

# Implementation of LMNtal Model Checkers: a Metaprogramming Approach

**\*Yutaro Tsunekawa Taichi Tomioka Kazunori Ueda**

Waseda University

META2016 Workshop Oct.30

# Abstract

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

## Purpose

- Rapid prototyping of new model checkers

## Achievement

- designed and implemented a **framework** that enables programmers to implement **metacircular interpreters** in **LMNtal**
- implemented **LTL and CTL model checkers** based on a **metacircular interpreter** in **LMNtal**

## 1 Background

## 2 Proposed method

## 3 Implementation and Examples

## 4 Related work and Conclusion

# Model Checking

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

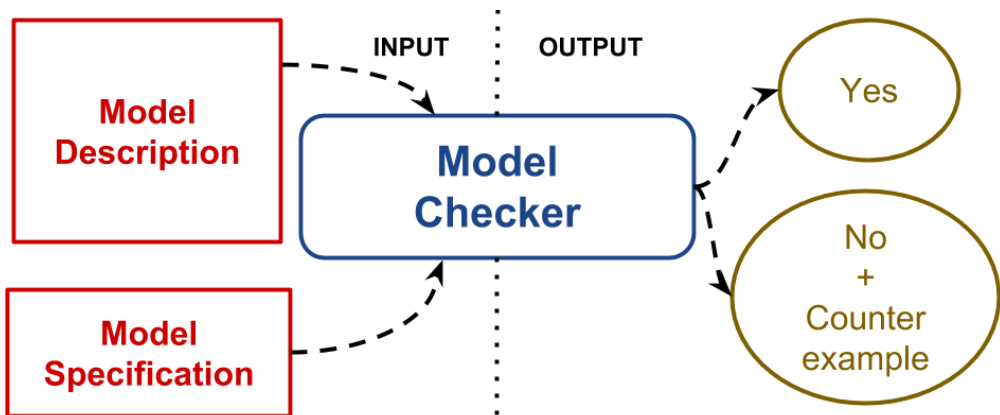
Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- Users just describe **models** and **specifications**
- **Counterexamples** are useful for debugging



# SLIM<sup>1</sup> and LMNtal<sup>2</sup>

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

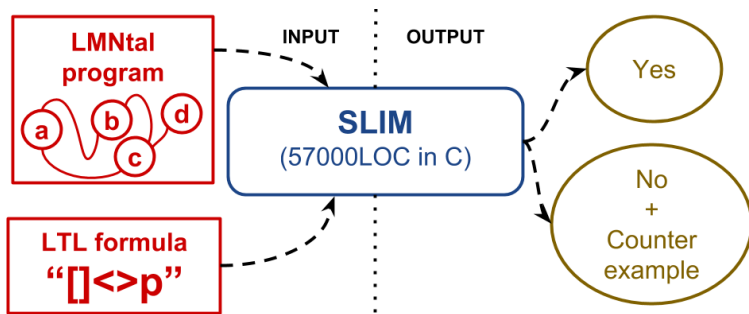
Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

SLIM: LTL model checker

LMNtal: Modeling language based on hierarchical graph rewriting system



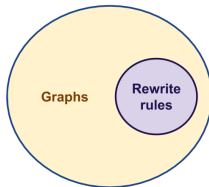
<sup>1</sup>M. Gocho, T. Hori, and K. Ueda.: Evolution of the LMNtal Runtime to a Parallel Model Checker, *Computer Software*, Vol. 21, No. 5, pp. 11–19, 2004

<sup>2</sup>K. Ueda.: LMNtal as a Hierarchical Logic Programming Language, *Theoretical Computer Science*, Vol. 410, No. 46, pp. 4784–4800, 2009

LMNtal programs = Graphs + Rewrite rules

- Highly expressive
  - Graphs are a super set of term rewriting system
- **Concurrent programs** are naturally modeled by **non-determinism** of rewriting
  - Another model checker for the graph rewriting system: GROOVE<sup>3</sup>

LMNtal = Graphs + Rewrite rules

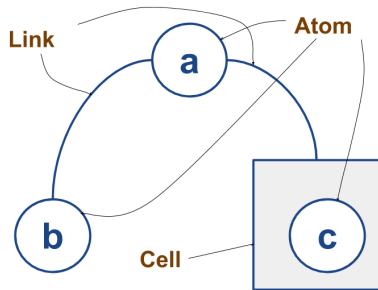


<sup>3</sup>Rensink, A.: The GROOVE Simulator: A Tool for State Space Generation, ProcApplications of Graph Transformations with Industrial Relevance, LNCS 3062, Springer-Verlag, pp.479–485, 2004.

