FOUNDATIONAL ASPECTS OF SIZE ANALYSIS: ANSWERS FOR THE EXERCISES

OLHA SHKARAVSKA, MARKO VAN EEKELEN, AND ALEJANDRO TAMALET

Institute for Computing and Information Sciences, Radboud University Nijmegen
e-mail address: shkarav@cs.ru.nl

Institute for Computing and Information Sciences, Radboud University Nijmegen
e-mail address: marko@cs.ru.nl

Institute for Computing and Information Sciences, Radboud University Nijmegen
e-mail address: tamalet@cs.ru.nl

Exercise 1. Type checking for append: experimenting with demo and reading the "manual" type checking from the exercise sheet.

Exercise 2. Type checking conspack : $\texttt{Int} \times \mathsf{L}_n(\texttt{Int}) \to \mathsf{L}_{n+1}(\texttt{Int})$:

```
conspack($x, l$) =
match $l$ with | nil ⇒ cons($x, l$)
               | cons($hd, tl$) ⇒ let $y = x - hd$
                                  in if $y$  then let $l' =$ conspack($x, tl$)
                                                  in cons($hd, l'$)
                                             else cons($x, l$)
```

(1) The AHA-language code for conspack is

```
letfun conspack(x, l) : Int L(Int, n) -> L(Int, n+1) =
match l with
| Nil -> Cons(x, l)
| Cons(h, t) ->
let w=-(x,h) in
```

2000 ACM Subject Classification: F.4.1[Mathematical logic and formal languages]: Mathematical logic – Lambda calculus and related systems, Logic and constraint programming; F.2.2 [Analysis of algorithms and problem complexity]: Non-numerical algorithms and problems; D.1.1 [Programming techniques]: Applicative (functional) programming. General Terms: Algorithms, Verification.

Key words and phrases: Shapely Functions, Size Analysis, Type Checking, Type Inference, Diophantine equations, Polynomial Interpolation.

This research is sponsored by the Netherlands Organisation for Scientific Research (NWO), project Amortised Heap Space Usage Analysis (AHA), grantnr. 612.063.511.

```
    if w then Cons(h, conspack(x, t))
         else Cons(x, l) fi
    in conspack(2, Cons(1, Cons(2, Cons(3, Nil))))
```

(2) Manual type-checking:

(a) The body of the function is a pattern-matching expression. Therefore, we have to prove two subgoals that correspond to the nil- and cons-branches respectively.
Applying the Match-rule first yields the Nil-branch subgoal:

$$n = 0;\ x\!:\mathtt{Int}, l\!:\mathsf{L}_n(\mathtt{Int})\ \vdash_\Sigma \mathsf{cons}(x, l)\!:\mathsf{L}_{n+1}(\mathtt{Int})$$

(b) Continue with the nil-branch. The expression in the previous subgoal is $\mathsf{cons}(x, l)$, so we apply the Cons-rule. We obtain the following subgoal:

$$n = 0 \vdash n + 1 = n + 1$$

which is trivially true.

(c) Now follow the cons-branch defined by the Match-rule applied to e_{conspack}.

$$x\!:\mathtt{Int},\ l\!:\mathsf{L}_n(\mathtt{Int})\ \vdash_\Sigma \mathsf{let}\ y = x - hd\ \mathsf{in\ if}\ \ldots\ \mathsf{then}\ \ \mathsf{else}\ :\mathsf{L}_n(\mathtt{Int})$$

This is a let-construct, so we apply the Let-rule to obtain two subgoals, corresponding to the let-binding and the let-body respectively.

(d) In the let-binding we have a subgoal

$$x\!:\mathtt{Int},\ hd\!:\mathtt{Int}\ \vdash_\Sigma x - hd\!:\tau^?$$

where $\tau^?$ is an unknown type, which we will reconstruct on the next step using the appropriate axiom. In the context we omit the program variables, on which the expression in the subgoal does not depend.

(e) The expression in the binding clause is an integer expression, so we apply BinOp. We obtain

$$\vdash \tau^? = \mathtt{Int}$$

Now the type $\tau^?$ is reconstructed and may be used in the let-body.

(f) In the let-body we have a subgoal with an if-expression. We apply the If-rule and obtain two subgoals corresponding the true- (when $y \neq 0$) and false- (when $y = 0$) branches.
The "true" subgoal is

$$x\!:\mathtt{Int}, l\!:\mathsf{L}_n(\mathtt{Int}),\ hd\!:\mathtt{Int},\ tl\!:\mathsf{L}_{n-1}(\mathtt{Int})\ \vdash_\Sigma \mathsf{let}\ l' = \mathsf{conspack}(x, tl)\ \mathsf{in\ cons}(hd, l')\!:\mathsf{L}_{n+1}(\mathtt{Int})$$

(g) The subgoal in the let-binding is

$$x\!:\mathtt{Int},\ tl\!:\mathsf{L}_{n-1}(\mathtt{Int})\ \vdash_\Sigma \mathsf{conspack}(x, tl)\!:\tau^?$$

(h) Applying function-application rule yields $\tau^? = \mathsf{L}_{(n-1)+1}(\mathtt{Int})$.

(i) The subgoal in the let-body is

$$l'\!:\mathsf{L}_{(n-1)+1}(\mathtt{Int}),\ hd\!:\mathtt{Int}\ \vdash_\Sigma \mathsf{cons}(hd, l')\!:\mathsf{L}_{n+1}(\mathtt{Int})$$

(j) Applying cons-rule yields $\vdash n+1 = (n-1)+1+1$ which is trivially true.

