Linear Temporal Logic of Rewriting Model Checker

Kyungmin Bae and Jose Meseguer

Department of Computer Science
University of Illinois, Urbana-Champaign

MVD'09

Overview

- State-based vs. Action-based Logic
  - LTL, CTL, CTL* / A-CTL*, Hennessy-Milner logic
  - System/Specification Mismatch Problem
- Linear Temporal Logic of Rewriting (LTLR)
  - Extension of LTL with spatial action patterns
  - Provide very expressive action patterns by rewriting
- Maude LTLR model checker
  - Extension of Maude LTL model checker
  - Support spatial action patterns
  - More general (user-definable) action patterns

Outline

- Motivating Example
- Linear Temporal Logic of Rewriting
- Model Checking Algorithm of LTLR
- Example
- Conclusion

Motivating Example (Specification as Rewriting Logic)

- Dekker's algorithm
  - Mutual exclusion algorithm
    - Two processes
    - Shared variables
    - Critical section (crit) terminate.
    - Remaining code (rem) may not terminate.
    - crit or rem do NOT touch other variables

```
repeat
 'c1 := 1 ;
 while 'c2 = 1 do
  if 'turn = 2 then
   'c1 := 0 ;
   while 'turn = 2 do skip od ;
   'c1 := 1
  fi
 od ;
 crit ;
 'turn := 2 ;'c1 := 0 ;
 rem
forever
```

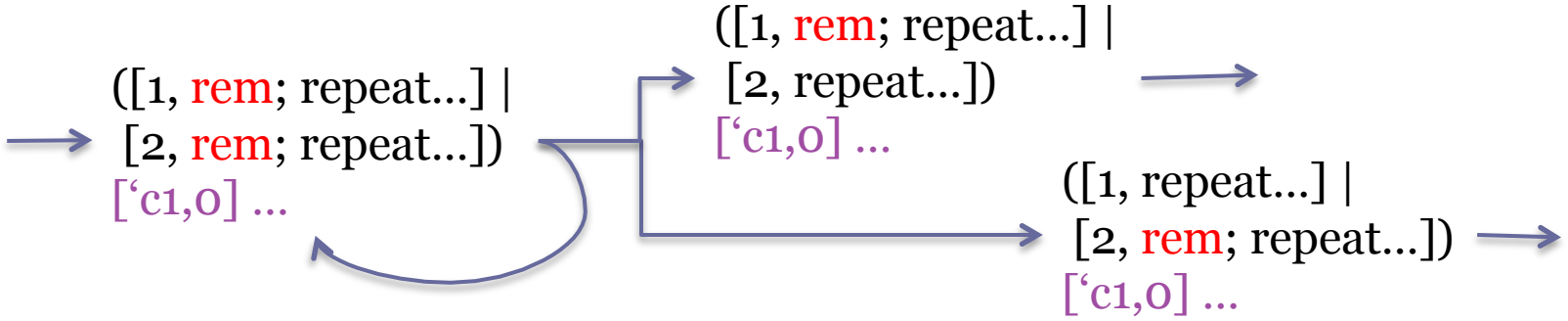
Process 1

Motivating Example (Specification as Rewriting Logic)

- State representation
  - Process (Id, Code), Shared Memory
  - ([1, repeat
    ‘c1 := 1 ; … ; ‘turn := 2 ; ‘c1 := 0 ; rem
    forever] |
    [2, repeat
    ‘c2 := 1 ; … ; 'turn := 1 ; 'c2 := 0 ; rem
    forever]
) [‘c1,0] [‘c2,0] [‘turn,1]
  - Term pattern : ([I, R] | PROC) M
    - Process Id I, Code R, Process PROC, Shared Memory M
    - Commutative operator |

Motivating Example (Specification as Rewriting Logic)

- Semantics as rewriting rules
  - [repeat] : ([I, repeat P forever ; R] | PROC) M
    => ([I, P ; repeat P forever ; R] | PROC) M .
  - [ift]: ([I, if T then P fi ;R] | PROC) M =>([I, P;R] | PROC) M
    if eval(T,M) == true.
  - [crit] : ([I, crit ; R] | PROC) M => ([I, R], PROC) M .
  - [rem] : ([I, rem; R] | PROC) M => ([I, R] | PROC) M .
  - [rem']: ([I, rem; R] | PROC) M => ([I, rem; R] | PROC) M .



