Chapter 7

Event reconstruction algorithm

The existing algorithms for analysis of finite state models of computing systems were developed for the purpose of systems verification [28]. The event reconstruction problem formulated in Chapter 6 is different from verification problems. Instead of *checking* that certain type of computations is impossible in the system, event reconstruction aims to *construct* possible computations that explain available evidence. As shown in Chapter 5, verification algorithms avoid construction of computations for efficiency reasons. They cannot be used directly to solve the event reconstruction problem.

This chapter describes an algorithm for solving the event reconstruction problem. It computes the meaning of the given evidential statement with respect to the given finite state machine. Since no such algorithm previously existed, it was decided to construct a simple algorithm, whose properties can be easily analysed. Performance improvement of the algorithm is left for future work.

The algorithm is described in three steps. First, a procedure for computing the meaning of fixed-length observation sequences is presented. Second, a procedure for computing the meaning of generic observation sequences is presented. Third, it is shown how the meanings of individual observation sequences can be combined into the meaning of the evidential statement. An

upper bound on the running time of the event reconstruction algorithm is then derived in Section 7.4, and a "proof-of-concept" implementation of the algorithm is described in Section 7.5.

7.1 Computing the meaning of a fixed-length observation sequence

Recall function Ψ^{-1} introduced in Section 6.2.4. It takes a set of computations $Y \subseteq C_T$ and produces the set of all computations, whose tails are in Y. In other words, it returns all possible backtracings of computations in Y.

Function Ψ^{-1} provides basic operation for automation of backtracing. Together with set intersection, it can be used to calculate the meaning of observation sequences that consist of fixed-length observations only. The idea is to take the set of all computations C_T as the starting point and iteratively backtrace it into the past using Ψ^{-1}. At each step, computations that do not possess observed property are discarded. This is achieved by intersecting the set of backtracings with the set of computations that possess property observed at the current step. The result of intersection is then used as input for the next invocation of Ψ^{-1}, and so on. The process continues until either all observations are explained, or the set of computations becomes empty. Please look at Figure 7.1, which illustrates this process for observation sequence

$$example \;=\; (\,(A,\,3,\,0),\,(B,\,2,\,0)\,)$$

If the set of computations produced at the last step of reconstruction is non-empty, its elements satisfy observation sequence *example* by construction. The set of partitioned runs $PR_{example}$ that explain *example* can be generated from these computations using function ψ and the *fixed* length of observations in *example*.

