Software Engineering using Algebraic Specification (AS)

Dorel Lucanu
"Alexandru Ioan Cuza" University
Faculty of Computer Science
Iasi, Romania
email: dlucanu@infoiasi.ro
Outlines

- formal methods in SE
- main paradigms: MSA, OSA, MA, HA, RWL
- what is an algebraic specification
- proof engineering
  - rewriting
  - induction
  - coinduction
- projects in progress
  - more expressivity: HA and model checking
  - integration: HA and CCS/pi-calculus
- conclusion
Formal methods in SE

- main concerns
  - specification
  - verification
  - refinement
- a formal method involves
  - specification languages
  - logics
  - tools
  - to produce right specifications
Specification languages

- a classification
  - model oriented (VDM, Z, CTL …)
  - algebraic (OBJ family, CASL, …)
  - process model (CSP, CCS, pi-calculs, …)
  - logical (HOL, PVS, …)
  - broad spectrum (LOTOS, …)
AS Paradigms

- Many Sorted Algebra (MSA)
- Order Sorted Algebra (OSA)
- Membership Algebra (MA)
- Hidden Sorted Algebra (HA)
- Rewriting Logic (RWL)
- ...

Dorel Lucanu
National University of Singapore 6/9/2003
### AS paradigms, logics & proof engines

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Logic</th>
<th>Proof engines</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSA, OSA</td>
<td>Equational Logic (MSEL, OSEL) (sound, complete)</td>
<td>Rewriting</td>
</tr>
<tr>
<td>MA</td>
<td>Membership Equational Logic (MEL) (sound, complete)</td>
<td>Rewriting, Membership deduction</td>
</tr>
<tr>
<td>RWL</td>
<td>Rewriting Logic (sound, complete)</td>
<td>Rewriting</td>
</tr>
<tr>
<td>HA</td>
<td>Behavioral Equational Logic (sound, incomplete)</td>
<td>Coinductive behavioral rew.</td>
</tr>
</tbody>
</table>
### Proof methodologies

<table>
<thead>
<tr>
<th>What specify</th>
<th>Proof methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADT</td>
<td>Confl. &amp; terminat. rew. Induction</td>
</tr>
<tr>
<td>Objects, systems (behavioral specs)</td>
<td>Coinduction</td>
</tr>
<tr>
<td></td>
<td>Circular Coinduction</td>
</tr>
<tr>
<td>Transitional systems</td>
<td>Induction</td>
</tr>
<tr>
<td></td>
<td>Backtracking</td>
</tr>
</tbody>
</table>
Comparing paradigms

BMEL

MSHorn$^=$

MEL

OSEL$^R$

RWL

MSEL

OSEL$^GM$

HL

Dorel Lucanu

National University of Singapore 6/9/2003
What is an algebraic specification?

- A specification \((\Sigma, E)\) includes:
  - The signature \(\Sigma\)
    - Sorts
    - A partial order on sorts (OSA)
    - Operational symbols
  - Properties of the operations (sentences) \(E\)
    - Simple equations
    - Conditional equations
    - Membership assertions (MEL)
    - Transition specifications (rewrite rules) (RWL)
Specification of naturals

obj NAT is
  sorts NzNat Nat .
  subsort NzNat < Nat .
  op 0 : -> Nat .
  op s_ : Nat -> NzNat .
  op _+_ : Nat Nat -> Nat .
  vars M N : Nat .
  eq M + 0 = M .
  eq M + s N = s(M + N) .
  *** ...
endo

Dorel Lucanu
National University of Singapore 6/9/2003
Sets of naturals

\textbf{obj \textsc{set} is}
   \textbf{including \textsc{nat} .}
\textbf{sort \textsc{natset} .}
\textbf{op \emptyset : \rightarrow \textsc{natset} .}
\textbf{op \textbf{ins} : \textsc{nat} \textsc{natset} \rightarrow \textsc{natset} .}
\textbf{op \_U\_ : \textsc{natset} \textsc{natset} \rightarrow \textsc{natset} .}