Motivating Example (Property Specification by Equations)

- Fairness property

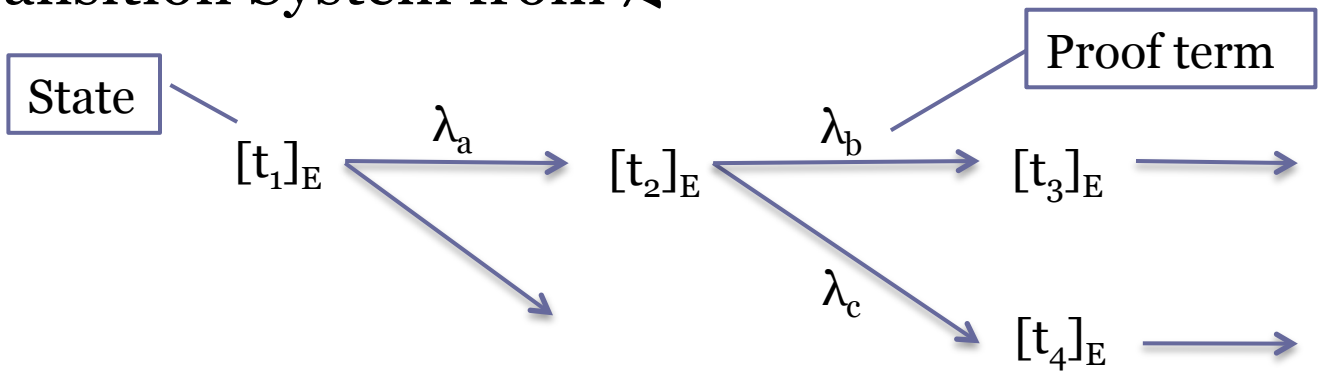
$$\Box\Diamond\, exec.p1 \Rightarrow \Box\Diamond\, crit.p1$$

- Proposition Definition in a state-based logic
  - Labeling operator |=
  - crit(p1) : p1 is in critical section.
    - ([p1, crit ; R] | PROC) M |= crit(p1) = true
  - exec(p1) : p1 is just executed.
    - Need to change a state and rules by adding p1 explicitly
      ([I, R] | PROC) M / p1 |= exec(p1) = true

Both states and actions are needed for the proposition!