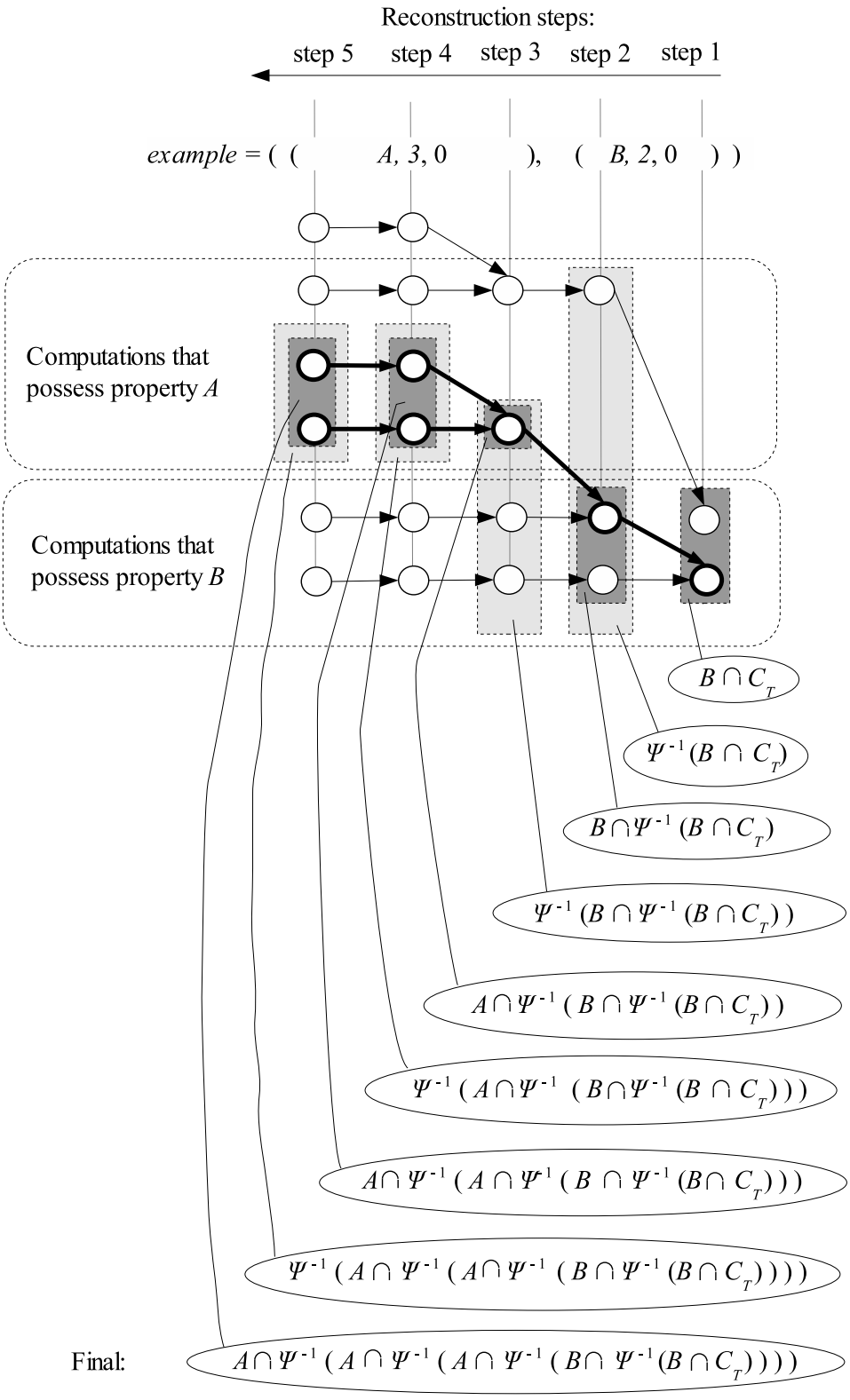

Figure 7.1: Finding explanations of a fixed-length observation sequence

```
1: function SolveFOS(fos)
2:     C_next ← C_T
3:     len ← ϵ
4:     for i ← (|fos| − 1) to 0 step -1 do
5:         observation (P, l, 0) ← fos_i
6:         len ← (l) · len
7:         for j ← 0 to l − 1 step 1 do
8:             C ← C_next ∩ P
9:             C_next ← Ψ^{-1}(C)
10:        end for
11:    end for
12:    return (C, len)
13: end function
```

Figure 7.2: Computing the meaning of a fixed-length observation sequence

A *map of partitioned runs (MPR)* is a representation for a set of partitioned runs. It is a tuple $pm = (C, len)$ where C is the set of initial computations, and len is a sequence of partition lengths. A single MPR represents the set of all partitioned runs whose initial computation is in C, and whose partitions have lengths $len_0, len_1, \ldots len_{|len|-1}$. Observe that the meaning of a fixed length observation sequence can be expressed by a single MPR.

The algorithm for computing the meaning of the given fixed-length observation sequence is presented in Figure 7.2. It implements the idea described above and returns an MPR that expresses the meaning of the given fixed-length observation sequence.

7.2 Computing the meaning of a generic observation sequence

The reconstruction process described above works, because the property observed at every step is known. This is because the length of run satisfying a fixed-length observation is equal to the observation's *min* parameter. For a generic observation $o = (P, min, opt)$, whose $opt \neq 0$, the length of explain-

ing run is not fixed, but is bounded between min and $min + opt$. As a result, single observation sequence represents many variants of linking observed properties to reconstruction steps. Consider, for example, observation sequence $example2 = (\,(A,\,1,\,3),\,(B,\,1,\,2)\,)$, which says that

- initially, property A was observed for at least 1 and at most 4 steps,
- then property B was observed for at least 1 and at most 3 steps.

This observation sequence represents twelve possible variants of linking properties to reconstruction steps:

$$\begin{array}{rrr} AB & ABB & ABBB \\ AAB & AABB & AABBB \\ AAAB & AAABB & AAABBB \\ AAAAB & AAAABB & AAAABBB \end{array}$$

Every one of these variants can be represented by a fixed-length observation sequence. Note that the meaning of *example2* is the union of explanations of each variant. Thus, the meaning of *example2* can be calculated in three steps:

1. Convert *example2* to a set of fixed-length observation sequences.
2. Calculate the meaning of each fixed-length observation sequence as described above.
3. Calculate the union of explanations of the fixed-length observation sequences.

Observe that the meaning of *example2* can be represented as a set of MPRs — each MPR representing the meaning of one of the fixed-length observation sequences.

The algorithm for computing the meaning of a generic observation sequence is given in Figure 7.3. The algorithm consists of two parts. First, lines 2–13 convert the given observation sequence *os* into a set of fixed-length observation

```
 1: function SolveOS(os)
 2:     F ← {ϵ}
 3:     for i ← 0 to |os| − 1 step 1 do
 4:         observation (P, min, opt) ← os_i
 5:         F_new ← ∅
 6:         for each partially constructed sequence f in F do
 7:             for j ← 0 to opt step 1 do
 8:                 f_new ← f · ( (P, min + j, 0) )
 9:                 F_new ← F_new ∪ {f_new}
10:             end for
11:         end for
12:         F ← F_new
13:     end for
14:     PM_os ← ∅
15:     for each fos in F do
16:         pm ← SolveFOS(fos)
17:         PM_os ← PM_os ∪ {pm}
18:     end for
19:     return PM_os
20: end function
```

Figure 7.3: Computing the meaning of a generic observation sequence

sequences F. After that, lines 14–18 use *SolveFOS* algorithm to compute the meaning of each fixed-length observation sequence in F. The resulting set of MPRs is returned in line 19.