(k) The "false" subgoal is

$$x : \texttt{Int}, l : \mathsf{L}_n(\texttt{Int}) \ \vdash_\Sigma \mathsf{cons}(x, l) : \mathsf{L}_{n+1}(\texttt{Int})$$

(l) Applying the Cons-rule yields

$$\vdash n+1 = n+1$$

which is trivially true.

Exercise 3. Type checking $\mathsf{sqdiff} : \mathsf{L}_n(\alpha) \times \mathsf{L}_m(\alpha) \to \mathsf{L}_{(n-m)^2}(\mathsf{L}_2(\alpha))$:

```
sqdiff($l_1, l_2$) =
match $l_1$ with | nil ⇒ cprod($l_2, l_2$)
                 | cons($hd_1, tl_1$) ⇒ match $l_2$ with | nil ⇒ cprod($l_1, l_1$)
                                                         | cons($hd_2, tl_2$) ⇒ sqdiff($tl_1, tl_2$)
```

Manual type-checking:

(1) The body of the function is, again, a pattern-matching expression. We consider two subgoals that correspond to the nil- and cons-branches respectively.
Applying the Match-rule first yields the Nil-branch subgoal:

$$n = 0;\ l_1 : \mathsf{L}_n(\alpha),\ l_2 : \mathsf{L}_m(\alpha) \ \vdash_\Sigma \mathsf{cprod}(l_2, l_2) : \mathsf{L}_{(n-m)^2}(\mathsf{L}_2(\alpha))$$

(2) Continue with the nil-branch. The expression in the subgoal the function call $\mathsf{cprod}(l_2, l_2)$, so we apply the function-call rule and obtain the following equations:

$$n = 0 \vdash (n-m)^2 = m^2$$
$$n = 0 \vdash 2 = 2$$

which are trivially true.

(3) Now follow the cons-branch defined by the Match-rule applied to e_{cprod}. This is again a pattern matching. Consider its nil-branch:

$$m = 0;\ l_1 : \mathsf{L}_n(\alpha),\ l_2 : \mathsf{L}_m(\alpha) \ \vdash_\Sigma \mathsf{cprod}(l_1, l_1) : \mathsf{L}_{(n-m)^2}(\mathsf{L}_2(\alpha))$$

(4) Continue with this nil-branch. The expression in the previous subgoal the function call $\mathsf{cprod}(l_1, l_1)$, so we apply the function-call rule and obtain the following equations:

$$m = 0 \vdash (n-m)^2 = n^2$$
$$m = 0 \vdash 2 = 2$$

which are trivially true.

(5) The subgoal in the cons-branch is

$$tl_1 : \mathsf{L}_{n-1}(\alpha),\ tl_2 : \mathsf{L}_{m-1}(\alpha) \ \vdash_\Sigma \mathsf{cprod}(tl_1, tl_2) : \mathsf{L}_{(n-m)^2}(\mathsf{L}_2(\alpha))$$

(6) Applying function-application rule yields two equations

$$\vdash (n-m)^2 = ((n-1)-(m-1))^2$$
$$\vdash 2 = 2$$

which are obviously true.

Exercise 4. Type checking of $\mathsf{scalar\_prod}$:

```
$\mathsf{scalar\_prod}(l_1, l_2) =$
$\mathsf{match}\ l_1\ \mathsf{with}\ |\ \mathsf{nil} \Rightarrow \mathsf{match}\ l_2\ \mathsf{with}\ |\ \mathsf{nil} \Rightarrow \mathsf{cons}(0, \mathsf{nil})$
                                                $|\ \mathsf{cons}(hd_2, tl_2) \Rightarrow \mathsf{scalar\_prod}(l_1, l_2)$ (* nontermination *)
                        $|\ \mathsf{cons}(hd_1, tl_1) \Rightarrow \mathsf{match}\ l_2\ \mathsf{with}\ |\ \mathsf{nil} \Rightarrow \mathsf{scalar\_prod}(l_1, l_2)$ (* nontermination *)
                                                          $|\ \mathsf{cons}(hd_2, tl_2) \Rightarrow \mathsf{let}\ l = \mathsf{scalar\_prod}(tl_1, tl_2)$
                                                                          $\mathsf{in\ let}\ y = hd_1 * hd_2$
                                                                            $\mathsf{in\ replace}(y, l)$
```

where $\mathsf{replace} : \texttt{Int} \times \mathsf{L}_n(\texttt{Int}) \to \mathsf{L}_n(\texttt{Int})$ is defined by

```
$\mathsf{replace}(x, l) =$
$\mathsf{match}\ l\ \mathsf{with}\ |\ \mathsf{nil} \Rightarrow \mathsf{nil}$
            $|\ \mathsf{cons}(hd, tl) \Rightarrow \mathsf{cons}(x + hd, tl)$
```

(1) The AHA-language presentation of these programs, for type checking $\mathsf{scalar\_prod} : \mathsf{L}_n(\texttt{Int}) \times \mathsf{L}_n(\texttt{Int}) \to \mathsf{L}_1(\texttt{Int})$, is

```
 letfun replace(x, l): Int L(Int, n) -> L(Int, n) =
match l with
| Nil -> Nil
|Cons(h, t) -> Cons(+(x,h), t)
in
letfun scalarprod(l, ll) : L(Int, n) L(Int, n) -> L(Int, 1) =
match l with
| Nil ->
match ll with
| Nil -> Cons(0, Nil)
| Cons(hh, tt) -> scalarprod(l, ll)
| Cons(h, t) ->
match ll with
| Nil -> scalarprod(l, ll)
| Cons(hh, tt) -> replace(*(h, hh), scalarprod(t, tt))
in scalarprod(Cons(3, Nil), Cons(2, Nil))
```

Manual type checking:

