Incremental proofs of termination, confluence and sufficient completeness of OBJ specifications

NAKAMURA Masaki (Toyama Pref. Univ.)
FUTATSUGI Kokichi (JAIST)

Apr. 14 (Mon) - 16 (Wed), 2014
Kanazawa, Japan
Abstract

• OBJ languages support equational reasoning based on term rewriting systems (TRS).

• There are three important properties of TRSs: termination, confluence and sufficient completeness.

• In this talk, we discuss about incremental proofs of those properties based on the module structure of OBJ specifications.
Equational reasoning by TRSs

• In equational reasoning, equations (axioms) are regarded as left-to-right rewrite rules

```
mod! NAT-OP{
  [Zero NzNat < Nat]
  op 0 : -> Zero {constr}
  op s_ : Nat -> NzNat {constr}
  ops (_+_)(_-_):Nat Nat -> Nat
  ops (_>=_) (_>_):Nat Nat ->
}
```

vars M N : Nat
eq N + 0 = N .
eq M + s N = s (M + N) .
...

Describe spec. by equations
Verification by rewrite rules

CafeOBJ> red in NAT-OP : s 0 + s s 0 .
-- reduce in NAT-OP : ((s 0) + (s (s 0))):Nat
[1]: ((s 0) + (s (s 0))):NzNat ---> (s ((s 0) + (s 0))):NzNat
[2]: (s ((s 0) + (s 0))):NzNat ---> (s (s ((s 0) + 0))):NzNat
[3]: (s (s ((s 0) + 0))):NzNat ---> (s (s 0))):NzNat
(s (s (s 0))):NzNat
```
Module imports

• OBJ languages support module imports

```plaintext
mod* ACCOUNT{
  pr(NAT-OP) [Account]
  op balance_ : Account -> Nat
  ops (deposit_) (withdraw_ _) : Nat Account -> Account {constr}
  op init : -> Account {constr}
  var A : Account
  var N : Nat
  eq balance init = 0 .
  eq balance (deposit N A) = balance A + N .
  ceq balance (withdraw N A) = balance A - N if balance A >= N .
  ceq withdraw N A = A if N > balance A .
}
```

import NAT-OP

Sorts and operators in the imported module can be used
Reduction in CafeOBJ

- Equations in both importing and imported modules are used for reduction

CafeOBJ> red in ACCOUNT : balance (withdraw (s s 0) (deposit (s 0) init)) .

[1(cond)]: ((s (s 0)) > (balance (deposit (s 0) init))):Bool
  --> (not ((balance (deposit (s 0) init)) >= (s (s 0)))):Bool
  ...
[9(cond)]: (false xor true):Bool
  --> (true):Bool

[10]: (balance (withdraw (s s 0) (deposit (s 0) init))):Nat
  --> (balance (deposit (s 0) init))):Nat

[11]: (balance (deposit (s 0) init))):Nat
  --> ((balance init) + (s 0))):Nat
  ...
[14]: (s (balance init)):NzNat
  --> (s 0)):NzNat
  (s 0)):NzNat

NAT-OP

eq N > M = not (M >= N) .

ACCOUNT

c eq withdraw N A = A
  if N > balance A .

NAT-OP

eq N + 0 = N .
eq M + s N = s (M + N) .

2014/4/14  SAS 2014
Fundamental properties

• The three fundamental properties of TRSs are important not only for reduction but also for specification description
  – Termination
    • the existence of normal forms (computable)
  – Confluence
    • the uniqueness of normal forms
  – Sufficient completeness
    • the well-definedness of (non-constructor) operators
Proving fundamental properties

• When a specification consists of more modules, the task of proving the fundamental properties might be heavier

• To obtain light-weight proof methods, incremental approaches are known to be effective
Incremental approach to $P$

- Let $M$ and $M'$ be modules s.t. $M$ imports $M'$ and $P'$ be a sufficient condition for $P$ such that $(P'(M)$ and $P'(M'))$ implies $P'(M \cup M')$

- Then, we can prove $P$ incrementally

![Diagram showing the incremental approach to proving $P$.]
Related work


<table>
<thead>
<tr>
<th></th>
<th>TRS</th>
<th>AC-TRS</th>
<th>Conditional TRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Termination</td>
<td>[1]</td>
<td>[2]</td>
<td>[3] ○</td>
</tr>
<tr>
<td>Confluence</td>
<td></td>
<td>[3]</td>
<td>○</td>
</tr>
<tr>
<td>Suff. Comp.</td>
<td></td>
<td>[3]</td>
<td>○</td>
</tr>
</tbody>
</table>

The main contribution of our work is to extend the result of [3] to specifications with conditional equations
Hierarchical extension for CTRS

- We give the notion of hierarchical extension, based on a TRS version in [2]

[Def] Let $M_1 = [\Sigma_1 \mid R_1]$ and $M_2 = [\Sigma_2 \mid R_2]$ be modules. The pair of $M_1$ and $M_2$ is called a hierarchical extension, denoted by $M_1 \leftarrow M_2$, if
  1. $\Sigma_1 \cap \Sigma_2 = \phi$
  2. $R_2$ is a CTRS over $\Sigma_1 \cup \Sigma_2$
  3. for all eq $f(...)=r \in R_2$ and ceq $f(...)=r$ if $c \in R_2, f \in \Sigma_2$

Operational termination

• Termination of TRSs is defined as the absence of infinite rewrite sequences
  
  -- $R = \{eq \ a = g(a)\}$
  
  -- $a \rightarrow g(a) \rightarrow g(g(a)) \rightarrow ...$

• Termination of CTRSs can be characterized by the notion of operational termination [5], which covers infinite sequence of both rewrites and condition checks
  
  -- $R = \{\text{ceq } a = b \text{ if } c, \ \text{eq } c = a\}$
  
  -- To reduce $a$, check $c$, $c \rightarrow a$, to reduce $a$, ...

Argument decreasing

• In [3], the notion of argument decreasing rules has been proposed
• We extend it to CTRS straightforwardly

[Def] $f >^1_R g \iff$ there exists $f(...) = r$ if $c$ in $R$ such that $g$ occurs in $r$ or $c$, and $>_R$ and $\sim_R$ are the strict part and the equivalent part of the ref. trans. closure of $>_R$

[Def] The rule $f(l_1,\ldots,l_m) = r$ if $c$ is $g$-argument decreasing if there is a subterm $r|_p = g(r_1,\ldots,r_n)$ such that \{l_1,\ldots,l_m\} $>_{\text{sub}}^\text{mul}$ \{r_1,\ldots,r_n\} or $c|_p = g(r_1,\ldots,r_n)$


2014/4/14  SAS 2014  12
Incremental proof of op. termination

• Like [3], we can prove operational termination incrementally by the notion of argument decreasing rules

[Lemma] If all equations $f(...) = r$ if $c$ in $R$ are $g$-argument decreasing for each $g \sim_R f$, then CTRS $R$ is operationally terminating

[Lemma] Let $M_1 = [\Sigma_1 \mid R_1] \leftarrow M_2 = [\Sigma_2 \mid R_2]$. If
(1) all $f(.) = r$ if $c$ in $R_1$ are $g$-argument decreasing for each $g \sim_{R_1} f$ and
(2) all $f(.) = r$ if $c$ in $R_2$ are $g$-argument decreasing for each $g \sim_{R_2} f$,
then
all $f(.) = r$ if $c$ in $R_1 \cup R_2$ are $g$-argument decreasing for each $g \sim_{R_1 \cup R_2} f$
Confluence

- [Def] A SP is **confluent** if all terms reduced from a term can be reduced into a same term

