On the Complexity of the Behavioral Properties

Grigore Roșu[1] Dorel Lucanu[2]

[1]Department of Computer Science
University of Illinois at Urbana-Champaign, USA
grosu@illinois.edu

[2]Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

November 2009, Eidhoven

Plan

A stack class

```
public class Array
{
  int[]! items;
  //...
  public void put(int elt,int idx)
    requires 0 <= idx;
  {
    items[idx] = elt;
  }

  public virtual int this [int idx]
  {
      [Pure]
      get
        requires 0 <= idx;
      {
        return items[idx];
      }
  }
}
```

```
class Stack
{
  Array a;
  int t;
  int err = System.Int32.MinValue;
  //...
  public void pop()
  {
     if (t > 0)
       t--;
  }

  public void push(int elt)
  {
     a.put(elt, ++t);
  }
}
```

the goal is to prove `pop(S.push(E)) = S`

A theory of indices

```
(theory IDX is
 sort Idx .
 including INT .
 op _+_ : Idx Int -> Idx .
 op _-_ : Idx Int -> Idx .
 op equal : Idx Idx -> Bool .
 op _<_ : Idx Idx -> Bool .
 op 0 : -> Idx .

 vars I J : Idx .

 eq (I + 1)- 1 = I .
 eq equal(I, I) = true .
 ceq equal(I, J) = false if I < J = true .
 eq I < I + 1 = true .
 eq 0 < I + 1 = true .
 ceq I - 1 < J = true if I < J = true .
endtheory)
```

A theory of arrays

```
(theory ARRAY is
 sorts Arr Elt .
 including IDX .
 op nil : -> Arr .
 op put : Elt Idx Arr -> Arr .
 op _‘[_‘] : Arr Idx -> Elt .

 vars I J : Idx . var A : Arr .
 var E : Elt .

 geq put(E, I, A) [J] =
       E    if equal(I, J) = true  []
       A[J] if equal(I, J) = false []
     .
endtheory)
```

An equational theory of stacks implemented with arrays

```
(theory STACKARRIMP is
 including ARRAY .
 sorts Stack .
 op <_‘,_> : Idx Arr -> Stack .
 op err : -> Elt .

 op empty : -> Stack .
 op push : Elt Stack -> Stack .
 op top_ : Stack -> Elt .
 op pop_ : Stack -> Stack .

 eq empty = < 0, nil > .
 eq push(E, < I, A >) = < I + 1, put(E, I + 1, A) > .
 geq top < I, A > =
       A[I] if 0 < I = true []
       err if 0 < I = false []
     .
 geq pop < I, A > =
       < I - 1, A > if 0 < I = true []
       < I, A > if 0 < I = false []
     .
endtheory)
```

An equational theory of stacks implemented with arrays

```
(theory STACKARRIMP is
 including ARRAY .
 sorts Stack .
 op <_‘,_> : Idx Arr -> Stack .
 op err : -> Elt .

 op empty : -> Stack .
 op push : Elt Stack -> Stack .
 op top_ : Stack -> Elt .
 op pop_ : Stack -> Stack .

 eq empty = < 0, nil > .
 eq push(E, < I, A >) = < I + 1, put(E, I + 1, A) > .
 geq top < I, A > =
       A[I] if 0 < I = true []
       err if 0 < I = false []
     .
 geq pop < I, A > =
       < I - 1, A > if 0 < I = true []
       < I, A > if 0 < I = false []
     .
endtheory)
```

pop(push(E:Elt, < I:Idx, A:Arr >)) $\neq$ < I:Idx, A:Arr >

An equational theory of stacks implemented with arrays

```
(theory STACKARRIMP is
 including ARRAY .
 sorts Stack .
 op <_‘,_> : Idx Arr -> Stack .
 op err : -> Elt .

 op empty : -> Stack .
 op push : Elt Stack -> Stack .
 op top_ : Stack -> Elt .
 op pop_ : Stack -> Stack .

 eq empty = < 0, nil > .
 eq push(E, < I, A >) = < I + 1, put(E, I + 1, A) > .
 geq top < I, A > =
       A[I] if 0 < I = true []
       err if 0 < I = false []
     .
 geq pop < I, A > =
       < I - 1, A > if 0 < I = true []
       < I, A > if 0 < I = false []
     .
endtheory)
```

DEMO using CIRC

`pop(push(E:Elt, < I:Idx, A:Arr >)) ≠ < I:Idx, A:Arr >`

Behaviorally equivalent stacks

- experiments:

