Verifying Liveness Properties of ML Programs

M M Lester   R P Neatherway   C-H L Ong   S J Ramsay

Department of Computer Science, University of Oxford

ACM SIGPLAN Workshop on ML, 2011–09–18

Motivation

- We want to verify properties of ML programs.
- We are particularly interested in liveness properties.
- Safety properties assert nothing bad ever happens.
  - A finite trace of a program can witness violation of safety.
  - Examples: Does a program ever divide by zero? Is a resource ever accessed without a lock?
- Liveness properties assert something good eventually happens.
  - Violation of liveness properties can be shown by infinite traces.
  - Examples: Does a program always terminate? Does it always respond? Is every file opened eventually closed?

Example

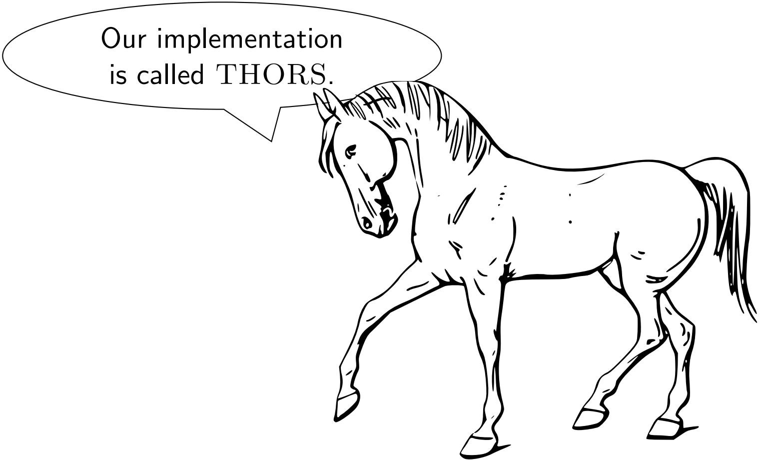
```
let rec g x = if b then close(x); g(open_in n)
                   else read(x); g(x) in
let s = open_in "foo" in g(s)
```

Is every file opened in this program eventually closed?

(Program courtesy of Naoki Kobayashi.)

Approach

- Model-checking: translate the problem into some equivalent, automatically solvable, abstract problem; then solve it.
- We have developed the theory to solve such a class of problems and implemented it.

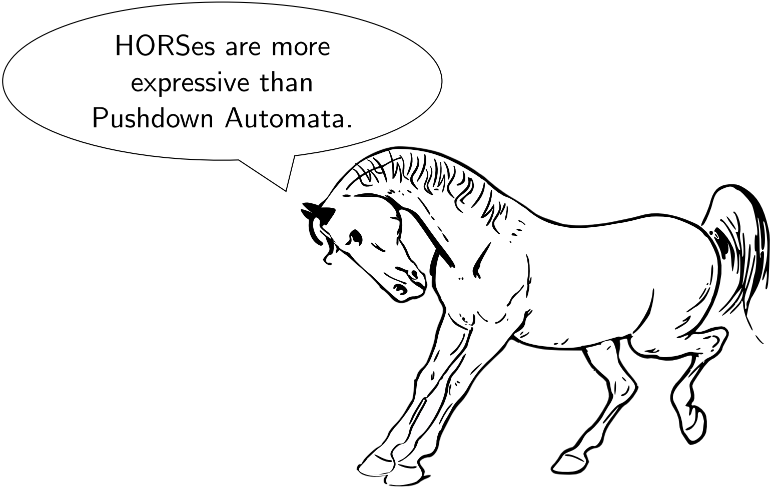


Outline

- Motivation
- Approach
- Class of Abstract Problem
- Previous Work
- Our Algorithm
- Conclusion

Class of Abstract Problem

We consider the problem of whether the tree generated by a Higher Order Recursion Scheme (HORS) $\mathcal{G}$ satisfies a property of Alternation Free Mu Calculus (AFMC) ϕ.



