Abstractions of Data Types

Ferucio Laurenţiu Ţiplea

Faculty of Computer Science

“Al.I.Cuza” University of Iasi

6600 Iasi, Romania

Aim

- Investigate three types of abstractions in the context of (abstract) data types, and provide preservations results that generalize preservation results known from:
  - Shape analysis
  - Predicate abstraction
  - McMillan's approach
  - Duplicating predicate symbols technique
  - etc.
- Investigate equationally defined abstractions in the context of (abstract) data types.

Framework

Abstract data types modeled by universal algebras

1. J. Mitchell. *Foundations of Programming Languages*, The MIT Press, 1996.
2. J. Loecks, H.-D. Ehrich, M. Wolf. *Algebraic Specification of Abstract Data Types*, in Handbook of Logic in Computer Science, vol 5, Clarendon Press, 2000, 217–316.
3. H. Ehrig, D. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, Springer-Verlag, 1985.
4. H. Ehrig, D. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, Springer-Verlag, 1990.

Terminology

- Data types modeled by universal algebras. Why?
  - mathematical precision
  - independence of implementation
  - axiomatic definition of operations
  - suitable to reason about operations and their properties
- Abstract data types modeled by classes of universal algebras closed under isomorphism. Why?
  - the closer under isomorphism corresponds to the similarity concept
- Specifications given by sets of equations
- Model = data type (algebra) which satisfies a specification

A Motivating Example

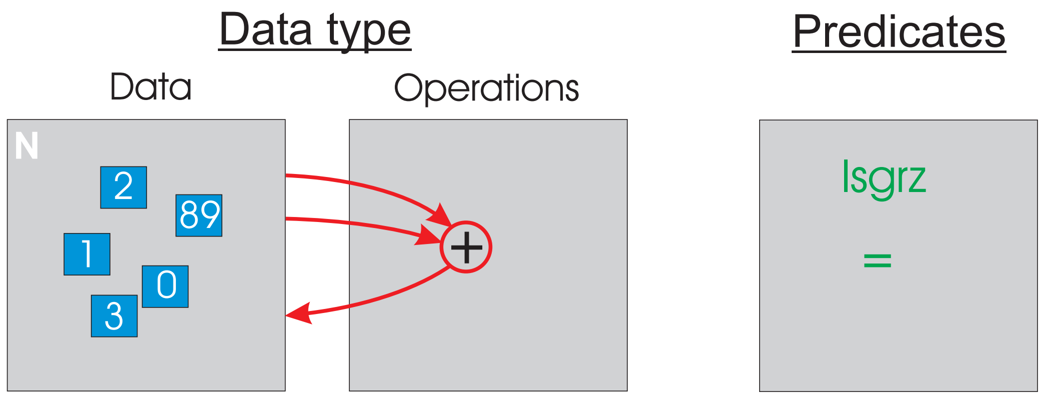


Figure: A data type $A = (\mathbf{N}, +^A)$ together with a set of predicates

The following property holds true:

$$(\forall x, y \in A)(Isgrz^A(x) \vee Isgrz^A(y) \;\Rightarrow\; Isgrz^A(x +^A y))$$

$|Spec1\rangle$

A Motivating Example (cont'd)