(a) Applying the match-rule to the body of $\mathsf{scalar\_prod}$ yields two subgoals. The first one corresponds to the nil-branch, which is in its turn, again, a pattern matching. We apply the match-rule again and obtain two subgoals. In the nil-branch we have

$$n = 0;\ l_1 : \mathsf{L}_n(\texttt{Int}),\ l_2 : \mathsf{L}_n(\texttt{Int})\ \vdash_\Sigma \mathsf{cons}(0, \mathsf{nil}) : \mathsf{L}_1(\texttt{Int})$$

(b) The expression $\mathsf{cons}(0, \mathsf{nil})$ is the sugared version of the let-bindings

$$\mathsf{let}\ l' = \mathsf{nil}\ \mathsf{in\ let}\ x = 0\ \mathsf{in\ cons}(x, l')$$

One can easily show that its type is $\mathsf{L}_{1+0}(\texttt{Int})$. Therefore, we obtain the trivially true equation $\vdash 1 = 1 + 0$.

(c) The cons-branch of the nil-branch of the body of $\mathsf{scalar\_prod}$ is the function call $\mathsf{scalar\_prod}(l_1, l_2)$, which yields the non-terminating computation. In any case, we have to check its type either. Applying the function-call rule we obtain

$$n = 0 \vdash \mathsf{L}_1(\mathtt{Int}) = \mathsf{L}_1(\mathtt{Int})$$

which is trivially true.

(d) Now consider the cons-branch of the body of $\mathsf{scalar\_prod}$. It is, again, a pattern-matching. First we type check its nil-branch, which yields a non-terminating computation.

$$n = 0;\ l_1 : \mathsf{L}_n(\mathtt{Int}),\ l_2 : \mathsf{L}_n(\mathtt{Int})\ \vdash_\Sigma \mathsf{scalar\_prod}(l_1, l_2) : \mathsf{L}_1(\mathtt{Int})$$

(e) Applying the function-application rule to the subgoal above we obtain, as earlier, $n = 0 \vdash \mathsf{L}_1(\mathtt{Int}) = \mathsf{L}_1(\mathtt{Int})$, which is trivially true.

(f) Consider the subgoal in the cons-branch of the cons-branch of the body:

$$\left.\begin{array}{l} l_1 : \mathsf{L}_n(\mathtt{Int}),\ l_2 : \mathsf{L}_n(\mathtt{Int}), \\ tl_1 : \mathsf{L}_{n-1}(\mathtt{Int}),\ tl_2 : \mathsf{L}_{n-1}(\mathtt{Int}), \\ hd_1 : \mathtt{Int},\ hd_2 : \mathtt{Int} \end{array}\right\} \vdash_\Sigma \left\{\begin{array}{l} \quad\mathsf{let}\ l = \mathsf{scalar\_prod}(tl_1, tl_2) \\ \mathsf{in\ let}\ y = hd_1 * hd_2 \\ \quad\quad \mathsf{in\ replace}(y, l) \end{array}\right\} : \mathsf{L}_1(\mathtt{Int})$$

(g) The binding in the outer let-expression generates the subgoal

$$tl_1 : \mathsf{L}_{n-1}(\mathtt{Int}),\ tl_2 : \mathsf{L}_{n-1}(\mathtt{Int})\ \vdash_\Sigma \mathsf{scalar\_prod}(tl_1, tl_2) : \tau^?$$

(h) Applying the function application rule we obtain $\tau^? := \mathsf{L}_1(\mathtt{Int})$.

(i) The binding in the inner let-expression generates the subgoal

$$hd_1 : \mathtt{Int},\ hd_2 : \mathtt{Int}\ \vdash_\Sigma hd_1 * hd_2 : \tau'^?$$

(j) Applying the binary-operation rule we obtain $\tau'^? := \mathtt{Int}$.

(k) For the let-body we have the subgoal

$$l : \mathsf{L}_1(\mathtt{Int}),\ y : \mathtt{Int}\ \vdash_\Sigma \mathsf{replace}(y, l) : \mathsf{L}_1(\mathtt{Int})$$

(l) The function-application rule yields $\vdash \mathsf{L}_1(\mathtt{Int}) = \mathsf{L}_1(\mathtt{Int})$, which is trivially true.

(2) The AHA-language presentation for type checking $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_m(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$, is the same as for the typing $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_n(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$, except the `letfun` row, where $\mathsf{scalar\_prod}$ is defined. Of course, now it is

```
letfun scalarprod(l, ll) : L(Int, n) L(Int, m) -> L(Int, 1)
```

The manual type-checking is similar to the one for the typing $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_n(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$:

(a) Applying the match-rule to the body of $\mathsf{scalar\_prod}$ yields two subgoals. The first one corresponds to the nil-branch, which is in its turn, again, a pattern matching. We apply the match-rule again and obtain two subgoals. In the nil-branch we have

$$n = 0, m = 0;\ l_1 : \mathsf{L}_n(\mathtt{Int}),\ l_2 : \mathsf{L}_m(\mathtt{Int})\ \vdash_\Sigma \mathsf{cons}(0, \mathsf{nil}) : \mathsf{L}_1(\mathtt{Int})$$

(b) The expression $\mathsf{cons}(0, \mathsf{nil})$ is the sugared version of the let-bindings

$$\mathsf{let}\ l' = \mathsf{nil}\ \mathsf{in}\ \mathsf{let}\ x = 0\ \mathsf{in}\ \mathsf{cons}(x, l')$$

One can easily show that its type is $\mathsf{L}_{1+0}(\mathtt{Int})$. Therefore, we obtain the trivially true equation $n = 0, m = 0 \vdash 1 = 1 + 0$.