LMNtal programs = **Graphs** + Rewrite rules

## Graph

$a(b, \{c\})$

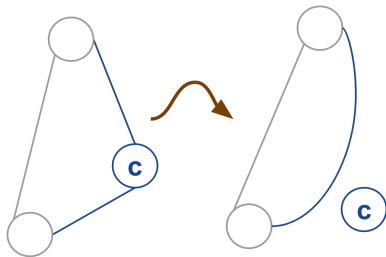


LMNtal programs = Graphs + Rewrite rules

## Rewrite rule

$c(X, Y) \text{ :- } X=Y, c$

- “=” connects links





# Non-determinism

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

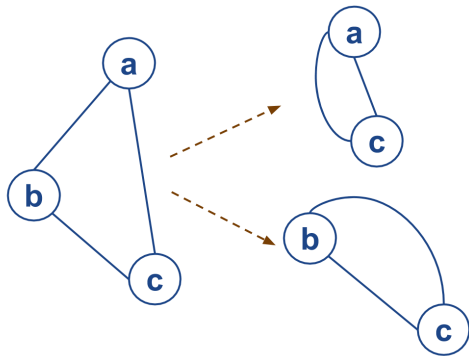
Implementation  
and Examples

Related work  
and  
Conclusion

If there are more than one rewriting pattern, rewriting is non-deterministic.

## Graph+Rewrite rule

```
a(b, c)
b(X, Y) :- X=Y
a(X, Y) :- X=Y
```



# Various model checkers

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

Checker	State Transition Graph	Desc	Spec
SPIN	Discrete transition system	Promela	LTL
NuSMV	Discrete transition system	SMV	LTL, CTL
UPPAAL	Timed automata	Timed automata	TCTL
PRISM	Markov decision processes	PEPA	PCTL
<b>SLIM</b>	Discrete transition system	<b>LMNtal</b>	LTL

# Research Question

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

## variations and extensions of SLIM

- “Real-time Model Checking using Explicit-time Methods in SLIM”<sup>3</sup>
  - Introduced real-time model checking to SLIM
- By *modifying* SLIM

## Problem

It is *not easy to modify* **complicated** and **large** software such as *model checkers*

## Research question

- How do we develop rapid prototypes of various model checkers *without modifying existing model checkers?*

---

<sup>3</sup>R. Shimizu, T. Kawabata, K. Ueda.: Real-time Model Checking using Explicit-time Methods in SLIM, JSSST, 2011.

## 1 Background

## 2 Proposed method

## 3 Implementation and Examples

## 4 Related work and Conclusion

# Proposed method

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

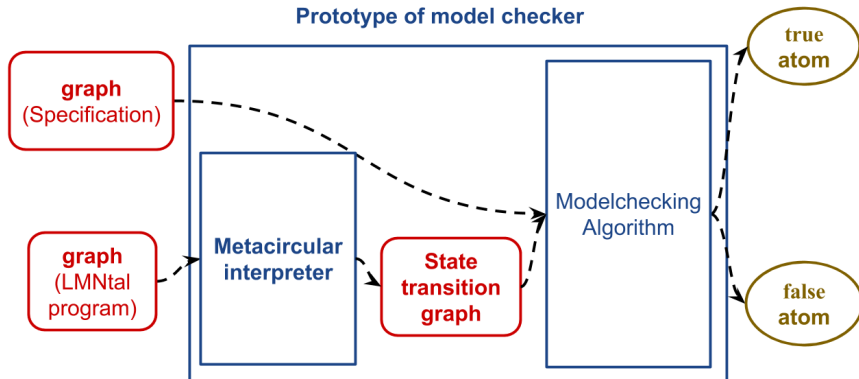
Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

Metaprogramming approach: **using a metacircular interpreter**

- Ease of implementation and modification



# Metacircular interpreter

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

Interpreters for the language  $\mathcal{L}$  implemented in  $\mathcal{L}$

- It is **easy to change** the **syntax** or **semantics** of the languages **without modifying** their implementations
- Prolog, Lisp

