Linear Temporal Logic of Rewriting Model Checker

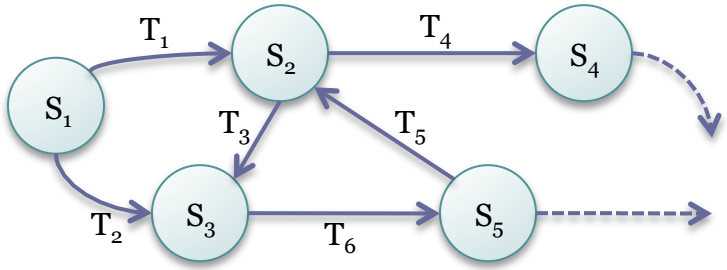
Kyungmin Bae and Jose Meseguer

Department of Computer Science
University of Illinois, Urbana-Champaign

Jan 5, 2010

Overview

Concurrent Model



S_i : State
T_i : Transition

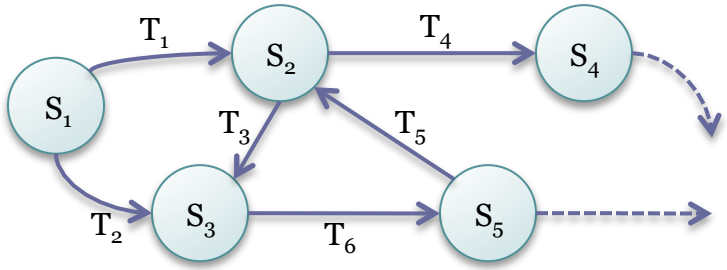
	State-based temporal logic	Event-based temporal logic
Atomic Propositions	On states ($S_1 \models p_1$, $S_1 \models p_2$, ...)	On transitions ($T_1 \models a_1$, $T_2 \models a_2$, ...)
Properties?	State-dependent	Event-dependent
Example	LTL, CTL, CTL*, ...	A-CTL*, Hennessy-Milner logic, ...

How about properties associated to both states and transitions?

➔ System/Specification Mismatch Problem

Overview

Concurrent Model
(by Rewriting Logic)



Linear Temporal Logic of Rewriting (= LTL + spatial action pattern)

Atomic Propositions	Either on states or on transitions ($S_1 \models p_1$, $T_1 \models a_1$, $S_1 \models p_2$, $T_2 \models a_2$, ...)

- Maude LTLR model checker
  - No state-space blow-up
  - Extension of Maude LTL model checker
  - Support spatial action patterns
  - More general (user-definable) action patterns

Outline

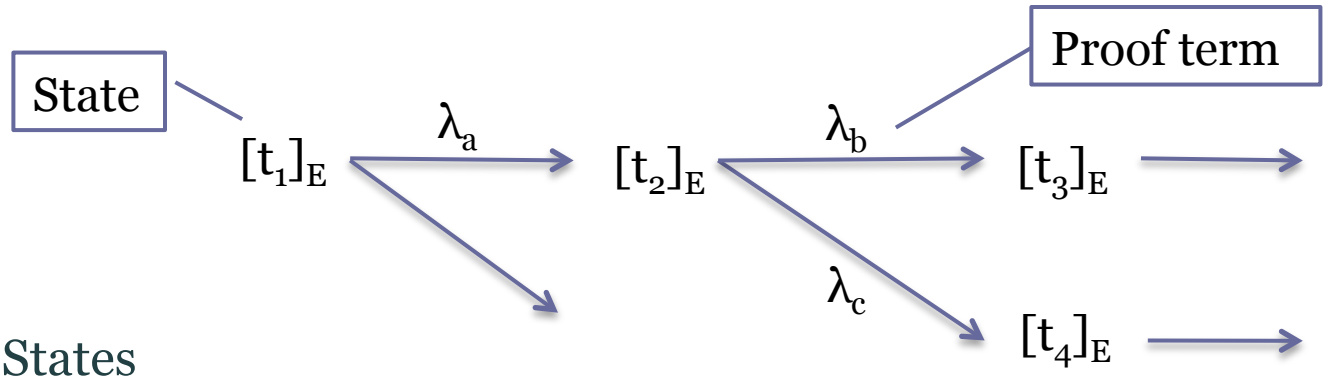
- Rewriting Logic
- Motivating Example
- Linear Temporal Logic of Rewriting
- Model Checking Algorithm of LTLR
- Some Examples
- Related Works and Conclusion