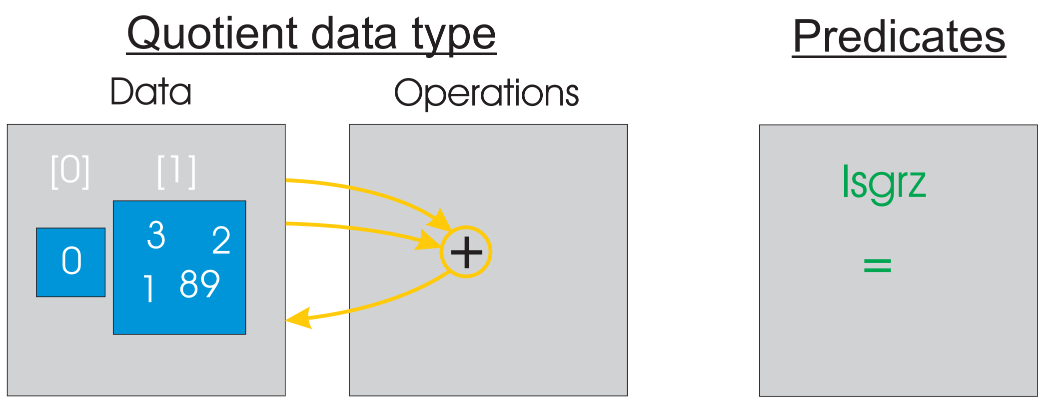


Figure: The quotient data type $A/_\rho = (\mathbf{N}/_\rho, +^{A/_\rho})$ together with a set of predicates

Let $Isgrz^{A/_\rho}$ be the interpretation of $Isgrz$ in $A/_\rho$ given by

$$Isgrz^{A/_\rho}([a]_\rho) \text{ iff } (\forall a' \in [a]_\rho)(Isgrz^A(a'))$$

The following property holds true:

$$(\forall x, y \in A/_\rho)(Isgrz^{A/_\rho}(x) \vee Isgrz^{A/_\rho}(y) \;\Rightarrow\; Isgrz^{A/_\rho}(x +^{A/_\rho} y))$$

$|Spec2\rangle$

A Motivating Example (cont'd)

Conclusions:

(1) the meta-language used to express properties of data types (algebras) should be specific to signatures and not to data types (algebras);

(2) data type reductions can be captured by congruences. In such a case, the operations are automatically redefined to operate on the quotient data type (algebra), but the predicates need a special treatment.

Logically Extended Signatures

- logical type
  - $w \in S^+$
  - $w = (nat, bool)$, $w = (nat, nat, bool)$
- logical S-sorted signature
  - Σ_L contains only logical symbols (predicate symbols)
  - $\Sigma_L = \{Isgrz, =\}$
- logically extended S-sorted signature
  - (Σ, Σ_L), where Σ is an S-sorted signature

(Σ, Σ_L)-algebras

- A Σ-algebra does the following:
  - associates domains to sorts
  - interprets the function symbols as operations of corresponding types
- A (Σ, Σ_L)-algebra does the following:
  - associates domains to sorts
  - interprets the function symbols as operations of corresponding types
  - interprets the logical symbols into $\{0, 1, \perp\}$

We use Kleene's 3-valued first order logic

$|Kleene\rangle$

Kleene's 3-valued First Order Logic

- first order formulas over (Σ, Σ_L) and X
  - $\mathcal{L}(\Sigma, \Sigma_L, X)$
- positive formulas
  - $\mathcal{L}^+(\Sigma, \Sigma_L, X)$
- assignment
  - $\gamma : X \rightarrow A$
- the interpretation function of φ into $\mathbf{A}$
  - $\mathcal{I}_{\mathbf{A}}(\varphi) : \Gamma(X, \mathbf{A}) \rightarrow A \cup \{0, 1, \perp\}$

$$\boxed{\mathbf{A} \models \varphi \quad \Leftrightarrow \quad (\forall \gamma : X \rightarrow A)(\mathcal{I}_{\mathbf{A}}(\varphi)(\gamma) = 1)}$$

Abstractions of Models

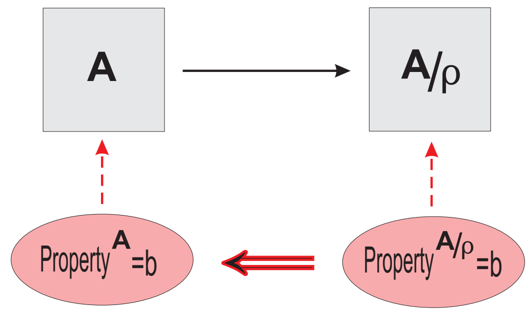
An abstraction of a (Σ, Σ_L)-algebra $\mathbf{A}$ is any couple consisting of:

- a quotient algebra of $\mathbf{A}$ under a congruence ρ ($\mathbf{A}/_{\rho}$), and
- an interpretation of the logical symbols into $\mathbf{A}/_{\rho}$

Congruences can be defined by:

- surjective homomorphisms
- sets of predicates
- partitions
- etc.

