# Recent Developments in FDR*

Philip Armstrong, Michael Goldsmith, Gavin Lowe, Joël Ouaknine,
Hristina Palikareva, A. W. Roscoe, and James Worrell

Department of Computer Science, Oxford University, UK
{phila,michael,gavinl,joel,hrip,awr,jbw}@cs.ox.ac.uk

**Abstract.** We describe and report upon various substantial extensions of the CSP refinement checker FDR including (i) the direct ability to handle real-time processes; (ii) the incorporation of bounded model checking technology; (iii) the development of conservative and highly efficient static analysis algorithms for guaranteeing livelock-freedom; and (iv) the development of automated CEGAR technology.

## 1 Introduction

FDR, standing for *Failures Divergence Refinement*, is the best-known tool supporting CSP [13, 21, 22]. It was originally released in 1991/2 and underwent a major re-write in 1994/5 to become FDR2. All subsequent developments have extended FDR2, including those reported in this paper. It is extensively described in [21, 22]. Most of its functionality is based around proving or refuting refinement between finite-state CSP processes, where refinement is over a selection of semantic models with different expressive powers. The best known of these are *traces*, which proves partial correctness, *failures*, which additionally handles deadlock, and *failures-divergences*, which captures a wide variety of total correctness properties. *Refinement* over one of these models is always reverse containment between the sets of relevant behaviours representing the processes.

Traditionally it has been an explicit model checker—capable of exploring millions of states per minute on a modern workstation—supported by state-space compression techniques and the partial-order compression *chase*, as described in [21, 22].

FDR has been widely used in research, teaching and industry [3, 6, 15], and is well known for its use in security analysis [14]. Until 2007 it was a product of Formal Systems (Europe) Ltd, but since 2008 it has been developed in Oxford University under support from EPSRC, ONR and industry. The present paper summarises the new features added in this latter phase.

In recent years FDR has been used as the back-end of verification engines aimed at notations other than CSP. Casper (security) and the shared-variable analyser SVA (see Chapters 18 and 19 of [22]) are examples, as well as a number of proprietary industrial tools.

* Research supported by EPSRC, ONR, Verum BV and Qinetiq.

## 2 New Techniques and Features

**Modelling and Verifying Real-Time Systems.** Two different timed extensions have been developed for CSP. The first is Timed CSP [20], which is a real-time interpretation of Hoare's CSP notation [13]. In its usual form it adds a single construct to CSP, namely $WAIT\,t$ which waits $t$ units of time before terminating successfully. It is possible to express a wide variety of time-based operations such as time-outs in terms of $WAIT\,t$ and standard CSP. Timed CSP generally assumes a continuous clock in the hands of external observers. A second form, *tock*-CSP, was introduced by Roscoe [21] as a medium for verifying discretely timed systems on FDR, most naturally with an internal clock signal. A special event *tock* represents the regular passage of time.

Thanks to the idea of *digitisation* introduced by Henzinger, Manna and Pnueli [12] and developed for Timed CSP by Ouaknine [17, 18], it is possible to establish theoretical connections among continuous Timed CSP, the Timed CSP language with a discrete semantics, and *tock*-CSP. It is therefore possible to model check Timed CSP programs, drawing conclusions about their continuous semantics by a relatively modest modification to FDR along the lines suggested in [17, 22]. This modification – described in more detail in [4] – takes the form of a mode within FDR that instructs it to interpret the syntax inside it as a Timed CSP process and translate it into semantically equivalent *tock*-CSP. It is possible, and frequently very useful, to mix Timed CSP, *tock*-CSP and ordinary CSP in the same script and indeed in the same process or refinement check.

We can check Timed CSP processes for refinement against specifications formulated in Timed CSP itself, in *tock*-CSP, or—when time is suitably abstracted—in ordinary CSP. As reported in [23], it is possible to check Timed CSP for noninterference (i.e. information flow) properties and therefore find or prove the absence of *timing channels* that pass information between mutual users of a system.