Useful interpreters are of moderate size

- Prolog: around 10 LOC
- Lisp: around 100 LOC

# Prolog metacircular interpreter

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- Input: Prolog programs
- Output: Answer substitutions

## basic\_metacircular\_interpreter.pl

```
prove(true).  
prove((Goal1, Goal2)) :-  
    prove(Goal1),  
    prove(Goal2).  
prove(Goal) :-  
    clause(Goal, Body),  
    prove(Body).
```

The most basic metacircular interpreter  
based on the **big-step semantics**

# Prolog metacircular interpreter

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

## generating\_proof\_trees.pl

```
:- op(500, xfy, <==).  
prove(true, true).  
prove((Goal1, Goal2), (Proof1, Proof2)) :-  
    prove(Goal1, Proof1),  
    prove(Goal2, Proof2).  
prove(Goal, Goal <== Proof) :-  
    clause(Goal, Body),  
    prove(Body, Proof).
```

- Prolog interpreter generating proof trees
- red parts are changed from basic\_metacircular\_interpreter.pl



# Features for implementations useful metacircular interpreters

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- Programs are expressed as fundamental data structures of the language (**Homoiconicity**)
- **Functionalities** of the implementation for program executions are available to programmers

## Prolog

- programs = terms
- functionalities for program executions = `clause`, `call`

## Lisp

- programs = lists
- functionalities for program executions = `eval`, `apply`

# Features for implementations useful metacircular interpreters

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- Programs are expressed as fundamental data structures of the language (**Homoiconicity**)
- **Functionalities** of the implementation for program executions are available to programmers

## Framework for metaprogramming in LMNtal

- **First-class rewrite rule**  $\Leftrightarrow$  Homoiconicity
- **APIs to use SLIM's features**  $\Leftrightarrow$  Functionalities for program executions

# First-class rewrite rule

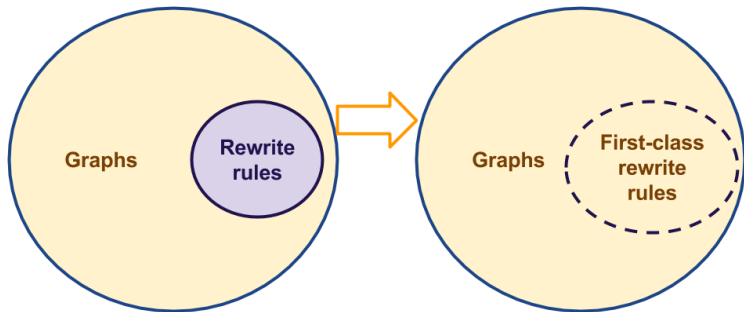
LMNtal is **not a homoiconic language**

⇒

**first-class rewrite rules** = hierarchical graphs behave like **rewrite rules**

LMNtal = Graphs + Rewrite rules

LMNtal = Graphs



# First-class rewrite rule

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

## Specification

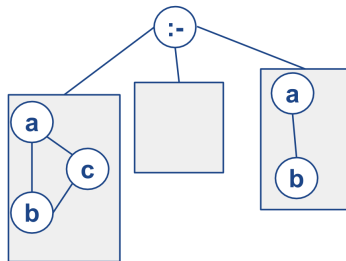
$\text{' :- ' } (\{Head\}, \{Guard\}, \{Body\})$

- expresses a rewrite rule,  $Head :- Guard \mid Body$

$\text{' :- ' } (\{a(b(X), c(X))\},$   
 $\{\},$   
 $\{a(X), b(X)\})$



$a(b(X), c(X)) :- a(X), b(X).$



# APIs to use SLIM's features

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- `rule.react_nd_set(RuleMem, GraphMem, RetRule, Ret)`
  - applies rewrite rules to graphs and returns all possible graphs by one step rewriting
- `membrane.eq(Mem0, Mem1, RetMem0, RetMem1, Ret)`
  - checks the isomorphism of two graphs
  - used to check **equivalence of states** with constructing state transition graphs