  $top(S),\ top(pop(S)),\ top(pop(pop(S))),\ \ldots$

- two stacks S and S' are behaviorally equivalent, $S \equiv S'$, iff

  $top(S) = top(S')$

  $top(pop(S)) = top(pop(S'))$

  $top(pop(pop(S))) = top(pop(pop(S')))$

  $\ldots$

  i.e., $S \equiv S'$ iff $C[S] = C[S']$ for all experiments C
- $top(*{:}Stack)$ and $pop(*{:}Stack)$ are called derivatives
- `pop(push(E:Elt, < I:Idx, A:Arr >))` $\equiv$ `< I:Idx, A:Arr >`

An behavioral theory of stacks implemented with arrys

```
(theory STACKARRIMP is
 including ARRAY .
 sorts Stack .
 op <_‘,_> : Idx Arr -> Stack .
 op err : -> Elt .

 op empty : -> Stack .
 op push : Elt Stack -> Stack .
 op top_ : Stack -> Elt .
 op pop_ : Stack -> Stack .

 eq empty = < 0, nil > .
 eq push(E, < I, A >) = < I + 1, put(E, I + 1, A) > .
 geq top < I, A > =
       A[I] if 0 < I = true []
       err if 0 < I = false []
     .
 geq pop < I, A > =
       < I - 1, A > if 0 < I = true []
       < I, A > if 0 < I = false []
     .

 derivative top *:Stack .
 derivative pop *:Stack .
endtheory)
```

An behavioral theory of stacks implemented with arrys

```
(theory STACKARRIMP is
 including ARRAY .
 sorts Stack .
 op <_‘,_> : Idx Arr -> Stack .
 op err : -> Elt .

 op empty : -> Stack .
 op push : Elt Stack -> Stack .
 op top_ : Stack -> Elt .
 op pop_ : Stack -> Stack .

 eq empty = < 0, nil > .
 eq push(E, < I, A >) = < I + 1, put(E, I + 1, A) > .
 geq top < I, A > =
       A[I] if 0 < I = true []
       err if 0 < I = false []
     .
 geq pop < I, A > =
       < I - 1, A > if 0 < I = true []
       < I, A > if 0 < I = false []
     .

 derivative top *:Stack .
 derivative pop *:Stack .
endtheory)
```

DEMO
using
CIRC

Streams

- a stream (of bits) S is an infinite sequence $b_1 : b_2 : b_3 : \dots$
  $zeroes = 0 : zeroes$
  $ones = 1 : ones$
  $blink = 0 : 1 : blink$
  $zip(B : S, S') = B : zip(S', S)$
- the derivatives are $hd(*{:}Stream)$ (head) and $tl(*{:}Stream)$ (tail)
- operation specifications in terms of hd() and tl():
  $hd(zeroes) = 0,\ tl(zeroes) = zeroes,\ \dots$
  $hd(zip(B : S, S')) = B,\ tl(zip(B : S, S')) = zip(S', S)$
- experiments: $hd(S), hd(tl(*{:}Stream)), hd(tl(tl(*{:}Stream))), \dots$
- two streams S and S' are behaviorally equivalent, $S \equiv S'$, iff
  $hd(S) = hd(S'),\ hd(tl(S)) = hd(tl(S')),$
  $hd(tl(tl(S))) = hd(tl(tl(S'))),\ \dots$
- example of behavioral equality: $blink \equiv zip(zeroes, ones)$

Streams

- a stream (of bits) S is an infinite sequence $b_1 : b_2 : b_3 : \ldots$
  $zeroes = 0 : zeroes$
  $ones = 1 : ones$
  $blink = 0 : 1 : blink$
  $zip(B : S, S') = B : zip(S', S)$
- the derivatives are $hd(*{:}Stream)$ (head) and $tl(*{:}Stream)$ (tail)
- operation specifications in terms of hd() and tl():
  $hd(zeroes) = 0,\ tl(zeroes) = zeroes, \ldots$
  $hd(zip(B : S, S')) = B,\ tl(zip(B : S, S')) = zip(S', S)$
- experiments: $hd(S), hd(tl(*{:}Stream)), hd(tl(tl(*{:}Stream))), \ldots$
- two streams S and S' are behaviorally equivalent, $S \equiv S'$, iff
  $hd(S) = hd(S'),\ hd(tl(S)) = hd(tl(S')),$
  $hd(tl(tl(S))) = hd(tl(tl(S'))), \ldots$
- example of behavioral equality: $blink \equiv zip(zeroes, ones)$

Formal Framework [1/2]

