Inference with Rewriting Rules

Lecture Note 04
CafeOBJ Team for JAIST-FSSV2010
Introduction to the theory of term rewriting systems (TRS), which gives us an efficient way to equational reasoning.

- The reduction command in OBJ languages is implemented based on TRS.
- For proving an equation in CafeOBJ, first decompose it (make proof passages), then apply the reduction command to them.
Overview of Lecture 04

♦ This lecture consists of two parts:
  • PART I : Introduction to TRS and its fundamental properties
    • Termination and Confluence
  • PART II : Advanced topic for OBJ specifications
    • Sufficient completeness
    • Conditional equations
    • Associative and Commutative operators
    and so on
Inference with Rewriting Rules

PART I

Introduction to TRS and its fundamental properties
Term rewriting system
The term rewriting system (TRS) gives us an efficient way to prove equations by regarding an equation as a left-to-right rewrite rule.

\[
\begin{align*}
eq N + 0 &= N . \\
eq M + s N &= s (M + N) .
\end{align*}
\]

\[
\begin{align*}
N + 0 &\rightarrow N . \\
M + s N &\rightarrow s (M + N) .
\end{align*}
\]
Redex and Rewriting

♦ A redex is an instance of the LHS of an equation
  ♦ [Convention] A variable is written in a capital letter

\[
\text{eq } M + s N = s (M + N)
\]

Redexs: \(0 + s 0\) \(s\ s\ 0 + s\ s\ 0\) \(s\ X + s\ s\ (Y + s\ Z)\)

♦ (1-step) Rewriting is a replacement of a redex with the corresponding instance of the RHS

\[
\begin{align*}
s\ (0 + s\ s\ 0) \\
\rightarrow s\ s\ (0 + s\ 0)
\end{align*}
\]

with\[
\begin{align*}
M & \leftarrow 0 \\
N & \leftarrow s\ 0
\end{align*}
\]
Redex and Rewriting

• A term which cannot be rewritten anymore is called a normal form
  • A term is a normal form \iff it has no redex

\[
\begin{align*}
s \ (0 + s \ 0) & \rightarrow s \ s \ (0 + 0) \rightarrow s \ s \ 0 \\
eq & N + 0 = N \\
eq & M + s \ N = s \ (M + N)
\end{align*}
\]
Variable conditions for TRS

♦ Rewrite rules should satisfy the following conditions of variables

1. Any LHS should not be a variable
   • E.g. by $N = N + 0$, reduction does not terminate

   \[
   s\ 0 \rightarrow s\ 0 + 0 \rightarrow (s\ 0 + 0) + 0 \rightarrow \ldots
   \]

2. Any variable in RHS should appear in LHS
   • E.g. by $0 = N \times 0$, a redex can be rewritten into infinitely many terms (by the single rule)

   \[
   0 \times 0 \leftrightarrow 0 \leftrightarrow s\ s\ s\ 0 \times 0 \ldots
   \]

   \[
   s\ 0 \times 0 \leftrightarrow 0 \leftrightarrow s^n\ 0 \times 0 \ldots
   \]
Bad equations ignored

♦ The reduction command in CafeOBJ uses equations satisfying the variable conditions only

CafeOBJ> mod* TEST{...  
eq N = N + 0.  
eq 0 = N * 0.  
}  
CafeOBJ> select TEST  
TEST> red 0.  
-- reduce in TEST : (0):Nat  
(0):Nat  
(0.000 sec for parse, 0 rewrites(0.000 sec), 0 matches)  
TEST>
Properties of TRS

♦ In general, TRS achieves only a partial equational reasoning because equations are directed
  • e.g. $b = c$ is not reduced to true, when $\{a = b, a = c\}$

       a
       / \
      /   \
    b     c
  $b \neq c$

  • If for any term, its normal form is computable and unique up to $=_E$, then all deducible equations can be proved by the reduction command

       t =_E t'
       red t = t'
       returns true
Definition of Termination

♦ A specification (or a TRS, or a equations set) is **terminating** if the length of every rewriting sequence is **finite**
Termination and normal forms

