Communicating Process Architectures 2004
Ian East, Jeremy Martin, Peter Welch, David Duce, and Mark Green (Eds.)
IOS Press, 2004

CSP: The Best Concurrent-System Description Language in the World – Probably!

Extended Abstract

Michael GOLDSMITH
Formal Systems (Europe) Ltd, 26 Temple Street, Oxford OX4 1JS, UK
Worcester College, Oxford, UK
michael@fsel.com

Abstract. CSP, Hoare's *Communicating Sequential Processes*, [1, 2] is one of the formalisms that underpins the antecedents of CPA, and this year celebrates its Silver Jubilee [3]. Formal Systems' own FDR refinement checker [4] is among the most powerful explicit exhaustive finite-state exploration tools, and is tailored specifically to the CSP semantics. The CSP_M ASCII form of CSP, in which FDR scripts are expressed, is the de-facto standard for CSP tools. Recent work has experimentally extended the notation to include a probabilistic choice construct, and added functionality into FDR to produce models suitable for analysis by the Birmingham University *PRISM* tool [5].

1 Introduction

The motivation for this work is provided by the increasing maturity of model-checking approaches to probabilistic analysis of concurrent systems, and the desire to move beyond the black-and-white precision of pure CSP refinement to richer grey-scale measures of confidence. Such probabilistic analysis is appropriate to 'natural' interference with a system, but may have less application in the case where deliberate malevolence is potentially involved: but even in this case it may play a role in quantifying the effect of simplifying assumptions, such as the improbability of guessing nonces or keys, or where an assailant can himself take advantage of probabilistic analysis [6, for example].

A number of attempts have been made to extend the theory of CSP with probabilistic constructs, from the original work of Lowe [7] to more recent efforts by Morgan *et al* [8]. There is an unfortunate tension between nondeterministic choice representing underspecification and probabilistic choice denoting some kind of run-time resolution. The classic refinement paradox of information-flow illustrates that these are very different concepts:

$$\mathit{SECURE} = \mathit{high}?\_ : \{0,1\} \rightarrow \; (\mathit{low}!0 \rightarrow \mathit{SECURE} \;{}_{0.5}\!\boxplus_{0.5}\; \mathit{low}!1 \rightarrow \mathit{SECURE})$$

(where the ${}_{0.5}\!\boxplus_{0.5}$ operator represents a fair coin toss) is reasonably free of information flow (allowing only the detection of activity on the *high* channel, not what that activity is); but if we replace the probabilistic choice with a nondeterministic one, then the resulting process is refined by

$$\mathit{INSECURE} = \mathit{high}?x : \{0,1\} \rightarrow \mathit{low}!x \rightarrow \mathit{INSECURE}$$

which patently is not!

There has so far been no entirely satisfactory treatment for combining the two concepts: either probabilistic choice fails to distribute over nondeterminism, or else some other desirable laws have to be sacrificed: in [8], since all probabilistic choices can effectively be made at process initiation, we have that one can detect when different instances of a process exist. Thus in general we have that P strictly probabilistically-refines $P \sqcap P$. For example, if we define

$$P = a \rightarrow STOP \; {}_{0{\cdot}5}\boxplus_{0{\cdot}5} \; b \rightarrow STOP$$

by 'multiplying out' the choices we get

$$P \sqcap P = (a \rightarrow STOP \; {}_{0{\cdot}5}\boxplus_{0{\cdot}5} \; b \rightarrow STOP) \; {}_{0{\cdot}5}\boxplus_{0{\cdot}5} \; (a \rightarrow STOP \sqcap b \rightarrow STOP)$$

which has only a probability of $0{\cdot}25$ of behaving like $a \rightarrow STOP$, as compared with the $0{\cdot}5$ probability in P.

While this treatment does give a well defined meaning to refinement, if we recall that $P \sqsubseteq Q$ is normally characterised as $P \sqcap Q = P$, there is clearly something a little counterintuitive going on (and that equivalence cannot hold, if $\sqsubseteq$ is to be reflexive).

We largely sidestep these difficulties by restricting our attention to a particular idiom of probabilistic analysis, which should nevertheless be rich enough to provide interesting and useful results.