This Timed CSP mode of FDR is a recent development and we have not yet had time to try it on serious industrial examples. We have, however, tried it on several well known benchmarks such as Fischer's mutual exclusion protocol and the puzzle in which a number of soldiers have to cross a bridge in pairs using a torch. In both of these it demonstrated great efficiency: Table 1 shows results for the first of these in comparison to Uppaal [2] and PAT [1]. Further results can be found in [4].

**Bounded Model Checking and Temporal *k*-Induction.** For the traces model of CSP, FDR now supports an alternative refinement engine employing symbolic techniques based on Boolean satisfiability (SAT). In particular, FDR features bounded model checking (BMC) [7], that can be used for bug detection, and temporal $k$-induction [11], which builds upon BMC, aims at establishing inductiveness of properties and is capable of both bug finding and establishing the correctness of systems. The symbolic engine [19] adopts FDR's implicit operational representation based on supercombinators [22], but explores this using

**Table 1.** Timed CSP. Times reported are in seconds, with ⋆ denoting memout. Comparison against Uppaal 4.0.13 and PAT 3.40. The columns titled PAT-zone and PAT-digit denote, respectively, PAT using zone abstraction and digitisation as underlying engines. All experiments were performed on a 2.6 GHz PC with 2 GB RAM running Linux Fedora.

| Benchmark | FDR | Uppaal | PAT-zone | PAT-digit |
|---|---|---|---|---|
| Fischer mutual exclusion-6 | 0 | 0 | 13 | 7 |
| Fischer mutual exclusion-7 | 0 | 0 | 196 | 85 |
| Fischer mutual exclusion-8 | 0 | 2 | ⋆ | ⋆ |
| Fischer mutual exclusion-10 | 2 | 20 | ⋆ | ⋆ |
| Fischer mutual exclusion-12 | 18 | 312 | ⋆ | ⋆ |

SAT rather than explicitly. For both BMC and $k$-induction, FDR offers configurable support for a SAT solver (MiniSAT, PicoSAT or ZChaff, all used in incremental mode), Boolean encoding (one-hot or binary), traversal mode (forward or backward), etc. The BMC engine sometimes substantially outperforms the original explicit state-space exploration, especially for complex tightly-coupled combinatorial problems, as reported in [19]. For $k$-induction, the completeness threshold blows up in all cases, due to concurrency, and, therefore, high performance depends on whether or not the property is $k$-inductive for some small value of $k$. Thus we have only seen SAT outperform FDR when there are counterexamples.

**Static Analysis for Establishing Livelock Freedom.** FDR now supports an alternative back end for establishing livelock freedom. Livelock, also called divergence, indicates that a process is unresponsive due to being engaged forever in internal computations. The new back-end relies on static analysis of the syntactic structure of a process rather than explicit state exploration. It employs a collection of rules to calculate a sound approximation of the fair/co-fair sets of events of a process [16]. The rules either safely classify processes as livelock-free or report inconclusiveness, thereby trading accuracy for speed. The algorithms generate and manipulate various sets of events in a fully symbolic way. The choice of an underlying symbolic engine is configurable, with support for using a SAT engine (based on MiniSAT 2.0), a BDD engine (based on CUDD 2.4.2), or running a SAT and a BDD analyser in parallel and reporting the results of the first one to finish. Experiments indicate that the static analyser is substantially more efficient than the exhaustive-search approach, outperforming it by multiple orders of magnitude whilst exhibiting a low rate of inconclusive results. We experimented with a wide range of benchmarks, including parameterised, parallelised, and piped versions of Milner's Scheduler, the Alternating Bit Protocol, the Sliding Window Protocol, the Dining Philosophers, Yantchev's Mad Postman Algorithm, etc., as reported in [16].