(c) The cons-branch of the nil-branch of the body of $\mathsf{scalar\_prod}$ is the function call $\mathsf{scalar\_prod}(l_1, l_2)$, which yields the non-terminating computation. In any case, we have to check its type either. Applying the function-call rule we obtain

$$n = 0 \vdash \mathsf{L}_1(\mathtt{Int}) = \mathsf{L}_1(\mathtt{Int})$$

which is trivially true.

(d) Now consider the cons-branch of the body of $\mathsf{scalar\_prod}$. It is, again, a pattern-matching. First we type check its nil-branch, which yields a non-terminating computation.

$$m = 0;\ l_1 : \mathsf{L}_n(\mathtt{Int}),\ l_2 : \mathsf{L}_m(\mathtt{Int})\ \vdash_\Sigma \mathsf{scalar\_prod}(l_1, l_2) : \mathsf{L}_1(\mathtt{Int})$$

(e) Applying the function-application rule to the subgoal above we obtain, as earlier, $m = 0 \vdash \mathsf{L}_1(\mathtt{Int}) = \mathsf{L}_1(\mathtt{Int})$, which is trivially true.

(f) Consider the subgoal in the cons-branch of the cons-branch of the body:

$$\left.\begin{array}{l} l_1 : \mathsf{L}_n(\mathtt{Int}),\ l_2 : \mathsf{L}_m(\mathtt{Int}), \\ tl_1 : \mathsf{L}_{n-1}(\mathtt{Int}),\ tl_2 : \mathsf{L}_{m-1}(\mathtt{Int}), \\ hd_1 : \mathtt{Int},\ hd_2 : \mathtt{Int} \end{array}\right\} \vdash_\Sigma \left.\begin{array}{l} \mathsf{let}\ l = \mathsf{scalar\_prod}(tl_1, tl_2) \\ \mathsf{in}\ \mathsf{let}\ y = hd_1 * hd_2 \\ \quad \mathsf{in}\ \mathsf{replace}(y, l) \end{array}\right\} : \mathsf{L}_1(\mathtt{Int})$$

(g) The binding in the outer let-expression generates the subgoal

$$tl_1 : \mathsf{L}_{n-1}(\mathtt{Int}),\ tl_2 : \mathsf{L}_{m-1}(\mathtt{Int})\ \vdash_\Sigma \mathsf{scalar\_prod}(tl_1, tl_2) : \tau^?$$

(h) Applying the function application rule we obtain $\tau^? := \mathsf{L}_1(\mathtt{Int})$.

(i) The binding in the inner let-expression generates the subgoal

$$hd_1 : \mathtt{Int},\ hd_2 : \mathtt{Int}\ \vdash_\Sigma hd_1 * hd_2 : \tau'^?$$

(j) Applying the binary-operation rule we obtain $\tau'^? := \mathtt{Int}$.

(k) For the let-body we have the subgoal

$$l : \mathsf{L}_1(\mathtt{Int}),\ y : \mathtt{Int}\ \vdash_\Sigma \mathsf{replace}(y, l) : \mathsf{L}_1(\mathtt{Int})$$

(l) The function-application rule yields $\vdash \mathsf{L}_1(\mathtt{Int}) = \mathsf{L}_1(\mathtt{Int})$, which is trivially true.

Exercise 5. Infer the size annotations for append for $d = 2$.

(1) The input of the inference procedure is $\mathsf{append} : \mathsf{L}_n(\alpha) \times \mathsf{L}_m(\alpha) \to \mathsf{L}_{p^?(n,m)}(\alpha)$, that is we supply the procedure with the underlying type and the list of the size variables, n, m assigned to the input types of append. The task is to reconstruct $p^?(n, m)$, assuming that the degree d of the polynomial $p^?$ is $d = 2$.

(2) A quadratic polynomial ($d = 2$) of two variables has *six* coefficients: $p^?(n,m) = a_{20}n^2 + a_{02}m^2 + a_{11}nm + a_{10}n + a_{01}m + a_{00}$. Therefore, to compute these coefficients, we must have the values $p(n_1, m_1), \ldots, p(n_6, m_6)$ of the polynomial in some six 2-dimensional nodes, $(n_1, m_1), \ldots, (n_6, m_6)$ such that the system

$$\left.\begin{array}{llllllll} a_{20}n_1^2 & + a_{02}m_1^2 & + a_{11}n_1m_1 & + a_{10}n_1 & + a_{01}m_1 & + a_{00} & = p(n_1, m_1) \\ \ldots & & & & & & \ldots \\ a_{20}n_6^2 & + a_{02}m_6^2 & + a_{11}n_6m_6 & + a_{10}n_6 & + a_{01}m_6 & + a_{00} & = p(n_6, m_6) \end{array}\right\}$$

where $a_{20},\ a_{02},\ a_{11},\ a_{10},\ a_{01},\ a_{00}$ are variables, has a unique solution.

(3) The system above has a unique solution if the nodes $(n_1, m_1), \ldots, (n_6, m_6)$ satisfy the NCA-configuration for six 2-dimensional nodes. E.g. three of them lie on a line on the Cartesian plane, two of them lie on another line, which is parallel to the first line, and the third one lies on yet another parallel line. We take $(1, 1),\ (2, 1), (3, 1)$ lying on $y = 1$, then $(1, 2), (2, 2)$ lying on $y = 2$ and $(1, 3)$ lying on $y = 3$.

(4) Now we generate six pairs of input lists for append with these pairs of lengths:
- $[1],\ [2]$ with the lengths $(1, 1)$ resp.,
- $[1, 2],\ [3]$ with the lengths $(2, 1)$ resp.,
- $[1, 2, 3],\ [4]$ with the lengths $(3, 1)$ resp.,
- $[1],\ [2, 3]$ with the lengths $(1, 2)$ resp.,
- $[1, 2],\ [3, 4]$ with the lengths $(2, 2)$ resp.,
- $[1],\ [2, 3, 4]$ with the lengths $(1, 3)$ resp.

