CS 225 – Data Structures

ZJUI – Spring 2022

Lecture 1: Introduction: Abstract Data Types

Klaus-Dieter Schewe

ZJU–UIUC Institute, Zhejiang University

International Campus, Haining, UIUC Building, B404

email: kd.schewe@intl.zju.edu.cn

1 *Introduction: Abstract Data Types*

This course is concerned with the question how to represent large collections of data on computers such that required access and update operations can be executed effectively and efficiently

Therefore, all structures we consider are **bulk data structures**

In particular, they are concerned with varying large amounts of data of the same type

In general, "large" includes data volumes that do not fit into main memory or not even into main memory and secondary storage of a single machine

That is, data structures are intrinsically linked to external memory management and also distribution

Tarski Structures

In **MATH 213 / CS 173: Discrete Mathematics / Discrete Structures** you have learnt the general concept of a **structure:**

Definition. A ***structure*** $\mathcal{S}$ over a signature $\Sigma = (\mathcal{P}, \mathcal{F})$ consists of

- a set B called the **domain** (or **base set**) of the structure;
- functions $f_S : B^n \to B$ for all function symbols $f \in \mathcal{F}$ with arity n;
- functions $p_S : B^n \to \{\mathbf{true}, \mathbf{false}\}$ for all function/relation symbols $p \in \mathcal{P}$ with arity n.

Here a signature Σ comprises two sets of symbols: function symbols in $\mathcal{F}$ and predicate (or relation) symbols in $\mathcal{P}$; each symbol has a fixed **arity** $n \in \mathbb{N}$

We can identify functions $p_S : B^n \to \{\mathbf{true}, \mathbf{false}\}$ with n-ary relations

$$\{(b_1, \dots, b_n) \in B^n \mid p_S(b_1, \dots, b_n) = \mathbf{true}\}$$

From Tarski Structures to Data Structures

Such structures in general are called **Tarski structures** or **universal algebras**

- They have been used to define the semantics of predicate logic, databases, states of sequential and recursive algorithms

- They have been tailored to many other discrete structures (posets, lattices, Boolean algebras, (natural) numbers, quotients structures (congruences), graphs, trees, etc.)

When we speak about **data**, we usually refer to the formal representation of **information** in a way that permits manipulation by machines (computers)

Thus, **data structures** refer to the formal representation of structures in computer memory

This includes the need of well-defined syntax and semantics (interpretation)

Finite Structures

Here we are interested only in **finite** structures, i.e. the domain is a finite set

This set may, however, be partitioned into subsets

Decisive for the structures of interest are the functions/relations defined on them:

- For instance, we may assume a partial or total order $\leq$, the latter one leading us to sequence data structures

- Another example (already briefly stressed in Discrete Mathematics) is given by n-ary relations forming the basis for (relational) databases—using relations as data structures

In general, such functions/relations allows us to **access** parts of the data stored in a data structure

Evolving Structures

Furthermore, we are interested in structures that are **evolving**, i.e. the structure is subject to **updates**

This aspect was already briefly stressed in Discrete Mathematics in the definitions of sequential and recursive algorithms

Here the emphasis will be on the elementary operations that are necessary for such updates

Combining the functions for the update of data structures together with the functions/relations defining the structure and enabling access we obtain the notion of **abstract data type** (**ADT**)

Abstract Data Types

To define formally the notion of an abstract data type we require:

- several sets, as we might define operations that produce results in a different set, e.g.
  - the length of a list is a natural number
  - the result of a comparison predicate is a truth value
  - retrieving elements from a sequence requires a set of sequences and a set of the elements of the sequence
- some of these sets may well be used as parameters (with operations on them defined by a different ADT)
- functions on cartesian products of these sets

Abstract Data Types – Definition

Definition. A ***(multi-sorted) abstract data type*** (**ADT**) consists of non-empty sets $S_1, \dots, S_k$ ($k \geq 1$) and operations $op_1, \dots, op_\ell$ ($\ell \geq 1$), each having the form

$$op_i : S^i_{i_0} \times \cdots \times S^i_{n_i} \to S^i$$

with $S^i, S^i_{i_j} \in \{S_1, \dots, S_k\}$.

An ADT provides the abstract description of a data structure enabling its usage without consideration of lower level representation and realisation of the functions and relations

The operations that are to be supported determine the most suitable implementation of an ADT by data structures

Some of the sets S_i may be parameters defined by a different ADT

Example

Let us take lists with elements in a set T—formally, such a list is a function $\ell : \{1, \dots, n\} \to T$ with $n \in \mathbb{N}$

Then the sets required for the definition of an ADT $LIST(T)$ are

- T
- L —the set of lists with elements in T
- $\mathbb{N}$ —the set of natural numbers
- $\mathbb{B}$ —the set of truth values

Then we might be interested in (partial) operations like the following:

- $get : L \times N \to T$, where $get(\ell, i)$ retrieves the i'th element of the list ℓ
- $length : L \to \mathbb{N}$, where $length(\ell)$ returns the number of elements of the list ℓ
- $in : T \times L \to \mathbb{B}$, where $in(x, \ell) = \textbf{true}$ iff x appears in the list ℓ

Example / cont.

Further operations of interest might be:

- $set : L \times \mathbb{N} \times T \to L$, where $set(\ell, i, x)$ is the list obtained from ℓ, in which the i'th element is updated to x
- $concat : L \times L \to L$, where $concat(\ell_1, \ell_2)$ is the list of length $length(\ell_1) + length(\ell_2)$ with elements from ℓ_1 first, followed by those in ℓ_2
- $append : L \times T \to L$ is the special case, where $append(\ell, x)$ is the list of length $length(\ell) + 1$ resulting from adding x at the end of ℓ
- $delete : L \times \mathbb{N} \to L$, where $delete(\ell, i)$ is the list of length $length(\ell) - 1$ resulting from ℓ by removing its i'th element
- $insert : L \times \mathbb{N} \times T \to L$, where $insert(\ell, i, x)$ is the list of length $length(\ell) + 1$ resulting from ℓ by adding a new i'th element x

Example / cont.

Further operations of interest might also be:

- $eq : L \times L \to \mathbb{B}$, where $eq(\ell_1, \ell_2) = \mathbf{true}$ iff the lists ℓ_1 and ℓ_2 contain the same elements in the same order

- $sort : L \to L$, where $sort(\ell)$ is the result of sorting the elements of ℓ according to a total order $\leq$ on T, i.e. we need to have also $\leq: T \times T \to \mathbb{B}$

- $sublist : L \times L \to \mathbb{B}$, where $sublist(\ell_1, \ell_2) = \mathbf{true}$ iff all elements of the list ℓ_1 appear in the same order in ℓ_2, equivalently: ℓ_1 results from ℓ_2 be a sequence of delete operations

We may extend or shorten the list of operations depending on the needs

We will usually drop one of the list arguments in the operations above assuming that this list is given—this is in accordance with the view of object-oriented programming, where methods are associated with objects

Data Structures Realising ADTs

There is a whole branch of Computing Science dealing with the specification and analysis of ADTs, known under the name **algebraic specifications**—here, we will not proceed much further in this direction

Our task is to turn the **abstract** data types into **concrete** (bulk) data structures

We will see that supporting all operations above lead to conflicts:

- Operations such as $get()$ can be optimally supported, if we enable direct access to all list elements
- However, direct access is inflexible and thus not optimal for operastions such as $insert()$ or $delete()$
- Furthermore, operation such as $sort()$ requires different support for reasonable short lists, long lists, or lists that require secondary storage or even remote storage