Foundations of Programming Languages and Software Engineering

Jan-Georg Smaus (Peter Thiemann)

Universität Freiburg

July 2011

- Abstract Data Types

Abstract Data Types

- We have learned about different datastructures, e.g. for dictionaries:
  - Search trees
  - Lists
  - Tables with hashing
- Implementations of these concepts may have different characteristics:
  - Memory usage
  - Efficiency
- Implementations should be exchangeable
- Abstract over the concepts, use ADTs!
  - Functional specification
  - Implementation independent
  - Different implementations of a single ADT are possible

ADTs are Special Signatures

Definition

Let Σ be a signature.

- A Σ-identity is a pair $(s, t) \in T(\Sigma, X) \times T(\Sigma, X)$. We write a Σ-identity as $s \approx t$ for emphasis.
- An ADT is a pair $(\Sigma, \mathcal{E})$ where
  - Σ is a signature,
  - $\mathcal{E} \subseteq T(\Sigma, X) \times T(\Sigma, X)$ is a set of Σ-identities.

Examples

An ADT for natural numbers

$$\Sigma_{nat} = \{\mathtt{zero}^{(0)}, \mathtt{succ}^{(1)}\}$$
$$\mathcal{E}_{nat} = \emptyset$$

Examples

An ADT for natural numbers

$$\Sigma_{nat} = \{\mathtt{zero}^{(0)}, \mathtt{succ}^{(1)}\}$$
$$\mathcal{E}_{nat} = \emptyset$$

An ADT for integers

$$\begin{aligned}\Sigma_{int} &= \{\mathtt{zero}^{(0)}, \mathtt{pred}^{(1)}, \mathtt{succ}^{(1)}\} \\ \mathcal{E}_{int} &= \{\mathtt{pred}(\mathtt{succ}(x)) \approx x, \\ &\qquad \mathtt{succ}(\mathtt{pred}(x)) \approx x\}\end{aligned}$$

Datatypes are Σ-Algebras

Definition

- An identity $s \approx t$ is valid in a Σ-algebra $\mathcal{A} = (A, \mathcal{J})$ iff $\mathcal{J}_\alpha(s) = \mathcal{J}_\alpha(t)$ for all variable assignments $\alpha : X \to A$.

Datatypes are Σ-Algebras

Definition

- An identity $s \approx t$ is valid in a Σ-algebra $\mathcal{A} = (A, \mathcal{J})$ iff $\mathcal{J}_\alpha(s) = \mathcal{J}_\alpha(t)$ for all variable assignments $\alpha : X \to A$.
- A datatype is a Σ-algebra $\mathcal{D}$.
- A datatype $\mathcal{D}$ implements the ADT $(\Sigma, \mathcal{E})$ iff every identity $s \approx t \in \mathcal{E}$ is valid in $\mathcal{D}$.
  (Note: We shall refine this definition later.)

Implementations of the ADT for Naturals

Implementation 1

- $\mathcal{D}nat' = (\mathbb{N}, \mathcal{J}')$, $\mathcal{J}'(\texttt{zero}) = 0$, $\mathcal{J}'(\texttt{succ})(x) = x + 1$ (note that x is meta-notation!).
- No identities, so all are valid.
- The function $\mathcal{J}'$ is bijective.

Implementations of the ADT for Naturals

Implementation 1

- $\mathcal{D}nat' = (\mathbb{N}, \mathcal{J}')$, $\mathcal{J}'(\texttt{zero}) = 0$, $\mathcal{J}'(\texttt{succ})(x) = x + 1$ (note that x is meta-notation!).
- No identities, so all are valid.
- The function $\mathcal{J}'$ is bijective.