# rule.react\_nd\_set

```
rule.react_nd_set(RuleMem, GraphMem, RetRule, Ret)
```

- applies rewrite rules in a cell  
*RuleMem* to a process in a cell  
*GraphMem*
- After rewriting, a list of rewritten  
processes is connected to *Ret*
- *RetRule* is connected to *RuleMem*

```
rule.react_nd({a(X) :- b(X)},  
              {a(1), a(2), a(3)},  
              retrule,  
              ret).
```



```
retrule({a(X) :- b(X)}),  
ret([b(1), a(2), a(3)},  
    {a(1), b(2), a(3)},  
    {a(1), a(2), b(3)}}).
```

## Example of rule.react\_nd\_set

## LMNtal modelchecker in LMNtal

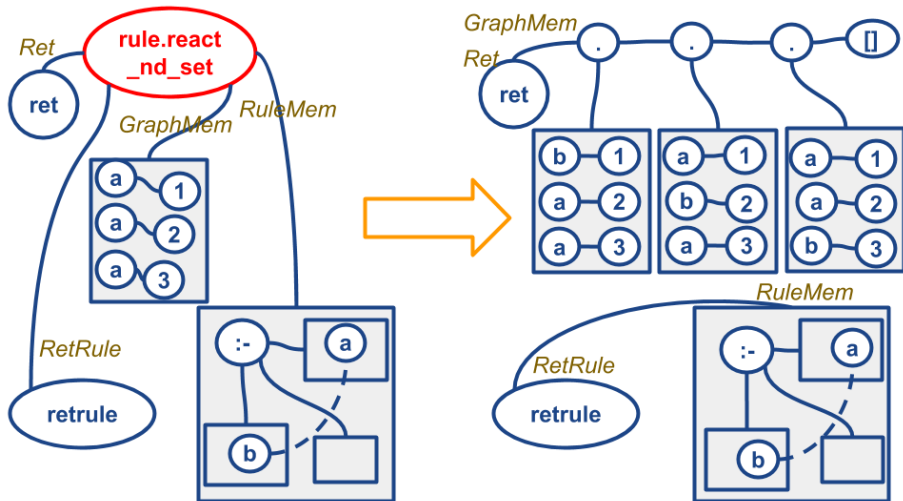
Tsunekawa  
Tomioaka  
Ueda

## Background

Proposed method

## Implementation and Examples

## Related work and Conclusion



# membrane.eq

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

`membrane.eq`(*Mem*<sub>0</sub>, *Mem*<sub>1</sub>, *RetMem*<sub>0</sub>, *RetMem*<sub>1</sub>, *Ret*)

- check isomorphism of graphs in a cell *Mem*<sub>0</sub> and graphs in a cell *Mem*<sub>1</sub>
- If they are isomorphic, connects a true atom to *Ret*
- otherwise, connects a false atom to *Ret*
- *RetMem*<sub>0</sub> is connected to *Mem*<sub>0</sub>, *RetMem*<sub>1</sub> is connected to *Mem*<sub>1</sub>

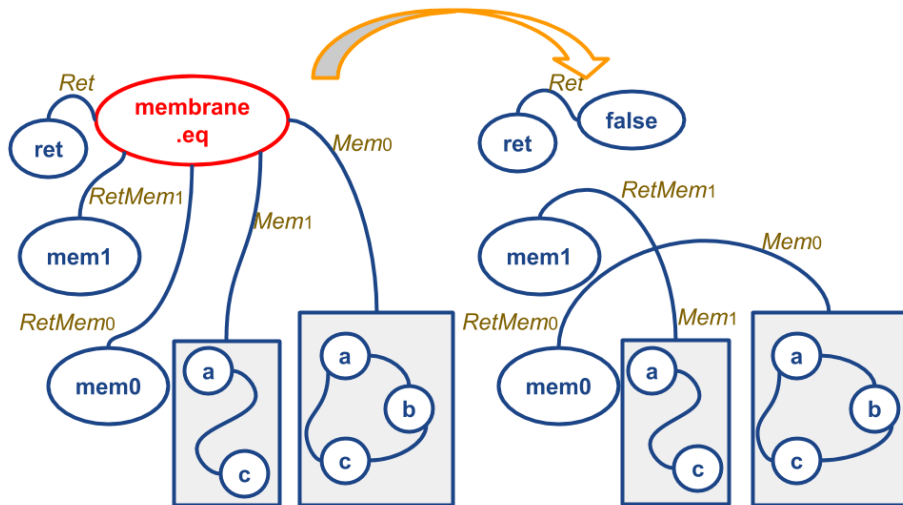
```
membrane.eq({a(b(X), c(X))},  
             {a(Y), c(Y)}  
             mem0,  
             mem1,  
             ret).
```



```
mem0({a(b(X), c(X))},  
     mem1({a(Y), c(Y)}),  
     ret(false).
```