(5) Now we run append on these data:

input lengths	input list 1	input list 2	output of append	length of the output
$(1,1)$	$[1]$	$[2]$	$[1,2]$	2
$(2,1)$	$[1,2]$	$[3]$	$[1,2,3]$	3
$(3,1)$	$[1,2,3]$	$[4]$	$[1,2,3,4]$	4
$(1,2)$	$[1]$	$[2,3]$	$[1,2,3]$	3
$(2,2)$	$[1,2]$	$[3,4]$	$[1,2,3,4]$	4
$(1,3)$	$[1]$	$[2,3,4]$	$[1,2,3,4]$	4

(6) Using the table above, we generate a system of linear equations w.r.t. the coefficients a_{10}, a_{01}, a_{00}:

$$\begin{array}{lllllll} a_{20} & + a_{02} & + a_{11} & + a_{10} & + a_{01} & + a_{00} & = 2 \\ 4a_{20} & + a_{02} & + 2a_{11} & + 2a_{10} & + a_{01} & + a_{00} & = 3 \\ 9a_{20} & + a_{02} & + 3a_{11} & + 3a_{10} & + a_{01} & + a_{00} & = 4 \\ a_{20} & + 4a_{02} & + 2a_{11} & + a_{10} & + 2a_{01} & + a_{00} & = 3 \\ 4a_{20} & + 4a_{02} & + 4a_{11} & + 2a_{10} & + 2a_{01} & + a_{00} & = 4 \\ a_{20} & + 9a_{02} & + 3a_{11} & + a_{10} & + 3a_{01} & + a_{00} & = 4 \end{array}$$

(7) Solve the system above. The solution is $a_{10} = a_{01} = 1$ and $a_{20} = a_{02} = a_{11} = a_{00} = 0$. Therefore, as for $d = 1$, we obtain $p^?(n, m) = n + m$. We have already checked that append : $\mathsf{L}_n(\alpha) \times \mathsf{L}_m(\alpha) \to \mathsf{L}_{n+m}(\alpha)$ is a correct type.

Exercise 6. Inferring annotations for conspack.

(1) The presentation of conspack in the AHA-language for inferring its size annotations in the demo is the same as the code for type checking, except that we delete size variables from types, so we have only underlying types in the signatures.

(2) Manual inference.

(a) Assign the size variable n to the input list and assume the degree $d = 1$ of the polynomial size function of conspack. Therefore, the polynomial size function is of the form $p(n) = an + b$ and we need to find its coefficients a and b.

(b) The polynomial $p(n) = an + b$ is defined by two different points on its graph. Let, e.g., $n_1 = 0$ and $n_2 = 1$.

(c) Generate two pairs input data where the length of the input lists are $n_1 = 0$ and $n_2 = 1$ respectively. E.g. take as the first input the pair $1, []$ and as the second input the pair $1, [1]$.

(d) "Run" the program on these pairs. The program outputs $[1]$ of the length 1 on the first pair and $[1, 1]$ of the length 2 on the second pair.

(e) Based on the size information from the tests, construct the system of linear equations w.r.t. a, b:

$$\begin{aligned} b &= 1 \\ a + \quad b &= 2 \end{aligned}$$

(f) Solving this system gives $a = 1$ and $b = 1$, therefore $p(n) = n + 1$.

(g) Type check conspack : $\mathtt{Int} \times \mathsf{L}_n(\mathtt{Int}) \to \mathsf{L}_{n+1}(\mathtt{Int})$. As we have seen earlier, this type is accepted.

Exercise 7. Inferring annotations for sqdiff.

(1) Inferring annotations for sqdiff in the inference part of demo (there it is called $u$sqdiff).

(2) Manual inference of annotations for sqdiff, assuming $d = 2$.

(a) The inference is very similar to the annotation inference for append, assuming that $d = 2$, where d is the degree of the corresponding polynomial size function. The input of the inference procedure is

$$\mathsf{sqdiff} : \mathsf{L}_n(\alpha) \times \mathsf{L}_m(\alpha) \to \mathsf{L}_{p^?(n,m)}(\mathsf{L}_{p_2^?(n,m)}(\alpha))$$

that is we supply the procedure with the underlying type and the list of the size variables, n, m assigned to the input types of append. The task is to reconstruct $p^?(n, m)$ and $p_2^?(n, m)$ assuming that the degree d of the polynomials $p^?$ and $p_2^?$ is $d = 2$.

(b) A quadratic polynomial ($d = 2$) of two variables has *six* coefficients: $p^?(n, m) = a_{20}n^2 + a_{02}m^2 + a_{11}nm + a_{10}n + a_{01}m + a_{00}$. Therefore, to compute these coefficients, we must have the values $p(n_1, m_1), \ldots, p(n_6, m_6)$ of the polynomial in some six 2-dimensional nodes, $(n_1, m_1), \ldots, (n_6, m_6)$ such that the system

$$\left.\begin{array}{llllllll} a_{20}n_1^2 & + a_{02}m_1^2 & + a_{11}n_1m_1 & + a_{10}n_1 & + a_{01}m_1 & + a_{00} & = p(n_1, m_1) \\ \ldots & & & & & & \ldots \\ a_{20}n_6^2 & + a_{02}m_6^2 & + a_{11}n_6m_6 & + a_{10}n_6 & + a_{01}m_6 & + a_{00} & = p(n_6, m_6) \end{array}\right\}$$

where a_{20}, a_{02}, a_{11}, a_{10}, a_{01}, a_{00} are variables, has a unique solution. Similar holds for $p_2^?$.