Implementation 2

- $\mathcal{D}nat'' = (\{0, 1, 2, 3\}, \mathcal{J}'')$, $\mathcal{J}''(\texttt{zero}) = 0$, $\mathcal{J}''(\texttt{succ})(x) = (x + 1) \mod 4$.
- No identities, so all are valid.
- The function $\mathcal{J}''$ is not injective (but surjective). $\mathcal{J}''(\texttt{zero}) = 0 = \mathcal{J}''(\texttt{succ}(\texttt{succ}(\texttt{succ}(\texttt{succ}(\texttt{zero})))))$

Implementations of the ADT for Integers (1)

Implementation 1

- $\mathcal{D}int' = (\mathbb{Z}, \mathcal{J}')$, $\begin{aligned} \mathcal{J}'(\texttt{zero})() &= 0 \\ \mathcal{J}'(\texttt{succ})(x) &= x + 1 \\ \mathcal{J}'(\texttt{pred})(x) &= x - 1 \end{aligned}$
- For arbitrary $\alpha : \{x\} \to \mathbb{Z}$ we have
  $\mathcal{J}'_\alpha(\texttt{pred}(\texttt{succ}(x))) = (\alpha(x) + 1) - 1 = \mathcal{J}'_\alpha(x)$
  $\mathcal{J}'_\alpha(\texttt{succ}(\texttt{pred}(x))) = (\alpha(x) - 1) + 1 = \mathcal{J}'_\alpha(x)$
- $\mathcal{J}'$ is surjective but not injective. Consider $\mathcal{J}'(\texttt{zero}) = 0 = \mathcal{J}'(\texttt{succ}(\texttt{pred}(\texttt{zero})))$.

Implementations of the ADT for Integers (2)

Implementation 2

- $\mathcal{D}int'' = (\{0, 1, 2, 3\}, \mathcal{J}'')$,
  $\mathcal{J}''(\mathtt{zero})() \quad = 0$
  $\mathcal{J}''(\mathtt{succ})(x) = x + 1 \mod 4$
  $\mathcal{J}''(\mathtt{pred})(x) = x - 1 \mod 4$
- For arbitrary $\alpha : \{x\} \to \mathbb{Z}$ we have
  $\mathcal{J}''_\alpha(\mathtt{pred}(\mathtt{succ}(x))) = \mathcal{J}''_\alpha(x)$
  $\mathcal{J}''_\alpha(\mathtt{succ}(\mathtt{pred}(x))) = \mathcal{J}''_\alpha(x)$
- $\mathcal{J}''$ is surjective but not injective.

Implementations of the ADT for Integers (3)

A Non-implementation

- $\mathcal{D}int''' = (\mathbb{N}, \mathcal{J}''')$
  $$\begin{aligned} \mathcal{J}'''(\mathtt{zero})() &= 0 \\ \mathcal{J}'''(\mathtt{succ})(x) &= x + 1 \\ \mathcal{J}'''(\mathtt{pred})(x) &= \begin{cases} x - 1 & x > 0 \\ 0 & x = 0 \end{cases} \end{aligned}$$
- Not an implementation:
  For $\alpha : X \to \mathbb{N}$ with $\alpha(x) = 0$ we have
  $$\mathcal{J}_\alpha(\mathtt{succ}(\mathtt{pred}(x))) = 1 \neq 0 = \mathcal{J}_\alpha(x)$$

Fixing the Problems

- Want to rule out implementations such as $\mathcal{D}nat''$ and $\mathcal{D}int''$
- Definition of "implementation" is too weak
- Needed: restriction on function $\mathcal{J}$
  - $\mathcal{J}$ is not necessarily injective (see $\mathcal{D}int'$)
  - Idea: $\mathcal{J}$ must be injective on the equivalence classes induced by the identities of an ADT.
    In other words: $\mathcal{J}$ should make those terms equal that are equal according to the identities of the ADT, but not more!

Equivalence Classes

Definition

Suppose R is an equivalence relation on some set M.

- The set $[x]_R := \{y \in M \mid x \, R \, y\}$ is called the equivalence class of x.
- $y \in [x]_R$ is called a representative of $[x]_R$.
- The quotient of M with respect to R is the set of equivalence classes induced by R, written $M/_R := \{[x]_R \mid x \in M\}$.

Note: For equivalence classes $[x]_R$ and $[y]_R$ we have either $[x]_R = [y]_R$ or $[x]_R \cap [y]_R = \emptyset$.

Congruence Relations

Definition

Suppose Σ is a signature and let R be an equivalence relation on $T(\Sigma, X)$.

- R is a congruence relation iff R is closed under Σ-operations, i.e. $t_i \; R \; t_i'$ implies $f(t_1, \ldots, t_i, \ldots, t_n) \; R \; f(t_1, \ldots, t_i', \ldots, t_n)$ for any $n \geq 0$, $f \in \Sigma^{(n)}$, and $t_1, \ldots, t_n, t_i' \in T(\Sigma, X)$.