# Example of `membrane.eq`



## 1 Background

## 2 Proposed method

## 3 Implementation and Examples

## 4 Related work and Conclusion

# LMNtal metacircular interpreters

LMNtal  
modelchecker  
in LMNtal

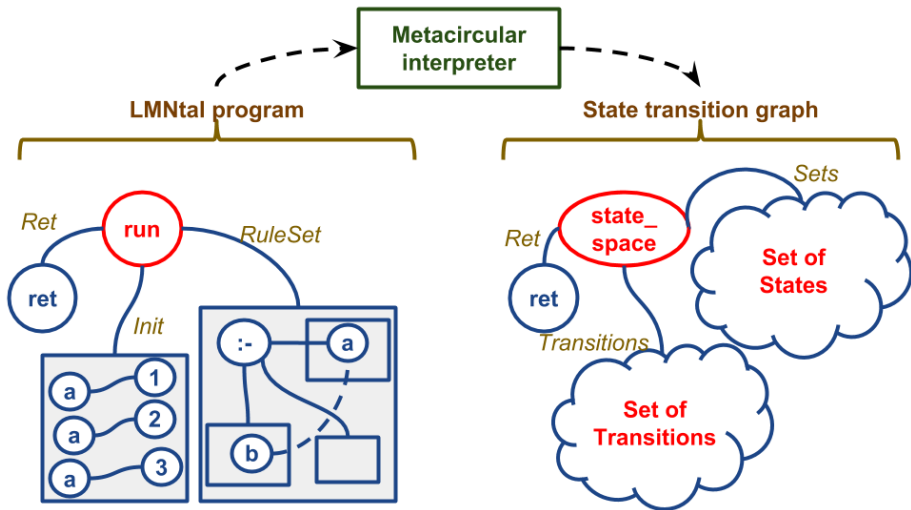
Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion



# Algorithm of metacircular interpreters

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

## Pseudocode: metacircular interpreters

```
 $S := \{s_0\}; T := \emptyset; Stack := \emptyset$   
push  $s_0$  Stack  
while  $Stack \neq \emptyset$   
   $s := pop\ Stack$   
   $succ := expand(s)$   
  forall  $s' \in succ$   
    if  $s'$  is a new state then  
       $S := S \cup \{s'\}$   
       $T := T \cup \{(s, s')\}$   
      push  $s'$  Stack  
    else if  $(s, s')$  is a new transition then  
       $T := T \cup \{(s, s')\}$   
  end forall  
end while
```

- LMNtal metacircular interpreters construct **state transition graphs** by DFS
- State transition graphs =  
Set of states  $S$   
+ Set of transitions  $T$

