Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 319 (2015) 387–401

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

The Expressiveness of CSP With Priority

A.W. Roscoe[1]

Oxford University Department of Computer Science

Abstract

The author previously [16,15] defined *CSP-like* operational semantics whose main restrictions were the automatic promotion of most τ actions, no cloning of running processes, and no negative premises in operational semantic rules. He showed that every operator with such an operational semantics can be translated into CSP and therefore has a semantics in every model of CSP. In this paper we demonstrate that a similar result holds for CSP extended by the priority operator described in Chapter 20 of [15], with the restriction on negative premises removed.

Keywords: CSP, operational semantics, priority

1 Introduction

As well as its denotational semantics in models such as traces $\mathcal{T}$ and failures-divergences $\mathcal{N}$, CSP [11] has a well-established operational semantics first described in SOS in [5,6], and congruence with that is perhaps the main criterion for the acceptability of any new semantic model.

The author previously created a class of *CSP-like* operational semantic definitions that automatically have semantics over every CSP model. In addition to a number of other restrictions on the full generality of Structured Operational Semantic (SOS) definitions, CSP-like ones are not permitted any negative premises: thus there can be no rule in which some action can fire only if one of its arguments *can not* perform some (either one or more) action(s).

There have been a number of proposals for adding priority to CSP. A straightforward one, because it does not involve building special semantic models or types of LTSs, was proposed in [15]. $\mathbf{Pri}_{\leq}(P)$, for a partial order on the events that processes perform, permits P an event x only when no higher priority event is possible. With restrictions on how the invisible event τ fits into $\leq$, this adds very usefully to CSP, for example by permitting the accurate description of real-time systems.

[1] Email: bill.roscoe@cs.ox.ac.uk

http://dx.doi.org/10.1016/j.entcs.2015.12.023

$\mathbf{Pri}_{\leq}(\cdot)$ is not CSP-like since it requires negative premises. Indeed it does not have a semantics in most CSP models. This raises the question of whether we can capture a notion of *Pri-CSP-like* operational semantics which includes this operator, where all Pri-CSP-like operators can be expressed in terms of CSP plus $\mathbf{Pri}_{\leq}(\cdot)$. Establishing such a notion is the job of the present paper.

In the next section, we remind ourselves about CSP and its operational semantics. We then recall CSP-like operational semantics and outline their expressiveness result. Finally we recall the definitions of $\mathbf{Pri}_{\leq}(\cdot)$ in terms of operational semantics and over $\mathcal{FL}$, the *finite linear* or *ready traces* model that can record an acceptance set before each event. In Section 3 we generalise the definition of *CSP-like* to achieve the goal set out above. The main result of this paper then follows, in which we show that any operator (or class of operators) with such Pri-CSP-like operational semantics can be simulated precisely in augmented CSP. The precision obtained by this simulation depends on whether or not the language involves the CSP concept of termination, represented ✓. However, for brevity this paper does not include the role of ✓ in CSP semantics: it is fully covered in the extended version [18].

As with [16], the primary motivation of this paper is to characterise what operators and languages can be translated into CSP (in this paper extended by $\mathbf{Pri}_{\leq}(\cdot)$) to identify which of these can be handled on the model checker FDR [9], which itself now supports this operator [2]. We give some examples of what is now representable in Section 5.

2 Background

2.1 The operational semantics of CSP

The SOS operational semantics [5,6] of CSP came along after its well-known denotational semantics. For CSP (without ✓ and sequential composition), the action labels come from $\Sigma \cup \{\tau\}$, where Σ is the *alphabet*, the actions that are visible to and controllable by the external observer, and τ is an invisible and uncontrollable event such that whenever it is enabled and another event does not happen quickly, it will. Given the process P, αP means its own set of Σ actions, which is usually just the visible events it uses.

In SOS style [13] we need rules to infer every action that each process can perform. The conditions that enable actions can be of three sorts:

- *Positive:* Some other process can perform a specific action. This other process is determined from the syntax of the process P whose transitions we are calculating. In our setting these other processes are, except in the case of recursion, arguments of the operator whose semantics we are defining.
- *Negative:* The same except the other process cannot perform a given action.
- *Side conditions* on the actions etc that appear.

[2] FDR3 supports two priority operators: `prioritisepo` is directly equivalent to the one used in this paper, while `prioritise` is a restricted case that does not require the programmer to construct an explicit partial order.