Property Preservation



- strong-preservation — if a set of properties with truth values true or false in the abstract system has corresponding properties in the concrete system with the same truth values;
- weak-preservation — if a set of properties true in the abstract system has corresponding properties in the concrete system that are also true;
- error-preservation — if a set of properties false in the abstract system has corresponding properties in the concrete system that are also false.

Types of Abstractions

$p^A(a'_1,\ldots,a'_n),\ a'_i \in [a_i]$	$p^{A/\rho}([a_1],\ldots,[a_n])$		
	$\forall\forall$-abs	$\forall\exists$-abs	$\exists^{0,1}\forall$-abs
all 1	1	1	1
all 0	0	0	0
$\perp$ and 0/1	$\perp$	0,$\perp$	$\perp$
0 and 1	$\perp$	0	1

Property Preservation – $\forall\forall$

Theorem Let $\mathbf{A}$ be a (Σ, Σ_L)-algebra, ρ a $\forall\forall$-abstraction of $\mathbf{A}$, and φ a formula. Then

$$\mathcal{I}_{\mathbf{A}/\rho}(\varphi)(\gamma) = b \;\;\Rightarrow\;\; (\forall \gamma' \in \gamma)(\mathcal{I}_{\mathbf{A}}(\varphi)(\gamma') = b),$$

for any $b \in \{0, 1\}$ and $\gamma \in \Gamma(X, \mathbf{A}/\rho)$.

Corollary $\forall\forall$-abstractions of (Σ, Σ_L)-algebras are strongly preserving w.r.t. formulas in $\mathcal{L}(\Sigma, \Sigma_L, X)$.

Property Preservation – $\forall\exists$

Theorem Let $\mathbf{A}$ be a (Σ, Σ_L)-algebra, ρ an abstraction of $\mathbf{A}$, and φ a formula in $\mathcal{L}^+(\Sigma, \Sigma_L, X)$. If ρ is an $\forall\exists$-abstraction then

$$\mathcal{I}_{\mathbf{A}/\rho}(\varphi)(\gamma) = 1 \;\Rightarrow\; (\forall\gamma' \in \gamma)(\mathcal{I}_{\mathbf{A}}(\varphi)(\gamma') = 1),$$

for all $\gamma \in \Gamma(X, \mathbf{A}/\rho)$.

Corollary $\forall\exists$-abstractions of (Σ, Σ_L)-algebras are weakly preserving w.r.t. formulas in $\mathcal{L}^+(\Sigma, \Sigma_L, X)$.

Property Preservation – $\exists^{0,1}\forall$

Theorem Let $\mathbf{A}$ be a (Σ, Σ_L)-algebra, ρ an abstraction of $\mathbf{A}$, and φ a formula in $\mathcal{L}^+(\Sigma, \Sigma_L, X)$. If ρ is an $\exists^{0,1}\forall$-abstraction then

$$\mathcal{I}_{\mathbf{A}/\rho}(\varphi)(\gamma) = 0 \;\Rightarrow\; (\forall \gamma' \in \gamma)(\mathcal{I}_{\mathbf{A}}(\varphi)(\gamma') = 0),$$

for all $\gamma \in \Gamma(X, \mathbf{A}/\rho)$.

Corollary $\exists^{0,1}\forall$-abstractions of (Σ, Σ_L)-algebras are error preserving w.r.t. formulas in $\mathcal{L}^+(\Sigma, \Sigma_L, X)$.