7.3 Computing the meaning of an evidential statement

The meaning of an evidential statement can be computed using a two-step procedure. First, the meanings of individual observation sequences are computed as described in the previous sections. Then the meanings of observation sequences are combined into the meaning of the entire evidential statement.

To combine the meanings of observation sequences, note that, to satisfy the evidential statement, a run must satisfy all of its observation sequences.

Thus, the problem is to identify the subset of runs, whose partitionings are present in the meanings of all observation sequences.

Let $pm_a = (len_a, C_a)$ and $pm_b = (len_b, C_b)$ be two MPRs. A run r can be partitioned by both pm_a and pm_b if and only if two conditions hold:

1. the initial computation of run r belongs to the initial computation sets of both MPRs: $r \in C_a$ and $r \in C_b$, and

2. both MPRs have equal total number of computation steps: $\Sigma\, len_a = \Sigma\, len_b$.

Clearly, if $\Sigma\, len_a \neq \Sigma\, len_b$, then the lengths of all computations represented by pm_a are different from the lengths of all computations represented by pm_b, and two MPRs have no common runs. Otherwise, the common runs are determined by the common set of initial computations $C_a \cap C_b$.

A *map of sequence of partitioned runs (MSPR)* $spm = (C, (len_0, \ldots, len_n))$ is a representation for a set of sequences of partitioned runs. C is the set of initial computations, and $len_0, \ldots\ len_n$ are lists of lengths that describe how to partition runs generated from the elements of C. MSPR is *proper* if and only if $\Sigma\, len_0 = \ldots = \Sigma\, len_n$.

The combination of two MPRs is defined by function *comb* that takes two MPRs and returns a proper MSPR:

$$comb(pm_a, pm_b) = \begin{cases} \emptyset & \text{, if } \Sigma\, len_a \neq \Sigma\, len_b \text{ or} \\ & \quad C_a \cap C_b = \emptyset \\ (\, C_a \cap C_b, (len_a, len_b), \,) & \text{, otherwise} \end{cases}$$

Suppose that the meanings of two observation sequences os_a and os_b are represented by two sets of MPRs called PM_a and PM_b respectively. The meaning of evidential statement $es = (os_a, os_b)$ is expressed by the set of proper MSPRs, which is obtained by combining every MPR from PM_a with every MPR from

```
 1: function SolveES(es)
 2:     SPM_es ← ∅
 3:     os ← es_0
 4:     PM_os ← SolveOS(os)
 5:     for each pm = (C, len) in PM_os do
 6:         SPM_es ← SPM_es ∪ {(C, (len))}
 7:     end for
 8:     for i ← 1 to |es| − 1 step 1 do
 9:         os ← es_i
10:         PM_os ← SolveOS(os)
11:         SPM_new ← ∅
12:         for each spm = (C_a, lenlist) in SPM_es do
13:             for each pm = (C_b, len) in PM_os do
14:                 C ← C_a ∩ C_b
15:                 if C ≠ ∅ and Σlen = Σlenlist_0 then
16:                     SPM_new ← SPM_new ∪ { (C, lenlist · (len_a) ) }
17:                 end if
18:             end for
19:         end for
20:         SPM_es ← SPM_new
21:     end for
22:     return SPM_es
23: end function
```

Figure 7.4: Computing the meaning of an evidential statement

PM_b :

$$\text{for all } x \in PM_a, \text{ for all } y \in PM_b, \ SPM_{es} = \cup\, comb(x, y)$$

This process can be extended to arbitrary number of observation sequences, thus providing a way to calculate meaning of an arbitrary evidential statement. The corresponding event reconstruction algorithm is given in Figure 7.4. The operation of the algorithm is divided into two parts. First, lines 2–7 compute the meaning of the first observation sequence in the evidential statement and make it the initial value for SPM_{es}. After that, the loop in lines 8–21 computes the meaning of the remaining observation sequences in the evidential statement and combines their meanings with SPM_{es}.

7.4 Running time of event reconstruction algorithm

This section derives an upper bound[1] on the running time of the event reconstruction algorithm *SolveES* described in the previous section.

7.4.1 Prefix based representation of computation sets

The running time of the event reconstruction algorithm cannot be estimated without first estimating the running time required for its basic operations $\Psi^{-1}(X)$ and $X \cap Y$. Their running times, in turn, depend on the chosen representation of computation sets.

For the purpose of this analysis, a *prefix based* representation of computation sets has been adopted. A prefix based representation of a computation set is a list

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

where $x_i \in C_T$, and $|x_i| > 0$.

The elements of L_X are called *prefixes*. Each prefix represents the set of all computations that begin with it. For example, the list of prefixes $(\,(a,b,c),\,(d,e)\,)$ represents the set of all computations of the forms $(a,b,c,\ldots)$ or $(d,e,\ldots)$. [2] This set includes, for example, computations (a,b,c), (a,b,c,d), (a,b,c,e,d), (d,e), (d,e,a). More precisely, the set of computations repre-

[1] Note that O-notation is not used in this section. This is because the running time of the event reconstruction algorithm depends on six parameters, and (since it is not clear which parameter grows faster) the use of O-notation does not bring much clarity into the resulting expression.