Higher Order Recursion Schemes (HORSes)

- Higher Order Recursion Schemes are simply-typed, tree-generating grammars with recursion.
- A HORS specifies a set of rewrite rules for non-terminal symbols, which must satisfy certain constraints.
- The value tree of a HORS is the potentially infinite, ranked tree of terminal symbols generated by repeated application of the rewrite rules, starting from the non-terminal S.

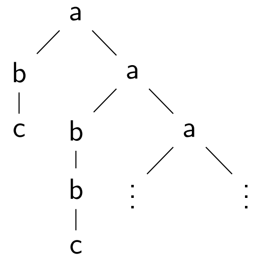
Example

These rules...

$$\begin{array}{rcl} S & \to & F\, b \\ F\, x & \to & a\,(x\, c)\,(F\,(G\, b\, x)) \\ G\, x\, y\, z & \to & x\,(y\, z) \end{array}$$

Note value trees may have infinitely many non-isomorphic subtrees.

...generate this tree:



Higher Order Recursion Schemes (HORSes)

Example

Recall the ML program introduced earlier:

```
let rec g x = if b then close(x); g(open_in n)
                    else read(x); g(x) in
let s = open_in "foo" in g(s)
```

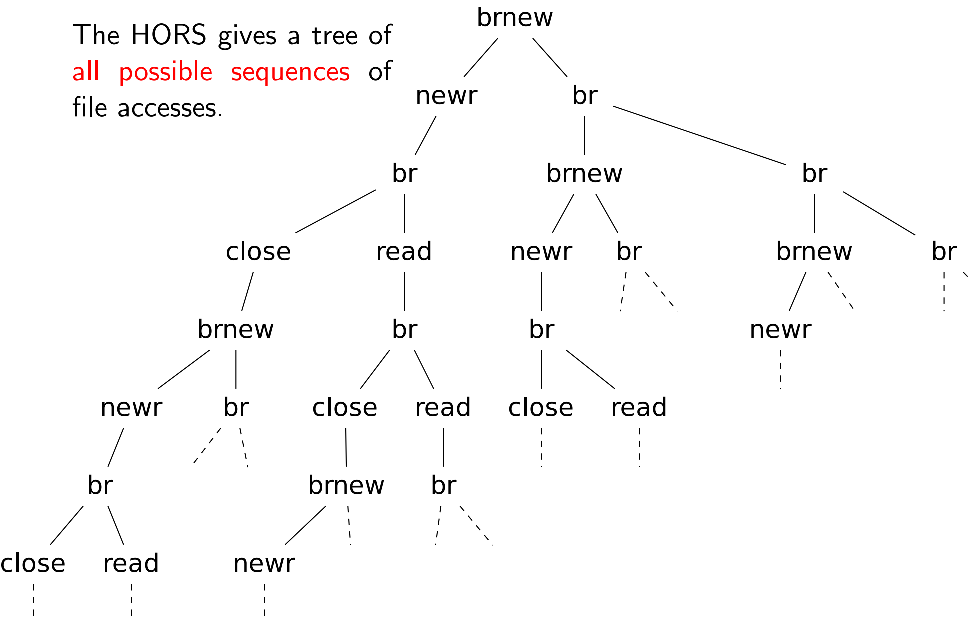
We abstract the program to the following HORS.

$$
\begin{array}{rcl}
S & \rightarrow & Newr\,(G\; end) \\
G\; k\; x & \rightarrow & br\,(Close\; x\,(Newr\,(G\; end)))\,(Read\; x\,(G\; k\; x)) \\
I\; x\; y & \rightarrow & x\; y \\
K\; x\; y & \rightarrow & y \\
Newr\; k & \rightarrow & brnew\,(newr\,(k\; I))\,(k\; K) \\
Close\; x\; k & \rightarrow & x\; close\; k \\
Read\; x\; k & \rightarrow & x\; read\; k
\end{array}
$$

Because HORSes are Call-By-Name, but ML is Call-By-Value, the abstraction involves a Continuation Passing Style transform.