Applications

The following formalisms can be viewed as particular cases of our approach (regarding the abstraction method and the corresponding preservation results):

- predicate abstraction
- shape analysis
- the technique of duplicating predicate symbols
- McMillan's approach

Abstractions of ADTs

- Abstract Data Type (ADT): class of algebras closed under isomorphism
  - monomorphic
  - polymorphic
- Specification of an ADT
  - syntax (fixes the “form”)
  - semantics (fixes the “meaning”)
- Initial specification
  - (syntax) $Sp = (\Sigma, E)$ where Σ is a signature and E is a set of Σ-equations
  - (semantics) $\mathcal{M}(Sp) = \{\mathbf{A} | \mathbf{A} \cong \mathbf{T}_{\Sigma,E}\}$

  $\mathcal{M}(Sp)$ is also called the monomorphic ADT defined by Sp

Abstractions of ADTs

Initial logically extended specification

- (syntax) $Sp = (\Sigma, \Sigma_L, E, \Sigma_L^{T_{\Sigma,E}})$
  - (Σ, Σ_L) is a logically extended signature
  - E is a set of Σ-equations
  - $\Sigma_L^{T_{\Sigma,E}}$ is a set of logical operations on $T_{\Sigma,E}$
- (semantics) $\mathcal{M}(Sp) = \{\mathbf{A} | \mathbf{A} \in Alg_{\Sigma,\Sigma_L} \ \wedge \ \mathbf{A} \cong \mathbf{T}_{\Sigma,\Sigma_L,E}\}$ where

$$\mathbf{T}_{\Sigma,\Sigma_L,E} = (T_{\Sigma,E}, \Sigma^{T_{\Sigma,E}}, \Sigma_L^{T_{\Sigma,E}})$$

Theorem $\mathbf{T}_{\Sigma,\Sigma_L,E}$ is an initial algebra in $\mathcal{M}(Sp)$.

The Keeping-up Program

- Z. Manna, A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992.

local x, y: integer where $x = y = 0$

$$P_1 :: \left[\begin{array}{l} l_0 : \ loop\ forever\ do \\ \left[\begin{array}{l} l_1 : \ await\ x < y + 1 \\ l_2 : \ x := x + 1 \end{array} \right] \end{array} \right] \quad \| \quad P_2 :: \left[\begin{array}{l} m_0 : \ loop\ forever\ do \\ \left[\begin{array}{l} m_1 : \ await\ y < x + 1 \\ m_2 : \ y := y + 1 \end{array} \right] \end{array} \right]$$

Global safety property: $\Box(|x - y| \leq 1)$

Specification of Keeping-up (I)

LSpec Keeping-up

sorts: nat

$vect(2)$

$bool$

opns: $Zero : nat$

$True, False : bool$

$Succ : nat \rightarrow nat$

$Conv : bool \rightarrow nat$

$Leq : nat\,nat \rightarrow bool$

$Add : nat\,nat \rightarrow nat$

$Trans : vect(2) \rightarrow vect(2)$

lopns: $GlobalSafety : vect(2)$

Specification of Keeping-up (II)

eqns:

$$Conv(False) = 0$$
$$Conv(True) = 1$$
$$Add(x, Zero) = x$$
$$Add(x, Succ(y)) = Succ(Add(x, y))$$
$$Leq(Zero, x) = True$$
$$Leq(Succ(x), Zero) = False$$
$$Leq(Succ(x), Succ(y)) = Leq(x, y)$$
$$Trans((x, y)) = (Add(x, Conv(Leq(x, y))), y)$$
$$Trans((x, y)) = (x, Add(y, Conv(Leq(y, x))))$$

leqns:

$$GlobalSafety_Q([(x, x]_Q) = 1$$
$$GlobalSafety_Q([(x, Succ(x)]_Q) = 1$$
$$GlobalSafety_Q([(Succ(x), x]_Q) = 1$$

Abstraction of Keeping-up

$$Succ(x) - Succ(y) = x - y$$

Abs of Keeping-up

vars: $x, y : nat$

abs: $[(Succ(x), Succ(y))]_Q = [(x, y)]_Q$

type: $\forall\forall$

Equivalence classes:

- $[[(Zero, Zero)]_Q]$
- $[[(Succ(Zero), Zero)]_Q]$
- $[[(Zero, Succ(Zero))]_Q]$

The Bakery Algorithm

- L. Lamport. *A New Solution of the Dijkstra's Concurrent Problem*, Communications of the ACM 17, 1974, 453–455.