[2] Symbols a, b, c, d, and e stand for state–event pairs of the form (q, ι), where $q \in Q$ and $\iota \in I$.

sented by a list of prefixes L_X is

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

Appendix B discusses properties of the prefix based representation and gives algorithms for computing $\Psi^{-1}(X)$ and $X \cap Y$ of sets of computations represented as lists of prefixes. To derive an upper bound on the running time of the event reconstruction algorithm, note that the prefix based representations have the following properties.

Prefix based representation of C_T. The set C_T can be represented as a list of all possible singleton prefixes L_{C_T}, whose length is

$$|L_{C_T}| = |Q||I| \tag{7.1}$$

where $|Q|$ and $|I|$ are sizes of sets of states and events respectively.

Size of prefix based representations with limited lengths of prefixes. Any set of computations P_m that restricts only the first m elements of its member computations can be represented by a list L_{P_m}, whose length is bounded by the number of all possible computation prefixes of length m:

$$|L_{P_m}| \leq |Q||I|^m \tag{7.2}$$

except for $m = 0$.

Checking set emptiness is a constant time operation. Checking emptiness of a set of computations C represented as a list of prefixes L_C amounts to checking emptiness of L_C, which can be performed in constant time.

Upper bound on the time required to compute set intersection. The time $t_{X \cap Y}$ required to compute the set intersection of two computation sets X and Y represented as lists of prefixes L_X and L_Y is bounded by

$$t_{X \cap Y} \leq cm|L_X||L_Y| \tag{7.3}$$

where c is an implementation-dependent constant, and m is the length of the longest prefix in either L_X or L_Y.

Upper bound on the length of the output of set intersection. The length $|L_{X \cap Y}|$ of the list of prefixes that represents the set intersection of two computation sets X and Y is bounded by

$$|L_{X \cap Y}| \leq |L_X||L_Y| \tag{7.4}$$

where $|L_X|$ and $|L_Y|$ are lengths of lists of prefixes that represent sets X and Y respectively.

Upper bound on the time required to compute $\Psi^{-1}(X)$. The time required to compute $\Psi^{-1}(X)$ is bounded by

$$t_{\Psi^{-1}(X)} \leq c|Q||I||L_X| \tag{7.5}$$

where c is an implementation dependent constant, $|L_X|$ is the length of the list of prefixes that represents the set X, and $|Q|$ and $|I|$ are sizes of sets of states and events respectively.

Upper bound on the length of the output of $\Psi^{-1}(X)$. The length $|L_{\Psi^{-1}(X)}|$ of the list of prefixes that represents the output of $\Psi^{-1}(X)$ is bounded by

$$|L_{\Psi^{-1}(X)}| \leq |Q||I||L_X| \tag{7.6}$$

where $|L_X|$ is the length of the list of prefixes that represents the set X, and $|Q|$ and $|I|$ are sizes of sets of states and events respectively.

7.4.2 An upper bound on the running time of *SolveFOS*

First, note that operations in lines 1–7 and 10–12 of the *SolveFOS* algorithm can be implemented to take constant time. The running times of lines 8 and 9 are bounded by inequalities 7.3 and 7.5 respectively.

Let l_{max} be an upper bound on the lengths of observations in the input observation sequence *fos*

$$\text{for all } 0 \leq i < |fos|, \quad l_i \leq l_{max} \tag{7.7}$$

where l_i is the length of the i^{th} observation in *fos*.

Let p be an upper bound on the length of prefixes used in prefix based representations of all observed properties in *fos*. Then, according to the inequality 7.2, the lengths of prefix based representations of observed properties are bounded by

$$\text{for all } 0 \leq i < |fos|, \quad |L_{P_i}| \leq |Q||I|^p \tag{7.8}$$

where L_{P_i} is a prefix based representation of the observed property P_i of the i^{th} observation in *fos*.

The running time of *SolveFOS* is equal to the following sum

$$t_{SolveFOS} = c_1 + \sum_{i=0}^{|fos|-1} \Big(c_2 + \sum_{j=0}^{l_i-1} \big(c_3 + t_{backtr_k} \big) \Big) \tag{7.9}$$

where

- c_1 represents the constant time spent in lines 1–3, 12, and in loop setup in line 4,

- c_2 represents the constant time spent in every iteration of the outer loop in lines 4–6, 11, and in loop setup in line 7,
- c_3 represents the constant time spent in every iteration of the inner loop in lines 7 and 10,
- t_{backtr_k} represents the time of the k^{th} reconstruction step performed in lines 8 and 9, where $k = j + \sum_{n=i+1}^{|fos|-1} l_n$, where l_n is the length of the n^{th} observation in fos.

