Foundations of Programming Languages and Software Engineering

Universität Freiburg

June 19, 2012

- 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

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

Implementation 2

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

Implementations of the ADT for Integers (1)

Implementation 1

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

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

- $\mathcal{D}int''' = (\mathbb{N}, \mathcal{J}''')$

$$\begin{aligned}
\mathcal{J}'''(\texttt{zero})() &= 0 \\
\mathcal{J}'''(\texttt{succ})(x) &= x + 1 \\
\mathcal{J}'''(\texttt{pred})(x) &= \begin{cases} x - 1 & x > 0 \\ 0 & x = 0 \end{cases}
\end{aligned}$$

- Not an implementation:
  For $\alpha : X \to \mathbb{N}$ with $\alpha(x) = 0$ we have

$$\mathcal{J}_\alpha(\texttt{succ}(\texttt{pred}(x))) = 1 \neq 0 = \mathcal{J}_\alpha(x)$$

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 $s_1 \; R \; t_1, \ldots, s_n \; R \; t_n$ then $f(s_1, \ldots, s_n) \; R \; f(t_1, \ldots, t_n)$. 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 $\approx_{\mathcal{E}_{int}}$

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

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

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!)

Example

Nat as a constructor-based ADT (CADT)

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

- $(\mathbb{N}, \mathcal{J}_1)$ 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 $s = \texttt{succ}^n(t)$ 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: $\Sigma = \{\mathtt{zero}, \mathtt{succ}\}, \mathcal{E} = \{\}, \Gamma = \Sigma$
$(\{0, 1, 2, 3\}, \mathcal{J}'')$ with $\mathcal{J}''(\mathtt{zero})() = 0$,
$\mathcal{J}''(\mathtt{succ})(x) = (x + 1) \mod 4$ 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 $\mathtt{zero} \neq \mathtt{succ}^4(\mathtt{zero})$ but
  $\mathcal{J}''(\mathtt{zero}) = 0 = \mathcal{J}''(\mathtt{succ}^4(\mathtt{zero}))$.

Example

Alternative implementation of the natural numbers

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

- $(\{a\}^*, \mathcal{J}''')$ 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 $s = \mathtt{succ}^n(t)$ with $n > 0$. Hence, $\mathcal{J}'''(s) = \mathcal{J}'''(t)\underbrace{a \ldots a}_{n} \neq \mathcal{J}'''(t)$.

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

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

Proof (cont.)

- Induction step for `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.

Richer ADTs

Definition (Linear Data Structure)

An ADT is called a linear data structure (LDS) iff there is an implementation with simple lists.

- LDS are aggregates, i.e. one element contains several elements of another sort.
- Types are now parameterized.
  - Examples: List(A), Array(A)
  - A is not a fixed type but rather a type parameter.
  - Compare with Java Generics: `List<A>`, `A[]`

Lists

Definition (Signature for Lists)

data type	List(A)	
operations	`empty`	$: \to \mathsf{List}(A)$
	`cons`	$: A \times \mathsf{List}(A) \to \mathsf{List}(A)$
	`head`	$: \mathsf{List}(A) \to A$
	`tail`	$: \mathsf{List}(A) \to \mathsf{List}(A)$
	`empty?`	$: \mathsf{List}(A) \to \mathsf{Boolean}$
	`app`	$: \mathsf{List}(A) \times \mathsf{List}(A) \to \mathsf{List}(A)$
	`len`	$: \mathsf{List}(A) \to \mathsf{Nat}$

- Definition parameterized over A
- Constructors: `empty`, `cons`
- Definition uses more than one type: More general notion of signature and arity needed

Lists (cont.)