Syntactic Quotient Algebras

Definition

Let Σ be a signature and R be a congruence on $T(\Sigma, X)$. For all $n \geq 0$, $f \in \Sigma^{(n)}$, and $t_1, \ldots, t_n \in T(\Sigma, X)$, define $\mathcal{J}^R$ as follows:

$$\mathcal{J}^R(f)([t_1]_R, \ldots, [t_n]_R) = [f(t_1, \ldots, t_n)]_R$$

Note

- $(T(\Sigma, X)/_R, \mathcal{J}^R)$ is a Σ-algebra. Its carrier elements are sets of terms.
- The representatives are arbitrary: Let $n \geq 0$, $f \in \Sigma^{(n)}$, and $s_1, t_1, \ldots, s_n, t_n \in T(\Sigma, X)$. If $s_1 \mathrel{R} t_1, \ldots, s_n \mathrel{R} t_n$ then $f(s_1, \ldots, s_n) \mathrel{R} f(t_1, \ldots, t_n)$. Hence, $[f(s_1, \ldots, s_n)]_R = [f(t_1, \ldots, t_n)]_R$, as R is a congruence.

Equational Theory

Definition

Let $(\Sigma, \mathcal{E})$ be an ADT. We define a relation $\approx_{\mathcal{E}}$ on $T(\Sigma, X)$ as the smallest relation such that

- $\approx_{\mathcal{E}}$ is a congruence relation;
- $\approx_{\mathcal{E}}$ contains $\mathcal{E}$, i.e. $s \approx t \in \mathcal{E}$ implies $s \approx_{\mathcal{E}} t$;
- $\approx_{\mathcal{E}}$ is closed under substitutions, i.e. $s \approx_{\mathcal{E}} t$ implies $\sigma(s) \approx_{\mathcal{E}} \sigma(t)$ for any substitution σ and all $s, t \in T(\Sigma, X)$.

Example

Congruence classes of $\approx_{\mathcal{E}_{int}}$

$$\Sigma_{int} = \{\mathtt{zero}^{(0)}, \mathtt{pred}^{(1)}, \mathtt{succ}^{(1)}\}$$

$$\begin{aligned}\mathcal{E}_{int} = \{&\mathtt{pred}(\mathtt{succ}(x)) \approx x,\\ &\mathtt{succ}(\mathtt{pred}(x)) \approx x\}\end{aligned}$$

$$\begin{aligned}[\mathtt{zero}]_{\approx_{\mathcal{E}_{int}}} = \{&\mathtt{zero},\\ &\mathtt{succ}(\mathtt{pred}(\mathtt{zero})),\\ &\mathtt{pred}(\mathtt{succ}(\mathtt{zero})),\\ &\mathtt{succ}(\mathtt{succ}(\mathtt{pred}(\mathtt{pred}(\mathtt{zero})))), \ldots \quad \}\end{aligned}$$

Revised Definition for ADT Implementations

Definition

A datatype $\mathcal{D} = (M, \mathcal{J})$ implements ADT $(\Sigma, \mathcal{E})$ with constructors $\Gamma \subseteq \Sigma$ if

- $(M, \mathcal{J})$ is a Σ-algebra (as before)
- All identities from $\mathcal{E}$ are valid in M (as before)
- For all $s, t \in T(\Gamma, \emptyset)$: $s \approx_{\mathcal{E}} t$ iff $\mathcal{J}(s) = \mathcal{J}(t)$ (new!)

Syntactic Implementations of ADTs

Theorem

Let $(\Sigma, \mathcal{E})$ be an ADT with constructors $\Gamma \subseteq \Sigma$. Then $\mathcal{D} = (T(\Sigma, \emptyset)/_{\approx_{\mathcal{E}}}, \mathcal{J}^{\approx_{\mathcal{E}}})$ is an implementation of $(\Sigma, \mathcal{E})$.