(c) The system above has a unique solution if the nodes $(n_1, m_1), \ldots, (n_6, m_6)$ satisfy NCA-configuration for six 2-dimensional nodes. E.g. three of them lie on a line on the Cartesian plane, two of them lie on another line, which is parallel to the first line, and the third one lies on yet another parallel line. We take $(1, 1)$, $(2, 1), (3, 1)$ lying on $y = 1$, then $(1, 2), (2, 2)$ lying on $y = 2$ and $(1, 3)$ lying on $y = 3$.

(d) Now we generate six pairs of input lists for sqdiff with these pairs of lengths:
- $[1]$, $[2]$ with the lengths $(1, 1)$ resp.,
- $[1, 2]$, $[3]$ with the lengths $(2, 1)$ resp.,
- $[1, 2, 3]$, $[4]$ with the lengths $(3, 1)$ resp.,
- $[1]$, $[2, 3]$ with the lengths $(1, 2)$ resp.,
- $[1, 2]$, $[3, 4]$ with the lengths $(2, 2)$ resp.,
- $[1]$, $[2, 3, 4]$ with the lengths $(1, 3)$ resp.

(e) Now we run sqdiff on these data:

input lengths	input list 1	input list 2	output of sqdiff	the outer length of the output	the inner length of the output
$(1,1)$	$[1]$	$[2]$	$[]$	0	?
$(2,1)$	$[1,2]$	$[3]$	$[[2,2]]$	1	2
$(3,1)$	$[1,2,3]$	$[4]$	$[[2,2],[2,3],[3,2],[3,3]]$	4	2
$(1,2)$	$[1]$	$[2,3]$	$[[3,3]]$	1	2
$(2,2)$	$[1,2]$	$[3,4]$	$[]$	0	?
$(1,3)$	$[1]$	$[2,3,4]$	$[[3,3],[3,4],[4,3],[4,4]]$	4	2

(f) Using the table above, we generate a system of linear equations w.r.t. the coefficients a_{10}, a_{01}, a_{00}:

$$\begin{array}{llllllll}
a_{20} & + a_{02} & + a_{11} & + a_{10} & + a_{01} & + a_{00} & = 0 \\
4a_{20} & + a_{02} & + 2a_{11} & + 2a_{10} & + a_{01} & + a_{00} & = 1 \\
9a_{20} & + a_{02} & + 3a_{11} & + 3a_{10} & + a_{01} & + a_{00} & = 4 \\
a_{20} & + 4a_{02} & + 2a_{11} & + a_{10} & + 2a_{01} & + a_{00} & = 1 \\
4a_{20} & + 4a_{02} & + 4a_{11} & + 2a_{10} & + 2a_{01} & + a_{00} & = 0 \\
a_{20} & + 9a_{02} & + 3a_{11} & + a_{10} & + 3a_{01} & + a_{00} & = 4
\end{array}$$

(g) Solve the system above. The solution is $a_{10} = a_{01} = a_{00} = 0$ and $a_{20} = a_{02} = 1$, $a_{11} = -2$. Therefore, we obtain $p^?(n, m) = n^2 + m^2 - 2nm = (n - m)^2$.

(h) To complete computations for the inner-size function $p^?_2(n, m)$ we need to perform two more tests instead of the ones that deliver undefinedness for $p^?_2(n, m)$. Let us run the program on the pairs of length $(4, 1)$ and $(3, 2)$. Proceed as above (for $p^?(n, m)$) to obtain $p^?_2(n, m) = 2$.

(i) We have already checked that sqdiff : $\mathsf{L}_n(\alpha) \times \mathsf{L}_m(\alpha) \to \mathsf{L}_{(n-m)^2}(\mathsf{L}_2(\alpha))$ is a correct type.

(3) Inferring annotations for sqdiff assuming that $d = 1$ (should fail).

(a) The input of the inference procedure is sqdiff : $\mathsf{L}_n(\alpha) \times \mathsf{L}_m(\alpha) \to \mathsf{L}_{p^?(n,m)}(\mathsf{L}_{p^?_2(n,m)}(\alpha))$, that is we supply the procedure with the underlying type and the list of the size variables, n, m assigned to the input types of sqdiff assuming $d = 1$.

A linear polynomial ($d = 1$) of two variables has *three* coefficients: $p^?(n,m) = a_{10}n + a_{01}m + a_{00}$. Therefore, to compute these coefficients, we must have the values $p(n_1,m_1), p(n_2,m_2), p(n_3,m_3)$ of the polynomial in some three 2-dimensional nodes, $(n_1,m_1), (n_2,m_2), (n_3,m_3)$ such that the system

$$\left.\begin{array}{llll} a_{10}n_1 & + a_{01}m_1 & + a_{00} & = p(n_1,m_1) \\ a_{10}n_2 & + a_{01}m_2 & + a_{00} & = p(n_2,m_2) \\ a_{10}n_3 & + a_{01}m_3 & + a_{00} & = p(n_3,m_3) \end{array}\right\}$$

where a_{10}, a_{01}, a_{00} are variables, has a unique solution. Similar holds for $p^?_2(n,m)$.

(b) The system above has a unique solution if the nodes $(n_1,m_1), (n_2,m_2), (n_3,m_3)$ satisfy NCA-configuration for three 2-dimensional nodes. E.g. two of them lie on a line on the Cartesian plane and the third one lies on another line and does not lie on the intersection of these two lines. We take $(1,1)$ and $(2,1)$ lying on $y = 1$ and $(1,2)$ lying on $y = 2$.

