each state p and event ι, such that δ(p, ι) = q do
 6:             Create new prefix x' ← ( (p, ι) ) · x
 7:             result ← result · (x')
 8:         end for
 9:     end for
10:     return result
11: end function
```

Figure B.3: Algorithm for computing $\Psi^{-1}(X)$

Computing $\Psi^{-1}(X)$. The algorithms for computing $\Psi^{-1}(X)$ using prefix-based representation of computation sets is given in Figure B.3. It is a direct implementation of definition of $\Psi^{-1}(X)$ given in Chapter 6. Assuming that all operations in the algorithm take constant time[1], both the running time and the number of created prefixes are $O(|Q||I||L_X|)$, because the outer loop iterates $|L_X|$ times, the inner loop iterates $|Q||I|$ times, and each iteration of the inner loop produces at most one element of the result.

[1] Observe that $\delta(q, \iota)$ can be implemented as a table lookup.