Co-Algebraic HD-automata: Modularization of Reducer

Roberto Raggi & Emilio Tuosto

joint work with

Gianluigi Ferrari, Ugo Montanari and Marco Pistore

Plan of the talk

Plan of the talk

- Just last news for profundians...

Plan of the talk

- Just last news for profundians...
- Data format

Implementation choices

- Guided by the theory

Implementation choices

- Guided by the theory
- then...`ocaml`

Implementation choices

- Guided by the theory
- then...`ocaml`
- polimorphism + encapsulation = modularity $\wedge$ extendability

Encapsulation

OCAML module system provides a neat separation between interfaces and their implementations:

Encapsulation

OCAML module system provides a neat separation between interfaces and their implementations:

```
module type Bisimulation =
sig

  type bundletype
  type resulttype

  val bisimilar : bundletype → resulttype option


end
```

Encapsulation

OCAML module system provides a neat separation between interfaces and their implementations:

module type *Bisimulation* =
sig

  type *bundletype*
  type *resulttype*

  val *bisimilar* : *bundletype* $\rightarrow$ *resulttype* option

end

module *Bisimulation* =
  functor(Bundle : BundleSig) $\rightarrow$
struct

  type *bundletype* = Bundle.quadtype list
  type *resulttype* = (int $\times$ int) list

  let *theta* = ...
  let *bisimilar* = ... (theta x y) ...
end

Encapsulation

OCAML module system provides a neat separation between interfaces and their implementations:

```
module type Bisimulation =
sig

  type bundletype
  type resulttype

  val bisimilar : bundletype → resulttype option


end
```

```
module Bisimulation =
  functor(Bundle : BundleSig) →
struct

  type bundletype = Bundle.quadtype list
  type resulttype = (int × int) list

  let theta = ...
  let bisimilar = ... (theta x y) ...
end
```

Advantages

Same algorithm for

- ordinary automata
- HD-automata

...and flexibility

Example:

If you want to change bisimulation, simply change `HD_bisimulation` and `HD_domination`,

if information needed during iterations changes, then change `HD_block`!

Data format

start q0

Data format

```
start q0
  state  q0  3
  state  q1  3
  state  q2  2
  state  q3  2    [ [1;2] ; [2;1] ]
  state  q4  3
```

Data format

```
start q0
  state  q0  3
  state  q1  3
  state  q2  2
  state  q3  2    [ [1;2] ; [2;1] ]
  state  q4  3
  q0  ->  q1     out[ 1 ; 2 ]     [ 1 ; 2 ; 3 ]
  q0  ->  q1     out[ 2 ; 1 ]     [ 1 ; 2 ; 3 ]
  q0  ->  q0     tau              [ 2 ; 1 ; 3 ]

  q1  ->  q2     in[ 1 ; 1 ]      [ 1 ; 2 ]
  q1  ->  q3     in[ 1 ; 2 ]      [ 1 ; 2 ]
  q1  ->  q4     in[ 1 ; 3 ]      [ 1 ; 2 ; 3 ]
  q1  ->  q4     bin[ 1 ]         [ 1 ; 2 ; * ]

  q2  ->  q2     bout[ 1 ; 2 ]    [ 1 ; 2 ]
```

Conclusion

Code is described on a draft paper

Conclusion

Next week on the web ;-)