Alk as a computational model

Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

2022

Introduction

This document includes an almost complete semiformal description of the Alk language as a computational model.

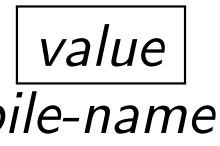
Plan

Memory model

- the memory is a set of variables
- a variable is a pair:

  mathematical notation $\quad$ *variabile-name* $\mapsto$ *value*

  graphical notation

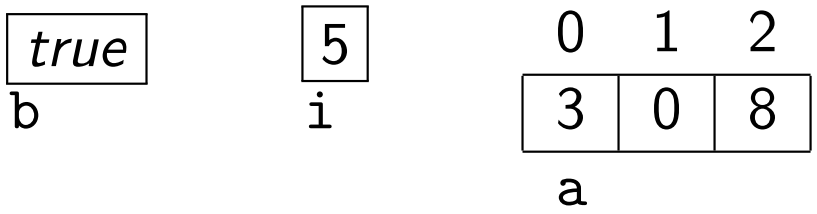
  

- a value is an object of an (abstract) data type
- examples of values:
  scalars
  arrays
  structures (records)
  lists
  maps
  ...
- *Val* denotes the set of all values

Examples of variables

math notation $\quad \mathtt{b} \mapsto \mathit{true} \quad \mathtt{i} \mapsto 5 \quad a \mapsto [3, 0, 8]$

graphical notation



Each notation is in fact the abstract representation of a function $\sigma : \{\mathtt{b}, \mathtt{i}, \ldots\} \to \mathit{Val}$ given by, e.g., $\sigma(\mathtt{b}) = \mathit{true}$, $\sigma(\mathtt{i}) = 5$, ...

Plan

Value dimension

Data type = values (constants) + operations

Each value is represented using a memory space.

For the values of each data type, the dimension/size of representation must be mentioned.

There are three ways to define the dimension of values:

- uniform: $|v|_{\text{unif}}$
- logarithmic: $|v|_{\log}$
- linear: $|v|_{\text{lin}}$

Scalars

booleans, integers, floating point numbers, strings,. . .

An important feature of these values is that they have finite representations.

Scalars (cont)

- integers:
  $Int = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
  - uniform dimension: $|n|_{\text{unif}} = 1$
  - logarithmic dimension: $|n|_{\log} = \log_2 abs(n)$
  - linear dimension: $|n|_{\text{lin}} = abs(n)$
- booleans:
  $Bool = \{false, true\}$
  - uniform dimension: $|b|_{\text{unif}} = 1$
  - logarithmic dimension: $|b|_{\log} = 1$
  - linear dimension: $|b|_{\text{lin}} = 1$
- floating point numbers:
  $Float =$ rational numbers
  - a rational number v is represented by a pair (m, n), where m is the mantissa (significand or coefficient) n is the coefficient
  - so, $|v|_d = |m|_d + |n|_d$, $d \in \{\text{unif}, \log\}$
- ...

We have $Int \cup Bool \cup Float \cup \ldots \subseteq Val$.

Arrays

- $a = [a_0, a_1, \ldots, a_{n-1}]$
- $|a|_d = |a_0|_d + |a_1|_d + \cdots + |a_{n-1}|_d$, $d \in \{\text{unif}, \log, \text{lin}\}$
- $Arr_n\langle V\rangle = \{\{[a_0, a_1, \ldots, a_{n-1}] \mid v_i \in V, i = 0, \ldots, n-1\}$
- $\bigcup_{n\geq 1} Arr_n\langle V\rangle \subset Val$ for each data-type $V \subset Val$
- bidimensional arrays are arrays of unidimensional arrays,
- tridimensional arrays are arrays of bidimensional arrays,
- etc.