Definition (Identities for Lists)

<u>identities</u>

$$\mathtt{head}(\mathtt{cons}(a, l)) = a$$

$$\mathtt{tail}(\mathtt{cons}(a, l)) = l$$

$$\mathtt{empty?}(\mathtt{empty}) = \mathit{true}$$
$$\mathtt{empty?}(\mathtt{cons}(a, l)) = \mathit{false}$$

$$\mathtt{app}(\mathtt{empty}, v) = v$$
$$\mathtt{app}(\mathtt{cons}(a, l), v) = \mathtt{cons}(a, \mathtt{app}(l, v))$$

$$\mathtt{len}(\mathtt{empty}) = \mathtt{zero}$$
$$\mathtt{len}(\mathtt{cons}(a, l)) = \mathtt{succ}(\mathtt{len}(l))$$

Heterogeneous Signatures

Definition

Let S be a set of sorts. A heterogeneous signature Σ is a set of function symbols where each $f \in \Sigma$ is associated with an arity $s \to s'$ where $s \in S^*$ and $s' \in S$.

Examples

- Arity of `empty`: $\epsilon \to \text{List}(A)$
- Arity of `cons`: $(A, \text{List}(A)) \to \text{List}(A)$

Previous definitions need to be generalized as well:

- An algebra has different carrier sets for every sort.
- Terms must respect the sorts associated with a function symbol to rule out illegal terms such as $\texttt{cons}(1, 1)$.
- Generalize congruence relation

Dealing with Partial Operations

- `head` and `tail` are partial operations:
  - $\texttt{head}(\texttt{empty}) = ??$
  - $\texttt{tail}(\texttt{empty}) = ??$
- One possible solution: Introduce a distinguished element $\perp_s$ for every sort s
  - $\texttt{head}(\texttt{empty}) = \perp_A$
  - $\texttt{tail}(\texttt{empty}) = \perp_{\mathsf{List}(A)}$
- All operations are strict in $\perp_s$, i.e. if one argument is $\perp_s$ the result is $\perp_{s'}$
  - $\texttt{head}(\perp_{\mathsf{List}(A)}) = \perp_A$
  - $\texttt{tail}(\perp_{\mathsf{List}(A)}) = \perp_{\mathsf{List}(A)}$
  - $\texttt{empty?}(\perp_{\mathsf{List}(A)}) = \perp_{\mathsf{Boolean}}$
  - $\texttt{len}(\perp_{\mathsf{List}(A)}) = \perp_{\mathsf{Nat}}$
  - ...

Search for Representatives

Proposition

Let t be a term of type List(A) without variables. Then one of the following holds:

- $t \approx_{\mathcal{E}} t'$ and $t' \in T(\Gamma, \emptyset)$ where $\Gamma = \{\mathtt{empty}, \mathtt{cons}\}$.
- $t \approx_{\mathcal{E}} \bot_{\mathsf{List}(A)}$

Proof. The proof is by induction on t.

- Case $t = \mathtt{empty} \in T(\Gamma, \emptyset)$. Trivial.
- Case $t = \mathtt{cons}(a, s)$ with $s \approx_{\mathcal{E}} s'$ and $s' \in T(\Gamma, \emptyset)$.
  Hence, $\mathtt{cons}(a, s) \approx_{\mathcal{E}} \mathtt{cons}(a, s') \in T(\Gamma, \emptyset)$.
- Case $t = \mathtt{head}(s)$ so t does not have type List(A).
- Case $t = \mathtt{tail}(s)$ with $s \approx_{\mathcal{E}} s'$ and $s' \in T(\Gamma, \emptyset)$.
  If $s' = \mathtt{empty}$ then $t \approx_{\mathcal{E}} \bot_{\mathsf{List}(A)}$.
  If $s' = \mathtt{cons}(a, s'')$ then $t \approx_{\mathcal{E}} s'' \in T(\Gamma, \emptyset)$.

Proof (cont.)