local x, y: integer where $x = y = 0$

$$P_1 :: \left[\begin{array}{l} 1: \; x := y+1; \\ 2: \; loop\ forever\ while \\ \qquad y \neq 0 \;\wedge\; x > y; \\ 3: \; critical\ section; \\ 4: \; x := 0; \end{array} \right] \quad \| \quad P_2 :: \left[\begin{array}{l} 1: \; y := x+1; \\ 2: \; loop\ forever\ while \\ \qquad x \neq 0 \;\wedge\; y \geq x; \\ 3: \; critical\ section; \\ 4: \; y := 0; \end{array} \right]$$

Safety property:

$$(\forall (x, x', y, y', z) \text{ reachable})(\neg CriticalSection(x, x', y, y', z))$$

Specification of Bakery (I)

LSpec Bakery

sorts: nat

$vect(5)$

opns: $Succ : nat \to nat$

$Trans : vect(5) \to vect(5)$

lopns: $CriticalSection : vect(5)$

vars: $x, x', y, y', z : nat$

Specification of Bakery (II)

eqns:

$$Trans((0,0,0,0,0)) = (1,1,1,1,0)$$
$$Trans((1,1,1,1,0)) = (1,2,1,1,0)$$
$$Trans((1,2,1,1,0)) = (0,0,1,1,2)$$
$$Trans((0,0,y,y',z)) = (Succ(y),1,y,y',1)$$
$$Trans((x,x',0,0,z)) = (x,x',Succ(x),1,2)$$
$$Trans((x,1,0,0,1)) = (x,2,0,0,1)$$
$$Trans((x,1,y,1,2)) = (x,2,y,1,2)$$
$$Trans((x,2,0,0,1)) = (0,0,0,0,0)$$
$$Trans((x,2,y,1,z)) = (0,0,y,1,2)$$
$$Trans((0,0,y,1,2)) = (0,0,y,2,2)$$
$$Trans((x,1,y,1,1)) = (x,1,y,2,1)$$
$$Trans((0,0,y,2,2)) = (0,0,0,0,0)$$
$$Trans((x,1,y,2,1)) = (x,1,0,0,1)$$

leqns:

$$CriticalSection_Q([(x,2,y,2,z)]_Q)$$

Abstraction of Bakery

Abs of Bakery

vars: $x_1, x'_1, x_2, x'_2, y_1, y'_1, y_2, y'_2 \ : \ nat$

abs: $[(x_1, x'_1, y_1, y'_1, 0)]_Q = [(x_2, x'_2, y_2, y'_2, 0)]_Q$

$[(x_1, x'_1, y_1, y'_1, 1)]_Q = [(x_2, x'_2, y_2, y'_2, 1)]_Q$

$[(x_1, x'_1, y_1, y'_1, 2)]_Q = [(x_2, x'_2, y_2, y'_2, 2)]_Q$

type: $\forall\forall$

Equivalence classes:

$$[[(1,1,0,0,1)]_Q]$$

$$[[(0,0,1,1,2)]_Q]$$

$$[[(0,0,0,0,0)]_Q] = \{[(0,0,0,0,0)]_Q, [(1,1,1,1,0)]_Q, [(1,1,2,1,0)]_Q\}$$

Conclusions

What we have done:

- general formalism for abstraction of (abstract) data types
- classification of abstractions w.r.t. the property preservation they assure
- equationally specified abstractions in the context of equationally specified abstract data types

What remains to be done:

- extensions to temporal logics
- overloading, ordered sorts, hidden sorts etc.

