ON THE COMPOSITIONAL CHECKING OF VALIDITY
(Extended Abstract)
Glynn Winskel
Computer Science Department, Aarhus University, Denmark

Introduction

This paper is concerned with deciding whether or not assertions are valid of a parallel process using methods which are directed by the way in which the process has been composed. The assertions are drawn from a modal logic with recursion, capable of expressing a great many properties of interest [EL]. The processes are described by a language inspired by Milner's CCS and Hoare's CSP, though with some modifications. The choice of constructors allows us to handle a range of synchronisation disciplines and ensures that the processes denoted are finite state. The operations are prefixing, a non-deterministic sum, product, restriction, relabelling and a looping construct. Arbitrary parallel compositions are obtained by using a combination of product, restriction and relabelling.

We are interested in deciding whether or not an assertion $A$ is valid of a process $t$. If it is valid, in the sense that every reachable state of $t$ satisfies $A$, we write $\models A : t$. Rather than perform the check $\models A : t$ monolithically, on the whole transition system denoted by the term $t$, we would often rather break the verification down into parts, guided by the composition of $t$. For instance if $t$ were a sum $t_0 + t_1$ we can ask what assertions $A_0$ and $A_1$ should be valid of $t_0$ and $t_1$ respectively to ensure that $A$ is valid of $t_0 + t_1$. This amounts to requiring assertions $A_0, A_1$ such that

$$\models A : t_0 + t_1 \iff \models A_0 : t_0 \text{ and } \models A_1 : t_1.$$

Once the assertions $A_0$ and $A_1$ are found, a validity problem for $t_0 + t_1$ is reduced to a problem to do with $t_0$ and another with $t_1$. Further, if the assertions can be found routinely only knowing the top-level operation, that e.g. the process is a sum, we are also told how to construct a process as a sum for which the assertion $A$ is valid: first find components $t_0$ and $t_1$ making $A_0$ and $A_1$ valid respectively.

This paper investigates the extent to which the composition of $t$ can guide methods for deciding $\models A : t$. It formulates new compositional methods for deciding validity, and exposes some fundamental difficulties. Algorithms are provided to reduce validity problems for prefixing, sum, relabelling, restriction and looping to validity problems for their immediate components—all these reductions depend only on the top-level structure of terms. The existence of these reductions rests on being able to 'embed' the properties of a term in the properties, or products of properties, of its immediate subterms. Because there is not such a simple embedding for the product construction of terms, as might be expected, similar reductions become much more complicated for products; although there are general results, and the reductions can be simple in special cases, the general treatment for products meets with fundamental difficulties. Whereas reductions for products always exist for this finite state language, they demonstrably no longer just depend on the top-level (product structure) of the term; in particular, a simple assertion is exhibited for which the size of the reduction must be quadratic in the number of states of the process. An attempt is thus made to explain what makes product different from the other operations with respect to compositional reasoning, and to delimit the obstacles to automated compositional checking of validity on parallel processes.
1 Transition systems and properties

The syntax, presented formally in the next section, will consist of process terms and assertions.

Process terms will denote labelled transition systems with distinguished initial states. A labelled transition system is a structure \((S, i, L, \text{tran})\) where \(S\) is a set of states containing a distinguished state \(i\), \(L\) is a set of labels, and \(\text{tran} \subseteq S \times L \times S\) is a set of transitions; as normal, we often write \(s \xrightarrow{\alpha} s'\) if \((s, \alpha, s') \in \text{tran}\). A state of a labelled transition system is reachable if it can be obtained as the end state of a sequence of transition beginning at the initial state.

A closed assertion is to denote a property of a labelled transition system, i.e., a subset of its reachable states. We write \(P(T)\) for the set of properties of a labelled transition system \(T\).

We construct labelled transition systems using the constructions of prefixing, sum, product, restriction, relabelling and looping starting from the nil process. These operations form the basis of our syntax for processes. We now describe these constructions. As has been stated, properties of a labelled transition system are identified with subsets of reachable states. The constructions in our language of transition systems are associated with maps. These prove useful in importing properties of immediate components of a term into a property of the term itself. Such mappings between properties are a key to compositional reasoning about processes. We introduce them alongside the constructions with which they are associated.

\textbf{nil}: The nil transition system is \(\{(i), i, \emptyset, \emptyset\}\).

\textbf{Prefixing}: For a label \(\alpha\) and a labelled transition system \(T = (S, i, L, \text{tran})\) the prefix \(\alpha T\) is obtained by adjoining a new initial state and introducing an \(\alpha\)-transition from it to the old initial state. More concretely:

\[
\alpha T = (S', \emptyset, L \cup \{\alpha\}, \text{tran}')
\]

where \(S' = \{\{s\} | s \in S\} \cup \{\emptyset\}\), and

\[(s_1, \beta, s'_1) \in \text{tran}' \text{ iff } (s_1 = \emptyset \& \beta = \alpha \& s'_1 = \{i\}) \text{ or } (s_1 = \{s\} \& s'_1 = \{s'\} \& (s, \beta, s') \in \text{tran}, \text{ for some } s, s')
\]

There is map \(S \to S'\) taking \(s \in S\) to the corresponding state \(\{s\} \in S'\). It extends to a map on properties \(P(T) \to P(\alpha T)\). It is convenient to name this map on properties after the prefixing operation and we define

\[
\alpha(-) : P(T) \to P(\alpha T)
\]

by taking \(\alpha U = \{\{s\} | s \in U\} \text{ for } U \in P(T)\).

\textbf{Sum}: Let \(T_0 = (S_0, i_0, L_0, \text{tran}_0)\) and \(T_1 = (S_1, i_1, L_1, \text{tran}_1)\) be labelled transition systems. Our nondeterministic sum operation \(T_0 + T_1\) is a little different from Milner's. It identifies disjoint copies of the transition systems at their initial states. We define

\[
T_0 + T_1 = ((S_0 \times \{i_1\}) \cup \{i_0\} \times S_1), (i_0, i_1), L_0 \cup L_1, \text{tran}')
\]

where

\[(\{s\}, \alpha, (s', i_1)) \in \text{tran}' \text{ iff } (s, \alpha, s') \in \text{tran}_0\]

and

\[(i_0, s), \alpha, (i_0, s') \in \text{tran}' \text{ iff } (s, \alpha, s') \in \text{tran}_1.
\]