♦ If a specification is terminating, then
  • every term has a normal form, and
  • possible to compute a normal form in finite times

♦ [Example] The following SP is not terminating:

\[
\text{eq } X + Y = Y + X .
\]

\[
\begin{align*}
\text{s } 0 + 0 & \rightarrow 0 + \text{s } 0 \rightarrow \text{s } 0 + 0 \rightarrow \ldots \\
\end{align*}
\]
Proving termination

♦ In general, termination is *undecidable*, i.e. no algorithm can decide whether a given term rewriting system is terminating or not
  • Several sufficient conditions for termination have been proposed

♦ We give a simple way to write terminating specifications in a hierarchical design
Hierarchical design

To describe a specification of an abstract data type, first describe a module for constructors, and next describe a module for each function.
BASIC-SP

♦ BASIC-SP is a module for data which includes all constructors of the specification to be described

BASIC-NAT

[Zero NzNat < Nat]
op 0 : -> Zero {constr }
op s_ : Nat -> NzNat {constr }

• A term consisting of only constructors and variables is called a constructor term
SP-f

♦ SP-f is a module specifying a function f
  • imports BASIC-SP and modules of pre-defined functions
  • E.g. \textsc{NAT*} imports \textsc{BASIC-NAT} and \textsc{NAT+}

♦ Equations satisfy the following conditions:
  • In the form of \texttt{eq } f(l_1, l_2, \ldots, l_n) = r \texttt{ where }

1. Each $l_i$ is a constructor term
2. The RHS $r$ is constructed from variables, constructors, pre-defined functions and decreasing recursive calls (of the LHS) only
Decreasing recursive call

♦ For \( F(l_1, \ldots l_n) = r \), a subterm \( f(t_1, \ldots t_n) \) of RHS \( r \) is called a recursive call of LHS

♦ A recursive call \( f(t_1, \ldots t_n) \) of \( f(l_1, \ldots l_n) \) is decreasing if

- All arguments \( t_i \) are subterms of \( l_i \) resp., and
- One of the arguments \( t_i \) is a strict subterm of \( l_i \)

\[
\begin{align*}
M + s N & \quad \text{equal}(s M, s N) \\
M + N & \quad \text{equal}(M, N) \\
M + s N & \quad s M + N
\end{align*}
\]
Example of SP-f: NAT+ and NAT*

First describe NAT+ which imports BASIC-NAT and whose equations satisfy the conditions

Next describe NAT* importing NAT+ similarly

\[
\begin{align*}
\text{eq } N \cdot 0 & = 0. \\
\text{eq } M \cdot s N & = M + (M \cdot N).
\end{align*}
\]

\[
\begin{align*}
\text{eq } N + 0 & = N. \\
\text{eq } M + s N & = s (M + N).
\end{align*}
\]

\[
\text{eq } f(l_1, l_2, \ldots, l_n) = r \text{ where }
\]

1. Each \( l_i \) is a constructor term

2. The RHS \( r \) is constructed from variables, constructors, pre-defined functions and decreasing recursive calls (of the LHS) only
SP-f is terminating

♦ Recursive path order (RPO) is one of most classical termination methods
  • Giving a precedence order $>$ on operation symbols
  • A well-founded order $>_\text{rpo}$ on terms obtained
  • If $>_\text{rpo}$ includes all rewrite rules, then terminating

♦ SP-f can be proved to be terminating by RPO
  • with a precedence order satisfying:
    • $f > g$ if SP-f imports SP-g, e.g. $\times > +$
    • $f > c$ if $c$ is constructor and $f$ is not, e.g. $\times, + > s, 0$
Into a single module

- The reduction command does not care module structure
- Thus, all equations can be in a single module if you have a hierarchy of operation symbols in mind

\[
\begin{align*}
\text{eq } N + 0 &= N \\
\text{eq } M * s N &= M + (M * N)
\end{align*}
\]

\[
\begin{align*}
\text{eq } N + 0 &= N \\
\text{eq } M + s N &= s (M + N)
\end{align*}
\]

\[
\begin{align*}
\text{eq } N * 0 &= 0 \\
\text{eq } M * s N &= M + (M * N)
\end{align*}
\]
Confluence
A specification is **confluent** if all terms obtained by rewriting from one ancestor term can be reduced into a common descendant term.
Confluence and normal forms

