A Co-Synthesis Approach Based On Symbolic Reachability Analysis

Francisco Assis M. do Nascimento
Universidade Luterana do Brasil
Departamento de Informática
assis@mozart.ulbra.tche.br
http://www.ulbra.tche.br/~assis

Abstract

In this paper we present a co-synthesis approach based on the state space reachability analysis of the system specification, which is captured using a partial order based model for concurrency. The Partial Order based Model (POM) combines the concept of relations between events from event structures for the modeling of intercommunicating concurrent behaviors and the Chu space semantics for these relations in terms of logical formulas. The POM is used in our approach as an intermediate representation, which permits a concise symbolic representation, mainly for systems with high degree of parallelism, and an efficient BDD based implementation of analysis and synthesis algorithms. We introduce the partial order based model for concurrency and how use it in modeling at the system level. We present a symbolic representation for the model and its implementation using BDDs and we give reachability analysis algorithms that are applied to the estimation of performance and costs during the co-synthesis and to the formal verification of properties. Finally we describe our co-synthesis approach based on the formal model.

1. Introduction

Our work is oriented to the co-synthesis of embedded distributed systems, which can be characterized as: consisting of concurrent processes that are constantly reacting to the occurrence of external events; the concurrent processes communicate in order to cooperate and so realize the system functionality; the timing of the operations performed by the processes is subject to constraints that must be satisfied in order to the system operates correctly. Since the objective of our co-synthesis approach is to generate an efficient implementation for a given such system, we are interested in a formalism able to model concurrent, communicating, reactive behaviors while abstracting from implementation details.

Most of the existing approaches to co-synthesis (for example, Castle [11], COSMOS [8], Olympus/Vulcan [6]) use procedural languages to capture the system specification and control/data flow graphs (CDFGs) as intermediate representation, on which the synthesis algorithms are applied. This internal representation is obtained by the parsing and data/control flow analysis [1] of the system specification. This permits to formulate the co-synthesis approaches independent of the language used to specify the system. But with CDFGs it is very difficult to implement design transformations that alter the control and/or data flow of the specification.

Moreover, as well noted by Chaiyakul et al. [3], most of the CDFGs based representations have an one-to-one correspondence between the constructs of the specification language and the nodes in the graph. So two different specifications in the language for a same behavior could result in two different graphs. Since the analysis and synthesis algorithms depend on the topology of the CDFGs, two different implementations could be produced for the same behavior.

Thus it would be interesting to define an intermediate representation that should be able to capture the specification in a unique way. This unique representation should be one where the most possible parallelism existing in the specification is captured. This gives the chance to the synthesis algorithms to explore all possible degrees of parallelism when generating the implementation for the system specification.

In our approach instead of developing a specification language, we are also more interested in defining an intermediate representation, which should be adequate to capture formally and concisely the various aspects of concurrency, and in developing efficient analysis and synthesis algorithms based on this intermediate representation. The intermediate representation based on the POM we developed captures all the parallelism in the system specification and does not reflect the structure imposed by the specific constructs of the specification language.

The use of partial order to the modeling of concurrent
A specific occurrence of an action with a junction of the properties will result in the POM of the process. Gupta [7], we consider the Chu space of each relation. Each state. In the POM definition, we propose a more practical approach based on relations between events. Each process is modeled as a set of relations between events and for each state. We present some preliminary results and conclusions.

2. POM – A Partial Order Based Model for Concurrency

In order to model a process using Chu spaces, one would need to define the set of events and what events occur at each state. In the POM definition, we propose a more practical approach based on relations between events. Each process is modeled as a set of relations between events and for each such relation we have a corresponding Chu space.

Adopting the concept of property of a Chu space given by Gupta [7], we consider the Chu space of each relation between events as a property of the process. Thus, a conjunction of the properties will result in the POM of the process. As in event structures [12], in POM an event models a specific occurrence of an action with a labeling function \( \lambda : E \rightarrow A \) indicating the action modeled by the event. We use the labeling function to introduce the notion of hierarchy when representing a design specification with POMs. An event can be labeled by an action, which is modeled by another POM.

With POM we have a causality (or precedence) relation between events to model that an event \( e \) causes an event \( e' \) and indicates that the occurrence of \( e' \) depends on the occurrence of \( e \). We also have a symmetric binary relation of conflict to represent that \( e \) is in conflict with \( e' \), i.e., both \( e \) and \( e' \) can never occur in a possible run of the process. Additionally we have a disjunctive enable relation to represent that a set of events \( DE \) enables the occurrence of an event \( e \), i.e., the occurrence of at least one of the events \( e \in DE \) is sufficient for \( e \) to occur. When two events \( e \) and \( e' \) are neither in conflict nor causally related, then they are said to be independent and can occur concurrently.