\ldots

\textbf{eq \textbf{ins}(N,\textbf{ins}(N',S)) = \textbf{ins}(N',\textbf{ins}(N,S))}
   \textbf{if} \; N > N' .
\textbf{eq \textbf{ins}(N, \textbf{ins}(N', S)) = \textbf{ins}(N, S)}
   \textbf{if} \; N == N' .
\textbf{eq \textbf{S \_U\_} \{} = \textbf{S} .
\textbf{eq \textbf{S \_U\_} \textbf{ins}(N, S') = \textbf{ins}(N, S \_U\_ S')} .
\textbf{endo}
Membership assertions

- sets of even naturals
  - sort EvNat.
  - subsort EvNat < Nat.
  - mb 0 : EvNat.
  - mb s s N : EvNat if N : EvNat.

or equivalently

  mb N : EvNat if N % 2 == 0.
Models

- A model is an algebra
  - \( \mathbb{N} \) = algebra of naturals
  - A sort is interpreted as a set
    - \([\text{Nat}]\) = \{0, 1, 2, \ldots\}, \([\text{NzNat}]\) = \{1, 2, \ldots\}
  - A subsort relation is interpreted as inclusion
    - \([\text{NzNat}] \subset [\text{Nat}]\)
  - Operation symbols are interpreted as operations
    - \([s] : [\text{Nat}] \rightarrow [\text{NzNat}]\)
    - \([s](x) = x + 1\)
Models

- equations are equalities that must be satisfied:
  \[ \mathbb{N} \models (\forall M, N) M + s \cdot N = s (M + N) \]
  because
  \[ x + (y+1) = (x + y) + 1 \]
  for all \( x, y \in [[\text{Nat}]] \)

- membership assertions are interpreted as membership relations that must be satisfied:
  \[ \mathbb{N} \models (\forall M) s \cdot s \cdot M : \text{EvNat} \text{ if } M : \text{EvNat} \]
  because
  \[ x + 2 \in [[\text{EvNat}]] \text{ if } x \in [[\text{EvNat}]] \]
  \( [[\text{EvNat}]] = \{0, 2, 4, \ldots\} \)
Algebra of ground terms $T_{\Sigma,E}$

- terms: a term is built from variables, constants, and operation symbols in the usual way

- examples of terms: $0$, $s0$, $ss0$, $x + sY$

- ground terms: no variable

- $T_{\Sigma,E} = \text{the set of ground terms modulo } E$
  
  $[0] = \{0, 0 + 0, 0 + (0 + 0), \ldots\}$
  
  $[s0] = \{s0, s0 + 0, 0 + s0,$
  
  $0 + s0 + 0, \ldots\}$
Abstract Data Types (ADT)

- M initial model iff $(\forall M')(\exists! h : M \rightarrow M')$
- initial models = standard models
  - no junks (is minimal)
  - no confusions
- two initial $(\Sigma, E)$-models are isomorphic
- \(T_{\Sigma, E}\) is initial
Proof engineering for ADT: rewriting

- equations are seen as rewrite rules
- one step rewriting relation:
  \[ t \Rightarrow t' \text{ iff } t' \text{ is obtained from } t \text{ replacing a subterm by another subterm according to a rewrite rule (equation) and a computed substitution} \]
  \[ t =*=> t' \text{ means zero, one or more rewriting steps} \]
Proof engineering for ADT: rewriting

- normal forms
  \[\text{nf}(t) = t' \iff t = \ast \Rightarrow t' \text{ and } t' \text{ is irreducible}\]
- normal form exists and it is unique if \(\ast \Rightarrow\) is terminating and confluent

\[
\begin{align*}
X + s(Y + 0) & \\
X + sY & \\
s(X + (Y + 0)) & \\
s(X + Y)
\end{align*}
\]

confluency property
BOBJ> in set.bob
BOBJ> select SET .
BOBJ> red ins(0, ins(1,{})) U ins(1,{}).