♦ If a specification is **confluent**, then
  - Each normal form is **unique** w.r.t. $=_{E}$
  - if it is terminating, **every term has a unique normal form**

♦ [Example] The following SP is not confluent:

\[
\begin{align*}
eq & \ (X + Y) + Z = X + (Y + Z) . \\
eq & \ \text{first}(X + Y) = X . \\
\end{align*}
\]
Confluence is another *undecidable* property, i.e. no algorithm can decide whether a given term rewriting system is confluent or not

- Confluence of a *terminating* TRS is *decidable*
For non-overlapping (and terminating) TRS

♦ A terminating and non-overlapping TRS is confluent
  • For any two redexs, their LHS do not share operation symbols

Termination is necessary for this proposition

[Exercise] Find a non-overlapping but non-confluent TRS
For overlapping (and terminating) TRS

♦ When overlapping, divided terms may not be convergent since a redex pattern may not be preserved
  - e.g. \{f(a) = b, a = c\}: f(c) \leftarrow \rightarrow f(a) \rightarrow b

♦ For an overlapping TRS,
  - Consider the most general unifiers for all overlaps

\[
\begin{align*}
eq M + 0 & = M . \\
eq M + s N & = s (M + N) . \\
eq (s M) + N & = s (M + N) .
\end{align*}
\]

- And, check if their direct two descendants, called critical pairs, are all convergent. If so, SP is confluent
Termination and Confluence

♦ A terminating and confluent SP gives us a terminating, sound and complete equational reasoning:
  1. Reduce both sides of a given equation
     • They have normal forms because of termination
  2. Compare their normal forms
     • If they are same, the equation is deducible from the axiom
       \[
       \text{NAT+ + EQL> red } 0 + s 0 = s 0 + 0 .
       \]
       (true) : Bool
     • If they are not same, the equation is not deducible from the axiom, because normal forms are unique
       \[
       \text{NAT+ + EQL> red } s 0 + s 0 = s 0 + 0 .
       \]
       ((s (s 0)) = (s 0)) : Bool
Termination and Confluence (cont.)

♦ When SP terminating and confluent, the reduction command can prove all deducible equations, however, it does not mean all equations in a model
  • A typical example is commutativity of _+_!
  • X + Y = Y + X holds for natural numbers, but this is not deducible from the axiom
    • By reflexive, symmetric, transitive, congruent and substitutive laws
  • You need other proof techniques, e.g. structural induction

\[
\begin{align*}
\text{NAT} + \vdash (\forall y)0 + y = y + 0 \\
\text{NAT} + \vdash (\forall y)s0 + y = y + s0, \quad \text{NAT} + \vdash (\forall y)ss0 + y = y + ss0, \\
\vdash (\forall x)(\forall y)x + y = y + x
\end{align*}
\]
Keywords

♦ Termination
  • Recursive path order
  • Dependency pairs

♦ Confluence
  • Critical pairs
  • Orthogonality (= left-linear + non-overlap)
    • without termination assumption
Inference with Rewriting Rules
PART II

Advanced topic for OBJ specifications
Algebraic specifications

♦ Sufficient completeness
  • Each non-constructor operators is completely defined for all constructor terms

♦ Equations with condition
  • E.g. \( \text{ceq} \ odd(N) = \text{false} \ if \ even(N) \).