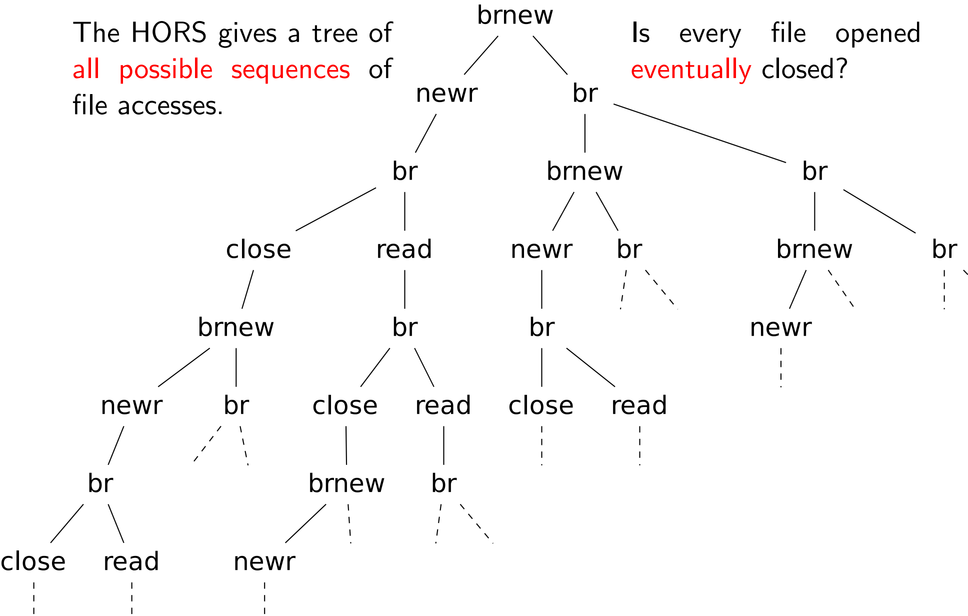
Higher Order Recursion Schemes (HORSes)

The HORS gives a tree of all possible sequences of file accesses.



Higher Order Recursion Schemes (HORSes)

The HORS gives a tree of all possible sequences of file accesses.

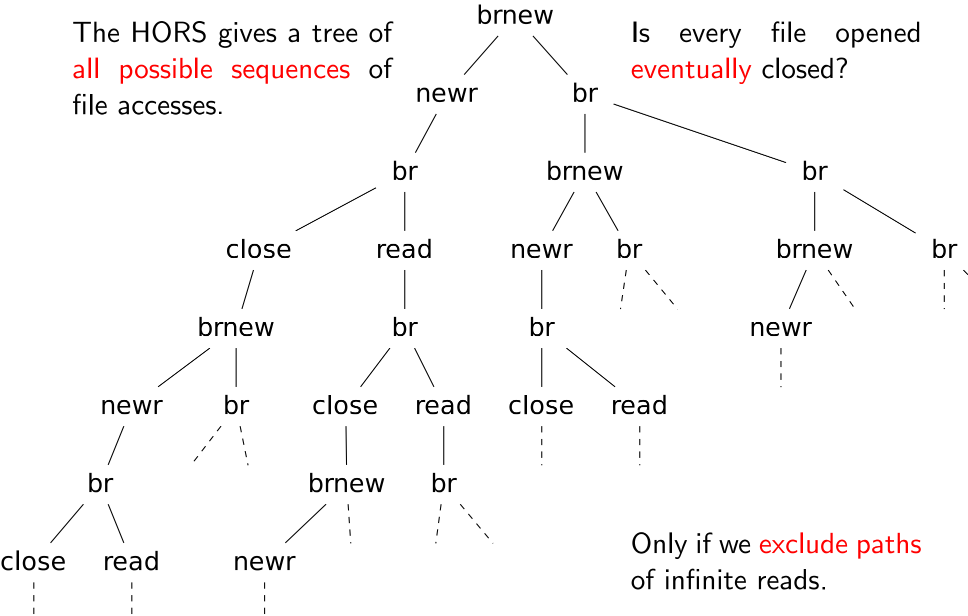


Is every file opened eventually closed?