Specification of $A = (\mathbf{N}, +^A)$

LSpec Nat

sorts: nat

opns: $Zero : nat$

$Succ : nat \to nat$

$Add : nat\, nat \to nat$

vars: $x, y : nat$

eqns: $Add(x, Zero) = x$

$Add(x, Succ(y)) = Succ(Add(x, y))$

$\langle Back$

Specification of $A = (\mathbf{N}, +^A)$

LSpec Nat

	sorts:	nat
	opns:	$Zero \,:\, nat$
		$Succ \,:\, nat \rightarrow nat$
		$Add \,:\, nat\, nat \rightarrow nat$
$\longrightarrow$	**lopns:**	$Isgrz \,:\, nat$
	vars:	$x, y \,:\, nat$
	eqns:	$Add(x, Zero) = x$
		$Add(x, Succ(y)) = Succ(Add(x, y))$
$\longrightarrow$	**leqns:**	$Isgrz_Q([Succ(x)]_Q) = 1$

$\langle Back|$

Specification of $A/_\rho = (\mathbf{N}/_\rho, +^{A/_\rho})$

LSpec Nat

sorts: nat

opns: $Zero : nat$

$Succ : nat \rightarrow nat$

$Add : nat\, nat \rightarrow nat$

lopns: $Isgrz : nat$

vars: $x, y : nat$

eqns: $Add(x, Zero) = x$

$Add(x, Succ(y)) = Succ(Add(x, y))$

leqns: $Isgrz_Q([Succ(x)]_Q) = 1$

Abs of Nat

vars: $x : nat$

abs: $[Succ(Succ(x))]_Q = [Succ(Zero)]_Q$

type: $\forall\forall$

$\langle Back|$

Kleene's 3-valued Interpretation

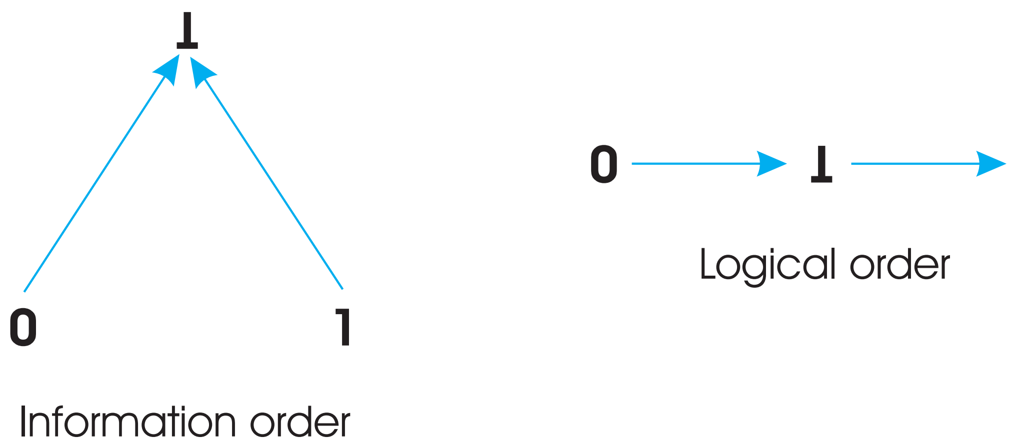
$[0] = \{0\}$ and $[1] = \{1, 2, \ldots\}$

$=^{A/\rho}$	$[0]$	$[1]$
$[0]$	1	0
$[1]$	0	$\perp$

$=^{A/\rho}([1], [1])$ is evaluated to $\perp$ because two arbitrary numbers in $[1]$ can be equal or different.

$\langle Back$

Kleene's 3-valued Interpretation



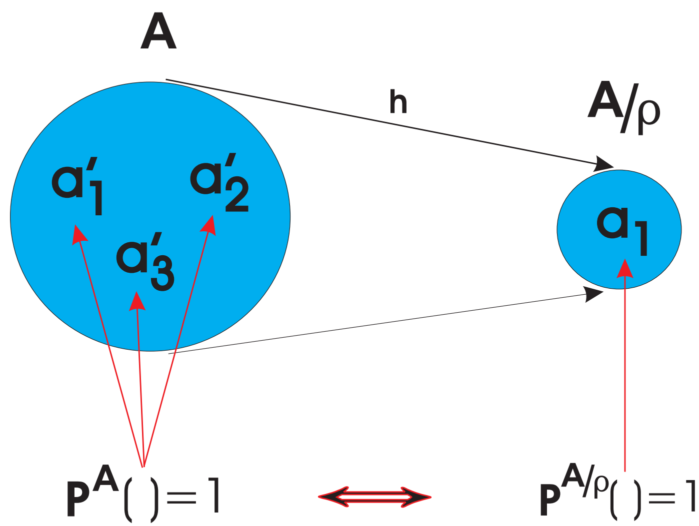
$\vee$	0	1	$\bot$
0	0	1	$\bot$
1	1	1	1
$\bot$	$\bot$	1	$\bot$

