Generalizing the Submodule Construction Techniques for Extended State Machine Models

Bassel Daou and Gregor v. Bochmann

School of Information Technology and Engineering (SITE), University of Ottawa.

bdaou@site.uottawa.ca, bochmann@site.uottawa.ca

Abstract. In previous research we extended the submodule construction techniques to cover a more expressive and compact behavioral model that handles data through parameterized interactions, state variables, and simple transition guards. The model was based on extended Input/Output Automata, and the algorithm on the Chaos concept. In this paper we generalize these extensions and improve the submodule construction techniques and algorithms. The generalizations include regular transition guards including equality and inequality, negation, conjunction and disjunction of predicates. The algorithm is improved by utilizing the concept of generic transitions (non refined transitions) that are refined as needed instead of considering all possible refinements of the Chaos. The algorithm selects needed refinements through dataflow relations bridging which involves forward propagation of definitions and backward propagation of usages. The new approach provides a more intuitive explanation of the submodule construction algorithm, gives justification for the number of variables in the new module and results in a much smaller and compact solution.

1. Introduction:

Submodule construction, also called equation solving or factorization, considers the following situation: An overall system is to be constructed which consists of several components. It is assumed that the specification S of the desired behavior of the system is given, as well as a specification of the behavior of all the components, except one. The process of submodule construction has the objective to find a specification for the latter component X such that when joined with all other components , referred to as the Context C, together provide a behavior consistent with the behavior specification S. If the modeling paradigm for the behavior specifications is sufficiently limited, e.g. finite state models, an algorithm for submodule construction can be defined [MeBo83, Parr89, LaXi90, JoLa91, LeQi90, PeYe98, DrBo99, CDFM01]. Submodule construction finds application in the synthesis of controllers for discrete event systems [BrWo94], for communication gateway design and protocol conversion [KeHa93, KNM97, TBD97].

Service Oriented Architecture (SOA) is an area of possible application for sub-submodule construction. The submodule construction techniques may be used in Services synthesis and implementation. However, these techniques need to be developed to fit SOA requirements, such as data manipulation and handling nonfinite state models.

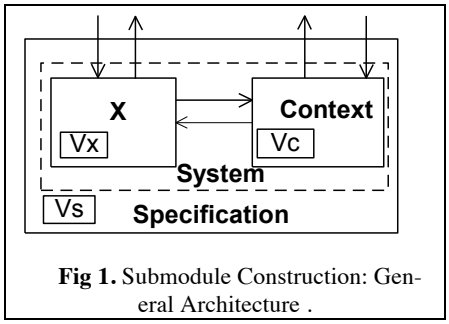

Fig 1. Submodule Construction: General Architecture .

In this paper we report the continuation of work published in FORTE 2005 [DaBo05] where we extended the sub-module construction techniques that has been limited in the past to finite state models. In this work we ease the restrictions that were applied to these extensions and we modify the solution approach and parts of the algorithm to fit the new model and to provide a smaller, more compact and more intuitive solution.

2. Model Extensions

In the previous model [DaBo05], data manipulation and value passing were achieved by extending finite automata models with parameterized interactions, local variables, simple transition guards and variable assignments. Transition guards were limited to the conjunction of equality predicates between variables and transition parameters. In this paper we eliminate this restriction so that guards can include disjunction and negation as well as inequality predicates between variables and parameters. Moreover, in the previous model, variables were only assigned parameter values; in the new model we ease this restriction to allow assignment of variables between one another.

3. New Solution Approach

Our previous algorithm followed the general steps of the submodule construction algorithm for finite state machines. It starts with a general superset of behaviors, called Chaos, it then removes unwanted behaviors through composition, hiding, determinization and bad or uncontrollable state removal. These steps were adapted for the extended specification paradigm. During the construction of a deterministic model, the effect of hidden guards and hidden variables was taken care of through state splitting transformations based on previously collected information about variable configurations. In the new approach we continue to use the same general outline of the algorithm, however, we use the duality concept to obtain a superset of the wanted behavior before hiding, instead of using Chaos machine concept.

We define the dual of a given behavior G as the most general matching behavior G' that when joined with G will never generate an output that is not specified among inputs accepted by G. Besides, G' always accepts as input any matching output of G.