Higher Order Recursion Schemes (HORSes)

The HORS gives a tree of all possible sequences of file accesses.

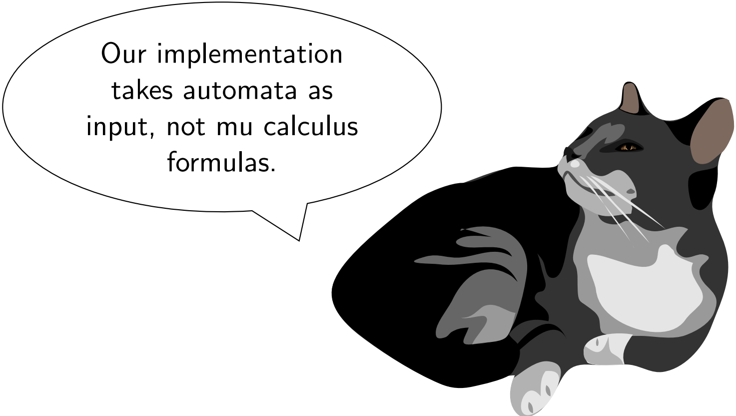
Is every file opened eventually closed?



Only if we exclude paths of infinite reads.

Alternation Free Mu Calculus (AFMC)

- Alternation Free Mu Calculus is a specification logic equivalent to Alternating Weak Tree automata and more expressive than CTL.
- Alternating Weak Tree automata are Alternating Parity Tree automata with the restriction that priorities must be monotonically decreasing for any sequence of states.



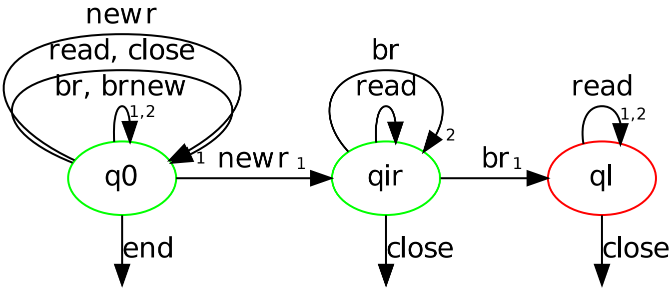
Alternation Free Mu Calculus (AFMC)

Example

To verify that all files opened are eventually closed, we check the CTL property, but ignore paths of infinite reads:

$$AG\,(newr \Rightarrow AX\ A(read\ U\ close))$$

The corresponding conjunctive AWT is:



The green states are accepting; the red state is rejecting.

Previous Work

...on model-checking HORSes

Kobayashi 2009		Ong 2006
Deterministic Trivial Tree automata		Alternating Parity Tree automata
Safety Fragment of Modal Mu Calculus		Modal Mu Calculus
first practical algorithm		algorithm suffers hyper-exponential blow-up in all cases

Even for alternating trivial tree automata, the decision problem is n-EXPTIME complete; that is, bounded by a tower of exponentials of height n.

