Information Processing Letters 135 (2018) 47–52

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Unifying theories of time with generalised reactive processes

Simon Foster[*], Ana Cavalcanti, Jim Woodcock, Frank Zeyda

Department of Computer Science, University of York, York, YO10 5DD, United Kingdom

A R T I C L E   I N F O

Article history:
Received 3 May 2017
Received in revised form 15 December 2017
Accepted 15 February 2018
Available online xxxx
Communicated by J.L. Fiadeiro

Keywords:
Formal semantics
Hybrid systems
Process algebra
Unifying theories
Theorem proving

A B S T R A C T

Hoare and He's theory of reactive processes provides a unifying foundation for the formal semantics of concurrent and reactive languages. Though highly applicable, their theory is limited to models that can express event histories as discrete sequences. In this paper, we show how their theory can be generalised by using an abstract trace algebra. We show how the algebra, notably, allows us to consider continuous-time traces and thereby facilitate models of hybrid systems. We then use this algebra to reconstruct the theory of reactive processes in our generic setting, and prove characteristic laws for sequential and parallel processes, all of which have been mechanically verified in the Isabelle/HOL proof assistant.

1. Introduction

The theory of reactive processes provides a generic foundation for the denotational semantics of concurrent languages. It is part of the Unifying Theories of Programming (UTP) [1,2] framework, which models computation using predicate calculus. The theory of reactive processes unifies formalisms such as CSP [3], ACP [4], and CCS [5]. This is possible due to its support for a large set of algebraic theorems that universally hold for families of reactive languages. The theory has been extended and applied to several languages including, stateful [6] and real-time languages, with both discrete [7] and continuous time [8,9].

Technically, the theory's main feature is its trace model, which provides a way for a process to record an interaction history, using an observational variable $tr : \mathrm{seq}\ Event$. In the original presentation, a trace is a discrete event sequence, which is standard for languages like CSP. The alphabet can be enriched by adding further observational variables; for example, $ref : \mathbb{P}\ Event$ to model refusals [1].

Though sequence-based traces are ubiquitous for modelling concurrent systems, other models exist. In particular, the sequence-based model is insufficient to represent continuous evolution of variables as present in hybrid systems. A typical notion of history for continuous-time systems are real-valued trajectories $\mathbb{R}_{\geq 0} \to \Sigma$ over continuous state Σ.

Although the sequence and trajectory models appear substantially different, there are many similarities. For example, in both cases one can subdivide the history into disjoint parts that have been contributed by different parts of the program, and describe when a trace is a prefix of another. By characterising traces abstractly, and thus unifying these different models, we provide a generalised theory of reactive processes whose properties, operators, and laws can be transplanted into an even wider spectrum of languages. We thus enable unification of untimed, discrete-time, and continuous-time languages. The focus of our theory is on traces of finite length, but the semantic framework is extensible.

We first introduce UTP and its applications (§2). We then show how traces can be characterised algebraically by a form of cancellative monoid (§3), and that this algebra encompasses both sequences and piecewise continuous functions (§4). We apply this algebra to generalise the the-

[*] Corresponding author.
E-mail address: simon.foster@york.ac.uk (S. Foster).

https://doi.org/10.1016/j.ipl.2018.02.017
0020-0190/© 2018 Elsevier B.V. All rights reserved.

ory of reactive processes, and show that its key algebraic laws are retained in our generalisation, including those for sequential and parallel composition (§5).

Our work is mechanised in our theorem prover, Isabelle/UTP[1] [10], a semantic embedding of UTP in Isabelle/HOL. We sometimes give proofs, but these merely illustrate the intuition, with the mechanisation being definitive. To the best of our knowledge, this is the most comprehensive mechanised account of reactive processes.

2. Background

UTP is founded on the idea of encoding program behaviour as relational predicates whose variables correspond to observable quantities. Unprimed variables (x) refer to observations at the start, and primed variables (x') to observations at a later point of the computation. The operators of a programming language are thus encoded in predicate calculus, which facilitates verification through theorem proving. For example, we can specify sequential programming operators as relations:

$$\begin{aligned} x := v &\triangleq x' = v \wedge y_1' = y_1 \wedge \cdots \wedge y_n' = y_n \\ P \,;\, Q &\triangleq \exists x_0 \bullet P[x_0/x'] \wedge Q[x_0/x] \\ P \lhd b \rhd Q &\triangleq (b \wedge P) \vee (\neg b \wedge Q) \end{aligned}$$

Assignment $x := v$ states that x' takes the value v and all other variables are unchanged. We define the degenerate form $I\!I \triangleq x := x$, which identifies all variables. Sequential composition $P \,;\, Q$ states that there exists an intermediate state x_0 on which P and Q agree. If-then-else conditional $P \lhd b \rhd Q$ states that if b is true, P executes, otherwise Q.