1.1 PRISM *Language*

PRISM supports a variety of probabilistic models: Discrete-Time Markov chains (DTMC), Continuous-Time Markov Chains (CTMC), and Markov Decision Processes (MDP). MDP support nondeterministic scheduling, but more importantly for our purposes, we need MDP for nondeterminism.

The *PRISM* language, like FDR's view of CSP, combines one or more 'modules' (leaf processes) using a variety of high-level process operators. The semantics of leaf processes are described in terms of 'Labelled Transition Systems', but these are rather dissimilar to the familiar event-labelled ones underlying the operational semantics of CSP_M: the nodes in the graph are particular assignments of values to state variables, and the arcs are labelled with probabilities. In fact, one should strictly distinguish two levels of arc: the first selecting nondeterministically between probability distributions, and then a partition of the probability space between the arcs within that distribution.

In an important extension to this scheme, the choice of distribution from any node may be constrained by labelling it with a named action. This makes that choice available only when it is also available in the zero or more other modules with which it must synchronise, according to the high-level system composition.

The tests that we support are to calculate the maximum and minimum probability, over all nondeterministic schedulers and environmental choice of events offered deterministically, that a counterexample state to some refinement query can be reached. In picking a path through these (internal and external) choices, the *PRISM* semantics allows the scheduling dæmon to take note of how any probabilistic choice is resolved, which the choice in question does not causally *happen-before* it (and which might therefore be scheduled to happen before it, in a non-causal sense).

This gives a quite different view of the world to that in [8], where the dæmon has to pick a path before any of the probabilistic choices is made. We recover idempotence of nondeterministic choice, but probabilistic choice still does not distribute over nondeterministic.

2 Extended Syntax

CSP_M is already endowed with two choice operators: '`|~|`' which represents internal (nondeterministic) choice, and '`[]`', which allows the environment to choose the initial event of the combined process and so control which of the two arguments proceeds to evolve; unless the event is common to both, in which case which 'wins' is nondeterministic.

The probabilistic choice operator '$_p\boxplus_{1-p}$' lies conceptually somewhere in between, so we have chosen a syntax reminiscent of both. When one considers the role that dot '`.`' plays in the formation of events and datatype values in CSP_M, it rapidly becomes clear that adding support for floating-point numerals is likely to be fraught with sorrow; so rather than probabilities directly, we allow probabilities of each branch to be expressed by an integer weight. Thus

$$P \ [m \tilde{} n] \ Q$$

corresponds to what we have been writing

$$P \ {}_{\frac{m}{m+n}}\boxplus_{\frac{n}{m+n}} \ Q$$

and we can express $P \ {}_{0{\cdot}5}\boxplus_{0{\cdot}5} \ Q$ as

$$P \ [1 \tilde{} 1] \ Q$$

or even

$$P \ [50 \tilde{} 50] \ Q$$

as we might express it colloquially.

Both the choice operators have distributed forms:

$$|\tilde{}| \ i : I @ P(i)$$
$$[] \ i : I @ P(i)$$

where in general an arbitrarily complex collection of generators and filter expressions can occur between the choice operator and the '`@`'.

For probabilistic choice, we have a similar construct:

$$\tilde{} \ i : I @ [w(i)] \ P(i)$$

where the weights $w(i)$ can also vary with the 'loop index' i. Here, as with nondeterministic choice, it doesn't make much sense if the generators and filters give rise to an empty construct. Otherwise the probability of each branch is its weight divided by the sum of the weights across all admitted values of the loop index.

Currently the bridge between CSP_M and *PRISM* is mediated by special forms of assertion in FDR:

`assert` *Spec* `[T=` *Impl* `:[probabilistic translation]`

`assert` *Spec* `[F=` *Impl* `:[probabilistic translation]`

Running such an assertion uses FDR's compiler and some of its other internal machinery to generate a *PRISM* model file, which can then be loaded manually into *PRISM* and checked against some supplied PCTL formulæ. It is clearly desirable in the longer term to automate

this process, and to provide a route back for analysing any counterexample with the FDR debugger.

