Recovering the Lattice of Repetitive Sub-functions

Guy-Vincent Jourdan[1], Hasan Ural[1], and Hüsnü Yenigün[2]

[1] School of Information Technology and Engineering (SITE), University of Ottawa, 800 King Edward Avenue, Ottawa, Ontario, Canada, K1N 6N5
{gvj, ural}@site.uottawa.ca
[2] Faculty of Engineering and Natural Sciences, Sabancı University, Tuzla, Istanbul, Turkey 34956
yenigun@sabanciuniv.edu

Abstract. Given a set of observations of an existing concurrent system with repetitive sub-functions, we consider the construction of an MSC graph representing the functionality of the concurrent system. We first introduce a formal structure that we call "lattice of repetitive sub-functions". This lattice provides us with a global view of all the repetitive sub-functions of the system and all the compatible observations. Using the lattice, we are able to propose an algorithm that constructs the MSC graph representation of the system functionality in a more general context than in previously published work.

1 Introduction

Often, depictions of individual intended behaviors of a concurrent system are given by designers as Message Sequence Charts (MSCs) [1,2]. An MSC is a visual description of a series of message exchanges among communicating processes in a concurrent system. Figure 1, left, shows an MSC of three processes exchanging a total of five messages. The message m_1 is sent by the process P_2 and received by the process P_3, which is represented by an arrow from P_2 to P_3 and labeled m_1. Each message exchange is represented by a pair of *send* and *receive* events. The local view of the message exchanges of a process (send and receive events of a process) is a total order, but the global view is a partial order. A tuple consisting of a local view for each process of the message exchanges depicted in an MSC uniquely determines that MSC. Thus, an MSC represents a partial order execution of a concurrent system which stands for a set of linearizations (total order executions of the system) determined by considering all possible interleavings of concurrent message exchanges implied by the partial order.

To describe a functionality that is composed of several sub-functionalities, an MSC graph (a graph with a source and a sink node where edges labeled by MSCs) can be used. An MSC corresponding to the concatenation of MSCs along a path from the source node to the sink node in an MSC graph is said to be in the language of the MSC graph. In the following, M^k means that M is repeated k times, and M^* means any number of repetitions of M. Figure 1, right,

P. Yolum et al.(Eds.): ISCIS 2005, LNCS 3733, pp. 956–965, 2005.

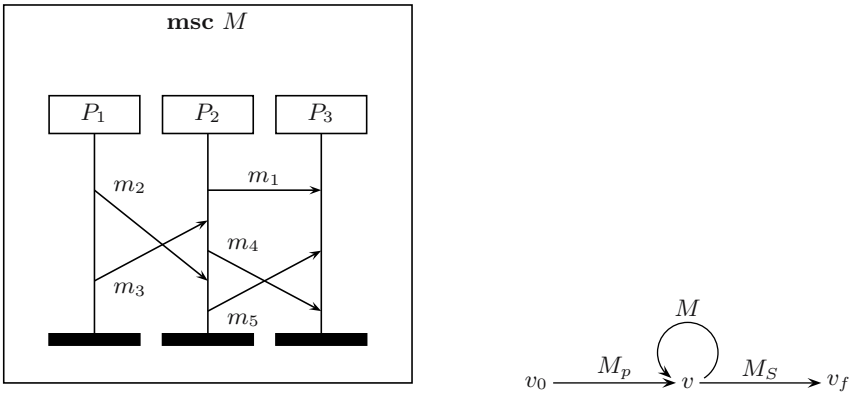

Fig. 1. An MSC of three processes (left) and an example MSC Graph (right)

shows an MSC Graph where the MSC M_p is followed by an arbitrary number of iterations of the MSC M, followed by the MSC M_s, which defines the language $M_p.M^*.M_s$. In this paper we assume that an MSC in the language of an MSC graph represents a system functionality from the initial state to the final state, without going through the initial state again during the execution.

Formal semantics associated with MSCs provides a basis for their analysis such as detecting timing conflicts and race conditions [3], non–local choices [4], model checking [5], and checking safe realizability [6,7].