====================================
reduce in SET : ins(0, ins(s 0, {})) U ins(s 0, {})
result NatSet: ins(0, ins(1, {}))
rewrite time: 63ms      parse time: 16ms
BOBJ>
red ins(0, ins(0, ins(0, {}))) == ins(0, {}).

reduce in SET : ins(0, ins(0, ins(0, {}))) == ins(0, {})
result Bool: true
rewrite time: 0ms    parse time: 0ms

BOBJ>
Proof engineering for ADT: induction

- initial truth = properties that are true in initial models
- (structural) induction = a method to prove initial truth
- example
  \[ \emptyset \cup S = S \]
  - it cannot be proved using rewriting (no chance even we use equational deduction)
- the method
Proof engineering for ADT: induction

- define first the constructors
  - for SET they are `{}` and `ins(N, S)`
  - Q: How do we find them?
- check the property for each constant constructor
  
  ```
  BOBJ> red {} U {} .
  result NatSet: {}
  ```
- for each non-constant constructors
  - formulate the inductive hypothesis
    ```
    BOBJ> op s : -> NatSet .
    BOBJ> op n : -> Nat .
    BOBJ> eq {} U s = s .
    ```
Proof engineering for ADT: induction

- check the inductive conclusion

BOBJ> red {} U ins(n, s) == ins(n, s).

===============================================
reduce in SET : {} U ins(n, s) == ins(n, s)
result Bool: true
rewrite time: 46ms
Hidden Algebra (HA): the problem

stack

obj STACK[X :: TRIV] is
  sort Stack .
  op push : Elt Stack -> Stack .
  op pop : Stack -> Stack .
  op top : Stack -> Elt .
  var S : Stack . var E : Elt .
  eq top(push(E, S)) = E .
  eq pop(push(E, S)) = S .
endo
HA: the problem

- stack: implementation with arrays

\[
\begin{array}{c}
S \\
2 & 8 & 5 & \ldots \\
\end{array}
\]

push \(S, 6\)

\[
\begin{array}{c}
push(S, \ 6) \\
2 & 8 & 5 & 6 & \ldots \\
\end{array}
\]

pop \(S\)

\[
\begin{array}{c}
pop(S) \\
2 & 8 & 5 & 6 & \ldots \\
\end{array}
\]
HA: the problem

but

\[ \text{pop}\left(\text{push}(S, 6)\right) \neq S \]
HA: the problem

- if we consider the experiments:
  
  \[
  \text{top}(\_),
  \]
  
  \[
  \text{top}(\text{pop}(\_)),
  \]
  
  \[
  \text{top}(\text{pop}(\text{pop}(\_)))
  \]
  
  \[\Rightarrow\] the results are the same: 5 8 2

- we say that S and S' are \textbf{behavioral equivalent} iff they return equal responses for each experiment
HA specification

- signature = (H, V, Σ, D)
  - H - hidden sorts (not observable)
  - V - visible sorts (observable)
  - Σ - (V U H)-sorted signature
  - D - Σ|_V – algebra (data algebra)

- sentences: Σ - equations

- specification = (Σ, Γ, E)
  - Γ ⊆ Σ s.t. Γ|_V = Σ|_V
  - E – a set of equations
  - operations in Γ - Σ|_V are called behavioral
HA: models

- model - $\Sigma$-algebra $M$ s.t. $M|_{\Sigma|V} = D$
- $\equiv$ is the behavioral equivalence

$$a \equiv b \text{ iff } (\forall \Gamma\text{-experiment } c)[[c]](a) =[[c]](b)$$

- equations are behaviorally satisfied:

$$[[\theta(t)]] \equiv [[\theta(t')]$$
HA: specification of objects

- hidden sorts – models the state space of the object
- operations:
  - methods: \( f : h v_1 \ldots v_n \rightarrow h \)
  - observers (attributes): \( f : h v_1 \ldots v_n \rightarrow v \)
- concurrent composition operator: \(_ \mid _\)
Unreliable Counter

bth UCOUNTER is
  sort Counter .
  pr NAT .
  op init : -> Counter .
  op dec : Counter -> Counter [ncong] .
  op read : Counter -> Nat .
  ops (inc) (reset) : Counter -> Counter .