```
(0 + s 0) + s s 0

 s 0 + s s 0

 s s 0

 s s(0 + s 0)

 s s s 0
```
Conditional critical pairs

• For terminating CTRSs, confluence can be proved by using the notion of conditional critical pairs (CCPs)
  – CCPs are made from overlapping rules

```plaintext
ceq balance (withdraw N A) = balance A - N if balance A >= N .
ceq withdraw N A = A if N > balance A .
```

[Def] A CCP (l, r) if c is joinable if for every such substitution, instances of l and r are reduced into a same term

[Prop] If a spec. is op. terminating and all CCPs are joinable then the spec is confluent
Non-overlapping modules

• We also give a condition for proving confluence incrementally

[Def] The root symbols of the left-hand sides of all equations of R are called defined symbols of R

[Def] A hierarchical ext. \( M_1 = [\Sigma_1 | R_1] \) \( \leftarrow \) \( M_2 = [\Sigma_2 | R_2] \) is non-overlapping if the left-hand side of each equation of \( R_2 \) does not involve any defined symbol of \( R_1 \).

[Theorem] For a non-overlapping hier. ext. \([\Sigma_1 | R_1] \leftarrow [\Sigma_2 | R_2]\), if all CCPs of \( R_1 \) and all CCPs of \( R_2 \) are joinable, then all CCPs of \( R_1 \cup R_2 \) are joinable.
Checking CCP

[Def] A CCP \((l, r)\) if \(c\) is joinable if for every such substitution, instances of \(l\) and \(r\) are reduced into a same term

- Checking joinability of a CCP \((l, r)\) if \(c\), we need to consider all substitutions s.t. \(c\theta = true\)
- If a CCP is unfeasible (no \(\theta. c\theta = true\)), it is joinable.
- It can be checked by a TRS for XOR normal forms

E.g.

\((balance\ A - N, balance\ A)\) if \(balance\ A \geq N\) and \(N > balance\ A\)

CafeOBJ> red in ACCOUNT: balance A \(\geq N\) and \(N > balance\ A\).
   -- reduce in ACCOUNT: (((balance A) \(\geq N\)) and (N > (balance A))):Bool
   (false):Bool
Sufficient completeness

• To formalize sufficient completeness, we introduce the notion of constructor operators and constraint sorts [4]

[Def] An operator is called constructor if it is declared with `{constr}`

[Def] A sort $s$ is called constrained if (1) $\text{op } f : \ldots \to s \{\text{constr}\}$ exists or (2) $s' < s$ for some constrained sort $s'$

mod! NAT-OP{ [Zero NzNat < Nat]
  op 0 : -> Zero  {constr}       op s_ : Nat -> NzNat  {constr}
  ops (_+_)(_-_): Nat Nat -> Nat  ops (_>=_)(_>_): Nat Nat -> Bool }

Sufficient completeness (cont.)

[Def] An operator \( \text{op} f : ... \rightarrow s \) is called constrained if its sort \( s \) is constrained.

[Def] Let \( T^{\text{SC}} \) be the set of all terms of constrained sorts constructed from only constrained operators and variables of loose variables (= non-constrained sort variables), let \( T^{\text{C}} \) be the set of all terms constructed from only constructors and loose variables. If each term of \( T^{\text{SC}} \) can be reduced into a term of \( T^{\text{C}} \), the specification is called sufficiently complete.

- For op. terminating CTRSs, sufficient completeness can be checked by the notion of reducibility.
Quasi-reducibility

- We give the notion of quasi-C-reducibility (straightforward extension of the notion of quasi-reducibility of TRSs)

[Def] A term $f(t_1...t_n)$ is basic if $f$ is constrained but not constructor and $t_1...t_n$ are of $T^C$

[Def] A CTRS is quasi-C-reducible if every basic terms are reducible (are not normal forms)

[Prop] If an op. terminating CTRS is quasi-C-reducible then it is sufficiently complete