The running time of the k^{th} reconstruction step is the sum of the running times of set intersection and backtracing. By monotonicity of addition and by inequalities 7.3 and 7.5, it is bounded by

$$t_{backtr_k} \leq c_4 p |L_{C_{next_k}}||L_{P_i}| + c_5 |Q||I||L_{(C_{next_k} \cap P_i)}| \tag{7.10}$$

where

- C_{next_k} is the "input" set of computations at the k^{th} reconstruction step,
- l_i is the length of observation fos_i,
- P_i is the observed property of observation fos_i,
- c_4 and c_5 are implementation dependent constants

Observe that the right-hand side of inequality 7.10 depends on the lengths of representations $L_{C_{next_k}}$ and $L_{(C_{next_k} \cap P_i)}$. To compute these values, note that $C_{next_{k+1}}$ is linked with C_{next_k} according to the following recurrence

$$\begin{aligned} C_{next_0} &= C_T \\ C_{next_{k+1}} &= \Psi^{-1}(C_{next_k} \cap P_i) \end{aligned}$$

An upper bound on the length of $L_{C_{next_{k+1}}}$ can be derived from this recurrence

using equation 7.1 and inequalities 7.4, and 7.6:

$$|L_{C_{next_0}}| = |L_{C_T}| = |Q||I|$$
$$|L_{C_{next_{k+1}}}| \leq |Q||I|(|L_{C_{next_k}}||Q||I|^p)$$

Solving this recurrence for k produces

$$|L_{C_{next_k}}| \leq |Q||I|(|Q|^2|I|^{p+1})^k = |Q|^{2k+1}|I|^{kp+k+1} \tag{7.11}$$

After substituting this result into the inequality 7.10, replacing $|L_{P_i}|$ and $|L_{(C_{next_k} \cap P_i)}|$ according to inequalities 7.8 and 7.4, and simplifying, the upper bound on the time of the k^{th} reconstruction step becomes

$$t_{backtr_k} \leq (c_4 p + c_5|Q||I|)(|Q|^{2(k+1)}|I|^{(k+1)(p+1)}) \tag{7.12}$$

An upper bound on the running time of *SolveFOS* can be obtained from the right-hand side of equation 7.9 by replacing t_{backtr_k} and l_i according to inequalities 7.12 and 7.7. After simplification, it becomes

$$t_{SolveFOS} \leq \Lambda + (c_4 p + c_5|Q||I|)\left(\frac{(|Q|^2|I|^{p+1})^{|fos|l_{max}+1} - (|Q|^2|I|^{p+1})}{(|Q|^2|I|^{p+1}) - 1}\right) \tag{7.13}$$

where

$$\Lambda = c_1 + c_2\,|fos| + c_3\,|fos|\,l_{max}$$

An upper bound on the length of the output of *SolveFOS* can be derived from the upper bound on the length of $L_{C_{next_k}}$ at the last step of event reconstruction using inequalities 7.11 and 7.4. After simplification, the resulting upper bound is

$$|L_{SolveFOS}| \leq (|Q|^2|I|^{p+1})^{|fos|l_{max}} \tag{7.14}$$

7.4.3 An upper bound on the running time of *SolveOS*

First, note that operations in lines 1–15 and 17–19 of the *SolveOS* algorithm can be implemented to take constant time. The execution of the line 16 requires time, whose upper bound is given by inequality 7.13.

Second, note that by definition of *infinitum* given in Chapter 6, the length of any computation that may have happened during the incident is bounded by *infinitum*. As a result, the length of any observation made during the incident is also bounded by *infinitum*, and

$$\text{for all } 0 \leq i < |os| \quad (min_i + opt_i) \leq infinitum \tag{7.15}$$

where min_i and opt_i are *min* and *opt* parameters of the i^{th} observation in the input observation sequence *os*.

The running time of *SolveOS* is equal to the following sum

$$t_{SolveOS} = c_6 + \sum_{i=0}^{|os|-1} (c_7 + \sum_{l=0}^{|F_i|-1} (c_8 + \sum_{j=0}^{opt_i} c_9)) + \sum_{l=0}^{|F_{|os|}|-1} (c_{10} + t_{SolveFOS}) \tag{7.16}$$

where

- c_6 represents the constant time spent in lines 1–2, 14, and 19, and in loop setups in lines 3 and 15,
- c_7 represents the constant time spent in every iteration (of the loop in lines 3–13) in lines 3–5, 12–13, and in loop setup in line 6
- c_8 represents the constant time spent in every iteration (of the loop in lines 6–11) in lines 6, and 11, and in loop setup in line 7,
- c_9 represents the constant time spent in every iteration of the inner loop in lines 7–10,
- F_i is the set of partially constructed fixed-length observation sequences at the beginning of the i^{th} iteration of the loop in lines 3–13,

- $F_{|os|}$ is the final set of fixed-length observation sequences produced by the $(|os|-1)^{\text{th}}$ iteration of the loop in lines 3–13,
- c_{10} represents the constant time spent in every iteration (of the loop in lines 15–18) in lines 15, 17, and 18.