Proof. Omitted

Example

Nat as a constructor-based ADT (CADT)

CADT: $\Sigma = \{\mathtt{zero}, \mathtt{succ}\}, \mathcal{E} = \{\}, \Gamma = \Sigma$
Implementation: $(\mathbb{N}, \mathcal{J}_1)$ with $\mathcal{J}_1(\mathtt{zero})() = 0$ and $\mathcal{J}_1(\mathtt{succ})(x) = x + 1$

- $(\mathbb{N}, \mathcal{J}_1)$ is Σ-algebra
- No identities to check
- Since $\mathcal{E} = \emptyset$, $\approx_{\mathcal{E}}$ is $=$. Suppose $s, t \in T(\Gamma, \emptyset)$.
    - If $s = t$ then $\mathcal{J}_1(s) = \mathcal{J}_1(t)$
    - Suppose $s \neq t$. Then $s = \mathtt{succ}^n(t)$ with $n > 0$. Hence, $\mathcal{J}_1(s) = \mathcal{J}_1(t) + n \neq \mathcal{J}_1(t)$.

Example

$\mathcal{D}int''$ is not an implementation of the natural numbers

CADT: $\Sigma = \{\texttt{zero}, \texttt{succ}\}, \mathcal{E} = \{\}, \Gamma = \Sigma$
$(\{0, 1, 2, 3\}, \mathcal{J}'')$ with $\mathcal{J}''(\texttt{zero})() = 0$,
$\mathcal{J}''(\texttt{succ})(x) = (x + 1) \mod 4$ is not an implementation.

- $(\{0, 1, 2, 3\}, \mathcal{J}'')$ is Σ-algebra
- No identities to check
- Since $\mathcal{E} = \emptyset$, $\approx_{\mathcal{E}}$ is $=$.
  We have $\texttt{zero} \neq \texttt{succ}^4(\texttt{zero})$ but
  $\mathcal{J}''(\texttt{zero}) = 0 = \mathcal{J}''(\texttt{succ}^4(\texttt{zero}))$.

Example

Alternative implementation of the natural numbers

CADT: $\Sigma = \{\texttt{zero}, \texttt{succ}\}, \mathcal{E} = \{\}, \Gamma = \Sigma$
Implementation: $(\{a\}^*, \mathcal{J}''')$ with $\mathcal{J}'''(\texttt{zero})() = \epsilon$,
$\mathcal{J}'''(\texttt{succ})(w) = aw$

- $(\{a\}^*, \mathcal{J}''')$ is Σ-algebra
- No identities to check
- Since $\mathcal{E} = \emptyset$, $\approx_{\mathcal{E}}$ is $=$.
  - If $s = t$ then $\mathcal{J}'''(s) = \mathcal{J}'''(t)$
  - Suppose $s \neq t$. Then $s = \texttt{succ}^n(t)$ with $n > 0$. Hence, $\mathcal{J}'''(s) = \mathcal{J}'''(t)\underbrace{a \ldots a}_{n} \neq \mathcal{J}'''(t)$.

Equivalence Classes for Terms Representing Integers

Suppose $\Gamma = \Sigma = \{\mathtt{zero}, \mathtt{succ}, \mathtt{pred}\}$,
$\mathcal{E} = \{\mathtt{succ}(\mathtt{pred}(x)) = x, \mathtt{pred}(\mathtt{succ}(x)) = x\}$
Question: What is $T(\Sigma, \emptyset)/_{\approx_{\mathcal{E}}}$?
Answer: Give a representative for every equivalence class.

Lemma

For every term $t \in T(\Sigma, \emptyset)$, exactly one of the following propositions holds

- A There exists $n > 0$ such that $t \in [\mathtt{succ}^n(\mathtt{zero})]_{\approx_{\mathcal{E}}}$.
- B $t \in [\mathtt{zero}]_{\approx_{\mathcal{E}}}$.
- C There exists $n > 0$ such that $t \in [\mathtt{pred}^n(\mathtt{zero})]_{\approx_{\mathcal{E}}}$.

Proof

The proof is by term induction over t.

- Base case: $t = \texttt{zero}$. Then B holds.

Proof (cont.)