```
op 0 : -> Zero  {constr} 
op s_ : Nat -> NzNat  {constr}
ops (_,+) (_,_) : Nat Nat -> Nat 
ops (_,>=,) (_,_) : Nat Nat -> Bool  
```

$0 + s s 0$, $(0 + s 0) - s 0$, $s 0 + s 0 >= s s s 0$, ...
Constructor preserving

• Even if proving sufficient completeness once, we need to check again when it is imported because basic terms may increase

\[\begin{align*}
op\ 0 & : \to \text{Zero} \ {\text{constr}} \\
onp\ s_\_ & : \text{Nat} \to \text{NzNat} \ {\text{constr}} \\
\text{ops} \ (\_+\_) \ (\_-\_) & : \text{Nat Nat} \to \text{Nat}
\end{align*}\]

\[\begin{align*}
op\ \text{inf} & : \to \text{Nat} \ {\text{constr}} \\
\text{eq inf} + N & = \text{inf} \ldots
\end{align*}\]

• We give a condition under which basic terms are preserved when imported
Constructor preserving (cont.)

[Def] A hierarchical ext. \( M_1 = [\Sigma_1 \mid R_1] \) \( \leftarrow \) \( M_2 = [\Sigma_2 \mid R_2] \) is constructor preserving if
1. for each op \( f : \ldots \rightarrow s \) \{constr\} in \( \Sigma_2 \), s is not in \( S_1 \)
2. for each \( s \) in \( S_1 \), no \( s' \) in \( S_2 \) exists such that \( s' < s \)

[Theorem] Let \( M_1 = [\Sigma_1 \mid R_1] \) \( \leftarrow \) \( M_2 = [\Sigma_2 \mid R_2] \) be terminating and constructor preserving. If
1. every basic terms of \( \Sigma_1 \) are reducible w.r.t \( R_1 \)
2. every basic terms of \( \Sigma_2 \) are reducible w.r.t. \( R_1 \cup R_2 \)
then \( (\Sigma_1 \cup \Sigma_2, R_1 \cup R_2) \) is sufficiently complete.
Checking reducibility of basic terms

[Def] A term \( f(t_1...t_n) \) is basic if \( f \) is constrained but not constructor and \( t_1...t_n \) are of \( T^C \)

• We can obtain a finite cover set for basic terms
• Take all applicable rules \( l = r \) if \( c \), make their disjunction, and check if it can be reduced to true

E.g. (cover set for balance)
{ balance init,
  balance (deposit X Y),
  balance (withdraw X Y) }

c eq balance (withdraw N A) = balance A - N
  if balance A >= N .
c eq withdraw N A  = A
  if N > balance A .

CafeOBJ> red in ACCOUNT : balance A >= N or N > balance A .
(true):Bool
Conclusion

• We proposed methods to prove op. termination, confluence and sufficient completeness incrementally by checking simple syntactical conditions: non-overlapping constructor-preserving hierarchical extension (fair extension [3])

• We also gave methods to check joinability of CCPs and quasi-C-reducibility for CTRS
  – hierarchical ext. + argument decreasing rules for termination
  – non-overlapping extension + joinable CCPs for confluence
  – constructor preserving extension + quasi-C-red. for suff. comp.

• Future work: AC symbols, proof score, etc