In both of these assertions, *Spec* must be a normal (in a nontechnical sense) nonprobabilistic process, while *Impl* is allowed to (and presumably does, as otherwise FDR would vastly outperform the route via *PRISM* to checking the refinement) contain probabilistic choices. The fact that *Spec* is nonprobabilistic means that it can unproblematically be normalised (in the technical sense of reducing it to a deterministic transition system, while retaining sufficient annotations to be able to reconstitute its nondeterministic behaviour), which is key to efficient refinement checking.

3 Translation

The translation of an individual leaf process into a *PRISM* module is relatively straightforward: as within FDR, we tabulate the transitions as between numbered states, and supply the resulting module with a program-counter variable which can be updated accordingly. (Each module requires its own distinct variable, as 'ownership' of a variable controls only which module can modify it, not its visibility, which is global.) Visible events correspond to transitions with a synchronisable action label; internal (τ-) transitions to anonymous, autonomous ones; and probabilistic choice gives rise to a probabilistic update of the state variable. For a variety of technical reasons, we forbid the (external) choice between a probabilistic choice and any other action.

The `system` section of a *PRISM* model now admits a rich enough set of operators to permit direct translation of the high-level operator tree, so we can simply enough create an overall model of the implementation *Impl*; but there is no direct way to compare that, within *PRISM*, with an analogous translation of *Spec*.

In fact we use a different strategy, based on another situation where we want to approach a refinement query indirectly.

3.1 Watchdog Transformation

It has long been known [9] that the hierarchical compression operators provided by FDR tend to work best when there is a lot of hidden activity that can be compressed away. One evident possible route towards maximising this is somehow to move the specification over to the right-hand-side of the refinement, in such a way that the resulting check is invariant under hiding; and then hide everything!

We have shown [10, 11] that this is indeed possible, and we have addressed the issue that it is not just hidden activity, but rather localised hidden activity, which is necessary in order to get real benefit from hierarchical compression. Unfortunately the simple execution of the watchdog transformation, where the specification is transformed into a monitor process which signals a failure of refinement either through an error-flag event or by deadlocking the system (in the traces and failures models respectively), yields a system where the hidden events are nearly all shared immediately below the outermost hiding, so that virtually no extra compression is obtained. We explain in the cited works how the syntax tree can be rebalanced to solve this problem, but CSP operators are generally not precisely associative and do not commute with one another, so the transformation is quite intricate.

More recently, to be reported in [3], we have been able to take advantage of another part of FDR's internal machinery, the supercompiler. Any CSP operator tree can be transformed, leaving the leaf processes untouched, into an equivalent one (unique up to reordering and choice of new event names) that uses only outward (inverse-functional) renaming at the

leaves, 'natural' alphabetised parallel, and functional renaming and hiding at the outermost level. Using this transformation, which FDR already makes use of for efficient exploration of the operational semantics of the system, allows the system to be expressed in a form which can be reordered and rebracketed at relatively little cost in either CPU cycles or, more importantly, intellectual effort.

In the interests of code re-use, as much as anything else, exactly the same approach has been followed in performing a watchdog transformation on the specification for *PRISM*. The only real difference is that the global visibility of *PRISM* variables and the fact that the actual specification property in the *PRISM* analysis is a PCTL formula which is expressed in terms of them together allow a slight simplification in detecting failure of refinement. In particular, the use of the interrupt operator in [11, §4] can be avoided. Some quite neat encodings have been found of, for instance, the slices through the minimal acceptances in the failures-model watchdog, and the resulting *PRISM* code is quite compact (if not much more readable than most autogenerated code).

4 Conclusions and Further Work

Only limited amounts of experimentation have been performed at the time of writing, but the results are encouraging: a variety of small technical examples have been checked to validate the translation (successfully), and a version of the classic Alternating-Bit Protocol with probabilistic media has been shown to refine its specification (as a small buffer) with probability 1. By the time of the conference there should be substantial results from its application to ad-hoc routing protocol analysis within the *FORWARD* project [12].

As mentioned above, a more streamlined workflow is desirable, and a facility for back-annotation into the FDR debugger. But it is here that the fact that we have used the FDR supercompiler may come into its own: given a path in terms of the state variables of the *PRISM* model, all that is required is to drop the special control variables (marking the presence of a trace error, or controlling the minimal acceptances of the watchdog), and the result is *precisely* the path that FDR itself would have generated internally from a failed refinement check, and passed to the debugger.