**CEGAR.** We developed abstraction/refinement schemes for the traces, failures and failures-divergences models and embedded them into a fully automated and compositional counterexample-guided abstraction refinement framework (CEGAR) [10]. An initially coarse abstraction of the system is iteratively refined (i.e. made more precise) on the basis of spurious counterexamples until either a genuine counterexample is derived or the property is proven to hold. We exploit the compositionality of CSP for the stages of initial abstraction, counterexample validation and abstraction refinement, extending the framework proposed in [9, 8]. Generally, we adopt lazy refinement strategies that yield coarser abstractions even though it takes a greater number of iterations to converge. Experiments can show performance enhancement when verifying both safety and liveness properties, as illustrated in Table 2.

**Table 2.** CEGAR. Times reported are in seconds, with * denoting a 30-minute timeout. The last column titled ♯ reports the number of iterations that it takes for CEGAR to converge. All experiments were performed on a 3.07GHz Intel Xeon processor with 8 GB RAM running Linux Ubuntu.

| **Property** | **Benchmark** | **FDR** | **CEGAR** | ♯ |
|---|---|---|---|---|
| Trace | Milner-10 | 0 | 0.03 | 21 |
| (safety), | Milner-20 | 158 | 0.07 | 41 |
| holds | Milner-30 | * | 0.16 | 61 |
| | Milner-100 | * | 4.42 | 201 |
| | Milner-200 | * | 40.01 | 401 |
| Deadlock | Mad Postman-3 | 4 | 0.03 | 4 |
| (liveness), | Mad Postman-5 | * | 0.22 | 4 |
| holds | Mad Postman-7 | * | 1.49 | 4 |
| | Mad Postman-9 | * | 7.13 | 4 |

The columns for FDR represent its use with none of its compression functions used. Compression can be used effectively on these systems, but of course requires skill in picking the right compression and compression strategy.

**New Semantic Models of CSP.** While traces, failures and failures-divergences remain the most generally used models in FDR, it is sometimes useful to have the expressive power of richer models. FDR now supports the *revivals* and *refusal testing* models [22], together with their divergence-strict analogues. We eventually hope to support almost the full range of models reported in Chapters 10–12 of [22], including some which support non-strict reasoning about divergence. Compositional reasoning about Timed CSP and priority each require one of these stronger models.

**Divergence-Respecting Weak Bisimulation.** The range of compression operators is now augmented with divergence-respecting weak bisimulation (DRWB):

the largest weak bisimulation which does not identify any immediately divergent node with one that is not. Typically, DRWB does not achieve quite the same degree of compression as the combination of strong bisimulation and *diamond* compression, which is frequently used with FDR. However, it has the great advantage that it is faithful to all CSP models and also inside the priority operator, something that is not true of diamond compression (which works for traces and failures based models only). DRWB compression was, for example, crucial to the efficiency of the timed noninterference work described above.

**A Priority Operator.** In [22] Roscoe proposed a priority operator for CSP that would fit within the general operational semantic framework of CSP and for which the most refined of the abstract semantic models described in that book would be compositional. It is a generalisation of the *timing priority* model that had been used for some time with *tock*-CSP models of timed systems, as discussed above. It is an operator which takes a CSP process in which no actions are intrinsically prioritised and returns another one of the same type. In theory, we can take any partial order on the actions of an operational semantics: ordinary visible actions together with the internal $\tau$ and termination signal $\checkmark$, in which the latter two are both maximal. The priority operator then blocks any action at an operational state when that state has one of higher priority. We support orders in which there are a number of distinct priority levels—sets of equal-priority events—that are linearly ordered, with the first of these sets of events the possibly empty set at the same priority level as $\{\tau, \checkmark\}$. These levels need not partition the entire alphabet, with any events outside their union neither blocking nor being blocked by any other. Thus $\mathit{prioritise}(P, \{\}, \{\mathit{tock}\})$ represents the operator that gives $\tau$ and $\checkmark$ higher priority than *tock*, with no other priorities enforced.

