The FC2TOOLS Set

Amar Bouali[1] and Annie Ressouche[2] and Valérie Roy[1] and Robert de Simone[2]

[1] CMA/Ecole des Mines de Paris, B.P. 207, F-06904 Sophia-Antipolis cedex
[2] INRIA , B.P. 93, F-06902 Sophia-Antipolis cedex

1 Presentation

The AUTO/GRAPH toolset [4] developed in our group was one of the pioneering softwares in the field of analysis and verification of networks of communicating processes. We describe here the next-generation AUTO/GRAPH, consisting of a modular tool suite interfaced around a common file description format named FC2. The format allows representation of single reactive automata as well as combining networks. This format was developed in the scope of Esprit BRA project 7166:CONCUR2 [2].

Of uttermost interest in the new implementation is that most analysis functions are implemented with redundancy using both *explicit* classical representation of automata, and also *implicit* state space symbolic representation using *Binary Decision Diagrams*. The two alternative techniques are shown to offer drastically different performances in different cases, with low predictibility. Then offering both kinds of implementation in a unified framework is a valuable thing in our view.

Both FC2EXPLICIT and FC2IMPLICIT commands perform synchronised product and reachable state space search. They can minimize results w.r.t. *strong, weak, branching* bisimulation notions, and produce the result as an FC2 automaton. They can also *abstract* the system with a notion of "abstract actions", each synthesizing a set of sequences of concrete behaviours (in this sense behavioural abstraction can be seen as reverse from refinement). In addition FC2IMPLICIT has a fast checker for *deadlocks, livelock* or *divergent* states, for which it produces counterexample paths in case of existence, while FC2EXPLICIT allows *compositional* reduction techniques, mostly in case of "observational" bisimulation minimisations.

We are currently extending these features of FC2IMPLICIT so that labeled predicates on states, hiding of behaviours irrelevant to specific analysis, and use of side observer automata would allow to check in practice for much wider types of properties, while keeping with the same algorithmic kernel, and with the renewed aim of *not* introducing an heterogeneous formalism for expression of correctness properties, like temporal logics or μ-calculus.

The tool suite is completed by the graphical editor AUTOGRAPH, which allows for graphical depiction of automata and networks as well as source recollection of counterexample paths back up to the original graphical network; the FC2LINK preprocessor, which merge multifile descriptions of hierarchical networks into a single file for later analysis and verification; the FC2VIEW postprocessor for

source recollection and display of counterexample paths, this time back up to the distributed FC2 files.

Further information on the toolset and its availability can be obtained from the WWW page `http://cma.cma.fr/Verification/verif-eng.html`.

2 The Tool Set

2.1 AUTOGRAPH

Renamed ATG for its new C++ implementation, AUTOGRAPH is a graphical display system for both labeled transition graphs and networks of communicating systems, in the tradition of process algebra graphical depiction. Objects in AUTOGRAPH can also be extensively annotated as allowed by the FC2 format standards. Figure 1 provides a (trivial) example of 3 dining philosophers drawn in AUTOGRAPH.

AUTOGRAPH can be used for edition, but also to visualise automata that were produced elsewhere, typically as an output of verification. Human guidance is then required for lay-out.

2.2 FC2EXPLICIT

This tool performs the following functions, on explicit representations of automata:

Global Automaton Generation. Straightforward.

Compositional Reductions. FC2EXPLICIT can perform automata minimisation with respect to *strong*, *weak* or *branching* bisimulation. When invoked on a network, the hierarchical model construction can be alternated with such reduction steps at intermediate stages. Traditional *Relational Coarsest Partitioning Algorithm* [3] is used to refine a state partition until fix-point.

Comparison with automata specifications. The equivalence checking problem is solved on the disjoint union of the two state spaces, by partitioning them as a whole. The main algorithmic improvement is that negative answer is produced as soon as a class contains no further state from one of the automata, which happens usually quite soon (if ever), long before reaching actual fix-point. Then a path leading to an arbitrary state without match from the other network is provided as counterexample. It can be visualised using AUTOGRAPH or FC2VIEW.

