Science of Computer Programming 74 (2009) 723–753

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Efficient symbolic computation of process expressions

Benoît Fraikin *, Marc Frappier

GRIL, Département d'Informatique, Université de Sherbrooke, Sherbrooke, Québec, Canada, J1K 2R1

ARTICLE INFO

Article history:
Received 27 February 2006
Received in revised form 13 February 2009
Accepted 14 February 2009
Available online 27 February 2009

Keywords:
Trace-based specifications
Black-box specifications
Process algebra
Information systems
Symbolic computation
Interpreter

ABSTRACT

This paper describes three optimization techniques for the EB^3 process algebra. The optimizations are expressed in a new deterministic operational semantics which is shown to be trace-equivalent to a traditional non-deterministic operational semantics. Internal action transitions are eliminated by an efficient preruntime analysis of the structure of a process expression. Execution environments are used to optimize variable instantiation using lazy evaluation. Non-determinism is eliminated by returning a choice between possible transitions. This new operational semantics is implemented in the EB^3 PAI process algebra interpreter to support the EB^3 method. The goal of this method is to automate the development of information systems using, among other mechanisms, *efficient symbolic computation* of process expressions.

1. Introduction

The project EB^{3}PAI (which stands for EB^3 Process Algebra Interpreter) is part of the APIS research project [1]. The objective of APIS is to support the rapid development of information systems (IS) from formal specifications by using *code generation* and *efficient specification execution*. APIS is based on the EB^3 (Entity-Based Black Box) method [2], which was specifically designed for IS specification.

In our viewpoint, an IS is a software system that helps an organization to collect and manipulate all its relevant data. IS are used in almost all areas of human activities where information must be stored, exploited and analyzed. Typical examples include management IS (e.g. accounting, human resource and production) which are used to support the business process of an organization.

An information system is generally characterized by large persistent data structures which are modified or queried by several users in concurrency. The distinctive characteristics of IS consist in managing complex relationships between data structures, of calculations involving several data structures, of processing large volume of data, and of preserving data integrity through concurrent updates. IS typically have little hard real-time constraints. Modern database management systems provide concurrency control mechanisms which simplify IS development.

An EB^3 specification consists essentially of two parts: (i) a process expression, called **main**, which defines the valid input traces of the IS, (ii) input–output (I–O) rules which assign an output to each input trace. The semantics of an EB^3 specification is given by a relation R defined on $I^+ \times O$, where I^+ denotes the set of non-empty traces defined over input set I and O denotes an output set. Hence, process expression **main** defines the domain of R; in EB^3, a process algebra is used solely to define the inputs.

The EB^3 process algebra is inspired from regular expressions, CSP [3], CCS [4], ACP [5] and LOTOS [6]. For instance, the process expression `a . (b | c)`, where | and . correspond to the well-known regular expression operators choice and

* Corresponding author. Tel.: +1 819 821 8000.
E-mail address: benoit.fraikin@USherbrooke.CA (B. Fraikin).
URL: http://www.dmi.usherb.ca/~gril (B. Fraikin).

0167-6423/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2009.02.002

concatenation, denotes the input traces {a, ab, ac}. Given some input–output rules (omitted here), a relation $R = \{(a \mapsto o_1), (ab \mapsto o_2), (ac \mapsto o_3)\}$ is associated to this specification. This specification means that the IS, from its initial state, must accept user input a and provide output o_1; if some other input is submitted by the user, it must be rejected and the user must be informed by an appropriate error message [7]. After accepting a, the IS must accept user input b and produce output o_2, or accept c and produce output o_3. A system is said to be correct with respect to a specification R if it can accept all input traces t in the domain of R (i.e. a trace of **main**) and produce, for each t, an output o such that $(t \mapsto o) \in R$.

We are currently working on effective tools to support EB^3. The tool EB^{3}PAI is an interpreter for EB^3 process expressions. EB^{3}PAI relies on several optimization techniques to handle non-deterministic process expressions, internal actions and quantified operators like choice and parallel composition with synchronization.

EB^{3}PAI executes an action by applying the transition rules of an operational semantics (in the Plotkin style used for CCS [4]) defined for the EB^3 process algebra. Basically, EB^{3}PAI efficiently computes on the fly a proof of a transition $E \xrightarrow{\sigma} E'$ to determine whether process expression E can accept user input σ. If σ can be accepted, then E' becomes the resulting process expression on which the next input is applied; otherwise σ is discarded and the current process expression does not change. Hence, EB^{3}PAI does not generate executable code to execute a process expression; rather, it is itself an abstract machine that executes a process expression. The state of the abstract machine is the abstract syntax tree (AST) of E.

