On the Complexity of the Behavioral Properties

Grigore Roşu\textsuperscript{1} \quad Dorel Lucanu\textsuperscript{2}

\textsuperscript{1}Department of Computer Science
University of Illinois at Urbana-Champaign, USA
grusu@illinois.edu

\textsuperscript{2}Faculty of Computer Science
Alexandru Ioan Cuza University, Iaşi, Romania
dlucanu@info.uaic.ro

November 2009, Eindhoven
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
(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 >)) ≠ < 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)

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

- experiments:
  \[ \text{top}(S), \text{top}(\text{pop}(S)), \text{top}(\text{pop}(\text{pop}(S))), \ldots \]
- two stacks \( S \) and \( S' \) are behaviorally equivalent, \( S \equiv S' \), iff
  \[ \text{top}(S) = \text{top}(S') \]
  \[ \text{top}(\text{pop}(S)) = \text{top}(\text{pop}(S')) \]
  \[ \text{top}(\text{pop}(\text{pop}(S))) = \text{top}(\text{pop}(\text{pop}(S')))) \]
  
  \ldots
  
  i.e., \( S \equiv S' \) iff \( C[S] = C[S'] \) for all experiments \( C \)
- \( \text{top}(\ast:\text{Stack}) \) and \( \text{pop}(\ast:\text{Stack}) \) are called derivatives
- \( \text{pop}(\text{push}(E:\text{Elt}, < I:\text{Id}, A:\text{Arr} >)) \equiv < I:\text{Id}, A:\text{Arr} > \)
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 < = false []

  derivative top *:Stack .
  derivative pop *:Stack .
endtheory)
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
  
  \[ \text{zip}(B : S, S') = B : \text{zip}(S', S) \]

- the derivatives are \( \text{hd}(\ast : \text{Stream}) \) (head) and \( \text{tl}(\ast : \text{Stream}) \) (tail)

- operation specifications in terms of \( \text{hd}() \) and \( \text{tl}() \):
  
  \[ \begin{align*}
  \text{hd}(\text{zeroes}) &= 0, \\
  \text{tl}(\text{zeroes}) &= \text{zeroes}, \ldots \\
  \text{hd}(\text{zip}(B : S, S')) &= B, \\
  \text{tl}(\text{zip}(B : S, S')) &= \text{zip}(S', S)
  \end{align*} \]

- experiments: \( \text{hd}(S), \text{hd}(\text{tl}(\ast : \text{Stream})), \text{hd}(\text{tl}(\text{tl}(\ast : \text{Stream}))), \ldots \)

- two streams \( S \) and \( S' \) are behaviorally equivalent, \( S \equiv S' \), iff
  
  \[ \begin{align*}
  \text{hd}(S) &= \text{hd}(S'), \\
  \text{hd}(\text{tl}(S)) &= \text{hd}(\text{tl}(S')), \\
  \text{hd}(\text{tl}(\text{tl}(S))) &= \text{hd}(\text{tl}(\text{tl}(S'))), \ldots
  \end{align*} \]

- example of behavioral equality: \( \text{blink} \equiv \text{zip}(\text{zeroes}, \text{ones}) \)
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 $hd(\ast:\text{Stream})$ (head) and $tl(\ast:\text{Stream})$ (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: $hd(S)$, $hd(tl(\ast:\text{Stream}))$, $hd(tl(tl(\ast:\text{Stream})))$, \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: $blink \equiv zip(zeroes, ones)$
many-sorted signatures: $(S, \Sigma)$
$(S = \text{set of sorts, } \Sigma = \text{set of operation names})$

\(\Sigma\)-equations: $(\forall X) t = u$

many-sorted abstract logic for equality \(\mathcal{L} = (\text{Form}_{\mathcal{L}}, \vdash_{\mathcal{L}})\):
for each signature \(\Sigma\)
- a set of \(\Sigma\)-sentences \(\text{Form}_{\mathcal{L}}^{\Sigma}\);
- \(\Sigma\)-specification is a signature \((S, \Sigma)\) and a set \(F\) of \(\Sigma\)-sentences;
- a satisfaction, or entailment relation \(\vdash_{\mathcal{L}}^{\Sigma}\) between \(\Sigma\)-specifications and \(\Sigma\)-equations;

examples:
- First-order logic with equality (FOL)
- Conditional equational logic (CEQ)
- (Unconditional) equational logic (EQ)
- Rewrite systems (REW)
- Join rewrite systems (JOIN)
a $\Sigma$-context for sort $h \in S$ is a $\Sigma$-term $C$ having precisely one occurrence of a (special) variable $\ast$ of sort $h$.

- behavioral signature is a pair $(\Sigma, \Delta)$, where $\Sigma$ is a signature and $\Delta$ is a set of $\Sigma$-contexts, which we call derivatives.