UTP variables can either encode program data, or behavioural information, in which case they are called *observational* variables. For example, we may have $ti, ti' : \mathbb{R}_{\geq 0}$ to record the time before and after execution. These enrich the semantic model, and are constrained by *healthiness conditions* that restrict permissible behaviours. For example, we can impose $ti \leq ti'$ to forbid reverse time travel.

Healthiness conditions are expressed as functions on predicates, such as $\mathbf{HT}(P) \triangleq P \wedge ti \leq ti'$, the application of which coerces predicates to healthy behaviours. When they are idempotent and monotonic, with respect to the refinement order $\sqsubseteq$, we can show, with the aid of the Knaster–Tarski theorem, that their image forms a complete lattice, which allows us to reason about recursion.

Healthiness conditions are often built from compositions: $\mathbf{H} \triangleq \mathbf{H}_1 \circ \mathbf{H}_2 \circ \cdots \circ \mathbf{H}_n$. In this case, idempotence and monotonicity of $\mathbf{H}$ can be shown by proving that each $\mathbf{H}_i$ is monotonic and idempotent, and each $\mathbf{H}_i$ and $\mathbf{H}_j$ commute. A set of healthy fixed-points, $[\![\mathbf{H}]\!] \triangleq \{P \mid \mathbf{H}(P) = P\}$, is called a *UTP theory*. Theories isolate the aspects of a programming language, such as concurrency, object orientation, and real-time programming. Theories can also be combined by composing their healthiness conditions to enable construction of sophisticated heterogeneous and integrated languages.

Our focus is the theory of reactive processes, with healthiness condition $\mathbf{R}$, which we formalise in Section 5. Reactive programs, in addition to initial and final states, also have intermediate states, during which the process waits for interaction with its environment. $\mathbf{R}$ specifies that processes yield well-formed traces, and that, when a process is in an intermediate state, any successor must wait for it to terminate before interacting. This theory uses observational variable *wait* to differentiate intermediate from final states, and *tr* to record the trace.

UTP theories based on reactive processes have been applied to give formal semantics to a variety of languages [1, 11,12], notably the *Circus* formal modelling language family [6], which combines stateful modelling, concurrency, and discrete time [7,13]. A similar theory has been used for a hybrid variant of CSP [9], with a modified notion of trace. Though sharing some similarities, these various versions of reactive processes are largely disjoint theories with distinct healthiness conditions. Our contribution is to unify them all under the umbrella of *generalised reactive processes*.

3. Trace algebra

In this section, we describe the trace algebra that underpins our generalised theory of reactive processes. We define traces as an abstract set $\mathcal{T}$ equipped with two operators: trace concatenation $^\frown : \mathcal{T} \to \mathcal{T} \to \mathcal{T}$, and the empty trace $\varepsilon : \mathcal{T}$, which obey the following axioms.

Definition 3.1 *(Trace algebra).* A trace algebra $(\mathcal{T}, ^\frown, \varepsilon)$ is a cancellative monoid satisfying the following axioms:

$$x ^\frown (y ^\frown z) = (x ^\frown y) ^\frown z \tag{TA1}$$

$$\varepsilon ^\frown x = x ^\frown \varepsilon = x \tag{TA2}$$

$$x ^\frown y = x ^\frown z \;\Rightarrow\; y = z \tag{TA3}$$

$$x ^\frown z = y ^\frown z \;\Rightarrow\; x = y \tag{TA4}$$

$$x ^\frown y = \varepsilon \;\Rightarrow\; x = \varepsilon \tag{TA5}$$

As expected, $^\frown$ is associative and has left and right units. Axioms TA3 and TA4 show that $^\frown$ is injective in both arguments. As an aside, TA3 and TA4 hold only in models without infinitely long traces, as such a trace x would usually annihilate y in $x ^\frown y$. Axiom TA5 states that there are no "negative traces", and so if x and y concatenate to ε then x is ε. We can also prove the dual law: $x ^\frown y = \varepsilon \;\Rightarrow\; y = \varepsilon$. From this algebraic basis, we derive a prefix relation and subtraction operator.

Definition 3.2 *(Trace prefix and subtraction).*

$$x \leq y \;\Leftrightarrow\; \exists z \bullet y = x ^\frown z$$

$$y - x \;\triangleq\; \begin{cases} \iota z \bullet y = x ^\frown z & \text{if } x \leq y \\ \varepsilon & \text{otherwise} \end{cases}$$

Trace prefix, $x \leq y$, requires that there exists z that extends x to yield y. Trace subtraction $y - x$ obtains that trace z when $x \leq y$, using the definite description operator

[1] https://github.com/isabelle-utp/utp-main.