The time of the i^{th} iteration of the outer loop in lines 3–13 depends on the size of F_i generated in the previous iteration. To obtain an upper bound on the size of F_i note that every iteration of the inner loop in lines 7–10 creates one new element for F_{i+1}, and that opt_i is bounded by the inequality 7.15. Thus, the size of F_i is bounded by the following recurrence

$$\begin{aligned} |F_0| &= 1 \\ |F_{i+1}| &\leq |F_i| \mathit{infinitum} \end{aligned}$$

Solving this recurrence for i produces

$$|F_i| \leq \mathit{infinitum}^i \tag{7.17}$$

An upper bound on $t_{SolveOS}$ can be derived from 7.16 by replacing $|F_i|$ and opt_i using inequalities 7.17 and 7.15. The resulting inequality after simplification becomes

$$t_{SolveOS} \leq c_6 + c_7\,|os| + (c_8 + c_9\,\mathit{infinitum})\left(\frac{\mathit{infinitum}^{|os|+1} - 1}{\mathit{infinitum} - 1}\right) + \sum_{l=0}^{\mathit{infinitum}^{|os|}} (c_{10} + t_{SolveFOS}) \tag{7.18}$$

Note that the length of any fixed-length observation sequence *fos* in $F_{|os|}$ is the same as the length of the original observation sequence:

$$|\mathit{fos}| = |os|$$

and that the length l of any observation in *fos* is bounded by inequality 7.15.

As a result, $t_{SolveFOS}$ in the right hand side of 7.18 can be replaced with the right hand side of inequality 7.13 with $|fos|$ replaced by $|os|$ and l_{max} replaced by *infinitum*. After simplification the resulting inequality becomes

$$t_{SolveOS} \leq c_6 + c_7 \left|os\right| + (c_8 + c_9 \left(infinitum + 1\right)) \left(\frac{infinitum^{|os|} - 1}{infinitum - 1} \right) + c_{10}\, infinitum^{|os|} + \Upsilon \quad (7.19)$$

where Υ represents the time spent on computing the meanings of individual fixed-length observation sequences:

$$\Upsilon = infinitum^{|os|} \left(c_1 + c_2 \left|os\right| + c_3 \left|os\right| infinitum + (c_4 p + c_5 |Q||I|) \left(\frac{(|Q|^2|I|^{p+1})^{|os|\, infinitum + 1} - (|Q|^2|I|^{p+1})}{(|Q|^2|I|^{p+1}) - 1} \right) \right)$$

An upper bound on the number of elements in the output of *SolveOS* can be derived as follows. Note that each iteration of the loop in lines 15–18 creates one element of the output set. Thus, by inequality 7.17, the number of elements in the output of *SolveOS* is bounded by

$$|SolveOS(os)| \leq infinitum^{|os|} \quad (7.20)$$

7.4.4 An upper bound on the running time of *SolveES*

First, note that operations in lines 1–3, 5–9, 11–13, and 15–22 of the *SolveES* algorithm can be implemented to take constant time. The running times of lines 4 and 10 are bounded by inequality 7.19. The running time of line 14 is bounded by inequality 7.3.

Let os_{max} be the upper bound on the lengths of observation sequences in

the input evidential statement es

$$\text{for all } 0 \leq i < |es|, \quad |es_i| \leq os_{max} \tag{7.21}$$

where es_i is the i^{th} observation sequence in es.

The running time of the event reconstruction algorithm *SolveES* depends on the number of observation sequences in the evidential statement es. For the purpose of the worst case analysis, it suffices to consider only evidential statements that consist of two or more observation sequences (i.e. $2 \leq |es|$), in which case the running time of *SolveES* is bounded by the following sum

$$t_{SolveES} \leq c_{11} + t_{SolveOS(es_0)} + \sum_{i=0}^{|SolveOS(es_0)|-1} c_{12} + \sum_{i=1}^{|es|-1} \Big(c_{13} + t_{SolveOS(es_i)} + \sum_{j=0}^{|SPM_i|-1} \big(c_{14} + \sum_{l=0}^{|SolveOS(es_i)|-1} (\ c_{15} + t_{(C_{a_{j_i}} \cap C_{b_l})} \ ) \big) \Big) \tag{7.22}$$

where

- c_{11} represents the constant time spent in lines 1–3, 22, and in loop setups in lines 5 and 8,
- c_{12} represents the constant time spent in every iteration of loop in lines 5–7,
- c_{13} represents the constant time spent in every iteration (of the loop in lines 8–21) in lines 8–9, 11, 20–21, and in loop setup in line 12,
- c_{14} represents the constant time spent in every iteration (of the loop in lines 12–19) in lines 12, 19, and in loop setup in line 13,
- c_{15} represents the constant time spent in every iteration (of the loop in lines 13–18) in lines 13, and 15–18,

- SPM_i is the value of SPM_{es} at the beginning of the i^{th} iteration of the loop in lines 8–21; it is the set of MSPRs obtained by combining the first i elements of es,

- $t_{SolveOS(es_i)}$ represents the time spent on computing the meaning of the i^{th} element of the evidential statement es,

- $t_{(C_{a_{j\,i}} \cap C_{b_l})}$ represents time spent on computing set intersection of the computation sets of spm and pm on the i^{th} iteration of the loop in lines 8–21, on the j^{th} iteration of the loop in lines 12–19, and on the l^{th} iteration of the loop in lines 13–18.