(c) Now we generate three pairs of input lists for sqdiff with lengths $(1,1)$, $(2,1)$ and $(1,2)$ respectively. Since sqdiff is shapely it does not matter what we put as elements in these lists. For instance, it may be arbitrary integer numbers. So, we generate three input pairs:
- $[1]$, $[2]$ with the lengths $(1,1)$ resp.,
- $[1,2]$, $[3]$ with the lengths $(2,1)$ resp.,
- $[1]$, $[2,3]$ with the lengths $(1,2)$ resp.

(d) Now we run sqdiff on these data:

input lengths	input list 1	input list 2	output of sqdiff	outer length of the output
$(1,1)$	$[1]$	$[2]$	$[]$	0
$(2,1)$	$[1,2]$	$[3]$	$[[2,2]]$	1
$(1,2)$	$[1]$	$[2,3]$	$[[3,3]]$	3

(e) Using the table above, we generate a system of linear equations w.r.t. the coefficients a_{10}, a_{01}, a_{00}:

$$\begin{array}{llll} a_{10} & + a_{01} & + a_{00} & = 2 \\ 2a_{10} & + a_{01} & + a_{00} & = 3 \\ a_{10} & + 2a_{01} & + a_{00} & = 3 \end{array}$$

(f) Solve the system above. The solution is $a_{10} = a_{01} = 1$ and $a_{00} = 0$. Therefore $p^?(n,m) = n + m$.

(g) Check the typing sqdiff : $\mathsf{L}_n(\alpha) \times \mathsf{L}_m(\alpha) \to \mathsf{L}_{n+m}(\mathsf{L}_2(\alpha))$. Type checking fails. E.g. in the nil-branch of the body of sqdiff, applying the function-application rule on cprod(l_2, l_2), we obtain $n = 0 \vdash n + m = m^2$ which is not valid.

Exercise 8. Inferring the size annotations for scalar_prod.

(1) Manual inference of the typing scalar_prod : $\mathsf{L}_n(\texttt{Int}) \times \mathsf{L}_n(\texttt{Int}) \to \mathsf{L}_1(\texttt{Int})$ is very similar to the inference for conspack.

(a) Assign the size variable n the input list and assume the degree $d = 1$ of the polynomial size function of scalar_prod. Therefore, the polynomial size function is of the form $p(n) = an + b$ and we need to find its coefficients a and b.

(b) The polynomial $p(n) = an + b$ is defined by two different points on its graph. Let, e.g., $n_1 = 0$ and $n_2 = 1$.
(c) Generate two pairs of input lists where the length of the input lists are $n_1 = 0$ and $n_2 = 1$ respectively. E.g. take as the first input the pair $[], []$ and as the second input the pair $[1], [2]$.
(d) "Run" the program on these pairs. The program outputs $[0]$ of the length 1 on the first pair and $[2]$ of the length 1 on the second pair.
(e) Based on size information from the tests, construct the system of linear equations w.r.t. a, b:

$$\begin{aligned} b &= 1 \\ a+ \quad b &= 1 \end{aligned}$$

(f) Solving this system gives $a = 0$ and $b = 1$, therefore $p(n) = 1$.
(g) Type check $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_n(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$. As we have seen earlier, this type is accepted.

(2) The AHA-language presentation for scalar_prod for the inference part of the demo is the same as for its checking part, except that we remove the size annotations from the types. We have inferred the less precise typing $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_m(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$, because the implemented procedure assigns automatically different variables n and m to the first and the second input lists respectively and starts with the degree $d_0 = 0 \leq 2$.

How to force the inference procedure not to infer the less precise type $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_m(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$ and infer the precise type $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_n(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$?

First, force the inference procedure to start with the degree $d \geq 1$. Then then the inference procedure will not find any test data, which lengths satisfy NCA configuration and on which the program terminates. This is because all 2-dimensional points where scalar_prod terminates lie on one line, $m = n$. Thus, the less precise type may not be inferred.

Second, to infer the precise type $\mathsf{scalar\_prod} : \mathsf{L}_n(\mathtt{Int}) \times \mathsf{L}_n(\mathtt{Int}) \to \mathsf{L}_1(\mathtt{Int})$, we may allow a user to assign size variables to input types manually.

Exercise 9. Inferring and checking the size annotations for filter.

(1) First, parsing the code for filter, we obtain the rewriting rules for its size function:

$$\begin{aligned} &\vdash f_{\mathsf{filter}}(0) \to 0 \\ n \geq 1 \quad &\vdash f_{\mathsf{filter}}(n) \to 1 + f_{\mathsf{filter}}(n-1) \mid f_{\mathsf{filter}}(n-1) \end{aligned}$$

(2) We assume the degree $d = 1$ for a polynomial lower $p_{\mathsf{filter}\ \min}(n)$ and an upper $p_{\mathsf{filter}\ \max}(n)$ bounds of $f_{\mathsf{filter}}(n)$. Thus we need to know values of $p_{\mathsf{filter}\ \min}(n) = a_{\min}n + b_{\min}$ and $p_{\mathsf{filter}\ \max}(n) = a_{\max}n + b_{\max}$ in two different points. Let it be the points $n_1 = 1, n_2 = 2$.
(3) Compute $f_{\mathsf{filter}}(n)$ in these points.
First, $f_{\mathsf{filter}}(1) = \{1 + f_{\mathsf{filter}}(0), f_{\mathsf{filter}}(0)\} = \{1, 0\}$.
Second, $f_{\mathsf{filter}}(2) = \{1 + f_{\mathsf{filter}}(1), f_{\mathsf{filter}}(1)\} = \{2, 1, 0\}$.
(4) We have that $p_{\mathsf{filter}\ \min}(1) = 0$, $p_{\mathsf{filter}\ \min}(2) = 0$, $p_{\mathsf{filter}\ \max}(1) = 1$, $p_{\mathsf{filter}\ \max}(2) = 2$.