One of the aims of the reverse engineering [8,9] is to recover the design of an existing system from the run time behavior of its implementation. In this paper, we consider the reverse engineering of designs of existing concurrent systems from given sets of observations of their implementations. We assume that we are given a set O of observations, each observation $o \in O$ being an arbitrary linearization of an MSC m from a set of MSCs that is not given. Some of the sub-functions of the system can be *repetitive*, in which case they can be called consecutively a different number of times in different runs of the system. We assume that a repetitive sub-function does not start (resp. end) at the initial (resp. final) state, and that every repetitive sub-function of the system (if any) is represented in the given set of observations at least twice: once with no occurrence, and once with two or more consecutive occurrences.

In [10], a method to infer repetitive sub-functions from a given set of observations is described. However, this method requires restrictive assumptions regarding the observations. In this paper, we introduce a new concept, the *lattice of the repetitive sub-functions*, to model a structure indicating all possible 2^n combinations of n repetitive sub-functions of the observed system. Using this concept, we are able to relax the following the assumptions made in [10]:

- Repetitive sub-functions do not have to be iterated the same number of times in each observation,
- Repetitive sub-functions need not be introduced in a specific order,
- The ordering of the sub-functions need not be totally unambiguous.

2 Preliminaries

A sub-function that is repeated in an observation will create a repeated pattern in the MSC corresponding to that observation. However, a simple pattern repetition is not enough. In order to deduce the existence of a repetitive sub-function, we need to have an evidence such as different number of iterations of the pattern within the same context.

Definition 1. *An MSC M is* the basic repetitive MSC *of MSC M' if $M' = M^k$ for some $k \geq 2$ and there does not exist a basic repetitive MSC of M.*

Consider the visual representation of an MSC M and imagine that we draw a line through M by crossing each process line exactly once, and without crossing any message arrows. Such a line divides M into two parts M_p (the part above the cutting line) and M_s (the part below the cutting line). M_p and M_s can be shown to be MSCs again. M_p and M_s are what we call a prefix of M and a suffix of M, respectively.

Definition 2. *Two MSCs M_1 and M_2 are said to* infer M to be repetitive within the context M_p–M_s *if all the following are satisfied:*

1. *M does not have a basic repetitive MSC,*
2. *$M_1 = M_p.M^k.M_s$ for some $k \geq 2$ and $M_2 = M_p.M_s$,*
3. *M is not a suffix of M_p and M is not a prefix of M_s.*

Definition 3. *A* common prefix (resp. suffix) *of two MSCs M_1 and M_2, is an MSC M, such that M is a prefix (resp. suffix) of both M_1 and M_2. The* maximal common prefix (resp. suffix) *of M_1 and M_2 is a common prefix (resp. suffix) M of M_1 and M_2 with the largest number of events.*

3 The Lattice of the Repetitive Sub-functions

The *lattice of the repetitive sub-functions* is a structure providing all possible selection of n repetitive sub-functions, including none of them (bottom of the lattice) and all of them (top of the lattice). We first look at the simple case, with only one level of repetitive sub-functions, and we then consider the case of nested repetitive sub-functions.

3.1 The Case Without Nested Repetitive Sub-functions

If two MSCs M_1 and M_2 infer an MSC M to be repetitive within the context M_p–M_s, we obtain a regular expression, $M_p.M^*.M_s$, which can be seen as a language whose alphabet is the set of MSCs used in the regular expression ($\{M_p, M_s, M\}$ in that case). M_1 and M_2 are two of the words of that language.

For example, lets consider the following three MSCs, corresponding to a given set of three observations: $M_1.M_3.M_5$, $M_1.M_2.M_2.M_3.M_5$ and $M_1.M_3.M_4.M_4.M_4.M_5$. The first two MSCs infer the language $M_1.M_2^*.M_3.M_5$, while the

first and third MSCs infer the language $M_1.M_3.M_4^*.M_5$, using the alphabet $\{M_1, M_2, M_3, M_4, M_5\}$.

Considering the general case of n repetitive sub-functions with no nested repetitive sub-functions, the general form of the top "language" (the top of the lattice), representing the selection of all repetitive sub-function, will be of the form $M_p.M_1^*.T_1.M_2^*.T_2.M_3^* \ldots M_{n-1}^*.T_{n-1}.M_n^*.M_s$, where $\forall i \leq n$, M_i is a non empty MSC representing a repetitive sub-function, T_i is a possibly empty "transition" MSC, M_p is the non-empty prefix and M_s is the non-empty suffix. Figure 2 shows an example with $n = 3$.