An upper bound on the running time of *SolveES* can be derived from the equation 7.22 using inequalities 7.21, 7.3, 7.19, 7.20, and 7.14. After replacing $|es_i|$ and $|SolveOS(es_i)|$ according to inequalities 7.21 and 7.20 respectively, moving common factors to the outside of summations, and initial simplification, the upper bound becomes

$$t_{SolveES} \leq c_{11} + c_{12}\, \mathit{infinitum}^{os_{max}} + c_{13}\,(|es| - 1) + \sum_{i=0}^{|es|-1} t_{SolveOS(es_i)} + \sum_{i=1}^{|es|} \Big(\sum_{j=0}^{|SPM_i|} \big(c_{14} + \sum_{l=0}^{\mathit{infinitum}^{os_{max}}} (c_{15} + t_{(C_{a_{j\,i}} \cap C_{b_l})})\big)\Big) \quad (7.23)$$

The value of the triple-nested sum depends on $|SPM_i|$ and on the time $t_{(C_{a_{j\,i}} \cap C_{b_l})}$. To derive an upper bound on $|SPM_i|$ note that the loop in lines 13–18 creates at most $|SolveOS(es_i)|$ elements of SPM_{i+1} for each element of SPM_i. Since the value of $|SolveOS(es_i)|$ is bounded by the inequality 7.20, the value of $|SPM_i|$ is bounded by the following recurrence

$$|SPM_1| \leq \mathit{infinitum}^{os_{max}}$$
$$|SPM_{i+1}| \leq |SPM_i|\, \mathit{infinitum}^{os_{max}}$$

Solving this recurrence for i produces

$$|SPM_i| \leq infinitum^{i\, os_{max}} \tag{7.24}$$

The time $t_{(C_{a_{j\,i}} \cap C_{b_l})}$ is bounded by the inequality 7.3

$$t_{(C_{a_{j\,i}} \cap C_{b_l})} \leq c_{16}\, |L_{C_{a_{j\,i}}}|\, |L_{C_{b_l}}|\, infinitum \tag{7.25}$$

where c_{16} is an implementation dependent constant.

Since C_b is produced by *SolveOS*, the size of $L_{C_{b_l}}$ is bounded by the inequality 7.14. The size of $L_{C_{a_{j\,i}}}$ depends on the previous iterations of the loop in lines 8–21, because each *spm* in SPM_{i+1} is obtained by combining one element of SPM_i with one element of output of $SolveOS(es_i)$. Thus, the size of $L_{C_{a_{j\,i}}}$ is bounded by the following recurrence

$$\begin{aligned} |L_{C_{a_{j\,1}}}| &\leq |L_{SolveFOS}| \\ |L_{C_{a_{j\,i+1}}}| &\leq |L_{C_{a_{j\,i}}}|\, |L_{SolveFOS}| \end{aligned}$$

Solving the recurrence for i produces

$$|L_{C_{a_{j\,i}}}| \leq |L_{SolveFOS}|^i \tag{7.26}$$

Substituting this bound into 7.25 and simplifying it using inequalities 7.21, 7.15, and 7.14 produces

$$t_{(C_{a_{j\,i}} \cap C_{b_l})} \leq (|Q|^2 |I|^{p+1})^{(i+1)\, |os_{max}|\, infinitum} \tag{7.27}$$

An upper bound on the running time of the event reconstruction algorithm can be obtained from the inequality 7.23 by replacing $t_{(C_{a_{j\,i}} \cap C_{b_l})}$, SPM_i, and $t_{SolveOS(es_i)}$ according to inequalities 7.27, 7.24, and 7.19, and simplifying. The resulting upper bound on the running time of the event reconstruction algo-

rithm is

$$t_{SolveES} \leq c_{11} + c_{12}\, infinitum^{os_{max}} + A + B + \Gamma \tag{7.28}$$

where A approximates the time spent on converting individual observation sequences of into sets of fixed-length observation sequences

$$A = |es| \left(\left(c_8 + c_9 \left(infinitum + 1 \right) \right) \left(\frac{infinitum^{os_{max}} - 1}{infinitum - 1} \right) + c_6 + c_7\, os_{max} + c_{10}\, infinitum^{os_{max}} \right)$$

B approximates the time spent on computing the meaning of the fixed length observation sequences

$$B = |es|\, infinitum^{os_{max}} \left((c_4 p + c_5 |Q||I|) \left(\frac{(|Q|^2 |I|^{p+1})^{os_{max}\, infinitum + 1} - (|Q|^2 |I|^{p+1})}{(|Q|^2 |I|^{p+1}) - 1} \right) + c_1 + os_{max}(c_2 + c_3\, infinitum) \right)$$