$\neg$	
0	1
1	0
$\bot$	$\bot$

$\wedge$	0	1	$\bot$
0	0	0	0
1	0	1	$\bot$
$\bot$	0	$\bot$	$\bot$

$\langle Back|$

∀∃-abstractions



- $p^{A/\rho}([a_1], \ldots, [a_n]) = 1$ if $(\forall i)(\forall a'_i \in [a_i])(p^A(a'_1, \ldots, a'_n) = 1)$
- $p^{A/\rho}([a_1], \ldots, [a_n]) = 0$ if $(\forall i)(\exists a'_i \in [a_i])(p^A(a'_1, \ldots, a'_n) = 0)$
- $p^{A/\rho}([a_1], \ldots, [a_n]) = \perp$, otherwise.

$\langle Back|$

Applications: Shape Analysis

Shape Analysis is a Data Flow Analysis technique mainly used for complex analysis of dynamically allocated data structures

- F. Nielson, H.R. Nielson, Ch. Hankin. *Principles of Program Analysis*, Springer-Verlag, 1999.

It is based on:

- "observing" the shape of these structures
- extracting a finite characterization of them in the form of a shape graph

The shape graph is an abstraction of the behavior of the original data type. The analysis goes on by using corresponding preservation results.

$\langle Back$

Applications: Shape Analysis

Example: original data type of acyclic lists (Sagiv, Reps, Wilhelm, 2002)

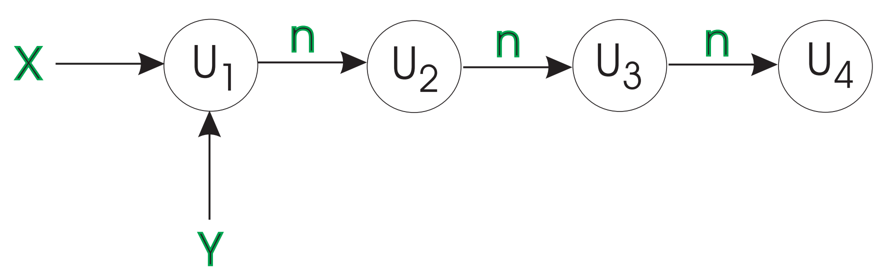
	x	y	t	e
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

$\leftarrow$ 2-valued logic

x, y, t and e are unary predicates

n is a binary predicate



⟨*Back*

Applications: Shape Analysis

Example: abstract data type of acyclic lists (the abstraction is driven by x, y, t and e)

	x	y	t	e
u_1	1	1	0	0
u_{234}	0	0	0	0

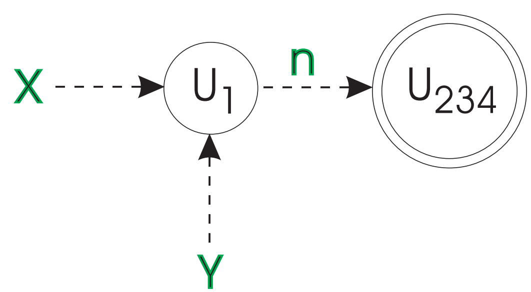
n	u_1	u_{234}
u_1	0	$\perp$
u_{234}	0	$\perp$

$\leftarrow$ 3-valued logic

$\leftarrow$ $\forall\forall$

x, y, t and e are unary predicates

n is a binary predicate



$\langle Back$

Applications: Shape Analysis

An embedding from S into S' is any surjective function $f : U^S \to U^{S'}$ such that

$$\mathcal{I}^S(p)(u_1, \ldots, u_k) \sqsubseteq \mathcal{I}^{S'}(p)(f(u_1), \ldots, f(u_k)),$$

for any any predicate symbol p of arity k and all $u_1, \ldots, u_k \in U^S$.