Rewriting Logic

- Term rewriting
  - Terms
    - $f(h(c_1, X), Y)$
    - functional operators, constants, variable
  - Rules
    - $h(X,Y) \rightarrow g(X)$
    - e.g., $f(h(c_1, g(c_2)), g(c_3))$ is rewritten to $f(g(c_1), g(c_3))$
      - with the substitution $\{X\backslash c_1, Y\backslash g(c_2)\}$
- What is the Rewriting Logic?
  - Defined by term rewriting
  - Logical framework
  - Formal specification of a concurrent system

Rewriting Logic

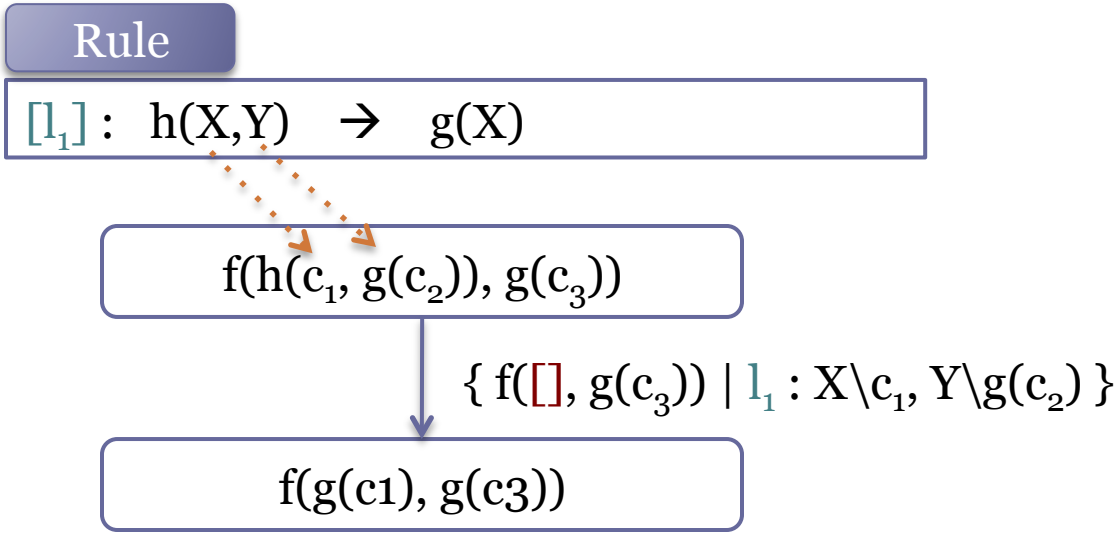
- Rewriting Logic $\mathcal{R} = (\Sigma, E, R)$
  - Signature Σ defines terms.
  - Equations E defines state spaces (by equivalence relations).
  - Rewriting rules R defines nondeterministic concurrent transitions.
- Transition System from $\mathcal{R}$



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

Rewriting Logic

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



Motivating Example (Specification as Rewriting Logic)

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

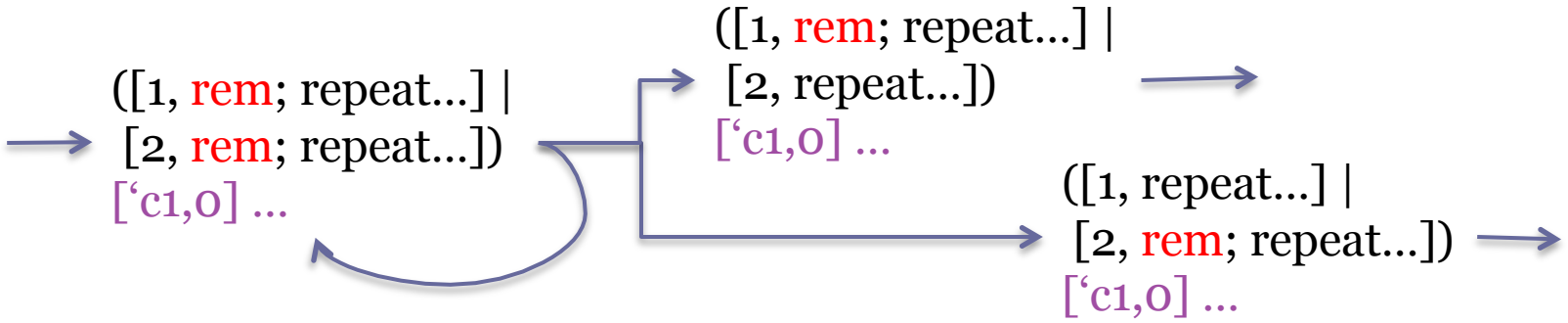
Process 1

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 representation : ([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
  []<> exec.p1 => []<> crit.p1

- Proposition Definition in a state-based logic
  - Labeling operator |= : State Prop -> Bool
  - 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!