This operator was implemented thanks to industrial funding from Verum after it was discovered [5] that priority plus other CSP operators such as renaming could be used to determine whether a system can still diverge when we disallow some infinite $\tau$ sequences in which hidden events from a set $M$ are infinitely often accompanied by offers of events from some set $A$. This was required for important availability checks in Verum's ASD tool [6], which has FDR embedded as its verification engine.

## 3 Technical Details, Availability and Usage

FDR is largely written in C++ and runs on Linux, Mac OS X and Solaris on SPARC. The binaries, as well as a user manual, are available for download from:

`http://www.cs.ox.ac.uk/projects/concurrency-tools/`.

There are two ways of using FDR: either through its own GUI or through a command-line interface that is primarily used by other verification tools which use FDR as a back end. Details can be found in the user manual. Collections of CSP scripts can be downloaded from `http://www.cs.ox.ac.uk/ucs/CSPM`.

## References


[1] PAT: Process analysis toolkit. `http://www.comp.nus.edu.sg/~pat/`.

[2] Uppaal. `http://http://www.uppaal.org/`.

[3] A. E. Abdallah. *Communicating sequential processes: the first 25 years*, volume 3525. Springer, 2005.

[4] P. Armstrong, G. Lowe, J. Ouaknine and A.W. Roscoe. Model checking Timed CSP. In *HOWARD*, to appear 2012 (Easychair, pub).

[5] P.Armstrong, P.J. Hopcroft and A.W. Roscoe. Fairness checking through priority. To appear 2012.

[6] G.H. Broadfoot and P.J. Hopcroft. A paradigm shift in software development. In Proceedings of Embedded World Conference 2012. Nurmemburg.

[7] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS*, pages 193–207. Springer-Verlag, 1999.

[8] Sagar Chaki, Edmund M. Clarke, Joël Ouaknine, Natasha Sharygina, and Nishant Sinha. Concurrent software verification with states, events, and deadlocks. *Formal Aspects of Computing*, 17(4):461–483, 2005.

[9] Sagar Chaki, Joël Ouaknine, Karen Yorav, and Edmund M. Clarke. Automated compositional abstraction refinement for concurrent C programs: A two-level approach. In *Electronic Notes in Theoretical Computer Science*, volume 89, 2003.

[10] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *CAV'00*. Springer LNCS, 2000.

[11] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. In *Electronic Notes in Theoretical Compututer Science*, volume 89, 2003.

[12] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *ICALP*, pages 545–558, 1992.

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

[14] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *TACAS '96*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

[15] C.M. O'Halloran. Acceptance based assurance. ASE 2001, IEEE.

[16] J. Ouaknine, H. Palikareva, A. W. Roscoe, and J. Worrell. Static livelock analysis in CSP. In *Proceedings of CONCUR 2011*, volume 6901 of *LNCS*, pages 389–403. Springer, 2011. Winner of the CONCUR 2011 Best Paper Award.

[17] Joël Ouaknine. Digitisation and full abstraction for dense-time model checking. In *TACAS*. Springer LNCS, 2002.

[18] Joël Ouaknine and James Worrell. Timed CSP = Closed Timed epsilon-automata. *Nord. J. Comput.*, 10(2):99–133, 2003.

[19] H. Palikareva, J. Ouaknine, and A. W. Roscoe. SAT-solving in CSP trace refinement. *Science of Computer Programming, special issue on Automated Verification of Critical Systems*, 2011. In press.

[20] G. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Automata, Languages and Programming*, pages 314–323, 1986.

[21] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[22] A. W. Roscoe. *Understanding Concurrent Systems*. Springer, 2011. `http://www.cs.ox.ac.uk/ucs/`.

[23] A. W. Roscoe and Jian Huang. Extending noninterference properties to the timed world. In *Proceedings of SAC 2006*, 2006.