Structures (Records)

Example: the plane point $(2, 7)$ is represented by the structure

$$\{\mathrm{x} \to 2\ \mathrm{y} \to 7\}$$

If $F = \{f_1, \ldots, f_n\}$ is the set of fields
then a structure (record) value is of the form $s = \{f_1 \to v_1, \ldots, f_n \to v_n\}$

$|s|_d = |v_0|_d + |v_1|_d + \cdots + |v_{n-1}|_d$, $d \in \{\mathrm{unif}, \log, \mathrm{lin}\}$

$Str\langle f_1{:}V_1, \ldots f_n{:}V_n\rangle = \{\{f_1 \to v_1, \ldots, f_n \to v_n\} \mid v_1 \in V_1, \ldots, f_n \in V_n\}$

Example of Fixed Size Linear Lists:
$FSLL = \{\mathtt{len}, \mathtt{arr}\}$
$Str\langle \mathtt{len} : Int, \mathtt{arr} : Arr_{100}\langle Int\rangle\rangle =$
$\{\{\mathtt{len} \to n\ \mathtt{arr} \to a\} \mid n \in Int, a \in Arr_{100}\langle Int\rangle\}$

$Str\langle f_1{:}V_1, \ldots f_n{:}V_n\rangle \subset Val$ for each structure $F = \{f_1{:}V_1, \ldots f_n{:}V_n\}$.

Linear lists

A list value is a sequence $l = \langle v_0, v_1, \ldots, v_{n-1} \rangle$.

$|l|_d = |v_0|_d + |v_1|_d + \cdots + |v_{n-1}|_d$, $d \in \{\text{unif}, \log, \text{lin}\}$

$LLin\langle V \rangle = \{\langle v_0, \ldots, v_{n-1} \rangle \mid v_i \in V, i = 0, \ldots, n\}$

Example: $LLin\langle Int \rangle$, $LLin\langle Arr_n \rangle$, $LLin\langle Arr_n\langle Float \rangle\rangle$

We have $LLin\langle V \rangle \subset Val$ for each data type V.

Maps

A map m is a set $\{k_1 \mapsto v_1, \ldots, k_n \mapsto v_n\}$ s.t. $i \neq j$ implies $k_i \neq k_j$

$|m|_d = \sum_{i=1}^{n}(|k_i|_d + |v_i|_d,\ d \in \{\text{unif}, \log, \text{lin}\}$

Sets

A set value is of the form $s = \{v_0, v_1, \ldots, v_{n-1}\}$, where $i \neq j$ implies $v_i \neq v_j$

$|s|_d = |v_0|_d + |v_1|_d + \cdots + |v_{n-1}|_d$, $d \in \{\text{unif}, \log, \text{lin}\}$

Complex values: digraphs

The digraph $D = (V, A)$, where $V = \{a, b, c\}$ and $E = \langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle, \langle b, a\rangle, \langle c, a\rangle$ can be represented by a structure (record) of the form (external adjacency lists)

```
D |-> { V -> ["a", "b", "c"]
        adj -> {"a" |-> < "b", "c">
                "b" |-> <"c", "a">
                "c" |-> <"a">
               }
      }
```

or of the form (sets)

```
D |-> { V -> { "a", "b", "c" }
        E -> { < "a", "b">, < "a", "c">,
               < "b", "c">, , < "b", "a">, , < "c", "a">
             }
      }
```

Plan

Data type (cont.)

Data type = objects + operations

Each operation *op* has a time cost $time(op)$.

For each operation of any data type must the cost time must be mentioned.

There three ways to measure the time (inherited from the value dimension):

uniform: $time_{\text{unif}}(op)$ – this does not depend on the dimension of the values
logarithmic: $time_{\log}(op)$ – uses the logarithmic dimension of values
linear: $time_{\text{lin}}(op)$ – uses the linear dimension of values

Operations with scalars

The operations with scalars are the usual ones and we do not list them here.

The uniform time for these operations is $O(1)$.

For the other cases, it is part of the meta-model and it must be specified for each particular analysis.

Operations with compound values

Are the usual ones and we do not list them here.