G' puts no restrictions on inputs that are not generated by G. In our model G' is obtained from G by labeling inputs as outputs and outputs as inputs. So G' has a set of variables V' matching the set of variables V of G.

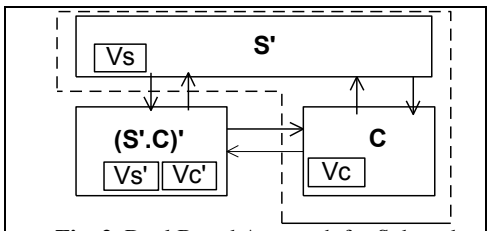

Fig. 2. Dual Based Approach for Submodule Construction.

Thus, as shown in figure 2 the superset of behaviors that is used in the new submodule construction algorithm will be the dual of C joined with the dual of S, *(C.S')'*, which is in general a much smaller set of behaviors than the Chaos machine. This approach provides the following benefits:

1. A better explanation for the number of variables needed for describing the most general behavior of the new module (in fact, a copy of the variables in S and in C suffice).
2. A smaller representation of the most general behavior for the new module which results from the fact that a single mapping of new module variables to the variables of S and C can be used, instead of having a solution with all possible permutations of variable mappings. Each variable of the new module is mapped to its original variable in either C or S.

The other aspect of our new approach is the use of the concept of generic "unrefined" transitions instead of using "interaction chaos" and the means of selecting only refinements that contribute to the solution. In our previous algorithm all possible refinements were explicitly considered, which though theoretically possible, becomes very unpractical for rather simple submodule construction problems. We need to note that in the behavior superset only transitions that are executed by the new module can be refined since we have full control over the new module. And thus two types of refinements are possible. These are: conditions on what the new module sends or receives, and options of where the new module stores received values.

Traditionally to overcome the effect of hiding in the case of finite label transition systems, transition closure and determinization were enough. However, when variables enter the picture as in our model we need to do something more, we basically need to make sure that variables are used properly, that is variables use the right values as defined in the specifications. We are especially interested in dataflow relations that cross machine borders. The Chaos solution explicitly generates all possible refinements and consequently all possible dataflow associations, however, not all these dataflow relations are needed or at least need to be identified, we only need to find all possible dataflow relations that simulate specification dataflow relations.

Thus the idea behind our approach is to perform dataflow analysis on the general behavior in order to identify the needed refinements. Accordingly, parameter value storage refinements are identified using forward propagation of definitions dataflow analysis. Meanwhile, conditions on received and sent data refinements are identified using backward propagation of usage dataflow analysis.

4. Algorithm Modifications:

The new algorithm manipulates guards using the disjunctive normal form (disjunction of conjuncts). So, a transition can be viewed as a group of transitions where each transition has a guard formed of a single conjunct of the original transition's conjuncts. The algorithm handles the conjuncts collectively when possible and separately when situation demands such as in some cases of the backward state and transition splitting. Regarding negation and inequality their effect is limited to the conformance predicate which checks whether a transition guard is enabled for a given matching relations which we represent using a variable partition.

Alogrithm 1. Submodule Construction Algorithm Steps [DaBo05]:	**Algorithm 2. New Submodule Construction Algorithm Steps:**
Given C, S:, ∑X Alphabet	Given C, S:, $\sum_X$ Alphabet
1. G1:=Chaos(∑X , |S.V|+|C.V|).(S'. C)	**1. G0:= (S.C)'.(S'.C)**
2. R := ComputePartitions(G1)	**2. G1 = AddRefinements(G0)**
3. G2 := Split(G1, ∑X , R,)	3. R := ComputePartitions(G1)
4. G3:= Hide(G2, (∑C U ∑S)-∑X, S.V U C.V)	4. G2 = Split(G1, $\sum_X$, R,)
5. G4:= Determinize(G3)	5. G3 :=Hide(G2, (∑C U ∑S) - ∑X , S.V U C.V)
6. X:=RemoveUncontrollableBehavior(G4)	6. G4 := Determinize(G3)
7. Return X	7. X := RemoveUncontrollableBehavior(G4)
	8. Return X

In the following we provide a high level outline of the new algorithm step "AddRefinements" sub-algorithm focusing on refinements added due to specification context definition and corresponding new module usage.