- if $\delta[\ast:h] \in \Delta$ then the sort $h$ is called a hidden sort. Remaining sorts are called data, or visible, sorts;

- experiment:
  - each visible $\delta[\ast:h] \in \Delta$ is an experiment, and
  - if $C[\ast:h']$ is an experiment and $\delta[\ast:h] \in \Delta$, then so is $C[\delta[\ast:h]]$
Plan
Behavioral equivalence

Behavioral satisfaction
\[ \mathcal{B} \models e \iff \mathcal{B} \vdash e, \text{ if } e \text{ is visible, and } \mathcal{B} \vdash C[e] \text{ for each experiment } C, \text{ if } e \text{ is hidden} \]

Behavioral equivalence of \( \mathcal{B} \): \( \equiv_{\mathcal{B}} \overset{\text{def}}{=} \{ e \mid \mathcal{B} \models e \} \)

A set of equations \( \mathcal{G} \) is behaviorally closed iff
\[ \mathcal{B} \vdash \text{visible}(\mathcal{G}) \text{ and } \Delta(\mathcal{G} - \mathcal{B}^\bullet) \subseteq \mathcal{G}, \]
where \( \mathcal{B}^\bullet = \{ e \mid \mathcal{B} \vdash e \} \)

Theorem

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

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

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

Examples:

- $\text{zip}(\star:\text{Stream}, S)$ and $\text{zip}(S, \star:\text{Stream})$ are special contexts, as well as any combination of these.
- if the stream operations $\text{odd}(S)$ and $\text{even}(S)$ are defined by
  \[
  \text{hd}(\text{odd}(S)) = \text{hd}(s) \quad \text{even}(S) = \text{odd}(\text{tl}(S))
  \]
  \[
  \text{tl}(\text{odd}(S)) = \text{even}(\text{tl}(S))
  \]
  then $\text{odd}(\star:\text{Stream})$ is not special:
  \[
  \text{hd}(\text{tl}(\text{odd}(\star:\text{Stream}))) = \text{hd}(\text{tl}(\text{tl}(\star:\text{Stream})))
  \]
  and the depth of $\text{hd}(\text{tl}(\text{tl}(\star:\text{Stream})))$ is larger than the depth of $\text{hd}(\text{tl}(\star:\))$
  the same is true for $\text{even}(\star:\text{Stream})$. 

G. Roșu, D. Lucanu (UIUC, UAIC)
If $\text{odd}(S)$ were special, then one would be able to wrongly “prove” by coinduction behavioral equivalences:

- assume a stream $a$ defined by $\text{hd}(a) = 0$ and $\text{tl}(a) = \text{odd}(a)$
- the following wrongly “proves” that $a \equiv \text{zeroes}$ by coinduction:
  - pick $a \sim \text{zeroes}$
  - show that $\text{hd}(a) = \text{hd}(\text{zeroes})$ (obviously)
  - show that $\text{tl}(a) \sim \text{tl}(\text{zeroes})$
    
    ($\text{tl}(a) = \text{odd}(a) \sim \text{odd}(\text{zeroes}) = \text{zeroes} = \text{tl}(\text{zeroes})$)
  - conclude that $a \equiv \text{zeroes}$ holds, because behavioral equivalence is the largest binary relation compatible with $\text{hd}$ and $\text{tl}$ ($\sim \subseteq \equiv$).

- This is a contradiction because the stream $a = 0:0:\text{ones}$ 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 \( \text{STREAM} \vdash \text{zeroes} = \text{ones} \), the we conclude
\[ 0 = \text{hd}(\text{zeroes}) = \text{hd}(\text{ones}) = 1. \]

Formal definition:
\( B = ((\Sigma, \Delta), F) \) is behaviorally consistent iff for any data equation \( e \), if \( B \models e \) (or, equivalently, \( B \vdash e \)) then \( B \upharpoonright V \vdash e \), where \( B \upharpoonright V \) is the “visible” restriction of \( B \) (i.e., \( \Sigma \upharpoonright V \)-specification consisting of the visible sentences in \( F \)).
Behavioral Well-Definedness

[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 $\Sigma$ a copy $\sigma'$ of each $\sigma \in \Sigma - (\Sigma|_V \cup \Delta)$
- and to $F$ a copy $\varphi'$ of each $\varphi \in F$, where $\varphi'$ is obtained by replacing each $\sigma \in \Sigma - (\Sigma|_V \cup \Delta)$ in $\varphi$ with $\sigma'$. 

Behavioral Well-Definedness

\( B \) well-defines term \( t \) with variables in \( X \), or \( t \) is well-defined by \( B \), iff \( B' \models (\forall X) t = t' \), where \( t' \) is obtained by replacing each \( \sigma \in \Sigma - (\Sigma | V \cup \Delta) \) in \( t \) with \( \sigma' \).