Model Abstraction. *Abstract Actions* allow us to define the atomicity level at which we want to observe an automaton. The idea is to consider terminated sequences of concrete behaviours as atomic and to call such a set abstract action. Reducing a global system w.r.t. a set of abstract actions results in a system conceptually simpler, where meaningful activities have been extracted. As a simple subcase, a single abstract action indicating improper (rational) behaviour can be presented for refutation; this implements language inclusion (in the complement language of the abstract action), and is noticed in FC2EXPLICIT by the

absence of transition in the resulting automaton.Example: `Abstract_Wrong = tau*.enter1.tau*.enter2` can specify lack of mutual exclusion property between two processes.

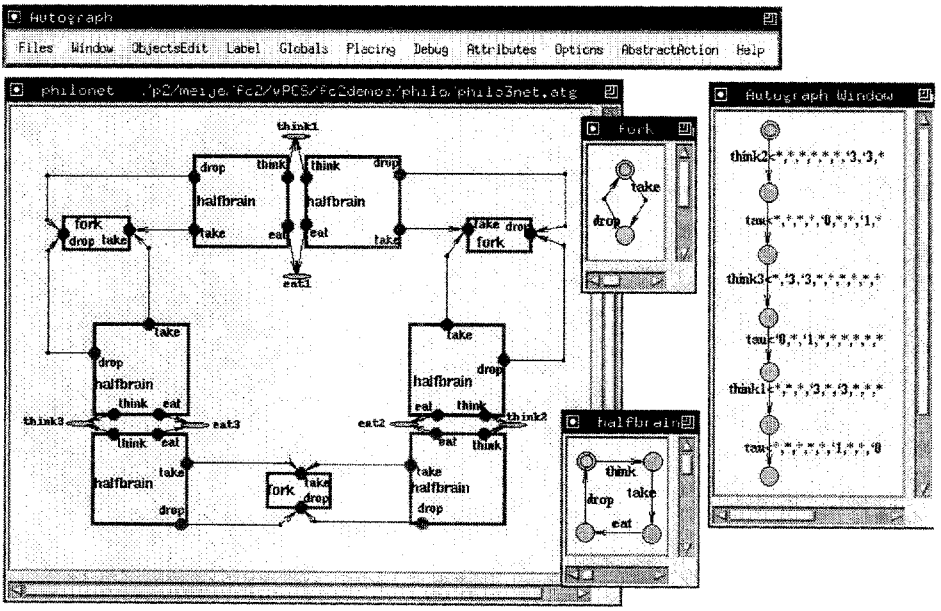

Fig. 1. The 3 dining philosophers specification

2.3 FC2IMPLICIT

performs the following functions, on BDD/implicit representation of states, using the TIGER BDD library:

Global Automata Generation. The reachable state space is of course evaluated in a breadth-first search strategy, applying event synchronisation vectors iteratively until fix-point, staring from initial state. Computation of reachable state computation can be refined to allow for on-line deadlock detection, and followed by livelock or divergent states detection on the result (a divergent state may perform infinite sequences of hidden "tau" actions, a livelock state can exhibit *only* such behaviour). When such an undesirable state is found, a counterexample path can be produced, and mapped by AUTOGRAPH or FC2VIEW back to the original network description. Figure 1 displays a deadlock path for the 3 philosophers (but *not* its distributed mapping).

Bisimulation Minimisation. Symbolic computation of bisimulation classes can also be applied from this BDD description of reachable states. following results from [1]. The (explicit) resulting minimal automaton can be produced in FC2 format on demand.

Bisimulation Checking. First a synchronised product of the two distinct networks is built. Then the same bisimulation partition as before is used, with the algorithmic improvement that it is aborted whenever a class contains no state from one side. Then a counterexample path can be produced, and mapped by AUTOGRAPH or FC2VIEW back to the original network description.