- many-sorted signatures: (S, Σ)
  (S = set of sorts, Σ = set of operation names)
- Σ-equations: $(\forall X)\, t = u$
- many-sorted abstract logic for equality $\mathcal{L} = (Form_{\mathcal{L}}, \vdash_{\mathcal{L}})$:
  for each signature Σ
  - a set of Σ-sentences $Form_{\mathcal{L}}^{\Sigma}$;
  - Σ-specification is a signature (S, Σ) and a set F of Σ-sentences;
  - a satisfaction, or entailment relation $\vdash_{\mathcal{L}}^{\Sigma}$ between Σ-specifications and Σ-equations;
- examples:
  - First-order logic with equality (FOL)
  - Conditional equational logic (CEQ)
  - (Unconditional) equational logic (EQ)
  - Rewrite systems (REW)
  - Join rewrite systems (JOIN)

Formal Framework [2/2]

- a Σ-context for sort $h \in S$ is a Σ-term C having precisely one occurrence of a (special) variable $*$ of sort h
- behavioral signature is a pair (Σ, Δ), where Σ is a signature and Δ is a set of Σ-contexts, which we call derivatives
- if $\delta[*{:}h] \in \Delta$ then the sort h is called a hidden sort.
  Remaining sorts are called data, or visible, sorts;
- experiment:
  each visible $\delta[*{:}h] \in \Delta$ is an experiment, and
  if $C[*{:}h']$ is an experiment and $\delta[*{:}h] \in \Delta$, then so is $C[\delta[*{:}h]]$

Plan

Behavioral equivalence

Behavioral satisfaction

$\mathcal{B} \Vdash e$ iff:

$\mathcal{B} \vdash e$, if e is visible, and

$\mathcal{B} \vdash C[e]$ for each experiment C, if e is hidden

Behavioral equivalence of B: $\equiv_{\mathcal{B}} \stackrel{def}{=} \{e \mid \mathcal{B} \Vdash e\}$

A set of equations $\mathcal{G}$ is behaviorally closed iff

$\mathcal{B} \vdash visible(\mathcal{G})$ and $\Delta(\mathcal{G} - \mathcal{B}^\bullet) \subseteq \mathcal{G}$,

where $\mathcal{B}^\bullet = \{e \mid \mathcal{B} \vdash e\}$

Theorem

(coinduction) *If $\vdash_{\mathcal{L}}$ is reflexive, monotonic, transitive, and Δ-congruent (if $E \vdash e$ then $E \vdash_{\mathcal{L}} \Delta[e]$), then the behavioral equivalence $\equiv$ is the largest behaviorally closed set of equations.*

Special Contexts

Context $\gamma[*{:}h]$ is special iff for any experiment C for γ there is some term t such that

1. $\mathcal{B} \vdash C[\gamma[*{:}h]] = t$ and
2. each occurrence of $*{:}h$ in t appears in a subterm which is an experiment of depth smaller than or equal to that of C.

Examples:

- $zip(*{:}Stream, S)$ and $zip(S, *{:}Stream)$ are special contexts, as well as any combination of these
- if the stream operations $odd(S)$ and $even(S)$ are defined by

$$hd(odd(S)) = hd(s) \qquad even(S) = odd(tl(S))$$
$$tl(odd(S)) = even(tl(S))$$

  then $odd(*{:}Stream)$ is not special:
  $hd(tl(odd(*{:}Stream))) = hd(tl(tl(*{:}Stream)))$ and the depth of $hd(tl(tl(*{:}Stream)))$ is larger than the depth of $hd(tl(*{:}))$
  the same is true for $even(*{:}Stream)$.

Special Contexts for Coinductive Contextual Reasoning

If $odd(S)$ were special, then one would be able to wrongly "prove" by coinduction behavioral equivalences:

- assume a stream a defined by $hd(a) = 0$ and $tl(a) = odd(a)$
- the following wrongly "proves" that $a \equiv zeroes$ by coinduction:
  - pick $a \sim zeroes$
  - show that $hd(a) = hd(zeroes)$ (obviously)
  - show that $tl(a) \sim tl(zeroes)$
    $(tl(a) = odd(a) \sim odd(zeroes) = zeroes = tl(zeroes))$
  - conclude that $a \equiv zeroes$ holds, because behavioral equivalence is the largest binary relation compatible with hd and tl ($\sim \subseteq \equiv$).
- This is a contradiction because the stream $a = 0{:}0{:}ones$ also satisfies the two equations of a.

Behavioral Consistency

Intuition:
the data is rigid from a behavioral point of view, that is, the hidden part can only use it but cannot distort it.
E.g., if $STREAM \vdash zeroes = ones$, the we conclude
$0 = hd(zeroes) = hd(ones) = 1$.