**Definition 1 (POM)** A POM \( P \) is a tuple \( (E, A, \prec, \#_\prec, \rightarrow, \lambda) \), where \( E = \{e_0, e_1, \ldots, e_n\} \) is the set of events, \( \prec \subseteq E \times E \) is the precedence relation, \( \#_\prec \subseteq E \times E \) is the conflict relation, and \( \rightarrow \subseteq E' \times E \) is the disjunctive enable relation, \( A = \{a_0, a_1, \ldots, a_n\} \) is the set of actions, and \( \lambda : E \rightarrow A \) is the labeling function. Each event \( e_i \) represents a specific occurrence of an action \( a_j = \lambda(e_i). \)

The concept of permitted states of a POM is similar to the configurations of event structures [12], i.e., the set of events that have occurred in a possible run of the POM.

**Definition 2 (States of a POM)** The set \( X_P \) of permitted states of a POM \( P = (E, A, \prec, \#_\prec, \rightarrow, \lambda) \) is the set of subsets \( x_i \subseteq E \) which are conflict free, i.e., \( \forall e, e' \in x_i, \neg(e \#_\prec e') \), and downwards-closed, i.e., \( \forall e, e', (e' \prec e) \wedge (e \in x_i) \Rightarrow e' \in x_i. \)

![Figure 1. Relations between events](image-url)

Each relation between events has a corresponding Chu space, which gives the semantics of the relation. The figure 1 presents the Chu space and the corresponding logical formula for each relation.

**Definition 3 (Symbolic Representation of a Process)** Let \( P = (E, A, \prec, \#_\prec, \rightarrow, \lambda) \) be the POM of a process, we define the symbolic representation of the process \( P \) as

\[
 f_P = \bigwedge_{r_i \in \prec} f_{r_i} \wedge \bigwedge_{r_j \in \#_\prec} f_{r_j} \wedge \bigwedge_{r_k \in \rightarrow} f_{r_k}
\]

where \( f_{r_i}, f_{r_j} \) and \( f_{r_k} \) are the logical formulas for the precedence, conflict and disjunctive enable relations, respectively.

In the example of the figure 2, we have five events, where \( a_0 \) precedes \( a_1 \), \( a_1 \) precedes \( a_2 \) and \( a_2 \), \( a_2 \) and \( a_3 \) are in conflict, and \( a_2 \) and \( a_3 \) enable the event \( e_4 \). The conjunction gives us the Chu space for the POM of the process. We use BDDs [2] to represent the logical formula of a POM and to implement the process composition operations.
In order to analyze some aspects of the entire system and for synthesis tasks (e.g., re-partitioning), a mechanism to compose the processes must be defined. Due to the lack of space we will not present here the POM compositions. In [4] we describe the parallel, sequential, choice and parallel with communications compositions, which permit to obtain the POM of the entire system from the POM of the processes.

In our approach the modeling of a system for co-synthesis consists of the system specification, the design constraints and the hardware/software libraries. The system specification describes the system in terms of a set of modules composed of concurrent communicating processes. Each process consists of local variables and a process body describing the behavior of the process. The behavior is described using sequential statements (e.g., assignment, conditional and iterative constructs) and a fork/join-like construct can be used to specify intra-process concurrency. Each process can send and receive data on the channels and read and write data on the interface of the module.

The behavior of each process is captured by a POM, which abstracts the control/data flow structure of the process specification. In order to define the behavioral representation based on POM first we must define what we consider an event and how the events and the relations between the events can be extracted from the system specification.

We define an event as each assignment of a value to a variable. Thus, in the compiler nomenclature [1], events correspond to three-address codes and each event is represented by a quadruple, where we have the operation, the operands and a result location. Each event has a type, which indicates if the event is an assignment, read/write operation on the interface, or send/receive operations on a channel.

The precedence relation between the events is obtained via the data/control flow analysis [1] of the system specification. Events in different branches of conditionals are modeled using conflict relations. Loops are modeled as an event with an associated POM to represent the body of the loop. The same idea is used to model the hierarchy of function and procedure calls. In [4] we give more details about system modeling using POM.

3. Reachability Analysis

In order to traverse the states of a POM, we need an efficient algorithm to compute the reachable states from a given state.

3.1. Implicit Transition Relation

We can consider the relation between a state \( x_i \) and its immediate successor \( x_j \) as a transition and we adopt the term implicit transition relation since this information is not explicitly represented.