# Implementation of metacircular interpreters

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

## 9 rewrite rules

### lmntal\_metainterpreter.lmn

```
%%Initialize
run@@Ret = run(Rs, {$ini[]}) :-
    Ret = exp(Rs, [{ $ini[] }], hash.put(hash.init, {$ini[]}), hash.init).

%%First loop(while)
exp0@@Ret = exp({ $rs[], @rs }, [], S, T) :-
    Ret = state_space(S, T).
exp1@@Ret = exp({ $rs[], @rs }, [{ $f[] | Stk }, S, T) :-
    Ret = suc({ $rs[], Stk, { $f[] }, rule.react_nd_set({ $rs[] }, { $f[] }, S, T)).

%%Second loop(for)
succ0@@Ret = suc(Rs, Stk, { $f[] }, [], S, T) :-
    Ret = exp(Rs, Stk, S, T).
succ1@@Ret = suc(Rs, Stk, From, [{ $t[] | Succ }, S, T) :-
    Ret = ns(Rs, Stk, From, { $t[] }, Succ, Res, hash.get(S, { $t[] }, Res), T).

%%Checking for the new staes
new_st0@@Ret = ns(Rs, Stk, { $f[] }, { $t[] }, Succ, some({ $s[] }, S, T) :-
    Ret = nt(Rs, Stk, { $f[] }, { $t[] }, Succ, Res, S, hash.get(T, { d({ $f[] }, { $t[] }) }, Res)).
new_st1@@Ret = ns(Rs, Stk, { $f[] }, { $t[] }, Succ, none, S, T) :-
    Ret = suc(Rs, [{ $t[] | Stk }, { $f[] }, Succ, hash.put(S, { $t[] }, hash.put(T, { d({ $f[] }, { $t[] }) }))).

%%Checking for the new transitions
new_tr0@@Ret = nt(Rs, Stk, From, { $t[] }, Succ, some({ $s[] }, S, T) :-
    Ret = suc(Rs, Stk, From, Succ, S, T).
new_tr1@@Ret = nt(Rs, Stk, { $f[] }, { $t[] }, Succ, none, S, T) :-
    Ret = suc(Rs, Stk, { $f[] }, Succ, S, hash.put(T, { d({ $f[] }, { $t[] }) })).
```

# LTL model checker<sup>45</sup>

LMNtal  
modelchecker  
in LMNtal

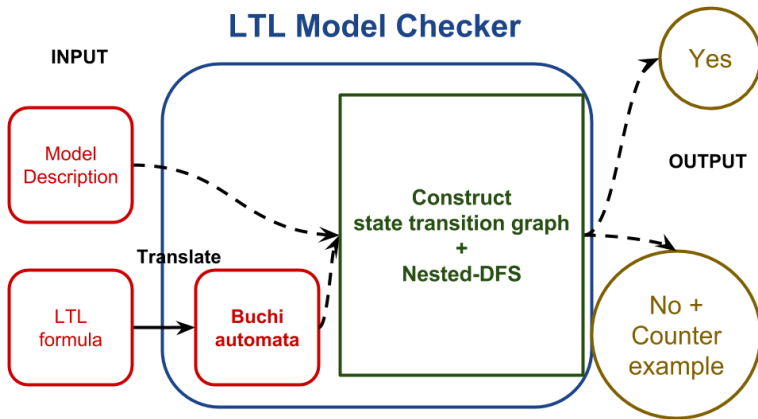
Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion



<sup>4</sup> Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-Efficient Algorithms for the Verification of Temporal Properties, Computer-Aided Verification, Springer-Verlag, pp.129–142, 1992.

<sup>5</sup> Vardi, M., Wolper, P.: Reasoning about infinite computations, Information and Computation, Vol. 115, No. 1, pp.1–37, 1994.

# LTL model checker in LMNtal

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

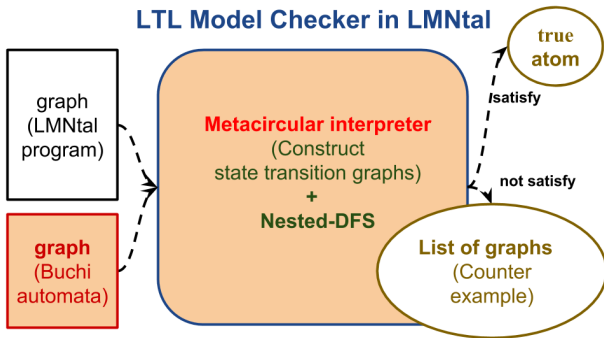
Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- LTL model checker is implemented by 46 rewrite rules
  - On-the-fly model checking by Nested-DFS
- We checked our implementation with 5 examples in the SLIM package.