(5) Using the obtained pair of points on the graph of $p_{\mathsf{filter}\ \min}$ we obtain that $p_{\mathsf{filter}\ \min}(n) = 0$. Using the obtained pair of points on the graph of $p_{\mathsf{filter}\ \max}$ we obtain that $p_{\mathsf{filter}\ \max}(n) = n$.

(6) Now we need to check that $p_{\mathsf{filter}\ \min}(n) = 0$ and $p_{\mathsf{filter}\ \max}(n) = n$ are correct lower and upper bounds. Proof of that amounts to proof of correctness of the following typing:

$$\mathsf{filter} : (\alpha \to \mathtt{Bool}) \times \mathsf{L}_n(\alpha) \to \mathsf{L}_{\{i\}_{0\le i\le n}}(\alpha)$$

(7) The proof of correctness reduces to the proof of the following predicates:
- $n = 0 \vdash \exists i.\ 0 \le i \le n \wedge i = 0$ (from the nil-branch),
- $n \ge 1, 0 \le i' \le n-1 \vdash \exists i.\ 0 \le i \le n \wedge i = 1 + i'$ (from the true-branch),
- $n \ge 1, 0 \le i' \le n-1 \vdash \exists i.\ 0 \le i \le n \wedge i = i'$ (from the false-branch).

It is easy to see that these predicates holds.

Exercise 10. Inferring and checking the size annotations for tails.

(1) First, parsing the code for tails, we obtain the rewriting rules for its size functions:

$$\begin{array}{rl} & \vdash f_{\mathsf{tails}\ 1}(0) \to 0 \\ n \ge 1 & \vdash f_{\mathsf{tails}\ 1}(n) \to 1 + f_{\mathsf{tails}\ 1}(n-1) \end{array}$$

and

$$\begin{array}{rl} n \ge 1 & \vdash f_{\mathsf{tails}\ 2}(n) \to f_{\mathsf{tails}\ 2}(n-1)\ (*\text{the sizes of lists in the tail of the output}*) \\ n \ge 1 & \vdash f_{\mathsf{tails}\ 2}(n) \to n\ (*\text{alternatively, the size of the head of the output}*) \end{array}$$

(2) We assume the degree $d = 1$ for a polynomial lower $p_{\mathsf{tails}\ 1\ \min}(n)$ and an upper $p_{\mathsf{tails}\ 1\ \max}(n)$ bounds of $f_{\mathsf{tails}\ 1}(n)$. Thus we need to know values of $p_{\mathsf{tails}\ 1\ \min}(n) = a_{1\ \min}n + b_{1\ \min}$ and $p_{\mathsf{tails}\ 1\ \max}(n) = a_{1\ \max}n + b_{1\ \max}$ in two different points. Let it be the points $n_1 = 1, n_2 = 2$.

The similar holds for $f_{\mathsf{tails}\ 2}(n)$.

(3) Compute $f_{\mathsf{tails}\ 1}(n)$ in these points.

First, $f_{\mathsf{tails}\ 1}(1) = 1 + f_{\mathsf{tails}\ 1}(0) = 1 + 0 = 1$.

Second, $f_{\mathsf{tails}\ 1}(2) = 1 + f_{\mathsf{tails}\ 1}(1) = 1 + 1 = 2$.

(4) Now, compute $f_{\mathsf{tails}\ 2}(n)$ in these points.

First, $f_{\mathsf{tails}\ 2}(1) = 1$.

Second, $f_{\mathsf{tails}\ 2}(2) = \{f_{\mathsf{tails}\ 2}(1), 2\} = \{1, 2\}$.

(5) For $f_{\mathsf{tails}\ 1}(n)$ (from its values in two points) we compute that $p_{\mathsf{tails}\ 1\ \min}(n) = p_{\mathsf{tails}\ 1\ \max}(n) = f_{\mathsf{tails}\ 1}(n) = n$.

(6) For the inner-size function, we have that $p_{\mathsf{tails}\ 2\ \min}(1) = 1$, $p_{\mathsf{tails}\ 2\ \min}(2) = 1$, $p_{\mathsf{tails}\ 2\ \max}(1) = 1$, $p_{\mathsf{tails}\ 2\ \max}(2) = 2$.

(7) Using the obtained pair of points on the graph of $p_{\mathsf{tails}\ 2\ \min}$ we obtain that $p_{\mathsf{tails}\ 2\ \min}(n) = 1$. Using the obtained pair of points on the graph of $p_{\mathsf{tails}\ 2\ \max}$ we obtain that $p_{\mathsf{tails}\ 2\ \max}(n) = n$.

(8) Now we need to check if we obtained correct upper and lower bounds. Proof of that amounts to proof of correctness of the following typing:

$$\mathsf{tails} : \mathsf{L}_n(\alpha) \to \mathsf{L}_n(\mathsf{L}_{\{i\}_{0\le i\le n}}(\alpha))$$

(9) The proof of correctness reduces to the proof of the following predicates:

- $n = 0 \vdash n = 0$ (from the nil-branch),
- $n \geq 1 \vdash n = 1 + (n - 1)$ (from the cons-branch, for the outer-size function $f_{\mathsf{tails}\ 1}$),
- $n \geq 1, 1 \leq i' \leq n - 1 \vdash \exists i.\ 1 \leq i \leq n \wedge i = i'$ (from the cons-branch, for the inner-size function $f_{\mathsf{tails}\ 2}$, the sizes if the lists in the tail of an output).
- $n \geq 1 \vdash \exists i.\ 1 \leq i \leq n \wedge i = n$ (from the cons-branch, for the inner-size function $f_{\mathsf{tails}\ 2}$, the size of the head of an output).

It is easy to see that these predicates holds.