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!
    - Proofterm |= ActionPattern = true

C : Context
R : Rule label
S : Substitution

Linear Temporal Logic of Rewriting

- Syntax of LTLR
  - Atoms
    - proposition P
    - spatial action pattern δ
  - Logical connective
    - Negation(¬), Conjunction(∧), Disjunction(∨), …
  - Temporal operator
    - Next(○), Globally([]), Future(<>), …

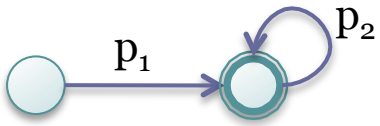
LTLR = LTL + δ

Linear Temporal Logic of Rewriting

- Semantics of LTLR
  - Path
    - $state_1 \rightarrow proofterm_1 \rightarrow state_2 \rightarrow proofterm_2 \rightarrow state_3 \rightarrow \ldots$
  - Spatial action pattern
    - $s_1, p_1, s_2, p_2, \ldots \models \delta$ if and only if $p_1 \models \delta$
  - Other cases are identical to those of LTL
    - $s_1, p_1, \ldots \models P$ if and only if $s_1 \models P$.
    - $s_1, p_1, \ldots \models \neg\varphi$ if and only if not $s_1, p_1, \ldots \models \delta$.
    - $s_1, p_1, \ldots \models \varphi_1 \wedge \varphi_2$ if and only if $s_1, p_1, \ldots \models \varphi_1$ and $s_1, p_1, \ldots \models \varphi_2$.
    - $s_1, p_1, \ldots \models \langle\rangle \varphi$ if and only if $s_i, p_i, \ldots \models \delta$ for some $i \geq 1$.
    - $s_1, p_1, \ldots \models [] \varphi$ if and only if $s_i, p_i, \ldots \models \delta$ for each $i \geq 1$.
    - ...

Model Checking Algorithm of LTLR

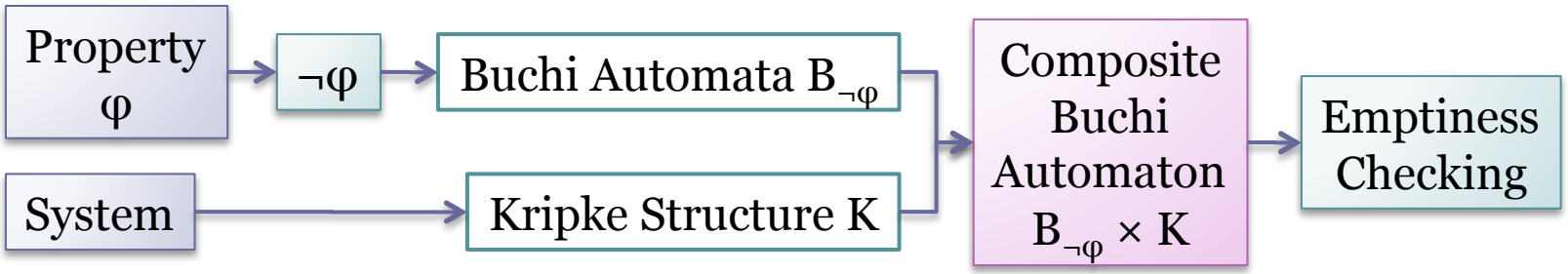
- LTL Model Checking
  - M |= φ ?
    - System path of M: $state_1$ $state_2$ $state_3$ …
    - Does every path of a system M satisfies φ?
      - Is there any path of M satisfying ¬φ?
  - Any LTL formula has an equivalent Buchi automaton.
    - e.g., $p_1 \wedge [] p_2$