♦ Associative and Commutative operators
  • E.g. \( \text{op}_+ : \text{Nat Nat} \to \text{Nat} \{\text{assoc comm}\} \)
Sufficient completeness
Definition of Sufficient completeness

♦ A specification is **sufficiently complete** if every term $t$ of a constraint sort $S^C$ has an equivalent constructor term $t'$, i.e. $t = t'$ can be deduced from the axiom

- Roughly speaking, every non-constructor constraint-sort operator $f$ (in $F - F^C$ and of $S^C$) is defined for all constructor terms

- Operator $+_+$ in the left SP is not defined for $(s\ 0, \ 0)$

- No constructor term $t$ such that $s\ 0 + 0 =_E t$
Sufficient completeness by TRS

♦ If SP is terminating and quasi C-reducible, then SP is sufficiently complete

- $f(t_1, \ldots, t_n)$ is a basic term if $f$ is not constructor ($F - FC$) and of a constraint sort ($SC$), and $t_i$ are all constructor terms

- TRS is quasi C-reducible if every basic term is not a normal form
  - In other words, LHSs of rewrite rules cover all basic terms

\[
\begin{align*}
\text{eq } N + 0 &= N \\
\text{eq } M + s N &= s (M + N) .
\end{align*}
\]

Every basic term $s^n 0 + s^m 0$ is an instance of $N + 0$ (when $m = 0$)

or an instance of $M + s N$ (when $m > 0$)
Conditional Equations
Conditional equations

♦ CafeOBJ allows us to write a condition for an equation
  • A condition is a term of Boolean sort \( \text{Bool} \)

\[
\begin{align*}
\text{op even} &: \text{Nat} \to \text{Bool} . \\
\text{eq even } 0 &= \text{true} . \\
\text{ceq even}(s N) &= \text{false if even } N . \\
\text{ceq even}(s N) &= \text{true if not (even } N) .
\end{align*}
\]

• All CafeOBJ modules implicitly import a built-in Boolean module \( \text{BOOL} \)

• You can use Boolean operators \text{not, and, or} \ldots \text{for writing conditions} without any explicit import of \( \text{BOOL} \)
A conditional equation is applied when the condition part is reduced into \textit{true}.

\begin{itemize}
  \item \texttt{eq even 0 = true}.
  \item \texttt{ceq even(s N) = false if even N}.
  \item \texttt{ceq even(s N) = true if not (even N)}.
\end{itemize}

\begin{itemize}
  \item \texttt{even(0) \rightarrow true}
  \item \texttt{even(s 0) \rightarrow false}
  \item \texttt{even(s s 0) \rightarrow true}
\end{itemize}

\begin{itemize}
  \item \texttt{not even(s 0) \rightarrow not false}
  \item \texttt{\rightarrow true}
\end{itemize}
Termination for conditional SP

To obtain termination of the reduction command for conditional SP, not only RHS but the condition part should also be cared.

\[
\text{ceq } \text{even}(s \ N) = \text{false if even } N .
\]
\[
\text{ceq } \text{even}(s \ N) = \text{true if not (even } N ) .
\]

\[
\text{ceq } f(X) = \text{true if } f(X) .
\]

```
INFINITE> red f(X:Elt) .
-- reduce in INFINITE : (f(X)):Bool
[Warning]:
Infinite loop? Evaluation of condition nests too deep,
terminates rewriting: f(X:Elt)
```

Lecture Notes 04, JAIST-FSSV2010
Confluence for conditional SP

♦ Most of conditional SPs overlap, since conditions are often used to write case-splitting for a same pattern

\[
\text{ceq even}(s \; N) = \text{false} \; \text{if} \; \text{even} \; N .
\]
\[
\text{ceq even}(s \; N) = \text{true} \; \text{if} \; \text{not} \; (\text{even} \; N) .
\]

• For a conditional TRS, we should consider conditional critical pairs
  • (false, true) if (even N) and not (even N)
  • CCP should be checked for each substitution satisfying the condition
• If for any two overlapping conditional rewrite rules, their conditions are distinct from each other, then the condition is false for each conditional CP (and each substitution)
  • SP can be treated as non-overlapping
Sufficient completeness for conditional SP

♦ For **sufficient completeness**
  • conditions of each pattern should cover all cases, i.e., \( c_0 \) or \( c_1 \) or ... or \( c_n = true \)

\[
\begin{align*}
\text{ceq even}(s \ N) &= \text{false if even } N. \\
\text{ceq even}(s \ N) &= \text{true if not (even } N). 
\end{align*}
\]