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

– 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 $B$ is productive for a term $t$ iff any $\Delta$-experiment over $t$ is evaluable.
– stream well-defined but not productive: $(\forall S) S = a$
– stream not productive when specified as infinitary trs but behaviorally productive:
  $zeros \rightarrow 0:zeros, \ f(x:s) \rightarrow g(f(s)), \ g(x:s) \rightarrow zeros$
[Hans Zantema, RTA 2009, Example 4]
Proposition

If the abstract logic $\mathcal{L}$

- is **monotone**, that is, if $(\Sigma, F) \vdash_{\mathcal{L}} e$ and $(\Sigma', F')$ is a $\Sigma'$-specification such that $\Sigma \subseteq \Sigma'$ and $F \subseteq F'$ then $(\Sigma', F') \vdash_{\mathcal{L}} e$, and

- is $\alpha$-**invariant**, that is, if $(\Sigma, F) \vdash_{\mathcal{L}} e$ then $(\Sigma[f'/f], F[f'/f]) \vdash_{\mathcal{L}} e[f'/f]$, where $\cdot[f'/f]$ substitutes fresh operation $f'$ for $f$, and

- has the **equational join property**, that is, $B \vdash (\forall X) t = w$ and $B \vdash (\forall X) u = w$ implies $B \vdash (\forall X) t = u$, then $B$ productive for term $t$ implies $B$ well-defined for term $t$. 
Theorem

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

1. If the rules in $R$ “do not introduce” operations in $\Sigma - (\Sigma \upharpoonright \mathcal{V} \cup \Delta)$, that is, if for each $(l \rightarrow r) \in R$ it is the case that if $l$ does not contain operations in $\Sigma - (\Sigma \upharpoonright \mathcal{V} \cup \Delta)$ then $r$ does not contain operations in $\Sigma - (\Sigma \upharpoonright \mathcal{V} \cup \Delta)$ either, then $B$ is well-defined on term $t$ if and only if $B$ is productive on term $t$; and

2. If $R$ terminates, and $\Sigma - (\Sigma \upharpoonright \mathcal{V} \cup \Delta)$ contains only operations of hidden result sort, and for every $f : \bar{s} \rightarrow h$ in $\Sigma - (\Sigma \upharpoonright \mathcal{V} \cup \Delta)$ and derivative $\delta[\star : h]$ in $\Delta$ there is some rule $\delta[f(\bar{x})] \rightarrow r$ in $R$, then $B$ is productive.
Plan
The class $\Pi^0_2$

A fragment of the arithmetic hierarchy:

$\Sigma^0_0 = \Pi^0_0 = \text{the set of recursive predicates}$

$\Sigma^0_1 = \{(\exists y) \mid r(x, y, z) \in \Sigma^0_0\} = \text{the set of r.e. predicates}$

$\Pi^0_2 = \{(\forall x)(\exists y) \mid r(x, y, z) \in \Sigma^0_0\}$

A canonical $\Pi^0_2$-complete problem is

$\text{Totality}(M) := (\forall x)(\exists n) \text{Stop}(x, n, M)$, asking whether computational device (Turing machine, program, rewrite system, etc.) $M$ stops on all its inputs

If $M$ is a trs, $\text{Totality}(M)$ is equivalent to terminating property

[J.G. Simonsen, RTA 2009]
Main result

A scheme for proving $\Pi^0_2$-completeness

Definition

An abstract logic $\mathcal{L}$ is called **Turing complete** iff:

1. $\vdash_{\mathcal{L}} \Sigma$ is recursively enumerable for each signature $\Sigma$, and
2. it can encode a universal computational model.

Theorem

If $\mathcal{L}$ be a Turing complete abstract logic, $\text{Problem}(\mathcal{B}, \text{inp})$ a behavioral problem, for each $(\Sigma_M, F_M)$ 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 $\text{pred}(x, n, \mathcal{B}(\Sigma_M, F_M))$ which holds if and only if $\mathcal{B}(\Sigma_M, F_M) \vdash C^x[\varphi(\text{inp})]$ and its Gödel number is $\leq n$;
3. $(\exists n) \text{pred}(x, n, \mathcal{B}(\Sigma_M, F_M))$ holds iff $(\exists n) \text{Stop}_L(\langle M, x \rangle = \downarrow, n, (\Sigma_M, F_M))$;

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

G. Roșu, D. Lucanu (UIUC, UAIC) Complexity of the Behavioral Properties November 2009, Eindhoven 25 / 1
The following problems are $\Pi_2^0$-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 $\gamma$ special?

**BehavioralProductivity**
- **Instance**: $\mathcal{B}, \mathcal{L}, t$
- **Question**: Is $\mathcal{B}$ productive for $t$?
Plan
Related Approaches


J. G. Simonsen. The $\Pi^0_2$-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.

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!