Because all combinations of the repetitive sub-functions are possible, we can observe every subset of the set of n M_i's, to a total of 2^n possible different languages. All of these languages can be ordered by inclusions: $\forall L_1, \forall L_2, L_1 \subseteq L_2$ if and only if $\forall w, w \in L_1 \Rightarrow w \in L_2$. Then, we are in fact looking at an hypercube of size n, the elements being the languages, where a language L_2 includes a language L_1 if L_2 contains all the repetitive sub-functions included in L_1.

The bottom "language" (the bottom of the lattice) is actually a constant, whose only word is the MSC of the observation with no repetitive sub-function. All other observations should "introduce" a new repetitive sub-function. All in all, n languages will be directly deduced from pairs of observations, and the others will be inferred.

There are several possible sets of observations that will allow the inference of the same lattice. In Figure 2, the black dots are deduced from pairs of observations and the white dots are the languages inferred by inclusion. However, not all combination of four nodes are valid. For example, the right-most combination is invalid, since M_3 is never inferred (never given in any observation, in fact).

Note that if a "transition" T_i is empty (that is, M_i and M_{i+1} are consecutive) and if no observation is provided showing both M_i and M_{i+1}, we cannot order M_i and M_{i+1}.

3.2 The Case of the Nested Repetitive Sub-functions

Observations containing occurrences of nested repetitive sub-functions will yield the top element of a structure whose general form is the same as the one given

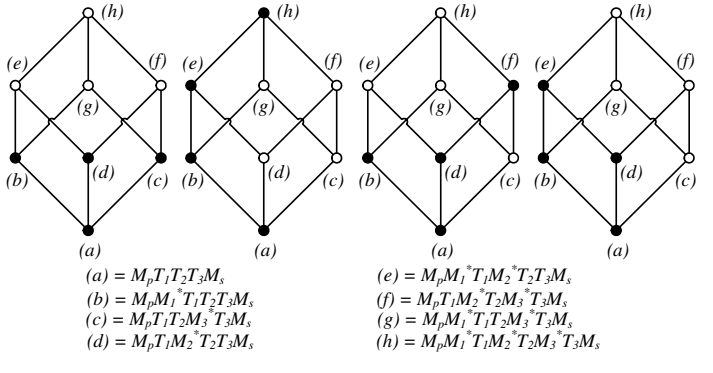

Fig. 2. Different sets of observations for the same lattice

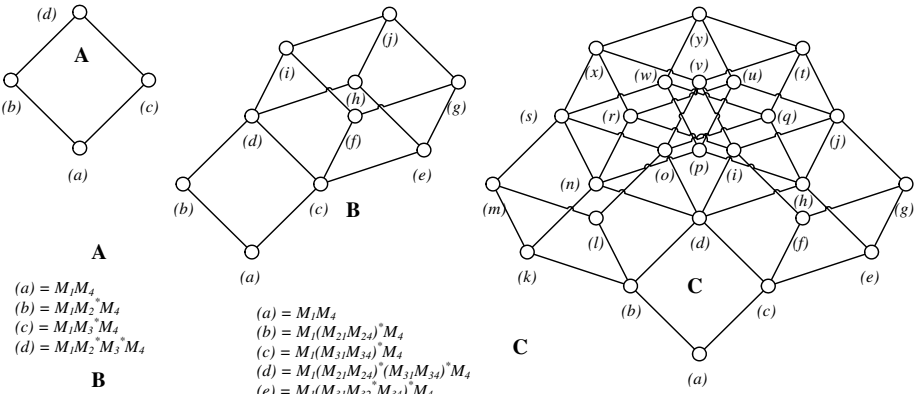

Fig. 3. Two repetitive sub-functions, each one having two nested repetitive sub-functions

in the previous section. In this general form, a sub-function containing k nested repetitive sub-functions will have a non empty prefix, a non empty suffix, and k repetitive sub-functions separated by possibly empty transition MSCs. Thus, this sub-function alone, with its k nested repetitive sub-functions, defines an hypercube of size k. Each of the nested repetitive sub-function can itself have nested repetitive sub-functions and the same idea would apply. That is, we first consider the "first level" repetitive sub-functions. With n first level repetitive sub-functions, we have an initial hypercube of size n, with each repetitive sub-function defining one "direction" or one "face" of the hypercube. Each repetitive sub-function appears in 2^{n-1} nodes of the hypercube.