- Induction Step for $t = \texttt{succ}(t')$. By the IH, one of the following holds for t':
  - A $t' \approx_{\mathcal{E}} \texttt{succ}^{(n)}(\texttt{zero})$ for $n > 0$: then $\texttt{succ}(t') \approx_{\mathcal{E}} \texttt{succ}(\texttt{succ}^{(n)}(\texttt{zero})) = \texttt{succ}^{(n+1)}(\texttt{zero})$. Since $n + 1 > 0$ we have case A.
  - B $t' \approx_{\mathcal{E}} \texttt{zero}$: then $\texttt{succ}(t') \approx_{\mathcal{E}} \texttt{succ}(\texttt{zero})$. We have case A with $n = 1$.
  - C $t' \approx_{\mathcal{E}} \texttt{pred}^{(n)}(\texttt{zero})$ for $n > 0$: Then $\texttt{succ}(t') \approx_{\mathcal{E}} \texttt{succ}(\texttt{pred}^{(n)}(\texttt{zero}))$.
    If $n = 1$ then $\texttt{succ}(\texttt{pred}(\texttt{zero})) \approx_{\mathcal{E}} \texttt{zero}$ so case B holds.
    If $n > 1$ then
    $\texttt{succ}(\texttt{pred}(\texttt{pred}^{(n-1)}(\texttt{zero}))) \approx_{\mathcal{E}} \texttt{pred}^{(n-1)}(\texttt{zero})$, so case C holds.

Proof (cont.)

- Induction step for $\texttt{pred}(t)$ analogous.

Equivalence Classes for Terms Representing Integers

Lemma

Suppose $n > 0, m > 0$. Then we have

- $\mathtt{zero} \not\approx_{\mathcal{E}} \mathtt{succ}^n(\mathtt{zero})$,
- $\mathtt{zero} \not\approx_{\mathcal{E}} \mathtt{pred}^n(\mathtt{zero})$,
- $\mathtt{succ}^n(\mathtt{zero}) \not\approx_{\mathcal{E}} \mathtt{pred}^m(\mathtt{zero})$,
- $\mathtt{succ}^n(\mathtt{zero}) \not\approx_{\mathcal{E}} \mathtt{succ}^m(\mathtt{zero})$ provided $n \neq m$, and
- $\mathtt{pred}^n(\mathtt{zero}) \not\approx_{\mathcal{E}} \mathtt{pred}^m(\mathtt{zero})$ provided $n \neq m$.

If follows that

$$\{\mathtt{succ}^n(\mathtt{zero}) | n > 0\} \cup \{\mathtt{zero}\} \cup \{\mathtt{pred}^n(\mathtt{zero}) | n > 0\}$$

is a set of representatives for $T(\Sigma, \emptyset)/_{\approx_{\mathcal{E}}}$.

Example

Integers as a CADT

CADT: $\Gamma = \Sigma = \{\mathtt{zero}, \mathtt{succ}, \mathtt{pred}\}$,
$\mathcal{E} = \{\mathtt{succ}(\mathtt{pred}(x)) = x, \mathtt{pred}(\mathtt{succ}(x)) = x\}$
Implementation: $(\mathbb{Z}, \mathcal{J})$ with
$\mathcal{J}(z) = 0, \mathcal{J}(\mathtt{succ})(x) = x + 1, \mathcal{J}(\mathtt{pred})(x) = x - 1$

- $(\mathbb{Z}, \mathcal{J})$ is a Σ-algebra
- All identities are valid (as seen before)
- An easy term induction shows for all $t \in T(\Sigma, \emptyset)$ that
  - if $t \approx_{\mathcal{E}} \mathtt{zero}$ then $\mathcal{J}(t) = 0$,
  - if $t \approx_{\mathcal{E}} \mathtt{succ}^n(\mathtt{zero})$ then $\mathcal{J}(t) = n$, and
  - if $t \approx_{\mathcal{E}} \mathtt{pred}^n(\mathtt{zero})$ then $\mathcal{J}(t) = -n$.

  Hence, if $s \approx_{\mathcal{E}} t$ then $\mathcal{J}(s) = \mathcal{J}(t)$.
  Conversely, if $s \not\approx_{\mathcal{E}} t$ then $\mathcal{J}(s) \neq \mathcal{J}(t)$ because $\mathcal{J}$ maps different representatives to different integers.

Summary

Slogan

Calculating with ADT $=$
applying term operations $+$
determining set of representatives.