For all the cases, the time cost is part of the meta-model and it must be specified for each particular analysis.

Plan

Plan

Expressions: syntax

Similar to that of C++:

- arithmetic expressions: `a * b + 2`
- relational expressions: `a < 5`
- boolean expressions: `(a < 5) && (a > -1)`
- set expressions: `s1 U s2` `s1 ^ s2` `s1 \ s2`
- function call: `f(a*2, b+5)`
- operation call for lists/array/...: `l.update(2,55)` `l.size()`

Instructions: syntax

- assignment: `a =` E`;` `a[i] =` E`;` `p.x =` E`;`
- function call: `quicksort(a);` `l.insert(2,77);`
- block: `{` Sts `}`
- conditional instructions:
  `if (` E `)` St
  `if (` E `)` St_1 `else` St_2
- iterative instructions:
  `while (` E `)` St
  `forall` X `in` S St
  `for (` X `=` E`;` E'`;` `++`X `)` S
- return: `return` E`;`
- sequential composition: St_1 St_2

Alk is extendable: it can be added new data type and operations, mentioning the dimensions and resp. the time costs.

Data types

Are predefined in Alk.

It does not exists variable declarations; we assume that there is some meta-information mentioning the type of each variable.

Example of program

```
/*
  This example includes the recursive version of the DFS algorithm.
  @input: a digraf D and a vertex i0
  @output: the list S of the verices reachable from i0
*/

 // the recursive function
 dfsRec(out D, i, out S) {
   if (!(i in S)) {
     // visit i
     S = S U {i};
     foreach j from D.adj[i]
       dfsRec(D, j, S);
   }
 }
```

```
// the calling algorithm
dfs(out D, i0) {
  S = emptySet;
  dfsRec(D, i0, S);
  return S;
}

// example of use
reached = dfs(D, i0);
```

Plan

Semantics: Expressions valuation

Consider a function $[\![\_]\!](\_) : \mathit{Expresii} \to (\mathit{Stare} \to \mathit{Valori})$, where $[\![E]\!](\sigma)$ return the value of the expression E computed in the state σ.

Example: Let σ be a state that includes $\mathtt{a} \mapsto 3 \quad \mathtt{b} \mapsto 6$. We have:
$[\![\mathtt{a} + \mathtt{b} * 2]\!](\sigma) =$
$[\![\mathtt{a}]\!](\sigma) +_{Int} [\![\mathtt{b} * 2]\!](\sigma) =$
$3 +_{Int} [\![b]\!](\sigma) *_{Int} [\![2]\!](\sigma) =$
$3 +_{Int} 6 *_{Int} 2 =$
$3 +_{Int} 12 = 15$

where $+_{Int}$ represents the algorithm for integer addition and $*_{Int}$ represents the algorithm for integer multiplication.

Time cost for evaluation

$time_d([\![\mathtt{a} + \mathtt{b} * 2]\!](\sigma)) =$
$time_{\mathtt{d}}([\![\mathtt{a}]\!](\sigma)) + time_{\mathtt{d}}([\![\mathtt{b}]\!](\sigma)) + time_{\mathtt{d}}(6 *_{Int} 2) + time_{\mathtt{d}}(3 +_{Int} 122)$,
$d \in \{\mathrm{unif}, \log, \mathrm{lin}\}$.

$\sigma = \mathtt{a} \mapsto 3 \quad \mathtt{b} \mapsto 6$

$time_{\log}([\![\mathtt{a}]\!](\sigma)) = \log 3,\ time_{\log}([\![\mathtt{b}]\!](\sigma)) = \log 6$
$time_{\mathrm{unif}}([\![\mathtt{a}]\!](\sigma)) = 1,\ time_{\mathrm{unif}}([\![\mathtt{b}]\!](\sigma)) = 1$
$time_{\mathrm{lin}}([\![\mathtt{a}]\!](\sigma)) = 3,\ time_{\mathrm{lin}}([\![\mathtt{b}]\!](\sigma)) = 6$