  - Find an accepted string of a Buchi automaton for ¬φ.
    - Path String $L(state_1)\ L(state_2)\ L(state_3)$ …
      - $L(state_i) = \{ p \in AP \mid (state_i \models p) = true \}$
        - L: a state labeling function
        - AP: a set of atomic propositions in φ

Model Checking Algorithm of LTLR

- LTL Model Checking
  - Path String: $L(s_1)\ L(s_2)\ L(s_3)\ \ldots$



s, s' : System states
b, b' : Automata states

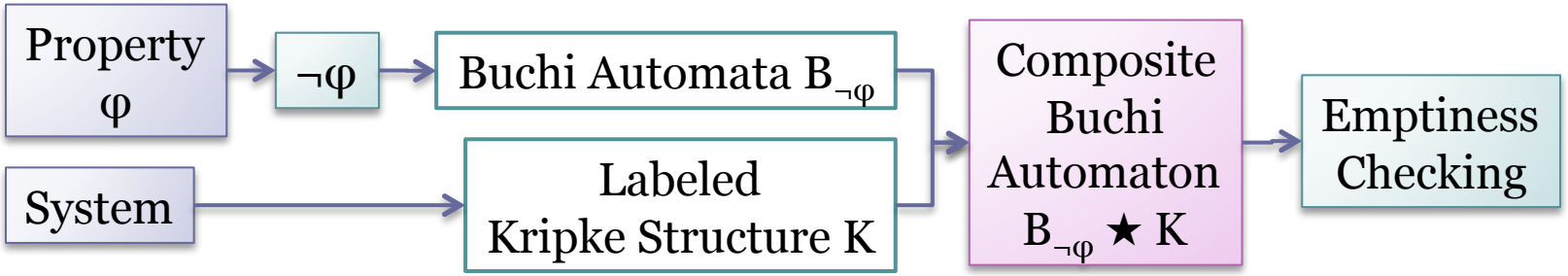
$$\frac{s \rightarrow s' \qquad b \xrightarrow{L(s)} b'}{(s,b) \rightarrow (s',b')}$$

Model Checking Algorithm of LTLR

- LTLR Model Checking
  - $M \models \varphi$
    - Find a path of M satisfies $\neg\varphi$
    - System path of M: $state_1$, $proofterm_1$, $state_2$, $proofterm_2$, $state_3$, ...
  - Path String $L(state_1)\ A_1\ L(state_2)\ A_2\ L(state_3)\ A_3$, ...
    - $L(state_i) = \{ p \in AP \mid (state_i \models p) = true \}$
    - $A_i = \{ a \in W \mid (proofterm_i \models a) = true \}$
      - W : a set of spatial action patterns in φ
  - Union Path string!
    - $(L(s_1) \cup A_1)\ (L(s_2) \cup A_2)\ (L(s_3) \cup A_3)$ ... [if $L(s_i) \cap A_i = \varnothing$]
    - Buchi automata of $\neg\varphi$ is the same with one of LTL case!

Model Checking Algorithm of LTLR

- LTLR Model Checking
  - Path string: $(L(s_1) \uplus A_1)\,(L(s_2) \uplus A_2)\,(L(s_3) \uplus A_3)\,\ldots$



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

$$\frac{s \xrightarrow{A} s' \quad b \xrightarrow{B} b'}{(s,b) \rightarrow (s',b')} \; B = L(s) \uplus A$$

Model Checking Algorithm of LTLR

- Theorem
  - 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)
  - Why does the above property fail?
    - Some processe may never execute.
    - rem may not terminate.
- Fairness Property 2
  - (<>[] enabled(1) -> []<> exec(1) /\
    <>[] enabled(2) -> []<> exec(2) /\ []<> ~ in-rem(1))
    ->
    ([]<> exec(1) -> []<> in-crit(1))

Example (Dining Philosopher Problem)

- Illustration

From Wikipedia

- Rewriting Logic representation
  - Philosopher
    - < ID : Philosopher | state : STATE, sticks : NAT >
    - STATE: thinking, hungry, eating

Example (Dining Philosopher Problem)

- Rewriting Logic representation
  - Initial states (with 5 Philosophers)
    - < 1 : Philosopher | state : thinking, stick : 0 > chopstick(0)
      ...
      < 5 : Philosopher | state : thinking, stick : 0 > chopstick(5)
  - 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 Works

- 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
  - Improve an user interface
  - Optimization
  - More generalization for rewriting system
    - Fairness condition

Thank you!