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

```plaintext
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.

\[
\begin{align*}
([1, \text{rem}; \text{repeat}... ] | \hspace{1cm} ([1, \text{rem}; \text{repeat}... ] | \hspace{1cm} ([1, \text{repeat}... ] | \\
[2, \text{rem}; \text{repeat}... ] ) & \rightarrow [2, \text{repeat}... ] ) \\
[\text{\text{'c1,0}]} & \rightarrow [\text{\text{'c1,0}}]
\end{align*}
\]
Motivating Example
(Property Specification by Equations)

• Fairness property

□◊ exec.\(p_1\) ⇒ □◊ crit.\(p_1\)

• Proposition Definition in a state-based logic
  ▫ Labeling operator |=
  ▫ crit(\(p_1\)) : \(p_1\) is in critical section.
    • \([p_1, crit ; R] \mid PROC\) M |= crit(\(p_1\)) = true
  ▫ exec(\(p_1\)) : \(p_1\) is just executed.
    • Need to change a state and rules by adding \(p_1\) explicitly
      \([I, R] \mid PROC\) M / \(p_1\) |= exec(\(p_1\)) = true

Both states and actions are needed for the proposition!
Rewriting Logic Representation

- Rewriting Logic Specification $R = (\Sigma, E, R)$
  - Signature, Equation $E$, Rewriting rule $R$

- Transition System from $R$
  - States
    - Equivalent classes of terms defined by equations
  - Actions
    - One-step proof terms defined by rewriting rules

\[ \begin{align*}
[t_1]_E & \xrightarrow{\lambda_a} [t_2]_E \\
[t_2]_E & \xrightarrow{\lambda_b} [t_3]_E \\
&t_3]_E \xrightarrow{\lambda_c} [t_4]_E
\end{align*} \]
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\}

\[
\begin{align*}
\text{Rule} \quad & \quad \text{[assign]} : ([I, Q := E ; R] | PR) M \rightarrow ([I, R] | PR) [Q,E] M \\
& \quad ([p1, a := 1 ; ...] | [p2, ...]) ... \\
& \quad \{[ ] | \text{assign} : I\{p1, Q\{a, E\{1, R\} ... , PR\{p2, ... \} , M\} ... \} \\
& \quad ([p1, ... ] | [p2, ...]) [a, 1] ...
\end{align*}
\]
Linear Temporal Logic of Rewriting

- **Syntax of LTLR**
  - Atoms
    - proposition P
    - spatial action pattern $\delta$
  - Logical connective
  - Temporal operator

- **Semantics of LTLR**
  - Path
    - $\text{state}_1 \rightarrow \text{proofterm}_1 \rightarrow \text{state}_2 \rightarrow \text{proofterm}_2 \rightarrow \text{state}_3 \rightarrow \ldots$
    - $s_1, p_1, s_2, p_2, \ldots \models \delta$ if and only if $p_1 \models \delta$

$LTLR = LTL + \delta$
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\} \models \{R\} = true.
    - \{C | R : S’\} \models \{R : S\} = true if \(S \subseteq S’\).
    - \{C | R : S\} \models \{C | R\} = true.
    - \{C | R : S’\} \models \{C | R : S\} = true if \(S \subseteq S’\).
- **And More!**
Model Checking Algorithm of LTLR

- LTL Model Checking

Property $\phi$ $\rightarrow$ Buchi Automata $B_{\neg\phi}$ $\rightarrow$ Composition $B_{\neg\phi} \times K$

Transition Label of $B = $ Set of state predicates

System $\rightarrow$ Kripke Structure $K$

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

$s \rightarrow s' \quad b \xrightarrow{L(s)} b' \
(s, b) \rightarrow (s', b')$
Model Checking Algorithm of LTLR

- **LTLR Model Checking**

  Transition Label of $B = \text{Set of state & action predicates}$

  No Difference with LTL

  Transition Label: Set of action patterns

  $s, s' : \text{System states}$
  $b, b' : \text{Automata states}$
  $L \quad : \text{State labeling function}$

  $s \xrightarrow{A} s' \quad b \xrightarrow{B} b'$
  $(s, b) \rightarrow (s', b')$

  $B = L(s) \cup A$
Model Checking Algorithm of LTLR

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

\[
K \models \varphi \iff L(K \star B_{\neg \varphi}) = \emptyset
\]
Example Revisited
(Dekker’s Algorithm with 2 processes)

- **Action patterns for execution**
  - \( \{C \mid R : S'\} \models \text{exec}(I) = \text{true} \text{ if } (I \setminus I) \subseteq S' \).

- **Fairness Property 1 (False)**
  - \([-\square]<> \text{exec}(1) \rightarrow [-\square]<> \text{in-crit}(1)\]

- **Fairness Property 2**
  - \([-\square]<> \text{exec}(1) \land [-\square]<> \text{exec}(2) \land [-\square]<> \sim \text{in-rem}(1) \rightarrow [-\square]<> \text{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 \ o : 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 \ o ; OTHER} |=
      partialGrab
      = true .

Kyungmin Bae  9/20/09
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
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!