Previous Work

... on model-checking HORSes

Kobayashi 2009	THORS	Ong 2006
Deterministic Trivial Tree automata	Alternating Weak Tree automata	Alternating Parity Tree automata
Safety Fragment of Modal Mu Calculus	Alternation Free Mu Calculus (includes CTL)	Modal Mu Calculus
first practical algorithm	also practical	algorithm suffers hyper-exponential blow-up in all cases

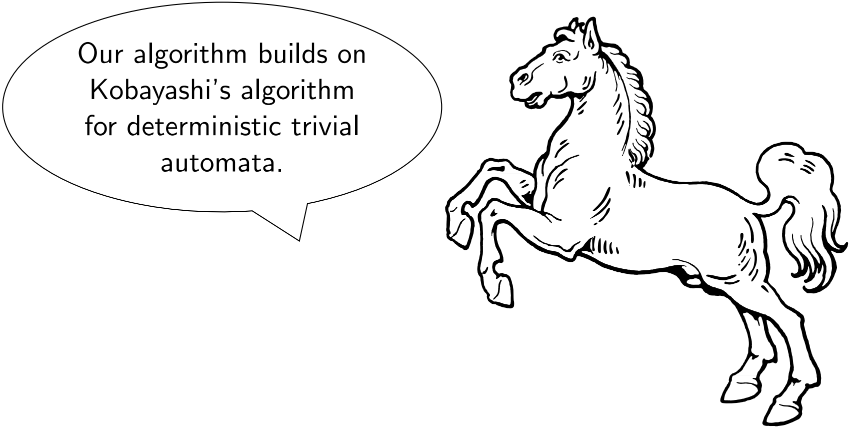
Even for alternating trivial tree automata, the decision problem is n-EXPTIME complete; that is, bounded by a tower of exponentials of height n.

Our Algorithm

Overview

Our algorithm solves the following decision problem:

For a HORS $\mathcal{G}$ and a property of AFMC ϕ, does the value tree generated by $\mathcal{G}$ satisfy ϕ?



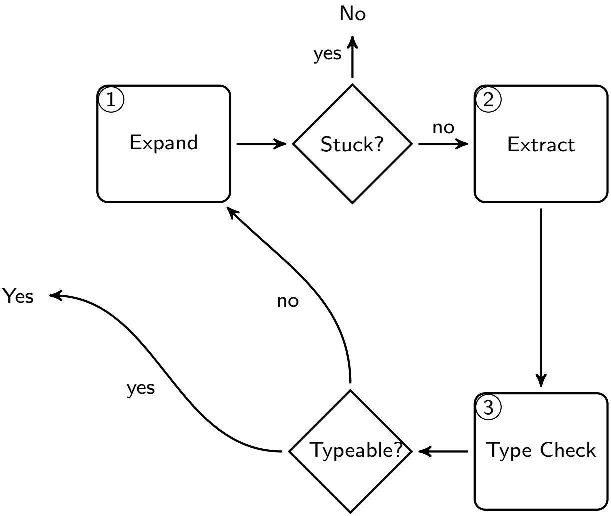
Our Algorithm

Kobayashi's Algorithm

- Kobayashi and Ong (2009) showed that the decision problem can also be solved by typing the symbols of the HORS using an intersection type system.
  - A consistent typing of the HORS indicates that the APT has a run over the value tree.
  - A parity game played over type environments and type bindings indicates whether the run is accepting or rejecting.
- Kobayashi's algorithm for deterministic trivial automata uses a similar but simpler type system.
- There are two key insights that make the algorithm practical:
  1. Types can be inferred heuristically by partially evaluating the HORS and examining how non-terminals are used.
  2. For a trivial automaton, any run is accepting, so there is no need to consider the parity game.

Our Algorithm

Kobayashi's Algorithm

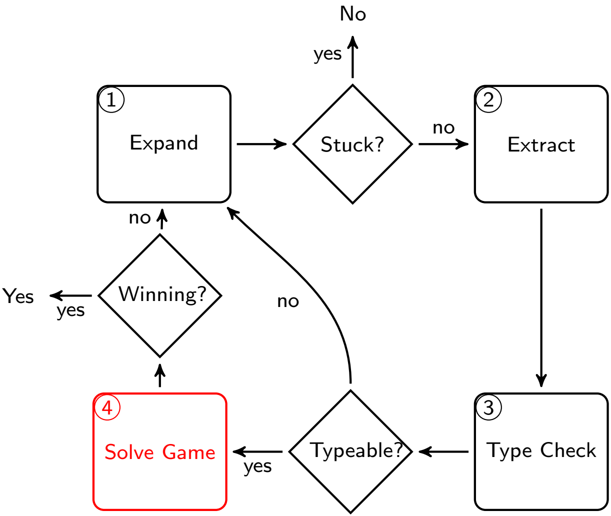


1. Partially evaluate the HORS; stop if a violating trace is found.
2. Infer possible types based on partial evaluation.
3. Discard inconsistent types.