Theorem (Embedding Theorem)

Let $S = (U^S, \mathcal{I}^S)$ and $S' = (U^{S'}, \mathcal{I}^{S'})$ be two structures, and f be an embedding from S into S'. Then, for every formula φ and every complete assignment γ for φ, $\mathcal{I}^S(\varphi)(\gamma) \sqsubseteq \mathcal{I}^{S'}(\varphi)(f \circ \gamma)$.

The embedding theorem is a particular case of our theorem regarding property preservation by $\forall\forall$-abstractions

$\langle Back|$

Applications: Duplicating Predicate Symbols

- E. Clarke, O. Grumberg, D.E. Long, 1994
- D. Dams, R. Gerth, O. Grumberg, 1997
- M. Bidoit, A. Boisseau, 2001

Basics:

- associate copies to predicate symbols, $P_\oplus$ and $P_\ominus$
- derive two versions of each formula, $\varphi_\oplus$ and $\varphi_\ominus$
  - $P(t_1,\ldots,t_n)_\oplus = P_\oplus(t_1,\ldots,t_n)$ and similar for $\ominus$
  - $(\varphi_1 \vee \varphi_2)_\oplus = ({\varphi_1}_\oplus \vee {\varphi_2}_\oplus)$ and similar for $\ominus$ and the other operators except for $\neg$
  - $(\neg\varphi)_\oplus = \neg(\varphi_\ominus)$ and $(\neg\varphi)_\ominus = \neg(\varphi_\oplus)$
- use $\varphi_\oplus$ for validation and $\varphi_\ominus$ for refutation

$\langle Back$

Applications: Duplicating Predicate Symbols

M. Bidoit and A. Boisseau (2001) use an universal algebra formalism to model security protocols and the technique of duplicating predicate symbols to verify security properties:

- messages = terms in a term algebra
- message exchanges = equations and formulas in a first order logic with equality
- states and reachability relation
- secrecy property: S's private key ($k^{-1}(S)$) remains secret

$$(\forall q_0, q : \ State)(\neg(q_0.I \models k^{-1}(S)) \wedge Reach(q_0, q) \ \Rightarrow \ \neg(q \models k^{-1}(S)))$$

$\langle Back$

Applications: Duplicating Predicate Symbols

The abstraction technique:

- abstractions are driven by epimorphisms $\mathbf{A} \xrightarrow{h} \mathbf{A}^h$
- $P_{\oplus}^{A^h}(b_1, \ldots, b_n)$ iff $(\forall i)(\forall a_i \in h^{-1}(b_i))(P^A(a_1, \ldots, a_n))$
- $P_{\ominus}^{A^h}(b_1, \ldots, b_n)$ iff $(\forall i)(\exists a_i \in h^{-1}(b_i))(P^A(a_1, \ldots, a_n))$

Now, one of the main results proved by Bidoit and Boisseau states that:

$$\mathbf{A}^h \models \varphi_{\oplus} \;\Rightarrow\; \mathbf{A} \models \varphi$$

and

$$\mathbf{A}^h \not\models \varphi_{\ominus} \;\Rightarrow\; \mathbf{A} \not\models \varphi.$$

$\langle Back$

Applications: Duplicating Predicate Symbols

In our approach we associate to each predicate P a new copy P' and interpret it as $\neg P$.

Theorem The following properties holds true:

- if ρ is a $\forall\exists$-abstraction of $\mathbf{A}$, then

$$\mathbf{A}/\rho \models \varphi' \quad \Rightarrow \quad \mathbf{A} \models \varphi$$

- if ρ is an $\exists^{0,1}\forall$-abstraction of $\mathbf{A}$, then

$$\mathbf{A}/\rho \not\models \varphi' \quad \Rightarrow \quad \mathbf{A} \not\models \varphi$$

where φ' is obtained from φ by replacing $\neg P$ by P' and $\neg Q'$ by Q.

$\langle Back|$