♦ For both **confluence** and **sufficient completeness**
  • For each basic term \( f(t) \), there exists a **unique** applicable conditional equation
    - \( N < 5, \ 7 \leq N \) (not sufficient completeness)
    - \( N < 5, \ 3 \leq N \) (may not be confluent)
Associative Commutative Operators
Commutative law

♦ As we know, an equation of commutative law causes infinite loop

\[
\text{eq \ } X + Y = Y + X .
\]

\[
s\ 0 + 0 \rightarrow 0 + s\ 0 \rightarrow s\ 0 + 0 \rightarrow \ldots
\]

♦ Commutative law is recommended to be specified as operators attributes

\[
\text{op } _+_: \text{Nat Nat } \rightarrow \text{Nat } \{ \text{comm} \}
\]
Associative law

♦ Associative law, another commonly-used equation, is also can be declared as operation attributes

\[
\text{op } \_+\_ \colon \text{Nat Nat } \to \text{Nat } \{ \text{assoc} \}
\]

• We can omit brackets as follows

\[
\text{NAT+A} \triangleright \text{red } 0 + s \ 0 + s \ s \ 0 . \ \\
\ldots
\]

[Error] no successful parse
"Symptom: ambiguous term"

\[
\text{NAT+A} \triangleright \text{red } 0 + s \ 0 + s \ s \ 0 . \ \\
\ldots
\]

\( (s \ (s \ (s \ 0))) \colon \text{Nat} \)
Use of AC operator

- A multiset (bag) can be specified by AC operator \( \_ \_ \), which just indicates the position of arguments

  \[
  \begin{align*}
  [\text{Elt} < \text{Bag}] \\
  \text{ops } a \ b \ c : & \to \text{Elt} \\
  \text{op } \_ \_ : & \text{Bag Bag } \to \text{Bag} \{ \text{assoc comm} \}
  \end{align*}
  \]

- By the subsort relation \( \text{Elt} < \text{Bag} \), each Elt-term is of \text{Bag}
- By AC-operator \( \_ \_ \), a sequence of Elt-terms is of \text{Bag}

\[\text{EQL + BAG> red } a \ a \ b \ b \ c \ c = a \ b \ c \ a \ b \ c.\]

\[\text{-- reduce in EQL + BAG :}\]
\[\ldots\]
\[(\text{true}) : \text{Bool}\]
Membership operation for bag

\[ E \text{ in } \text{Bag} \text{ is reduced into true when Bag includes } E \text{ anywhere} \]

\[
\text{op } \_ \_ : \text{Bag Bag } \rightarrow \text{Bag } \{ \text{assoc comm } \} \\
\text{op } \_ \text{in}_\_ : \text{Elt Bag } \rightarrow \text{Bool} \\
\text{var E : Elt } \quad \text{var B : Bag} \\
\text{eq E in (E B) = true .}
\]
AC Rewriting

♦ One step AC Rewriting $\rightarrow_{AC}$ is defined as the composition of $(=_{AC} \circ \rightarrow)$

\[
\text{eq E in (E B) = true.}
\]

\[
\begin{align*}
c \text{ in } (a \ (c \ b)) &= c \text{ in } ((c \ b) \ a) \\
&= c \text{ in } (c \ (b \ a)) \\
&\rightarrow \text{ true}
\end{align*}
\]

For a given term, the number of AC-equivalent terms is finite

\[
\begin{align*}
a (b \ c), &\ (a \ b) \ c, \ a \ (c \ b), \ (a \ c) \ b, \ b \ (a \ c), \ (b \ a) \ c, \ b \ (c \ a), \ (b \ c) \ a, \\
c \ (a \ b), &\ (c \ a) \ b, \ c \ (b \ a), \ (c \ b) \ a
\end{align*}
\]
[Attention] Even if SP seems to be terminating, AC attribute may make it non-terminating

\[
\text{BAG2}\]