  var C : Counter .
  eq read(init) = 0 .
  eq val(reset(C)) = 0 .
  eq read(inc(C)) = read(C) + 1 .

end
Unreliable Counter

- $\Gamma = \{ \text{read}(), \text{reset}(), \text{inc}() \}$
- dec() is unreliable: the result of the experiment read(dec(C)) is not predictable
- therefore dec() is not congruent (ncong)
HA: proof engineering

- rewriting is not always sound
  - behavioral rewriting
    - disregards the beh. non-cong. opns
- semantics of a behavioral specification = the category of all models
  - induction is not appropriate (it is good only for the initial model, if any)
- behavioral reasoning is of coalgebraic nature
  - coinduction
- circular coinductive rewriting
HA: coinduction

- **Input:**
  \[ B = (\Sigma, \Gamma, E) \]
  \( t, \ t' \) \( \Sigma \)-terms (representing states)

- **Output**
  Are \( t \) and \( t' \) \( \Gamma \)-behavioral equivalent?

- **Method**
  1. define an appropriate relation \( R \)
  2. prove that \( R \) is a \( \Gamma \)-behavioral congruence
  3. if \( t R t' \) then return YES
Coinductive rewriting in BOBJ

BOBJ> open UCOUNTER .
BOBJ> ops c : -> Counter .
BOBJ> red reset(c) == init .
==========================================
reduce in UCOUNTER : reset(c) == init
result Bool: false
rewrite time: 0ms     parse time: 0ms
BOBJ> cred reset(c) == init .
==========================================
c-reduce in UCOUNTER : reset(c) == init
using cobasis for UCOUNTER:
  op read : Counter -> Nat
  --------------------------------------
result: true
c-rewrite time: 0ms     parse time: 0ms
Can HA be model checked?

- behavioral attributes $\Rightarrow$ queries
- behavioral methods $\Rightarrow$ actions
  - execution of a behavioral method produces a transition (in a model)
- queries defines atomic CTL formulas
- canonical CTL model
- cooperative activity of the model checking algorithm and HA proof engines
Cells

bth CELL is

sort Cell .

pr DATA−CELL .

op init : −→ Cell .

op read : Cell −→ Data .

op write : Cell Data −→ Cell .

var C : Cell . var D : Data .

eq read(write(C, D)) = D .

end
HA and CCS

- Buffers

  bth CELL1 is
    inc CELL * (sort Cell to Cell1).
  end

  bth CELL2 is
    inc CELL * (sort Cell to Cell2).
  end

  bth BUFFER2 is
    inc (CELL1 | CELL2)*
    (sort Configuration to Buffer2).
  end
HA and CCS

- in any state any operation is possible
- so, any scenario is possible
- sometimes we wish to investigate what happens for some given scenarios
- these scenarios can be described by CCS processes
HA and CCS

- considering the actions:
  \[ i \equiv \text{write.\text{CELL1}(\_, D)} \]
  \[ t | \sim t \equiv \text{write.\text{CELL2}(\_, \text{read.\text{CELL1}(\_)})) \]
  \[ \sim o \equiv \text{read.\text{CELL2}()} \]

- a scenario is:
  \[ C1 \equiv i.o.C1 \]
  \[ C2 \equiv i.o.C2 \]
  \[ B \equiv \text{new t} (\{t/o\}C1 | \{t/i\}C2) \]
Conclusions

- **Advantages**
  - sentences are very simple
  - a good mathematical support
  - rigorous semantics
  - specifications are executable
  - there are software tools

- **Drawbacks**
  - mathematical notation
  - … too much mathematics
  - a gap between specification and implementation
Conclusions

- What do we can do
  - accompany AS with more powerful and intuitive software tools
  - integrate AS with complementary methods
  - fill the gap between specification and implementation