Formal definition:
$\mathcal{B} = ((\Sigma, \Delta), F)$ is behaviorally consistent iff for any data equation e, if $\mathcal{B} \Vdash e$ (or, equivalently, $\mathcal{B} \vdash e$) then $\mathcal{B}\restriction_V \vdash e$, where $\mathcal{B}\restriction_V$ is the "visible" restriction of $\mathcal{B}$ (i.e., $\Sigma\restriction_V$-specification consisting of the visible sentences in F).

Behavioral Well-Definedness [1/2]

[inspired by Hans Zantema's paper RTA 2099]

Intuition: $\mathcal{B}$ well-defines t iff any "clone" t' of t behaves like t, i.e., t and t' are behaviorally equivalent

The questions is what is a "clone"?

Given $\mathcal{B} = ((\Sigma, \Delta), F)$, let $\mathcal{B}'$ extend $\mathcal{B}$ by

- adding to Σ a copy σ' of each $\sigma \in \Sigma - (\Sigma\restriction_V \cup \Delta)$
- and to F a copy φ' of each $\varphi \in F$, where φ' is obtained by replacing each $\sigma \in \Sigma - (\Sigma\restriction_V \cup \Delta)$ in φ with σ'.

Behavioral Well-Definedness [2/2]

$\mathcal{B}$ well-defines term t with variables in X, or t is well-defined by $\mathcal{B}$, iff $\mathcal{B}' \Vdash (\forall X)\, t = t'$, where t' is obtained by replacing each $\sigma \in \Sigma - (\Sigma\restriction_V \cup \Delta)$ in t with σ'.

Example: the stream a specified by the equations $hd(a) = 0$ and $tl(a) = odd(a)$ is not well-defined, since
$hd(tl^2(a)) = hd(tl(odd(a))) = hd(odd(tl^2(a))) = hd(tl^2(a)) = \ldots$

the same is true for any clone a' of a, therefore no chance to show that $hd(tl^2(a)) = hd(tl^2(a'))$.

Behavioral Productivity [1/3]

– strong related to the behavioral well-Definedness, but are NOT identical

– inspired from the similar notion for the infinitary term rewriting systems, but, again, the two notions are NOT the same

– intuitively, a behavioral specification $\mathcal{B}$ is productive for a term t iff any Δ-experiment over t is evaluatable.

– stream well-defined but not productive: $(\forall S)\, S = a$

– stream not productive when specified as infinitary trs but behaviorally productive:

$zeros \to 0{:}zeros, \;\; f(x{:}s) \to g(f(s)), \;\;\; g(x{:}s) \to zeros$

[Hans Zantema, RTA 2009, Example 4]

Behavioral Productivity [2/3]

Proposition

If the abstract logic $\mathcal{L}$

- is **monotone**, that is, if $(\Sigma, F) \vdash_{\mathcal{L}}^{\Sigma} e$ and (Σ', F') is a Σ'-specification such that $\Sigma \subseteq \Sigma'$ and $F \subseteq F'$ then $(\Sigma', F') \vdash_{\mathcal{L}}^{\Sigma'} e$, and
- is α-**invariant**, that is, if $(\Sigma, F) \vdash_{\mathcal{L}}^{\Sigma} e$ then $(\Sigma[f'/f], F[f'/f]) \vdash_{\mathcal{L}}^{\Sigma} e[f'/f]$, where $\cdot[f'/f]$ substitutes fresh operation f' for f, and
- has the **equational join property**, that is, $\mathcal{B} \vdash (\forall X)\, t = w$ and $\mathcal{B} \vdash (\forall X)\, u = w$ implies $\mathcal{B} \vdash (\forall X)\, t = u$,

then $\mathcal{B}$ productive for term t implies $\mathcal{B}$ well-defined for term t.