\[
\begin{align*}
\text{[Elt} & \text{< Bag]} \\
\text{ops} & 0 \ 1 \ : \rightarrow \ \text{Elt} \\
\text{op} & \_ \_ \ : \ \text{Bag} \ \text{Bag} \rightarrow \ \text{Bag} \ \{ \text{assoc} \ \text{comm} \} \\
\text{var} & \text{E} : \ \text{Elt} \\
\text{eq} & (\text{E} \ \text{E}) = 0 \ 1 .
\end{align*}
\]

\[
\begin{align*}
0 \ (0 \ 1) & \overset{A}{=} (0 \ 0) \ 1 \\
\rightarrow & \ (0 \ 1) \ 1 \\
\overset{A}{=} & \ 0 \ (1 \ 1) \\
\rightarrow & \ 0 \ (0 \ 1) \ ...\end{align*}
\]
[Attention] Even if SP seems to be confluent, AC attribute may make it non-confluent

```
begin-with-zero(0 B) = true .
begin-with-zero(1 B) = false .
```
System specification
The above discussions can be applied to system specifications (Behavioral specifications)

- Transitions (actions) are regarded as constructors
  - A constructor term stands for a reachable state

- Observations are defined for all constructor terms, simultaneously in a module

\[
\begin{align*}
eq o_1(f_A(S)) & = o_1(S) + o_2(S, 0) . \\
eq o_2(f_A(S), N) & = o_1(S) * o_2(S, N) . \\
\end{align*}
\]

Precedence order for RPO: \( o_1 = o_2 > f_A = f_B = f_C > * > + \)
OTS/CafeOBJ specification

♦ OTS/CafeOBJ specifications are regarded as a special case of behavioral specifications
  • Only observations are behavioral
  • All transitions are defined completely (behaviorally coherent)
    • Quasi reducibility for OTS/CafeOBJ specifications:
      • \( o(\ell(t_1, \ldots, t_n)) \) is not a normal form for any \( o: \) observation, \( \ell: \) transition, and \( t_i: \) constructor terms
      • With termination, for any reachable state \( s \) and observation \( o, o(s) \) can be reduced into a term \( u \) which does not have any transition symbols

\[
\begin{align*}
\text{eq } o_1(\ell(S, 0)) &= \ldots \\
\text{eq } o_1(\ell(S, s N)) &= \ldots \\
\text{eq } o_2(\ell(S, N), M) &= \ldots 
\end{align*}
\]
Keywords

♦ Sufficient completeness
  • Ground reducibility
  • Reducible operation symbols

♦ Conditional equation
  • Effective termination
  • Operational termination
  • Conditional critical pairs

♦ AC operators
  • AC rewriting (or AC-TRS)
Conclusion

♦ To obtain SP whose reduction works well
  • First of all, describe as a terminating SP
  • Next, care about overlap and reducibility

♦ In verification, adding equations of lemma or I.H. may change the behavior of reduction
  • Anyway, for soundness, the above properties are not needed strictly
  • If reduction returns true, it is true
References and tools

♦ Term Rewriting systems, by Terese (Cambridge Univ. Press, 2003)
  • Textbook of TRS: containing Conditional TRS and AC-TRS

  • Reducibility for behavioral specifications

♦ AProVE : http://aprove.informatik.rwth-aachen.de/
  • Most powerful termination prover, which supports CTRS and AC-TRS

♦ Maude tools
  • Coming Lecture 11 on May 4th (Thu.) by Dr. Francisco Duran
Exercise

♦ Write specifications of the following operators on natural numbers:
  • Subtraction operation:
    • \(\_\_\_\_\_ : \text{Nat Nat} \rightarrow \text{Nat}\)
  • Greater-than operation:
    • \(\_\_\_\_\_ : \text{Nat Nat} \rightarrow \text{Bool}\)
  • Modulo operation:
    • \(\_\_\_\_\_\_ : \text{Nat Nat} \rightarrow \text{Nat}\)
  • GCD (Greatest Common Divisor) operation:
    • \(\text{gcd} : \text{Nat Nat} \rightarrow \text{Nat}\)