Semantics: Configurations

A configuration is a pair $\langle \textit{piece-of-program}, \textit{state} \rangle$

Example:

$\langle \texttt{x = x + 1; y = y + 2 * x;}, \quad \texttt{x} \mapsto 7\ \texttt{y} \mapsto 12 \rangle$

$\langle \texttt{s = 0; while (x > 0) \{s = s+x; x = x-1;\}}, \quad \texttt{x} \mapsto 5\ \texttt{s} \mapsto -15 \rangle$

Semantics: Execution steps

An execution step is a transition relation between configurations:

$$\langle S, \sigma \rangle \Rightarrow \langle S', \sigma' \rangle$$

iff

executing the first instruction from S in the state σ we obtain the piece of prgram S', which follows to be executed in the state σ'

Execution steps are described by rules $\langle S_1, \sigma_1 \rangle \Rightarrow \langle S_2, \sigma_2 \rangle$, where $S_1, S_2, \sigma_1, \sigma_2$ are terms with variables (patterns).

To compute the time of an execution step, we describe how compute the time for each rule application.

Semantics: Assignment

assignment: $\mathtt{x} = E;$

- *informal*: evaluate E and assign the result to the variable $\mathtt{x}$
- *formal*:
  $\langle \mathtt{x} = E; S, \sigma \} \rangle \Rightarrow \langle S, \sigma' \rangle$
  where σ of the form $\ldots \mathtt{x} \mapsto v \ldots$ and σ' dof the form
  $\ldots \mathtt{x} \mapsto [\![\mathtt{E}]\!](\sigma) \ldots$ (the rest is the same as in σ).

Time cost:
$time_d(\langle \mathtt{x} = E; S, \sigma \rangle \Rightarrow \langle S, \sigma' \rangle) = time_d([\![E]\!](\sigma)) + |[\![E]\!](\sigma)|_d$
where $d \in \{\mathrm{unif}, \log, \mathrm{lin}\}$.

Semantics: `if` Command

if: `if (`E`) then` S `else` S'

- *informal*: evaluate e; if the result is *true*, then execute S, else execute S'
- *formal*:

  $\langle$`if (`E`) `S` else `S' $S'', \sigma\rangle \Rightarrow \langle S\ S'', \sigma\rangle$ dacă $[\![E]\!](\sigma) = \mathit{true}$

  $\langle$`if (`E`) `S` else `S' $S'', \sigma\rangle \Rightarrow \langle S'\ S'', \sigma\rangle$ dacă $[\![E]\!](\sigma) = \mathit{false}$

Time cost:
$\mathit{time}_d(\langle$`if (`E`) `S'` else `S'' $S, \sigma\rangle \Rightarrow \langle \_, \sigma\rangle) = \mathit{time}_d([\![E]\!](\sigma))$
$d \in \{\mathrm{unif}, \mathrm{log}, \mathrm{lin}\}$.

Semantics: `while` command

while: `while (`E`)` S

- *informal*: evaluate e; if the result is *true*, then execute S, then evaluate again e and ...; otherwise the execution of the instruction stops
- *formal*: it is described using `if`:
  $\langle$`while (`e`)` S $S', \sigma\rangle \Rightarrow$
  $\langle$`if (`e`) { `S `; while (`e`)` S `} else { }`$S', \sigma\rangle$

Time cost:
$time_d(\langle$`while (`E`) then` S `else` S' $S, \sigma\rangle \Rightarrow \langle$`if (`$e$`) ...`$S, \sigma\rangle) = 0$,
$d \in \{\text{unif}, \log, \text{lin}\}$.

Semantics: Function call

Consider `f(a,b)` $\{\ S_f\ \}$.

We have to add stacks to the configurations.

