




часть пятая (1)




ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



СПЕЦИФИКАЦИИ

законченное описание поведения системы,
которую требуется разработать




ПРОЦЕДУРНАЯ АБСТРАКЦИЯ

Процедура – это отображение набора значений входных аргументов в выходной набор результатов с возможной модификацией входных значений. Набор входных или выходных значений или оба этих набора могут быть пусты.




Преимущества абстракции

Абстракция представляет собой некоторый способ отображения. При этом “абстрагируемся” от несущественных подробностей, описывая лишь те, которые имеют непосредственное отношение к решаемой задаче.





Виды абстракций

- Абстракция через параметризацию
 - Абстракция через спецификацию
- 



Абстракция через параметризацию


абстрагируемся от конкретных
используемых данных

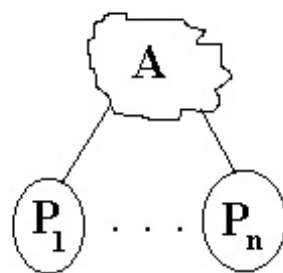




Абстракция через спецификацию


фокусируем внимание на особенностях, от которых зависит пользователь, и абстрагируемся от подробностей реализации этих особенностей







Абстракция

Реализации






Абстракция через спецификацию наделяет
структуру программы двумя
отличительными особенностями.





Первая из этих особенностей заключается в локальности, которая означает, что реализация одной абстракции может быть создана или рассмотрена без необходимости анализа реализации какой-либо другой абстракции.






Второй особенностью является модифицируемость. Если реализация абстракции изменяется, но ее спецификация при этом остается прежней, то эти изменения не повлияют на оставшуюся часть программы.



Спецификация процедурных абстракций

Спецификация процедуры состоит из заголовка и описания функции, выполняемой процедурой.

Заголовок содержит имя процедуры, номер, порядок и типы входных и выходных параметров. Кроме этого, выходные параметры могут, а входные должны быть поименованы.




Пример:


```
remove_dups=proc (a:array[int])  
  sqrt=proc (x:real) returns (rt:real)
```

В общем виде

```
<имя>=proc  
  ([<вх.пар>:<тип>[,<вх.пар>:<тип>...]])  
  returns  
    ([<вых.пар>:<тип>[,<вых.пар>:<тип>...]])
```



Семантическая часть спецификации состоит из трех предложений:

- **requires** (требуется)
 - **modifies** (модифицирует)
 - **effects** (эффекты)
- 


Шаблон спецификации для процедурных абстракций:

Pname = proc (...) returns (...)

requires //этот оператор задает
необходимые требования


modifies //этот оператор идентифицирует
все модифицируемые входные данные

effects //этот оператор описывает
выполняемые функции




Предложение `requires` задает ограничения, накладываемые на абстракцию.


Предложение `requires` необходимо в том случае, если процедура является **частичной**, т.е. ее поведение для некоторых входных значений недетерминировано.




Если же процедура **глобальна**, т.е. ее поведение определено для всех входных значений, то предложение `requires` может быть опущено.




Оператор `modifies` задает список имен входных параметров, модифицируемых процедурой. Если входные параметры не указаны или не модифицируются, то это предложение может быть опущено.





Предложение effects описывает работу
процедуры со значениями, не охваченные
предложением requires.



Например:

`concat=proc (a,b:string) returns (ab: string)`
`effects` по возврату `ab` есть новая строка,
содержащая символы из `a` (в том порядке,
в котором они расположены в `a`), за
которыми следуют символы из `b` (в том
порядке, в котором они расположены в `b`).

Например:

`Remove_dupls=proc (a:array [int])`

`modifies a`

`effects` удаляет из массива `a` все

повторяющиеся элементы. Нижняя граница `a` остается без изменений, однако порядок следования оставшихся элементов может изменяться, если, например, перед вызовом `a=[1:4, 13,6,3,6]`, то по возвращению массив `a` имеет нижнюю границу, равную 1, и содержит три элемента 3, 13 и 6, расположенных в некоторой неопределенной последовательности.


Например:

Search=proc (a:array[int],x:int) returns (i:int)
requires массив a упорядочен по
возрастанию.


effects Если элемент x принадлежит массиву
a, то возвращается значение i такое, что
 $a[i]=x$; в противном случае значение i на
единицу больше, чем значение верхней
границы массива a


Создание процедурных абстракций

Процедура является недоопределенной, если для определенных значений входных параметров на выходе вместо единственного правильного результата имеется набор допустимых результатов. Реализация может ограничивать этот набор только одним значением, однако он может быть любым из числа допустимых.





Неопределенная абстракция может иметь детерминированную реализацию, т.е. такую, которая, будучи вызванной два раза с идентичными входными данными, выполняется одинаково.






Важным свойством процедур является обобщаемость, которая достигается путем использования параметров вместо переменных.