# CTL model checking<sup>6</sup>

LMNtal  
modelchecker  
in LMNtal

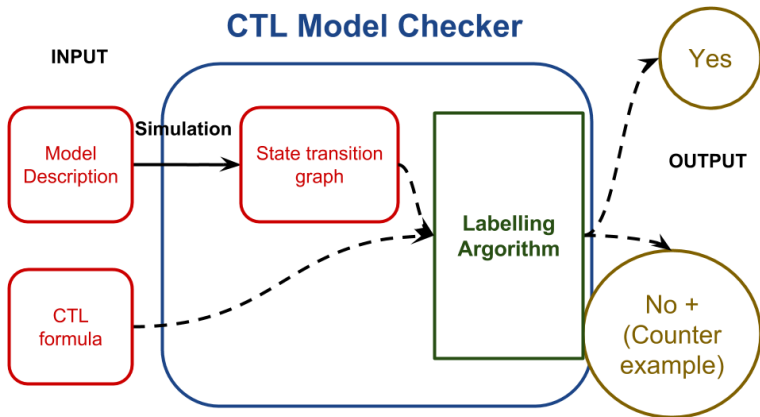
Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion



<sup>6</sup>Clarke, E. M., Emerson, E. A.: Design and synthesis of synchronization skeletons using branching time temporal logic, Proc. Workshop of Logic of Programs, LNCS 131, Springer-Verlag, pp.52–71, 1981.



# CTL model checker in LMNtal

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

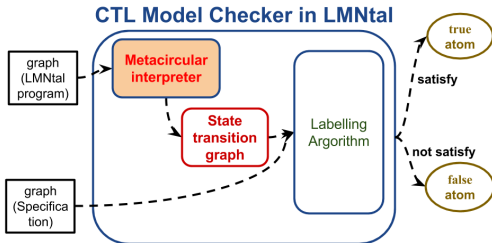
Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- CTL model checker is implemented by 71 rewrite rules
  - Abstraction of states
  - composed of procedures that computes a set of states satisfying each CTL operator
  - Modularity
- We checked our implementation with 4 examples including “Model of the oven”<sup>7</sup>



<sup>7</sup> Clarke, E. M., Grumberg, O., Long, D. E., Model Checking, MIT Press, 1999.

# Model of the oven

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

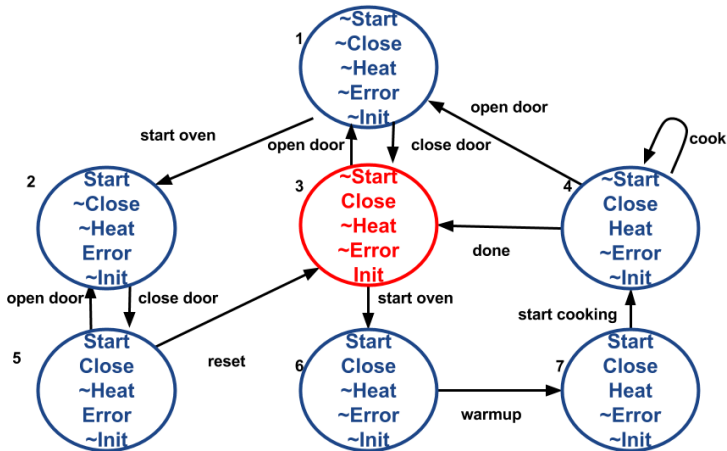
Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

$AG (EF \textit{Init}) =$  “the initial state is reachable from any state”



## 1 Background

## 2 Proposed method

## 3 Implementation and Examples

## 4 Related work and Conclusion

# Related work

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

McErlang<sup>8</sup>: model checker for Erlang implemented in Erlang

- **Similarity**: source language = implementation languages
- **Difference**: **flexibility** of the program manipulation
  - Erlang: functions = first-class objects,  $\neq$  data structures
  - LMNtal: rewrite rules = first-class objects, = data structures

---

<sup>8</sup>Fredlund, L., Svensson, H.: McErlang: a model checker for a distributed functional programming language, Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, pp.125–136, 2007.

# Future work

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- **Improving the efficiency** of the metacircular interpreter
  - The present metacircular interpreter is **2–3 orders of magnitude slower** than SLIM
- **Implementation of various model checkers** other than LTL model checkers and CTL model checkers.
  - **TCTL model checking** for the timed automata
  - **PCTL model checking** for the Markov decision processes

# Conclusion

LMNtal  
modelchecker  
in LMNtal

Tsunekawa  
Tomioka  
Ueda

Background

Proposed  
method

Implementation  
and Examples

Related work  
and  
Conclusion

- proposed a **metaprogramming approach** to developing prototypes of various model checkers
- designed **frameworks** for metaprogramming
- implemented **model checkers** based on **metacircular interpreters** in LMNtal

**Thank you for the listening**