For each first level repetitive sub-function having nested repetitive sub-functions (say k of them), we replace all 2^{n-1} nodes having that repetitive sub-function with the corresponding hypercube of size k. We then repeat that process for each nested level.

The example in Figure 3 shows two repetitive sub-functions $M_1.M_2^*.M_3^*.M_4$ (A), M_3 itself having two repetitive sub-functions ($M_3 = M_{31}.M_{32}^*.M_{33}^*.M_{34}$, B) and M_2 having two repetitive sub-functions ($M_2 = M_{21}.M_{22}^*.M_{23}^*.M_{24}$, C).

4 Recovery of Repetitive Sub-functions

With the help of the lattice, we can derive a repetitive sub-function recovery algorithm that has fewer assumptions than the ones assumed in [10]. We do keep the following assumptions from [10]:

1. The initial observation (without repetitive sub-function calls), and each repetitive sub-function having nested repetitive sub-functions, has a non empty, repetitive sub-function free prefix and a non empty, repetitive sub-function free suffix.
2. Repetitive sub-functions have no common prefix with the part of the MSC that starts just after them and no common suffix with the part of the MSC that leads to them.
3. Repetitive sub-functions starting at the same point do not alternate.
4. Every repetitive sub-function is introduced "individually" by at least one observation. That is, for every repetitive sub-function M, there is at least one observation o and a set of observations S such that:
   (a) M is repeated at least twice in o
   (b) M does not appear at all in any observation of S
   (c) Every other repetitive sub-function appearing at least once in at least one observation in S is also introduced individually within the set of observations of S.

   In other words, it is possible to introduce the repetitive sub-functions one at a time.

A major difference with [10] is that once a repetitive sub-function has been introduced, it can then be used (or not used), and used any number of times in the other observations. In the case of nested repetitive sub-functions, the observation that introduces the nested repetitive sub-function iterates that nested repetitive sub-function in only one occurrence of the outer repetitive sub-function.

4.1 Description of the Algorithm

We first consider the following simplified case:

- The ordering of the repetitive sub-functions is never ambiguous (there is always either a non empty transition between two consecutive repetitive sub-functions, or these repetitive sub-functions are provided incrementally).
- There are no nested repetitive sub-functions.
- There are no unnecessary observations (i.e. we are working with $n + 1$ observations to uncover n repetitive sub-functions).

We are going to construct the lattice corresponding to a given set of observations step by step, starting from the bottom. Initially, we look for observations that allow us to deduce a single repetitive sub-function (assumption 4 ensures us that there is at least one). Once we have deduced all these repetitive sub-functions, we complete that portion of the lattice and infer the top of it. Then, we look for observations that allow us to deduce just one repetitive sub-function in addition to the ones that have been already deduced (again, thanks to assumption 4, there is always one at least). We complete that part of the lattice, infer the top, etc. until we are done.

We will use "topLabel" to store the current top of the lattice. Initially, topLabel is the bottom of the lattice (which is simply the MSC of the shortest observation of the whole set).

```
topLabel = the MSC of the shortest observation
S = set of all observations minus topLabel
while S ≠ ∅ do
  /* First phase: discovering the next set of repetitive sub-functions */
  for all observations o ∈ S do
    Let prefix = MaxCommonPrefix(topLabel, o)
    Let suffix = MaxCommonSuffix(topLabel, o)
    if prefix ≠ null and suffix ≠ null and prefix.suffix ∈ topLabel then
      Let M be the portion of the MSC of o between prefix and suffix
      /* MSC representation of o=prefix.M.suffix */
      if M_1 = basic_repetitive_MSC(M) then
        /* M = M_1^k for some k. We have found prefix.M_1^*.suffix */
        Add prefix.M_1^*.suffix as a successor of topLabel in the lattice
        Remove o from S
      end if
    end if
  end for
  /* second phase: reconstruction phase */
  Close the lattice by combining all the found successors of topLabel.
  Assign topLabel to the top of the lattice.
end while
```

After the first pass through the while loop, topLabel is a regular expression, using the repetition($*$). The functions MaxCommonPrefix and MaxCommonSuffix presented in [10] must be adapted as follows: trace along topLabel. Once you reach a repetition (you reach an M^*), there are two options: either follow M or skip it altogether. Thanks to assumption 2, you see immediately which way to go. If M is followed, you can loop through M any number of times. If the trace ends in the middle of M, then push back the common prefix to the beginning of that repetitive sub-function M.