Важной характеристикой процедур является простота. Процедура должна обладать хорошо определенным и легко объяснимым назначением, независимым от контекста ее использования.







АБСТРАКЦИЯ ДАННЫХ


Дает возможность добавлять в базовый уровень новые типы данных.







Типы данных должны включать в себя абстракции как через параметризацию, так и через спецификацию.







Абстракции через параметризацию могут
быть осуществлены точно также, как и для
процедур, - использованием параметров
там, где это имеет смысл






Абстракции через спецификацию
достигаются за счет того, что операции
представляются как часть типа





абстракция данных = <объекты, операции>.



Спецификации для абстракций данных

Dname = data type is // список операций

Descriptions

**// Здесь приводится описание абстракций
данных**

Operations

**//Здесь задаются спецификации для всех
операций.**

End dname



Пример:

Спецификация абстракции данных intset



Пример:

Наборы целых чисел `intset` - это неограниченные множества целых чисел с операциями создания нового, пустого набора, проверки данного целого числа на принадлежность данному набору `intset` и добавления или удаления элементов. В секции описания, описываются наборы целых чисел в терминах математических наборов. Отмечаем, что наборы целых чисел - изменяемые, и перечисляем все изменяемые операции.

S_{post} - значение S при возврате.

Пример:

Intset=**data type** is create, insert, delete, member, size, choose

Descriptions

Наборы целых чисел intset - это неограниченных математические множества целых чисел. Наборы целых чисел изменяемые: операции insert и delete добавляют и удаляют целые числа из набора.

Operations

Пример:

create = **proc()** **returns** (intset)

effects Возвращает новый, пустой набор intset

insert = **proc** (s:intset, x:intset)

modifies s

effects Добавляет x к элементам s; после добавления – возврат, $S_{\text{post}} = S \cup \{x\}$,
где S_{post} – это набор значений в s при возврате из insert.

Delete = **proc** (s: intset, x: int)

modifies s

effects Удаляет x из s (т.е. $S_{\text{post}} = S - \{x\}$).

Member = **proc** (s:intset, x: int) **returns** (bool)

effects Возвращает значение true, если $x \in S$

size = **proc** (s:intset) **returns** (int)

effects Возвращает число элементов в s

choose = **proc** (s: intset) **returns** (int)

requires набор s не пуст

effects возвращает произвольный элемент s.

End intset

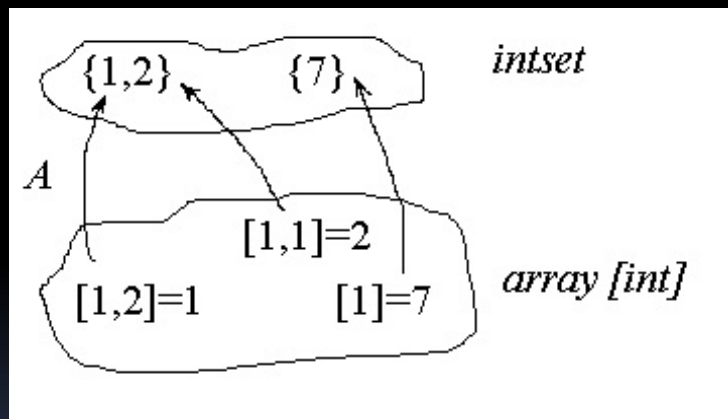
Функция абстракции

Функция абстракции отображает объекты представления в абстрактные объекты:

$$A : \text{rep} \rightarrow \bar{A}$$

Здесь \bar{A} обозначает набор абстрактных объектов. Для каждого объекта представления r , $A(r)$ является абстрактным объектом $a \in \bar{A}$, который представляет r


Например, функция абстракции реализации
`intset` отображает массив `array [int]` в набор
`intset`





Функция абстракции


важнейшая информация о реализации. Она определяет конкретное представление, т.е. то, каким образом объекты представления реализуют абстрактные объекты




Пример:

// Типичный стек - это последовательность $[e_1, \dots, e_n]$, где e_n - элемент со старшим индексом

//Функция абстракции есть
 $A(r)=[r[\text{low}(r)], \dots, r[\text{high}(r)]]$



//A(r)=[r[high(r)],...,r[low(r)]]



Инвариант представления

Условие, которому удовлетворяют все законные объекты, называется инвариантом представления. Инвариант представления I есть предикат.

$I: \text{rep} \rightarrow \text{boolean}$

который принимает значение true для законных объектов представления.

Например:

для `intset` мы можем делать следующий инвариант представления:

//Инвариант есть для всех целых i, j , таких, что $\text{low}(r) \leq i < j \leq \text{high}(r)$

// $r[i] \neq r[j]$



//Инвариант представления есть true



Параметризованные абстракции данных

Set = **data type** [t:type] **is** set, insert, ~set, member, size, choose, del

requires t имеет операцию equal : proctype (t,t) returns (bool), т.е. условие равенства t.