The original operational semantics of the EB^3 process algebra, proposed in [2], is not adequate for an efficient symbolic computation. This paper proposes a new set of transition rules on which EB^{3}PAI is based. The transition rules of [2] suffer from three main problems.

First, they allow non-determinism, which means that an action can sometimes be executed by several transitions, leading to different process expressions. Since the EB^3 process expression **main** defines the traces that must be accepted by the IS, an interpreter must find the appropriate execution path to accept a given trace. A naive interpreter based on the rules of [2] must sometimes backtrack and try other execution paths for past (accepted) actions, in order to accept a new one. Note that we are dealing here with *process expression non-determinism*, which is distinct from *I–O rules non-determinism*. I–O rules allows for the specification of several outputs for a given input trace, which is sometimes desirable for IS specification. For instance, in a travel agency, the choice of the ordering for a list of flights which match a set of criteria maybe non-deterministic.

Second, internal actions, which are not visible to the environment, can also require the interpreter to backtrack, or they can induce infinite loops (divergence) when trying to execute an action.

Third, the rules of [2] use syntactic substitution on the AST, which means that every occurrence of a variable is replaced by its substituted term. This can lead to significant overhead in transition computation and high memory usage for large interleave quantifications, because each interleaved process differs from the others only in the substituted text (*cf.* Fig. 4 in Section 4.2).

The proposed set of rules is proved to be trace-equivalent to the one defined in [2]. These new rules are more complex, because they are meant to be used for efficient execution. They have been implemented in EB^{3}PAI in order to evaluate their efficiency from an experimental standpoint, taking into account practical implementation issues like persistency of large ASTs, large quantification sets, and memory usage in order to minimize redundancy.

For various patterns of IS ([2], Section 5) which are derived from the structure of the business model (entity-relationship model), EB^{3}PAI can execute an action in linear time with respect to the size of the specification (i.e. the number of terms and operators in the process expression) and logarithmic time with respect to the number of entities of an entity type in the business model. The current version of EB^{3}PAI is implemented in Java; it uses the OODBMS ObjectStore PSE PRO (which is also implemented in Java) to handle the persistency of its internal state (i.e. an AST) and large collections of objects.

A companion paper [8] proposes algorithms to efficiently execute large interleave quantifications, which are fundamental components of an IS. Large interleave quantifications are used to model the entities (i.e. instances or objects) of an entity type (i.e. class) and the relationships between entity types. An entity type in an IS can easily contain thousands of entities.

The APIS framework supports the EB^3 method; it includes EB^{3}PAI and other components which are illustrated in Fig. 1. A complete EB^3 specification includes five elements represented in the upper part of the figure. The user interacts with the IS through a web interface generated by DCI-WEB [9] from a formal specification of the user interface interaction. The web interface calls EB^{3}PAI to determine if the user input is valid. EB^{3}PAI tries to execute this input event on the process expression. If it succeeds, it calls an update program which has been generated by EB^{3}TG [10] to update a relational database that contains the value of IS entity attributes and then calls a query program to compute the output associated to this input event; if EB^{3}PAI fails to accept the input event, it reports an informative message to explain the error to the user. Entity attributes are formally specified by recursive functions on the set of traces accepted by **main**. Entity types are defined by an entity-relationship (ER) diagram. Component EB^{3}IO is under development.

The EB^3 process algebra differs in a number of aspects from traditional process algebras, in order to streamline the specification of IS. The first important distinction, as illustrated in Fig. 1, is that outputs are not specified using the process algebra, but from recursive functions defined on input traces. A process algebra provides operators to define ordering constraints on actions that can communicate with the environment; it does not include state variables like those found in a state-machine language like B [11] or Z [12]. It has been recognized by several authors that data management is hard to specify using solely a process algebraic approach. A number of proposals were made to combine a process algebra with a state-machine specification language to manage data. The key idea is that process algebra operators define the ordering of actions; state variables and state-machine operations manage the data. The CSP‖B [13] approach combines a CSP specification with a B specification [11]. CSP actions are matched with B operations; when a CSP action is executed,