The next states of a given state are its immediate successors in the partial order. Since the successors of a state \( x_i \) contain all the events that occurred in \( x_i \), in the computation of the next states of \( x_i \) we remove from the logical formula the predecessor states of \( x_i \) and the states that do not include the events of \( x_i \). After that we remove the state \( x_i \) and look for the least minterms of the so resulting logical formula. By least minterms we mean the minterms that represent the initial states of a POM. In [4] we present details of this algorithm.

3.2. Explicit Transition Relation

We can also represent explicitly the transitions between the states, when we include for each event \( a \), an event \( a' \) corresponding to the value of \( a \) at the next states. For each relation between events we can have a corresponding logical formula of the explicit transition relation. As before, the transition relation for a process is obtained by the conjunction of these logical formulas. With the explicit transition relation the successors of a given state are computed by a relational product as used in the traditional finite state machine traversal algorithm:

\[
\text{POM_traversal}(\text{POM}, \text{from}) \begin{cases} \text{to} = \text{relational_product}(\text{POM}, \text{from}); \\ \text{unprime}(\text{to}); \\ \text{reduce}(\text{to}); \\ \text{while} (\text{to} != 0) \{ \\ \text{ns} = \text{get_satisfy}; \\ \text{visit_transition}(\text{from}, \text{ns}); \\ \text{POM_traversal}(\text{POM}, \text{ns}); \\ \} \\ \} \end{cases}
\]

In order to attack the state space explosion problem we use partial order reduction techniques [5]. We traverse only the transitions from the state \( \text{from} \) to the states where all independent events have occurred.

![Figure 2. Example of POM for a process](image)
3.3. Estimation of Performance and Cost

At each procedure call visit\_transition(from, ns) we have the sets $E_{\text{from}}$ of events occurred in the state from and $E_{\text{ns}}$ of events occurred in ns. The set $E_I = E_{\text{ns}} - E_{\text{from}}$ contains the events occurred in the transition from the state from to the state to. The events in $E_I$ are independent. We use the following formulas to estimate timing and cost in hardware and software:

$$t_{h\text{w}}(\text{from}, \text{ns}) = \max\{t_{h\text{w}}(\text{e}) | \text{e} \in E_I\}$$

$$T_{h\text{w}} = \max\{t_{h\text{w}}(\text{from}, \text{ns})\}$$

$$c_{h\text{w}}(\text{from}, \text{ns}) = \sum_{\text{e} \in E_I} c_{h\text{w}}(\text{e})$$

$$C_{h\text{w}} = \sum_{\text{e} \in E_I} C_{h\text{w}}(\text{e})$$

At each transition (from, ns) the timing in hardware $t_{h\text{w}}$ is calculated as the maximal execution time of the independent events $E_I$. The total timing $T_{h\text{w}}$ in hardware is the maximal timing of all transitions. The cost in hardware $c_{h\text{w}}$ consists of the sum of the implementation costs of each event in $E_I$ minus the cost of the shared resources. Components already allocated can be reallocated to implement other events. The total cost in hardware $C_{h\text{w}}$ is the sum of the implementation cost of all transitions. The estimation of performance and cost in hardware and software are very similar, except that in the software implementation we must serialize the independent events, i.e., we assume there is no parallelism in software.

4. Co-Synthesis Approach

In our co-synthesis approach, the POM is used as an internal representation for the system specification. On this internal representation are applied the repartitioning and performance analysis algorithms. The co-synthesis tasks include to build a partitioning tree (PT) by regrouping events and joining processes. These operations can be efficiently implemented using logical operations on the BDDs of the processes. The partitioning tree represents the alternative implementations for the system. A performance analysis of these implementations permits to estimate timing and resources information. These information are included in the partitioning tree, which is then used to guide the choice of the implementation that better fits the specified timing and resources constraints.

The partitioning tree has two levels. In the first level the events of the processes are regrouped according to the synchronizations between them. This produces an initial partition that will be the first alternative implementation to be analyzed. In the second level, the processes are composed resulting in new alternative implementations with possibly reduced communication costs (due to the elimination of communication channels). Based on the estimates obtained by the performance analysis of each alternative implementation, we choose one that better satisfies the specified timing and resources constraints.

5. Conclusions

We have presented a formal model for concurrency that permits a concise symbolic representation with efficient algorithms to implement analysis and synthesis based on the model.

Currently, we are implementing the algorithms to build the partitioning tree and to perform the hardware/software partitioning of each suggested partition present in the partitioning tree. We are also working on the implementation of a model checker for POM.

References