Description

Общие наборы set - неограниченные математические наборы. Эти наборы изменяемые: операции insert и del добавляют и уничтожают элементы набора.

Operations

set = **proc()** **returns** (set[t])

effects Возвращает новый пустой набор

insert = **proc**(s:set[t],x:t)


modifies s

effects Добавляет x к элементам s; после insert
Возвращается $S_{\text{post}} = S \cup \{x\}$



Изменяемость

Абстракции данных либо изменяемы (с объектами, значения которых могут изменяться), либо неизменяемы




Классы операций

- **Примитивные конструкторы**. Эти операции создают объекты соответствующего им типа, не используя никаких объектов в качестве аргумента. Примером таких операций является операция create для набора intset.
- **Конструкторы**. Эти операции используют в качестве аргументов объекты соответствующего им типа и создают другие объекты такого же типа.
- **Модификаторы**. Эти операции модифицируют объекты соответствующего им типа. Например, операции insert и delete - модификаторы для наборов intset. Очевидно, что только изменяемые типы, могут иметь модификаторы.
- **Наблюдатели**. Эти операции используют в качестве аргументов объекты соответствующего им типа и возвращают результаты другого типа. Они используются для получения информации об объектах. Сюда относится, например, операции size, member, choose для наборов intset.



Полнота


Тип данных является полным, если он обеспечивает достаточно операций для того, чтобы все требующиеся пользователю работы с объектами могли быть проделаны с приемлемой эффективностью





Полнота

В общем случае абстракция данных должна иметь операции по крайней мере трех из четырех рассматриваемых нами классов. Она должна иметь примитивные конструкторы, наблюдатели и либо конструкторы (если она неизменяемая), либо модификаторы (если она изменяемая).





ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ




Choose = **proc** (s:intset) **returns** (int)


requires s - не пустой набор.


Effects Возвращает произвольный элемент s






Устойчивая программа - это такая
программа, которая ведет себя корректно
даже в случае ошибки







В идеальном случае программа должна продолжать работать после ошибки, осуществляя некоторую аппроксимацию ее поведения при отсутствии ошибки. Говорят, что такая программа обеспечивает хорошую деградацию.





Метод, который гарантирует устойчивость,
заключается в использовании процедур,
заданных на всей области определения.





Защитное программирование – это методология составления программы, при которой программа защищает саму себя от ошибок.






Разделим область определения D на
несколько подмножеств.

$$D = D_0 \cup D_1 \cup \dots \cup D_n$$

Процедура ведет себя по-разному на
каждом из них.





Например: Для операции choose

$D_0 = \{\text{все непустые наборы}\}$

$D_1 = \{\text{пустой набор}\}$





$p: D_0 \rightarrow \text{обычный } (R_0)$

$D_1 \rightarrow \text{имя}_1 (R_1)$

.....

$D_n \rightarrow \text{имя}_n (R_n),$





Например:

choose: $D_0 \rightarrow \text{обычный(int)}$

$D_1 \rightarrow \text{empty()}$





signals // здесь приводится список имен и
результатов исключительных ситуаций



Например:

```
choose = proc(i:intset) returns(int)  
  signals(empty)
```

```
search=proc(a:array[int], x: int)  
  returns(ind:int) signals(not_in,  
    duplicate(ind1:int))
```

Например:

choose = **proc**(s:intset) **returns**(int)
 signals(empty)

effects Если $\text{size}(s)=0$, то сигнализировать об
 исключительной ситуации empty, иначе
 произвольный элемент s .

Например:

Search = **proc**(a:array[int], x:int) **returns**(ind: int)
 signals(not_in, duplicate (ind1:int)

requires Массив a упорядочен в возрастающем порядке.

Effects Если $x \in a$ один раз, то вернуть ind, такой, что $a[ind]=x$; Если $x \in a$ более чем один раз, то сигнализировать об исключительной ситуации duplicate (ind1), где ind1 - индекс для одного из x, иначе сигнализировать об исключительной ситуации not_in.

Область определения процедуры search есть

$$D_0 = \{ \langle a, x \rangle \mid x \in a \text{ равно один раз} \}$$


$$D_1 = \{ \langle a, x \rangle \mid x \text{ не принадлежит } a \}$$

$$D_2 = \{ \langle a, x \rangle \mid x \in a \text{ более чем один раз} \}$$



АБСТРАКЦИЯ ИТЕРАЦИИ

Абстракция итерации или итераторы являются некоторым обобщенным итерационных методов, имеющихся в большинстве языков программирования.





Спецификация

iname=iter (...) yields (...) signals (...)



Например:

elements = **iter** (s:intset) **yields** (int)

requires s не модифицируется в теле цикла.

Effects выдает элементы s в некотором произвольном порядке, причем каждый элемент только один раз.



Вопросы?