2. Katja Laboratory
Attribute Functions with Katja

Lecture Compilers SS 2011

Jan Schäfer (slides by Jean-Marie Gaillourdet)

Software Technology Group
TU Kaiserslautern

Generic Navigation on Term-Positions

`parent()`	parent node in tree, or null if root
`root()`	top most element of term position structure
`lsib()`	left sibling
`rsib()`	right sibling
`size()`	number of children, *n* for a *n*-tuple, or length of a list
`get(int n)`	get the nth child
`position()`	`this.parent().get(this.position()) == this` or `-1`

All functions returns the best possible type, safe to assume `SortPos`.

Path Navigation

Path

A path is a `List<Integer>` denoting the sequence of arguments to `get()` in order to reach a certain term position (usually starting from the root).

`List<Integer> path()`	get the path from the *root*() to *this*
`List<Integer> pathFrom(SortPos p)`	get the path to reach *this* from *p*

`SortPos follow(List<Integer> path)`

List Terms

list element sort: `E`, list sort: `Es`

`Es appFront(E e),Es appBack(E e)`	append front / back
`Es addAll(KatjaList<? extends E>) es`	add all elements of `es`
`Es conc(Es es)`	concatenate with `es`
`Es reverse()`	reverse the list
`Es remove(E e)`	remove `e`
`Es removeAll(KatjaList<E> es)`	remove all element of `es`
`E first(), E last()`	get first / last element
`Es front(), Es back()`	get all but last / first elements
`boolean contains(E e)`	check if `e` is contained
`boolean containsAll(KatjaList<E> es)`	check if all elements of `es` are contained

List Terms as Sets

Handle lists as sets.

`Es toSet()`	remove all duplicates
`Es setAdd(E e)`	add element
`Es setRemove(E e)`	remove element `e`
`boolean setEquals(Es es)`	compare two lists as sets
`Es setIntersection(Es es)`	intersection
`Es setUnion(Es es)`	union
`Es setWithout(Es es)`	without

Tree walking

- Post-Order (from left to right)

`postOrder()`	get next element in post-order
`postOrderStart()`	get first element to visit subtree in post-order

- Pre-Order (from left to right)

`preOrder()`	get next element in pre-order
`preOrderSkip()`	get next element in pre-order skipping all children

Visitor

- every sort provides a nested interface
  `VisitorType<E extends Throwable>`
  - has a method `visitS(S s) throws E` for every sort which might appear below
  - there is `katja.common.NE`, which is subtype of `Throwable`, to declare no exception
- every sort provides a nested abstract static class
  `Visitor<E extends Throwable>`
  - it implements `VisitorType<E>`
  - it provides implementations for all methods `visitV` if `V` is a variant
- every root position sort provides
  `DefaultVisitor<E extends Throwable>>`
  - implements all `VisitorType`s of sorts which may appear in position structure
  - provides default implementations of all methods
- to start visiting an term/position of type `S` call `visitS`

Visitor (2)

- when overriding method `visitS(...)` there are two options
  - call `super.visitS(...)`
  - call `this.visitT(...)` for every child!

Utilities

- `String KatjaElement.Utilities.prettyPrint(String s)` It allows to post-process the output of a `toString()` method in order to prettify the output.
- `cast()` cast to the "right" type
  `<T>cast()` cast to `T`

Extended Expressions

```
let x = in 4 + x end

let x = 5 + let y = 3 in y + 2 end in x - 2 end

let x = 4 in let x = 2 in x - 2 end end
```

Extendend Expressions

```
specification Expressions
[... backend block ...]
external Int
external String

root Expression Pos

Expression = Literal ( Int value )
           | Variable ( String name )
           | Plus ( Expression left, Expression right)
           | Minus ( Expression left, Expression right)
           | Mult ( Expression left, Expression right)
           | Div ( Expression left, Expression right)
           | Let ( Variable definedVar, Expression assigned,
                   Expression body )
VariablePosList * VariablePos
```

Problem 1

- Attribute function: `LetPos declaration(VariablePos var)`
  - find the binding let for a variable occurence
- Attribute function: `VariablePosList usages(LetPos let)`
  - find all usages of the variable bound by this let

Problem 2

- implement a method to remove unused lets

Problem 3

- implement a method to inline lets which are used only once