- Case $t = \mathtt{empty?}(s)$ but then t does not have type List(A).
- Case $t = \mathtt{app}(s_1, s_2)$ with $s_1 \approx_{\mathcal{E}} s_1'$ and $s_2 \approx_{\mathcal{E}} s_2'$ and $s_1', s_2' \in T(\Gamma, \emptyset)$.
  - If $s_1' = \mathtt{empty}$ then $t \approx_{\mathcal{E}} \mathtt{app}(\mathtt{empty}, s_2) \approx_{\mathcal{E}} s_2 \approx_{\mathcal{E}} s_2'$.
  - If $s_1' = \mathtt{cons}(a, s_1'')$ then
  $$\begin{aligned} t = \mathtt{app}(s_1, s_2) &\approx_{\mathcal{E}} \mathtt{app}(\mathtt{cons}(a, s_1''), s_2) \\ &\approx_{\mathcal{E}} \mathtt{cons}(a, \mathtt{app}(s_1'', s_2)) \end{aligned}$$
  By the IH, we have $\mathtt{app}(s_1'', s_2) \approx_{\mathcal{E}} s'$ with $s' \in T(\Gamma, \emptyset)$.
- Case $t = \mathtt{len}(s)$ but then t does not have type List(A).

Another Example

Sequences

- Also know as Array or Vector
- Parameterized over type of elements
- Fixed number of elements
- Direct access to elements (constant time)

Arrays

Definition (Signature for Arrays)

data type	Array(A)	
operations	`new`	: Nat $\times$ A $\rightarrow$ Array(A)
	`update`	: Array(A) $\times$ Nat $\times$ A $\rightarrow$ Array(A)
	`get`	: Array(A) $\times$ Nat $\rightarrow$ A
	`len`	: Array(A) $\rightarrow$ Nat

- Functional arrays
- In imperative languages: `update` operation changes the array

Arrays (cont.)

Definition (Identities for Arrays)

<u>identities</u>

(i.1) $\mathtt{get}(\mathtt{new}(n,x),i) = x$ **if** $i < n$

(i.2) $\mathtt{get}(\mathtt{update}(a,i,x),i) = x$ **if** $i < \mathtt{len}(a)$

(i.3) $\mathtt{get}(\mathtt{update}(a,j,x),i) = \mathtt{get}(a,i)$ **if** $i \neq j$

(i.4) $\mathtt{update}(\mathtt{update}(a,i,x),i,y) = \mathtt{update}(a,i,y)$

(i.5) $\mathtt{update}(\mathtt{update}(a,j,x),i,y) = \mathtt{update}(\mathtt{update}(a,i,y),j,x)$ **if** $i \neq j$

(i.6) $\mathtt{update}(\mathtt{new}(n,x),i,x) = \mathtt{new}(n,x)$ **if** $i < n$

(i.7) $\mathtt{len}(\mathtt{new}(n,x)) = n$

(i.8) $\mathtt{len}(\mathtt{update}(a,i,x)) = \mathtt{len}(a)$

New: Conditional identities

Examples

Calculating with representatives

Suppose the carrier set of A is $\{a, b, c\}$.

1. $\mathtt{get}(\mathtt{new}(10, a), 5) \overset{\text{(i.1)}}{\approx_{\mathcal{E}}} a$
2. $\mathtt{get}(\mathtt{new}(10, a), 11) \approx_{\mathcal{E}} \perp_A$
3. $\mathtt{get}(\mathtt{update}(\mathtt{update}(\mathtt{new}(10, a), 5, b), 0, c), 5)$
   $\overset{\text{(i.3)}}{\approx_{\mathcal{E}}} \mathtt{get}(\mathtt{update}(\mathtt{new}(10, a), 5, b), 5) \overset{\text{(i.2)}}{\approx_{\mathcal{E}}} b$
4. $\mathtt{update}(\mathtt{update}(\mathtt{new}(10, a), 5, b), 5, a)$
   $\overset{\text{(i.4)}}{\approx_{\mathcal{E}}} \mathtt{update}(\mathtt{new}(10, a), 5, a) \overset{\text{(i.6)}}{\approx_{\mathcal{E}}} \mathtt{new}(10, a)$
5. $\mathtt{get}(\mathtt{update}(\mathtt{new}(0, a), 1, a), 1) \approx_{\mathcal{E}} \perp_A$
   (i.2) is not applicable because
   $\mathtt{len}(\mathtt{update}(\mathtt{new}(0, a), 1, a)) \approx_{\mathcal{E}} 0$.