Let's consider the following set of MSCs corresponding to a given set of four observations as an example:

1. $M_p.M_t.M_s$
2. $M_p.M_a.M_a.M_a.M_t.M_s$
3. $M_p.M_t.M_b.M_b.M_s$
4. $M_p.M_a.M_t.M_b.M_b.M_b.M_c.M_c.M_s$

Initially, topLabel = $M_p.M_t.M_s$ In the first pass through, topLabel and observations 2) find $M_p.M_a^*.M_t.M_s$ and topLabel and observations 3) find $M_p.M_t.M_b^*.M_s$. The lattice, a square, is closed, and $M_p.M_a^*.M_t.M_b^*.M_s$ is inferred as the top. In the second pass, topLabel = $M_p.M_a^*.M_t.M_b^*.M_s$ with observations 4) find $M_p.M_a^*.M_t.M_b^*.M_c^*.M_s$. The lattice, a cube, is closed, and $M_p.M_a^*.M_t.M_b^*.M_c^*.M_s$ is inferred as the final top.

4.2 Adding the Nested Repetitive Sub-functions

If there are nested repetitive sub-functions, the above algorithm will not find them. To handle this case, we have to waive the restriction that MaxCommonPrefix must not finish inside a repetitive sub-functions. We must trace the max-

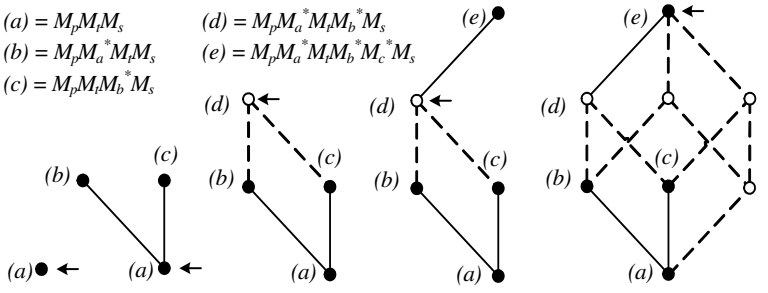

Fig. 4. The algorithm step by step

imal common prefix within the repetitive sub-function. If it finishes inside it, we trace the maximal common suffix and if it too finishes at the same point in the repetitive sub-function, then that is the candidate for the starting point of the nested sub-function.

Thus, the strategy becomes the following: we first iterate the algorithm as before, until going through the for loop (line 5) does not add new repetitive sub-functions. At that point, we have identified all the "first level" repetitive sub-functions. We then continue the algorithm, this time allowing the max common prefixes and max common suffixes to stop inside a first level repetitive sub-function. We iterate until, again, the for loop does not add new repetitive sub-functions. At that point, we have found all the second level repetitive sub-functions (repetitive sub-functions inside a repetitive sub-functions). Now, we allow suffixes and prefixes finishing inside second level repetitive sub-functions and find third level repetitive sub-functions. We go one until we have exhausted the set of observations.

4.3 Waiving the Non-ambiguity Condition

Consider the following set of MSCs corresponding to a given set of four observations:

1. $M_p.M_s$
2. $M_p.M_a.M_a.M_a.M_s$
3. $M_p.M_b.M_b.M_b.M_s$
4. $M_p.M_a.M_b.M_c.M_c.M_c.M_s$

By applying the algorithm of Section 4.1 we will first find M_a and M_b, but we don't know their respective order until later, when we find M_c.

We need to adapt the algorithm so that it records the ambiguity: during the lattice completion phase (line 19), we may end up with several repetitive sub-functions starting at the same point. We need thus to enhance our regular expression and allow alternative. In the example above, the first discovery phase finds $M_p.M_a^*.M_s$ and $M_p.M_b^*.M_s$, so the first reconstruction phase can't decide whether the top should be $M_p.M_a^*.M_b^*.M_s$ or $M_p.M_b^*.M_a^*.M_s$. So we record $M_p.(M_a^*|M_b^*).M_s$.