Rewriting Logic Representation

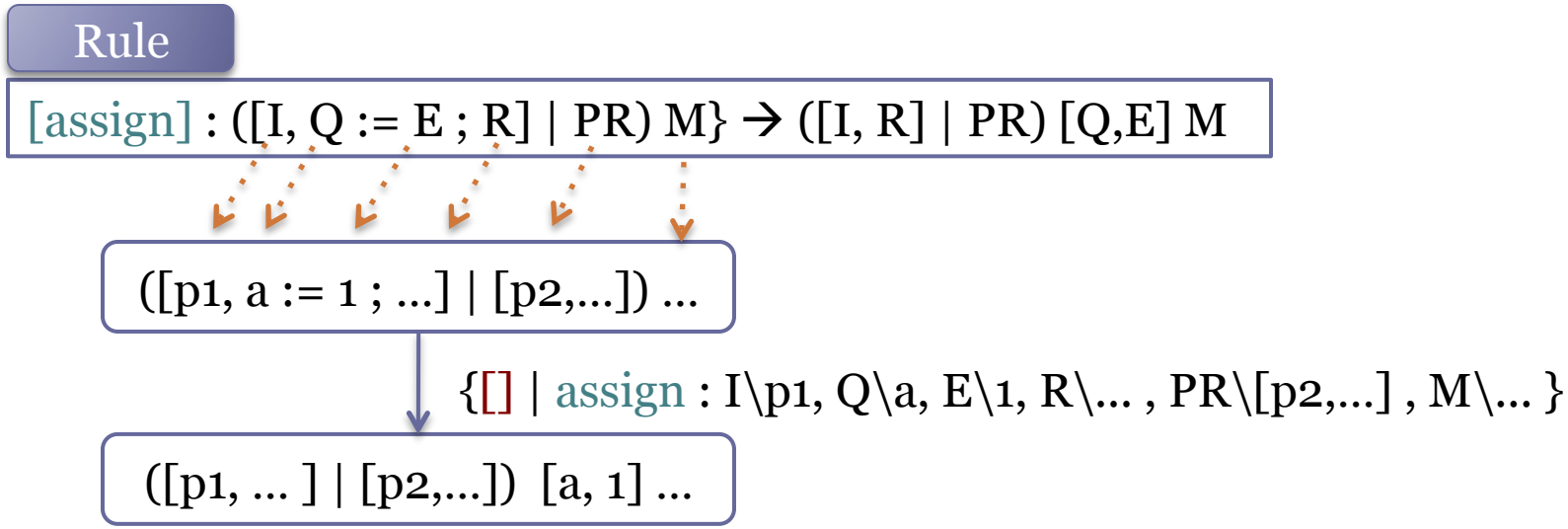
- Rewriting Logic Specification $\mathcal{R} = (\Sigma, E, R)$
  - Signature, Equation E, Rewriting rule R
- Transition System from $\mathcal{R}$



  - States
    - Equivalent classes of terms defined by equations
  - Actions
    - One-step proof terms defined by rewriting rules

Rewriting Logic Representation

- One Step Proof Term
  - Presents a term rewriting by a rule
  - Involves a context, a rule label, and a substitution
  - {CONTEXT | LABEL : SUBSTITUTION}



Linear Temporal Logic of Rewriting

- Syntax of LTLR
  - Atoms
    - proposition P
    - spatial action pattern δ
  - Logical connective
  - Temporal operator

LTLR = LTL + δ

- Semantics of LTLR
  - Path
    - $state_1 \rightarrow proofterm_1 \rightarrow state_2 \rightarrow proofterm_2 \rightarrow state_3 \rightarrow \ldots$
  - $s_1, p_1, s_2, p_2, \ldots \models \delta$ if and only if $p_1 \models \delta$