Γ approximates the time spent on combining the meanings of observation sequences

$$\Gamma = \frac{\left(infinitum\, (|Q|^2 |I|^{p+1})^{infinitum} \right)^{(|es|+1)\, os_{max}} - \left(infinitum\, (|Q|^2 |I|^{p+1})^{infinitum} \right)^{2\, os_{max}}}{\left(infinitum\, (|Q|^2 |I|^{p+1})^{infinitum} \right)^{os_{max}} - 1} + \frac{\left(infinitum^{(|es|-1)\, os_{max}} - 1 \right) \left(c_{14} + c_{15}\, infinitum^{os_{max}} \right) infinitum^{os_{max}}}{infinitum^{os_{max}} - 1}$$

In summary, assuming that the event reconstruction algorithm uses prefix based representation of computation sets, its running time is no more than *exponential* in the length of evidential statement, maximal length of observation sequences, the value of *infinitum* and the maximal length of prefixes used for representing observed properties. At the same time, the running time is no more than *polynomial* in the size of the state space and the number of possible events.

7.5 Implementation of the event reconstruction algorithm

The event reconstruction algorithm has been implemented as a "proof-of-concept" Common Lisp program, whose source code is collectively given by the Appendices C.2, C.3, and C.7. It was developed using CMU Common Lisp 18c running on a Pentium PC. This section describes the interface of the program.

The program provides a set of constants, macros, and functions for defining observation sequences and evidential statements, computing their meaning, and visualising the results of event reconstruction.

Observed properties are defined using two macros: `defprop1` and `defprop2`.

Macro `(defprop1 name1 (c0) exp1)` defines constant with name `name1` that represents the set of computations, whose first element `c0` satisfies logical expression `exp1`. Formally, it defines property of the form $P_{name1} = \{c \,|\, c \in C_T,\ exp1(c0)\}$.

Macro `(defprop2 name2 (c0 c1) exp2)` defines constant with name `name2` that represents the set of computations, whose first element `c0` and second element `c1` satisfy logical expression `exp2`. Formally it defines property $P_{name2} = \{c \,|\, c \in C_T,\ exp2(c0, c1)\}$.

Observation sequences are represented by Lisp lists. For example, observation sequence *example2* from Section 7.3 can be defined as follows:

```
(defprop1 *A* (c0) ...)
(defprop1 *B* (c0) ...)
(defconst *EXAMPLE2* `((,*A* 1 2) (,*B* 1 3)))
```

The meaning of observation sequence is computed using function `solve-os`. It takes an observation sequence and returns a list of MPRs that describes the meaning of the given observation sequence. For example, the meaning of *example2* is computed by

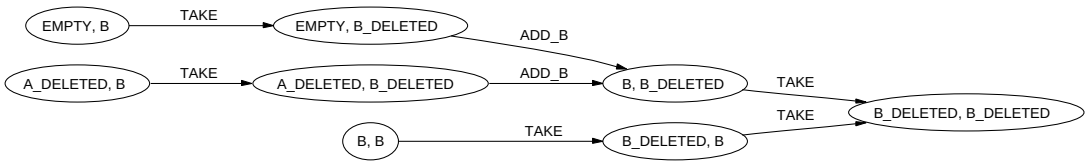

Figure 7.5: Sample output of the program

```
(solve-os *EXAMPLE2*)
```

Evidential statements are represented by Lisp lists. For example, the evidential statement $es = (os1, os2)$ can be defined as follows

```
(defconst *OS1* ...)
(defconst *OS2* ...)
(defconst *ES* '(,*OS1* ,*OS2*))
```

The *meaning of evidential statement* is computed using function `solve-es`. It takes an evidential statement as input and returns a list of MSPRs that describe the meaning of the given evidential statement. For example, the meaning of *es* is computed by

```
(solve-es *ES*)
```

To *visualize the meaning of evidential statement*, function `draw` is provided. It takes the meaning of evidential statement and creates a tree of possible scenarios[3]. An example tree is shown in Figure 7.5 of the incident.

7.6 Summary

This chapter presented an algorithm for event reconstruction based on the formalisation of event reconstruction given in Chapter 6. The algorithm performs event reconstruction in three major steps:

[3] The output of `draw` is a file for DOT utility [37]. The latter should be manually invoked to draw the tree.

1. Convert all observation sequences in the evidential statement into sets of fixed-length observation sequences.

2. Compute the meanings of the resulting fixed-length observation sequences.

3. Combine the meanings of the fixed-length observation sequences into the meaning of the evidential statement.

The running time of the algorithm has been analysed. The derived upper bound on the running time of the algorithm is exponential in the length of evidential statement, maximal length of observation sequences, the value of *infinitum*, and the maximal length of prefixes used for representing observed properties. However, the upper bound is polynomial in the size of the state space and the number of possible events.

The algorithm has been implemented as a Common Lisp program. The next chapter further evaluates its capabilities by applying it to the analysis of example problems of digital forensic analysis.