Algorithm 3: **AddRefinements (G)**

- CX = { (t1, t2, c1, s1) | t1 is a transition where the definition of C variable c1 simulates definition of specification variable s1 and t2 is the transition where the corresponding usage of s1 takes place in X }
- Done = {} //*represent handled define-use associations*.
- Loop While not (CX = {})
  - Remove (t1, t2, c1, s1) from CX
  - Done := Done U {(t1, t2,c1, s1)}
  - CX := CX U ({ (t3, t2, c3, s1) | t3 is a transition where c1 is used to define c3 such as c3:=c4} – Done)
  - For each t in {t | t has an output interaction sent from C to X, where a parameter p of t takes c1 value}
    - If t already has an assigning s1 to a parameter p2 other than p
      - Replicate t3 replace s1:= p2 with s1 = p
    - Else Add s1:=p to t3

The algorithm is guaranteed to stop since the possible dataflow relations existing are finite and the algorithm does not handle dataflow relation that has been already handled.

5. Conclusion and Future Work

This paper continues the work done on extending submodule construction techniques for finite state machines to more expressive behavioral models. We have eased the restriction on the model mainly allowing conjunction, disjunction, explicit negation and state variables equality predicates in state transition guards. We have presented a new solution approach that improves the practicality and efficiency of the algorithm, justifies the number of variables used in the new module, and results in a smaller solution by considering a standard mapping of new module variables to context and specification variables. We have provided an outline of the new algorithm that is based on dataflow analysis mainly backward propagation of criteria and forward propagation of definitions. This work will be the basis for adding more extensions to the behavioral model, such as considering functions and general predicates over variables which we are currently considering.

References

[BrWo94] B. A. Brandin, and W.M. Wonham. Supervisory Control of Timed Discrete Event Systems. IEEE Transactions on Automatic Control, Vol. 39, No. 2, pp. 329-342, 1994.

[CDFM01] V. Carchiolo, N. De Francesco, A. Fantechi, G. Mangioni, "ESA: an approach to Systems Design by Equation Solving". FMICS'2001, Paris, France, July 2001.

[DaBo05] B. Daou and G.V. Bochmann. Submodule Construction for Extended State machine Models. FORTE 05,pp. 396-410, 2005.

[DrBo99] J. Drissi, and G.V. Bochmann. Submodule Construction for Systems of I/O Automata. Tech. Rep. no. 1133, DIRO, University of Montreal, 1999.

[JoLa91] B. Jonsson, K.G. Larsen. On the complexity of equation solving in behavior expression algebra. TAPSOFT'91, vol. 1, LNCS 493, pp. 381-396, 1991.

[KeHa93] S.G. Kelekar, G. W. Hart. Synthesis of Protocols and Protocol Converters Using the Submodule Construction Approach. PSTV93, pp. 307-322, 1993.

[KNM97] R. Kumar, S. Nelvagal, and S. I. Marcus. A Discrete Event Systems Approach for Protocol Conversion. Discrete Event Dynamical Systems: Theory and Applications, Vol. 7, No. 3, pp. 295-315, 1997.

[LaXi90] K. Larsen, L. Xinxin. Equation solving using modal transition systems. LICS'90, 1990.

[LeQi90] P. Lewis and H. Qin. Factorization of finite state machines under observational equivalence. LNCS 458, Springer, 1990.

[MeBo83] P. Merlin, and G. v. Bochmann. On The Construction of Submodule Specifications and Communication Protocols, ACM Trans. On Programming Languages and Systems. Vol. 5, No. 1, pp. 1-25, 1983

[PeYe98] A. Petrenko and N. Yevtushenko. Solving Asynchronous Equations. FORTE'98, (1998), 231-247.

[Parr89] J. Parrow. Submodule Construction as Equation Solving in CCS. Theoretical Computer Science, Vol. 68, 1989.

[TBD97] Z. Tao, G. v. Bochmann and R. Dssouli. A Formal Method For Synthesizing Optimized Protocol Converters And Its Application To Mobile Data Networks. Mobile Networks & Applications, Vol.2, No. 3, pp. 259-69, 1997.