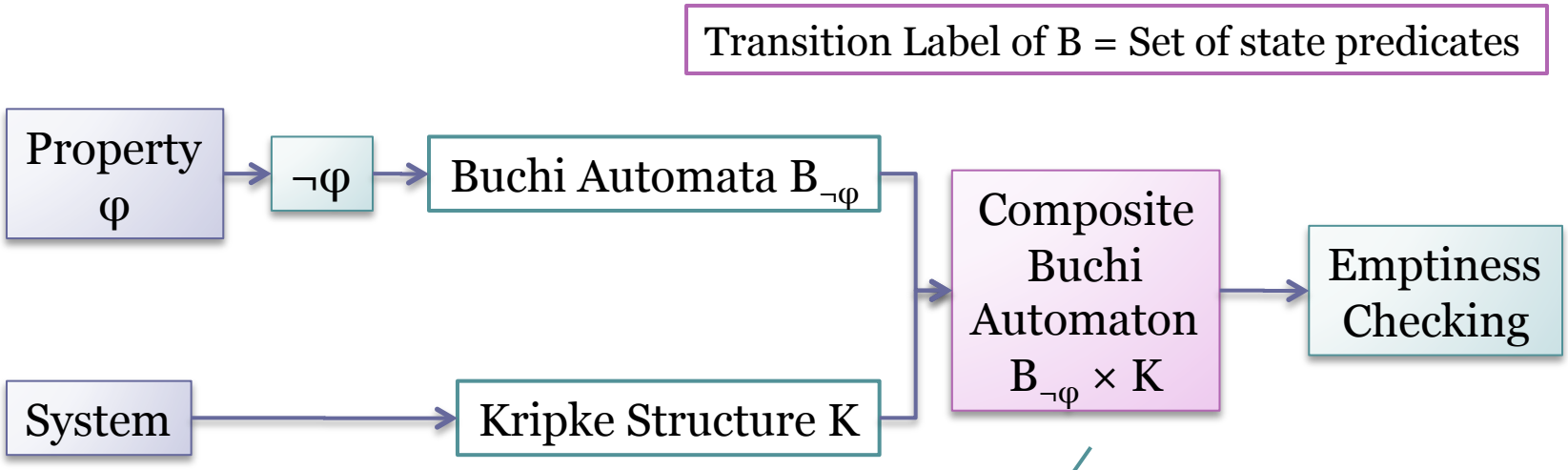
Linear Temporal Logic of Rewriting

- Spatial Action Pattern
  - Involves a set of proof terms
    - {assign} : { ... | assign : ... }
    - {assign : I \ p1} : { ... | assign : I \ p1 ; ... }
  - Definition
    - {C | R : S} |= {R} = true .
    - {C | R : S'} |= {R : S} = true if S ⊂ S' .
    - {C | R : S} |= {C | R} = true .
    - {C | R : S'} |= {C | R : S} = true if S ⊂ S' .
  - And More!