The evaluation `f`(e_1,e_2) consists of:

$\langle \texttt{f}(e_1, e_2)\ S, \sigma, \mathit{Stack}\rangle \Rightarrow$

$\langle S_f, \sigma \cup \{\texttt{a} \mapsto [\![e_1]\!](\sigma)\ \texttt{b} \mapsto [\![e_2]\!](\sigma)\}, (S, \sigma)\ \mathit{Stack}\rangle \Rightarrow^*$

$\langle v, \sigma', (S, \sigma)\ \mathit{Stack}\rangle \Rightarrow$

$\langle v\ S, \mathit{updateGlobals}(\sigma, \sigma'), \mathit{Stack}\rangle$

Assumption: the time cost of a function call is the sum of time for parameters evaluation and the time for executing the function body.

Computation (execution)

A computation (an execution) is a sequence of execution steps:
$\tau = \langle S_1, \sigma_1 \rangle \Rightarrow \langle S_2, \sigma_2 \rangle \Rightarrow \langle S_3, \sigma_3 \rangle \Rightarrow \dots$

The cost of a computation:
$time_d(\tau) = \sum_i time_d(\langle S_i, \sigma_i \rangle \Rightarrow \langle S_{i+1}, \sigma_{i+1} \rangle),\ d \in \{unif, log, lin\}$

Computation: example

$\langle$ `if (x > 3) x = x + y; else x = 0; y = 4;` $, x \mapsto 7\ y \mapsto 12\rangle \Rightarrow$
$\langle$ `x = x + y; y = 4;` $, x \mapsto 7\ y \mapsto 12\rangle \Rightarrow$
$\langle$ `y = 4;` $, x \mapsto 19\ y \mapsto 12\rangle \Rightarrow$
$\langle \cdot, x \mapsto 19\ y \mapsto 4\rangle$

We used:
$[\![x > 3]\!](x \mapsto 7\ y \mapsto 12) = \mathit{true}$
$[\![x + y]\!](x \mapsto 7\ y \mapsto 12) = 19$
$[\![4]\!](x \mapsto 19\ y \mapsto 12) = 4$

The cost:
uniform cost: 3 (= the number of steps)
logarithmic cost: $\log 7 + \log 12 + \log 19 + \log 4$
linear cost: $7 + 12 + 19 + 4$

Computation: example

$\langle$ `while (i > 5) i--;` $, \mathtt{i} \mapsto 6\ \mathtt{x} \mapsto 12\rangle \Rightarrow$
$\langle$ `if (i > 5) { i --; while (i > 5) i--; }` $, \mathtt{i} \mapsto 6\ \mathtt{x} \mapsto 12\rangle \Rightarrow$
$\langle$ `{i --; while (i > 5) i--; }` $, \mathtt{i} \mapsto 6\ \mathtt{x} \mapsto 12\rangle \Rightarrow$
$\langle$ `i --; while (i > 5) i--;` $, \mathtt{i} \mapsto 6\ \mathtt{x} \mapsto 12\rangle \Rightarrow$
$\langle$ `while (i > 5) i--;` $, \mathtt{i} \mapsto 5\ \mathtt{x} \mapsto 12\rangle \Rightarrow$
$\langle$ `if (i > 5) { i --; while (i > 5) i--; }` $, \mathtt{i} \mapsto 5\ \mathtt{x} \mapsto 12\rangle \Rightarrow$
$\langle \cdot, \mathtt{i} \mapsto 5\ \mathtt{x} \mapsto 12\rangle$

We used:
$[\![\mathtt{i} > 5]\!](\mathtt{i} \mapsto 6\ \mathtt{x} \mapsto 12) = \mathit{true}$
$[\![\mathtt{i}--]\!](\mathtt{i} \mapsto 6\ \mathtt{x} \mapsto 12) = 5$
$[\![\mathtt{i} > 5]\!](\mathtt{i} \mapsto 5\ \mathtt{x} \mapsto 12) = \mathtt{false}$

The cost:
uniform cost: 5 (= the number of steps)
logarithmic cost: $\log 6 + \log 6 + \log 5$