McScM - Feature # 42: [Experiment] Ignore abstract node regions in Cegar's trace checking

Status:	Closed	**Priority:**	High
Author:	Grégoire Sutre	**Category:**	
Created:	04/07/2011	**Assignee:**	Grégoire Sutre
Updated:	10/13/2011	**Due date:**	10/09/2011
Subject:	[Experiment] Ignore abstract node regions in Cegar's trace checking		
Description:	Every abstract counterexample corresponds to a path in the model's automaton. This path might be feasible even though the abstract counterexample is spurious. Moreover, if the path is not feasible, the trace-checking engine may provide a better path invariant when abstract nodes are ignored. To test this idea, modify Cegar's @trace_of_fwd_path@ and @trace_of_bwd_path@ functions so that they simply use @Region.top@ in place of abstract node regions. This is just an idea to experiment.		

History

09/06/2011 03:31 pm - Grégoire Sutre

In McMillan's _Lazy Abstraction with Interpolants_ paper, the regions labeling nodes of the abstract reachability tree are discarded for refinement. Indeed, the path interpolant is generated from the sequence of transition formulas, and the latter only depend on the program's instructions.

So it might be interesting to experiment with the passing/discarding of abstract regions for the LazyAbstraction module as well.

A uniform solution would be to have a functor that takes a trace-checking engine and returns a new trace-checking engine where the passed state regions are replaced by the top region. The resulting trace invariant must then be reduced to conform with the specification of trace checkers.

09/16/2011 05:28 pm - Grégoire Sutre

- Assignee set to Grégoire Sutre

- Priority changed from Normal to High

- Target version set to 1.2

- Start date set to 09/16/2011

- % Done changed from 0 to 50

Irreducibility of trace invariants turns out to be important for the in-progress Lazy Abstraction engine. To conform with McMillan's paper, it is desirable to drop regions labeling nodes, but the resulting trace invariant is not irreducible anymore.

Hence, we give this issue higher priority. The implementation should provide a functor taking as input a trace-checking engine and returning a trace-checking engine where state regions are replaced by top in the input trace. To avoid useless reductions, it will probably be necessary to _tag_ trace-invariants with a boolean telling whether they are (known to be) irreducible or not.

09/16/2011 05:29 pm - Grégoire Sutre

- Status changed from New to In Progress

10/13/2011 03:07 pm - Grégoire Sutre

- Due date set to 10/09/2011

- Status changed from In Progress to Closed

- % Done changed from 50 to 100

The functor @TraceCheckerUtil.DiscardStates@ has been added in commit 1170.

On second thought, it did not seem crucial to add irreducibility information in the trace invariants. So we keep the property that trace-checking engines shall always return irreducible invariants.