A consistent typing shows existence of a run.
Under a trivial acceptance condition, all runs are accepting.

Our Algorithm

Outline of Algorithm



Solve a game played over type environments and type bindings.

If there is a winning strategy for the game, the run is accepting. Otherwise, the run is rejecting.

For a deterministic or conjunctive automaton, the run is unique. Otherwise, there may be multiple runs.

Our Algorithm

Use of Game and Non-Weakening Type System

- Because we use automata with a non-trivial acceptance condition, we must consider the parity game. To remain practical, we:
  1. only allow automata with a weak acceptance condition;
  2. forbid weakening in the type system.

$$\frac{}{\Gamma, x : \theta \vdash x : \theta}\ (\textsc{Var}) \qquad \frac{}{x : \theta \vdash x : \theta}\ (\textsc{Var-NW})$$

- This ensures we need only consider a small, relevant fragment of the full parity game.
- The resulting parity game is a weak Büchi game, which is solvable in linear time. (Solution of arbitrary parity games is in $NP \cap co\text{-}NP$.)

Our Algorithm

About the Game

- The automaton accepts the value tree of the HORS if and only if there is a winning strategy (for E) in a certain game.
- The game is played by two players, A and E, who take turns to build a path in a graph with two kinds of nodes:
    1. An E-node is a binding of a type to a non-terminal.
       When the path ends in an E-node, E plays as the next node a type environment under which the binding is provable.
       E loses if she cannot play.
    2. An A-node is a type environment.
       When the path ends in an A-node, A plays a binding from the environment.
       A loses if he cannot play.
- Every node in the game is either accepting or rejecting.
- Because the game is a weak Büchi game, an infinite path has either finitely many accepting or finitely many rejecting states.
- When play is infinite, E wins if and only if the path has only finitely many rejecting states.