We thus need to modify MaxCommonPrefix and MaxCommonSuffix again: now, as we trace inside topLabel, we may reach a point where there are several choices: take any of the repetitive sub-functions starting there, or move on.

What we need to do is try all the repetitive sub-functions and follow the one that works. Two repetitive sub-functions can now have the same prefix, so we may end up following several repetitive sub-functions at the same time, but because ultimately all repetitive sub-functions are different, we will be able to decide which repetitive sub-function we must really follow before the end of the first iteration. We must record what repetitive sub-functions have been taken because we can go through several ones at that point, one after the other (the sub-functions cannot alternate according to assumption 3).

If at least two repetitive sub-functions start from the same point and have a common prefix, it is possible that MaxCommonPrefix finishes before we have reached a point allowing to decide which sub-function we are actually following. Again because all repetitive sub-functions are ultimately distinct, MaxCommonSuffix will resolve the problem: it may also end up in the middle of several repetitive sub-functions, but the set of repetitive sub-functions found by MaxCommonPrefix and the set of repetitive sub-functions found by MaxCommonSuffix will have at most one sub-function in common (the right one, if there is one).

Finally, to eventually waive the ambiguity, we have to enhance line 11: when we find a repetitive sub-function, if we have followed ambiguous branches during MaxCommonPrefix and/or during MaxCommonSuffix, then the order in which we have followed these branches is the order of the repetitive sub-functions and the ambiguity is removed.

In the example, after inferring $M_p.(M_a^*|M_b^*).M_s$ as the current top, in the next phase we trace it against $M_p.M_a.M_b.M_c.M_c.M_c.M_s$. We successfully locate M_c^*, and at that time we waive the $M_a|M_b$ ambiguity and finish with $M_p.M_a^*.M_b^*.M_c^*.M_s$.

Note that in fact, we can finish all the observations and still have ambiguity in the final top of the lattice. This means that the order of the ambiguous repetitive sub-functions was never provided. We can report that fact to the user.

4.4 Waiving the No Non-necessary Observations Conditions

So far, each observation adds something at one point. We actually don't need that. If an observation is "redundant", in that it doesn't help discovering any repetitive sub-function, at one point the current "topLabel" will be able to already generate that observation. That is, in the algorithm it simply means that MaxCommonSuffix will consume the complete observation. At that point, we just need to discard it. Note that a redundant observation can still be useful to waive ordering ambiguity. If that is the case, then we must simply record the order before discarding the observation.

5 Conclusion

We have introduced the "lattice of repetitive sub-functions", a new formal structure providing a global view of the compatible observations of a system having repetitive sub-functions. We have described an algorithm to construct an MSC graph of the functionality of a system built from a set of observations. The new

algorithm introduced in this paper is an improvement over the solution presented in [10] in that some of the most restrictive assumptions on the set of observations are being waived. The last strong assumption remaining is the assumption 4 of the Section 4, stating that each repetitive sub-function must be introduced "individually". In future work, we will attempt to waive this assumption as well.

References

1. ITU Telecommunication Standardization Sector: ITU-T Recommendation Z.120. Message Sequence Charts (MSC96). (1996)
2. Rudolph, E., Graubmann, P., Gabowski, J.: Tutorial on message sequence charts. Computer Networks and ISDN Systems–SDL and MSC **28** (1996)
3. Alur, R., Holzmann, G.J., Peled, D.: An analyzer for message sequence charts. Software Concepts and Tools **17** (1996) 70–77
4. Ben-Abdallah, H., Leue, S.: Syntactic detection of progress divergence and non–local choice in message sequence charts. In: 2nd TACAS. (1997) 259–274
5. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: 10th International Conference on Concurrency Theory, Springer Verlag (1999) 114–129
6. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. In: 22nd International Conference on Software Engineering. (2000) 304–313
7. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Transactions on Software Engineering **29** (2003) 623–633
8. Chikofsky, E., Cross, J.: Reverse engineering and design recovery. IEEE Software **7** (1990) 13–17
9. Lee, D., Sabnani, K.: Reverse engineering of communication protocols. In: IEEE ICNP'93. (1993) 208–216
10. Ural, H., Yenigun, H.: Towards design recovery from observations. In: FORTE 2004, LNCS 3235. (2004) 133–149