Observers and Annotated Global States.. A great deal of practical verification is usually conducted by compiling the property to establish into an automaton-like structure to act on the side of the observed process, with possibly additional annotations on states and transitions of various sorts (*success*, *failure* or *recur* states, *don't care* transitions,...). Verification then starts by constructing a synchronised product of the (usually large) network state space with the (usually smaller) state space of the observer structure. Such observers can actually be represented without additional theory in the same FC2 format as processes, and particular sets of states and transitions are just used to restrict (or introduce new) relations in algorithms. We are still working on "easy" description of such sets, so as to "hide' as much as possible intricate temporal logic formulation from the non-expert user.

3 Example

We just illustrate the basic verification features on our simple *dining philosophers* problem from figure 1.

We now suppose these three parts (the fork, halfbrain automata and the network) have been independently translated (by ATG) into distinct FC2 files, and then linked together by FC2LINK into file `philo3.fc2`.

We use symbolic methods based on BDDs for an easy evaluation of global state spaces and deadlock checking.

```
0-duick$ fc2implicit -dead -fc2 philo3.fc2 > deadpath.fc2
--- fc2implicit: Making reachable state space
--- fc2implicit: State space depth: 13
--- fc2implicit: First deadlock(s) detected at depth 7
--- fc2implicit: Reachable states: <<214>> -- BDD nodes: <<85>>
--- fc2implicit: Global automaton has 2 DEADLOCKS state(s) -- BDD nodes: <<27>>
0-duick$
```

The first detected deadlocks were found at depth 7. With the option `-fc2`, a diagnostic path was extracted into `deadpath.fc2`. It is displayed in AUTOGRAPH on the left of figure 1. Now clicking appropriately on states or transitions there would highlight contributing locations on the original graphical network.

The following session shows time figures for larger state spaces These were computed on a Sparc10 Workstation with 64 Mb memory. Timing include synthesis of diagnostic paths or production of automata onto files. The full-size automaton for 6 philosophers would be 46654 states, while compositional reductions deals with intermediate constructs of at most 1512 states.

```
philo$ time fc2implicit -dead -fc2 philo8 > deadpath8.fc2
--- fc2implicit: Reachable states: <<1679614>> -- BDD nodes: <<245>>
--- fc2implicit: Global automaton has 2 DEADLOCKS state(s) -- BDD nodes: <<77>>
--- fc2implicit: First deadlock detected at depth 16
real    1m52.21s user   1m39.08s sys    0m0.46s

philo$ time fc2explicit -comp -w obs_philo6_rec > opr6_eW.fc2
--- fc2min: Automaton has 728 states
real    3m4.85s user    2m2.30s sys     0m11.55s
```

Larger experiments were conducted, showing the possibilities of the tools. For the next future we are concentrating in replacing sequential automata components by *synchronous reactive processes* (such as produced by the ESTEREL language for instance), to be able to deal with asynchronous networks of synchronous processes. We have promising initial results in this direction, mainly from the fact that both domains allow partition of the transition relation for simpler symbolic application on implicit state spaces.

References

1. A. Bouali and R. de Simone. Symbolic bisimulation minimisation. In *Fourth Workshop on Computer-Aided Verification*, volume 663 of *LNCS*, pages 96–108, Montreal, 1992. Springer-Verlag.
2. A. Bouali, A. Ressouche, V. Roy, and R. de Simone. The FCTOOLS user manual. In *final ESPRIT-BRA CONCUR2 deliverables*, Amsterdam, October 1995. Available by ftp from `cma.cma.fr:pub/verif` as file *fc2userman.ps*.
3. P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
4. V. Roy and R. de Simone. AUTO and autograph. In R. Kurshan, editor, *proceedings of Workshop on Computer Aided Verification*, New-Brunswick, June 1990. AMS-DIMACS.