Behavioral Productivity [3/3]

Theorem

Let $\mathcal{L} = \mathrm{JOIN}$ and let $\mathcal{B} = ((\Sigma, \Delta), R)$ be a behavioral specification (i.e., a Σ-term rewrite system R with a set of derivatives Δ). Then:

1. *If the rules in R "do not introduce" operations in $\Sigma - (\Sigma\restriction_V \cup \Delta)$, that is, if for each $(l \to r) \in R$ it is the case that if l does not contain operations in $\Sigma - (\Sigma\restriction_V \cup \Delta)$ then r does not contain operations in $\Sigma - (\Sigma\restriction_V \cup \Delta)$ either, then $\mathcal{B}$ is well-defined on term t if and only if $\mathcal{B}$ is productive on term t; and*
2. *If R terminates, and $\Sigma - (\Sigma\restriction_V \cup \Delta)$ contains only operations of hidden result sort, and for every $f : \overline{s} \to h$ in $\Sigma - (\Sigma\restriction_V \cup \Delta)$ and derivative $\delta[*{:}h]$ in Δ there is some rule $\delta[f(\overline{x})] \to r$ in R, then $\mathcal{B}$ is productive.*

Plan

The class Π^0_2

A fragment of the arithmetic hierarchy:
$\Sigma^0_0 = \Pi^0_0 =$ the set of recursive predicates
$\Sigma^0_1 = \{(\exists y) \mid r(x, y, z) \in \Sigma^0_0\} =$ the set of r.e. predicates
$\Pi^0_2 = \{(\forall x)(\exists y) \mid r(x, y, z) \in \Sigma^0_0\}$

A canonical Π^0_2-complete problem is
$\textsc{Totality}(M) := (\forall x)(\exists n)\ \textsc{Stop}(x, n, M)$, asking whether computational device (Turing machine, program, rewrite system, etc.) M stops on all its inputs

If M is a trs, $\textsc{Totality}(M)$ is equivalent to terminating property [J.G. Simonsen, RTA 2009]

A scheme for proving Π_2^0-completeness

Definition

An abstract logic $\mathcal{L}$ is called **Turing complete** iff:
(1) $\vdash_{\mathcal{L}}^{\Sigma}$ is recursively enumerable for each signature Σ, and
(2) it can encode a universal computational model.

Theorem

If $\mathcal{L}$ be a Turing complete abstract logic,
$\textsc{Problem}(\mathcal{B}, inp)$ *a behavioral problem,*
for each (Σ_M, F_M) encoding a machine M there is $\mathcal{B}(\Sigma_M, F_M)$ s.t.:

1. *there is a bijective mapping between the inputs x of M and the experiments in $\mathcal{B}(\Sigma_M, F_M)$; let C^x denote the context associated to x;*
2. *there is a recursive predicate $pred(x, n, \mathcal{B}(\Sigma_M, F_M))$ which holds if and only if $\mathcal{B}(\Sigma_M, F_M) \vdash C^x[\varphi(inp)]$ and its Gödel number is $\leq n$;*
3. *$(\exists n)\ pred(x, n, \mathcal{B}(\Sigma_M, F_M))$ holds iff $(\exists n)\ \textsc{Stop}_{\mathcal{L}}(\langle M, x\rangle =\downarrow, n, (\Sigma_M, F_M))$;*

then $\textsc{Totality}(M) \iff \textsc{Problem}(\mathcal{B}(\Sigma_M, F_M), inp)$.

Π^0_2-completeness

The following problems are Π^0_2-complete:

BehavioralEquivalence
Instance : $\mathcal{B}, \mathcal{L}, e$
Question : $\mathcal{B} \vdash_{\mathcal{L}} e$?

BehavioralConsistency
Instance : $\mathcal{B}, \mathcal{L}$
Question : Is $\mathcal{B}$ consistent?

BehavioralWell – Definedness
Instance : $\mathcal{B}, \mathcal{L}, t$
Question : Does $\mathcal{B}$ well defines t?

SpecialContext
Instance : $\mathcal{B}, \mathcal{L}, \gamma$
Question : Is γ special?

BehavioralProductivity
Instance : $\mathcal{B}, \mathcal{L}, t$
Question : Is $\mathcal{B}$ productive for t?

Plan

Related Approaches

S. Buss and G. Roșu. Incompleteness of behavioral logics. In *Proceeding of CMCS'00*, volume 33 of *ENTCS*, pages 61–79. Elsevier, 2000.

G. Roșu. Equality of streams is a Π_2^0-complete problem. In *Proceedgins of ICFP'06*, pages 184–191. ACM, 2006.

J. G. Simonsen. The Π_2^0-completeness of most of the properties of rewriting systems you care about (and productivity). In *Proceedings of RTA'09*, volume 5595 of *LNCS*, pages 335–349, 2009.

H. Zantema. Well-definedness of streams by termination. In *RTA 2009*, volume 5595 of *LNCS*, pages 164–178. Springer, 2009.

Future work

- it would be nice to have in CIRC specification static analysis means checking for behavioral properties
- integrate CIRC into a uniform and integrated rewriting-based framework for the design and analysis of programming languages (e.g., for verifying invariants)

Thanks!