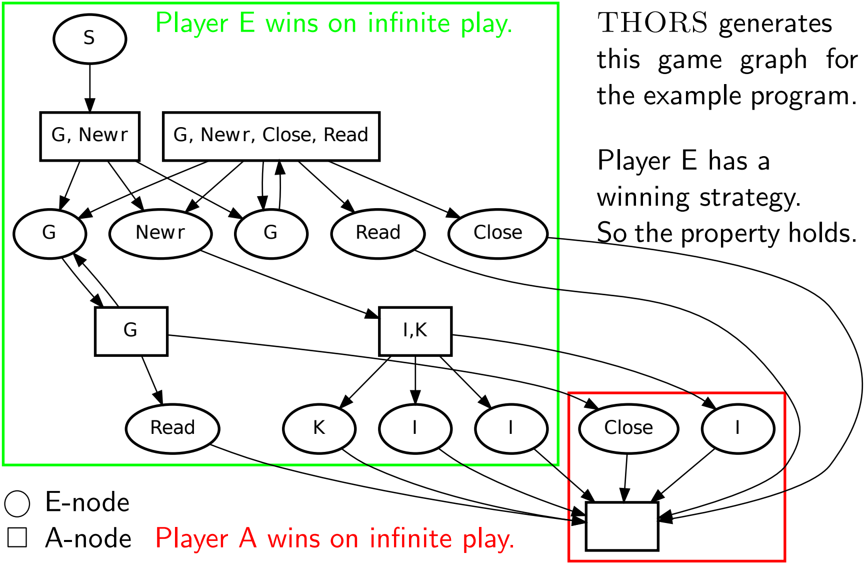
Our Algorithm

About the Non-Weakening Type System

- Types in the type system:
  - Types in the system are simply-typed intersection types.
  - Atomic types are states of the automaton.
  - Type environments can bind many types to each non-terminal.
- Intuition of binding a type to a non-terminal:
  - If S has type q_0, the tree generated by S is accepted from q_0. This is exactly what we want to verify, so play in the parity game starts from node $S : q_0$.
  - If F has type $q_1 \to q_2$ and x gives a tree accepted from q_1, then $F\ x$ gives a tree accepted from q_2.
- We do not allow weakening and:
  - thus every binding in a type environment must be used in a derivation under that environment;
  - this massively reduces the number of environments E can play in the game, making it small enough to construct explicitly;
  - the restriction does not prevent E from winning.

Our Algorithm

Example



THORS generates this game graph for the example program.

Player E has a winning strategy.
So the property holds.

Our Algorithm

Status of Implementation

You can try THORS online, although it is not ready for production use. This table shows how THORS performs in various tests with HORSes and AWTs of different sizes:

Example	*HORS Order*	*HORS Rules*	*AWT States*	*Total Time (ms)*	*Value Tree Nodes*	*Game Nodes*	*Property Class*
D1	4	7	2	1	19	16	Det Weak
D2	4	7	3	1	26	17	Con Weak
D2-ex	4	7	3	1	26	-	Alt Trivial
intercept	4	15	2	35	200	31	Con Weak
imperative	3	6	3	129	200	17	Det Weak
boolean2	2	15	1	1	13	-	Det Trivial
order5-2	5	9	4	19	200	37	Det Co-Triv
lock1	4	12	3	2	32	32	Det Co-Triv
order5-v-dwt	5	11	4	163	400	53	Det Weak
lock2	4	11	4	109	800	-	Det Trivial
example2-1	1	2	2	190	200	-	Det Trivial

Conclusion

Our Contributions

- We have developed the theory of an algorithm for practical verification of Alternation Free Mu Calculus (AFMC) properties of Higher Order Recursion Schemes (HORSes).
  - AFMC is a logic that is more expressive than CTL.
- We have implemented our algorithm in a tool, THORS, which is available to test online: `http://mjolnir.cs.ox.ac.uk/`
- We believe our work is particularly applicable to verifying liveness properties of ML programs.
- More details are available in our technical report: `http://mjolnir.cs.ox.ac.uk/papers/thors.pdf`

Conclusion

Future Work

- Within: Develop better heuristics and other performance improvements for THORS.
- Without: Automate abstraction of ML programs.
- Beyond: Can we find an algorithm for practical verification of a larger class of properties, such as CTL*?
- Beside: Is there another practical technique for verification, perhaps one that does not rely on partial evaluation of the program?

Conclusion

Online Materials

- THORS web interface:
  http://mjolnir.cs.ox.ac.uk/
- Technical report:
  http://mjolnir.cs.ox.ac.uk/papers/thors.pdf
- Extended abstract:
  http://mjolnir.cs.ox.ac.uk/papers/thors-ml2011.pdf
- Talk slides:
  http://mjolnir.cs.ox.ac.uk/papers/thors-slides.pdf

References

Naoki Kobayashi and C.-H. Luke Ong.
Complexity of model checking recursion schemes for fragments of the modal mu-calculus.
In *ICALP (2)*, pages 223–234, 2009.

Naoki Kobayashi and C.-H. Luke Ong.
A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes.
In *LICS*, pages 179–188, 2009.

Naoki Kobayashi.
Types and higher-order recursion schemes for verification of higher-order programs.
In *POPL*, pages 416–428, 2009.

C.-H. Luke Ong.
On model-checking trees generated by higher-order recursion schemes.
In *LICS*, pages 81–90, 2006.