C : Context
R : Rule label
S : Substitution

Model Checking Algorithm of LTLR

- LTL Model Checking

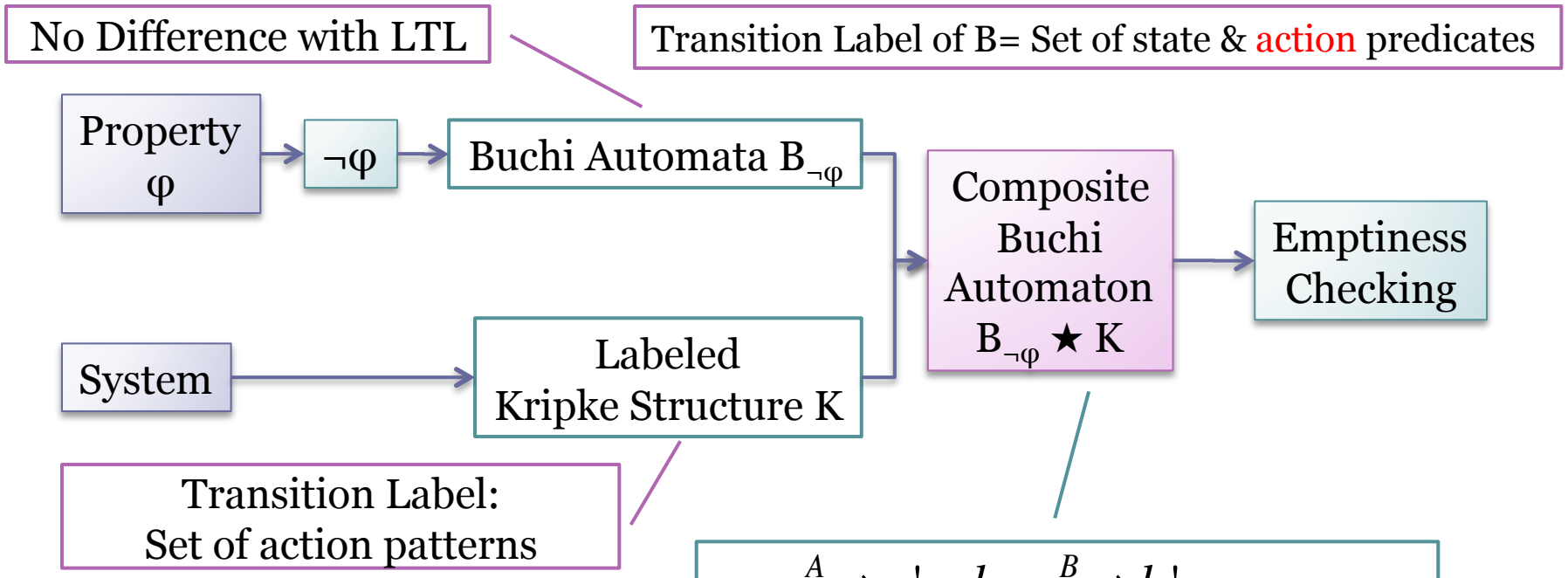


s, s' : System states
b, b' : Automata states
L : State labeling function

$$\frac{s \rightarrow s' \qquad b \xrightarrow{L(s)} b'}{(s,b) \rightarrow (s',b')}$$

Model Checking Algorithm of LTLR

- LTLR Model Checking



s, s' : System states
b, b' : Automata states
L : State labeling function

Model Checking Algorithm of LTLR

- Theorem (Correctness)
  - Given a labeled Kripke structure K and a LTLR formula φ, there is a Buchi automaton $B_{\neg\varphi}$ such that

$$K \models \varphi \quad \Leftrightarrow \quad L(K \star B_{\neg\varphi}) = \varnothing$$

Example Revisited (Dekker's Algorithm with 2 processes)

- Action patterns for execution
  - {C | R : S'} |= exec(I) = true if ('I \ I) ⊂ S' .
- Fairness Property 1 (False)
  - []<> exec(1) -> []<> in-crit(1)
- Fairness Property 2
  - []<> exec(1) /\ []<> exec(2) /\ []<> ~ in-rem(1) -> []<> in-crit(1)

Example (Dining Philosopher Problem)

- Rewriting Logic representation
  - Philosopher
    - < ID : Philosopher | state : STATE, sticks : NAT >
    - STATE: thinking, hungry, eating
  - Rules
    - [hungry]: < ID : Philosopher | state : thinking >
      => < ID : Philosopher | state : hungry > .
    - [grab] : < ID : Philosopher | state : hungry, sticks : N >
      chopstick(STICK)
      => < ID : Philosopher | state : eat?(N+1), sticks : N+1 >
      if ID can use stick STICK .
    - [stop] : < ID : Philosopher | state : eating >
      => < ID : Philosopher | state : thinking, sticks : 0 >
      chopstick(ID) chopstick(right(ID)) .

Example (Dining Philosopher Problem)

- Deadlock-free? (False)
  - [] ~ deadlock
- Deadlock-free Solutions (5 philosopher)
  - Each philosopher always grabs higher chopstick first.
    - ([] ~ lowerFirst) -> [] ~ deadlock
    - {C | 'grab : 'ID \ I ; 'STICK \ J ; 'N \ 0 ; *OTHER*} |= lowerFirst
      = true
      if (I < 5 and J== I) or (I == 5 and J== right(I))
  - Each philosopher always grabs all chopstick at once.
    - ([] ~ partialGrab) -> [] ~ deadlock
    - {C < I : Philosopher | sticks : 1 > | 'grab : 'N \ 0 ; *OTHER*} |=
      partialGrab
      = true .

Related Work

- TLR*, and previous work
  - J. Meseguer, The temporal logic of rewriting, 2007
  - K. Bae and J. Meseguer, A rewriting-based model checker for the linear temporal logic of rewriting, 2008
- SE-LTL and its extension
  - S. Chaki, E. Clarke, et al., State/event-based software model checking, 2004
  - S. Chaki, E. Clarke, et al., State/event software verification for branching-time specifications, 2005
- ESTL for Petri net
  - E. Kindler and T. Vesper, ESTL: A temporal logic for events and states, 1998

Conclusion and Future Work

- Linear Temporal Logic of Rewriting
  - LTL + spatial action patterns
- Maude LTLR model checker
  - Spatial action patterns by equations
- Future work
  - Optimization
  - Fairness condition
  - More generalization for rewriting system

Thank you!