One less attractive feature of this approach is that all regularity in the input process has been lost as a side-effect of the compilation process: neither state numbers nor event numbers reflect any symmetry or structure in the leaf processes. Since it is precisely this regularity which enables BDD-related technology to operate effectively on large problems (and *PRISM* is based on Multi-Terminal BDDs), this bodes ill for scaling the approach to significant examples; it remains to be seen how significant an issue this is in practice. It may be possible to address this likely problem by a symbolic compilation strategy, which can preserve the structure of process definitions and of the corresponding data components of events; a pilot version of such a tool is currently under development.

The purely CSP watchdog transformation scheme has difficulty handling the full failures-divergences model, since it is not possible for the watchdog to stop the implementation from diverging when the specification wants to allow it to (and so should stop it). It is also hardly worth adopting any of the rather more intrusive transformations that would allow this to be simulated, since there are no significant (fixed-specification) refinement queries in that model which are invariant under hiding, so the exercise would be more than a little academic. Neither of these considerations apply in the case of the *PRISM* models, however: it is easy for the watchdog to set a variable to inhibit any further action by the implementation, as part of its entry into a state corresponding to a divergent state of the implementation; and PCTL can express the necessary eventualities to capture livelock-freedom of the combination. So this is an enhancement to be anticipated.

However useful the bridge between the two tools proves in the long run, the transformation has a certain elegance of its own. It illustrates once more the unintended extra utility of some of the constructs within FDR, whose sole motivation were efficiency and robustness in refinement checking.

5 Acknowledgements

Much of the work in this paper was carried out as part of the DTI Next Wave Technologies and Markets project *FORWARD* [12], building upon research undertaken for QinetiQ Trusted Information Management System Assurance Group.

The assistance of the *PRISM* design team at Birmingham University, in particular Dave Parker, is gratefully acknowledged.

References

[1] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[2] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998. ISBN 0-13-6774409-5, pp. xv+565.

[3] Ali Abdullah, Cliff Jones, and Jeff Sanders, editors. *25 Years of CSP*. Springer Verlag, To appear, 2005? Workshop at Institute for Computing Research, London South Bank University, 7–8 July 2004, organised by Formal Aspects of Computing Science BCS Specialist Group.

[4] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement: FDR 2 User Manual*, 1992-2004.

[5] PRobabilistIc Symbolic Model checker. http://www.cs.bham.ac.uk/∼dxp/prism/.

[6] Vitaly Shmatikov. Probabilistic analysis of anonymity. In *IEEE Computer Security Foundations Workshop (CSFW)*, pages 119–128, 2002.

[7] Gavin Lowe. Pravda: A tool for verifying probabilistic processes. In *Proceedings of the Workshop on Process Algebra and Performance Modelling*, number CSR-2693, pages 57–64. Department of Computer Science, University of Edinburgh, 1993.

[8] Carroll Morgan, Annabelle McIver, Karen Seidel, and Jeff Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 3(1–000), 1995.

[9] A.W. Roscoe, P.H.B. Gardiner, M.H. Goldsmith, J.R. Hulance, D.M. Jackson, and J.B. Scattergood. Hierarchical compression for model-checking CSP *or* How to check 10^{20} dining philosophers for deadlock. In *Proceedings of TACAS Symposium, Aarhus, Denmark*, 1995.

[10] Irfan Zakiuddin, Nick Moffat, Michael Goldsmith, and Tim Whitworth. Property based compression strategies. In *Proceedings of Second Workshop on Automated Verification of Critical Systems (AVoCS 2002)*. University of Birmingham, April 2002.

[11] Michael Goldsmith, Nick Moffat, Bill Roscoe, Tim Whitworth, and Irfan Zakiuddin. Watchdog transformations for property-oriented model-checking. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, pages 600–616, Pisa, September 2003. Formal Methods Europe.

[12] QinetiQ, Birmingham University, Formal Systems, and Oxford University. *FORWARD*: A Future of Reliable Wireless Ad-hoc networks of Roaming Devices. `http://www.forward-project.org.uk`.