Executable Semantics for D-Clean

Viktória Zsók Pieter Koopman Rinus Plasmeijer

Department of Programming Languages and Compilers, Faculty of Informatics,
Eötvös Loránd University, Budapest, Hungary
Nijmegen Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands [1]

[1] Supported by TÁMOP-4.2.1/B-09/1/KMR-2010-0003

Overview

D-Clean at a glance

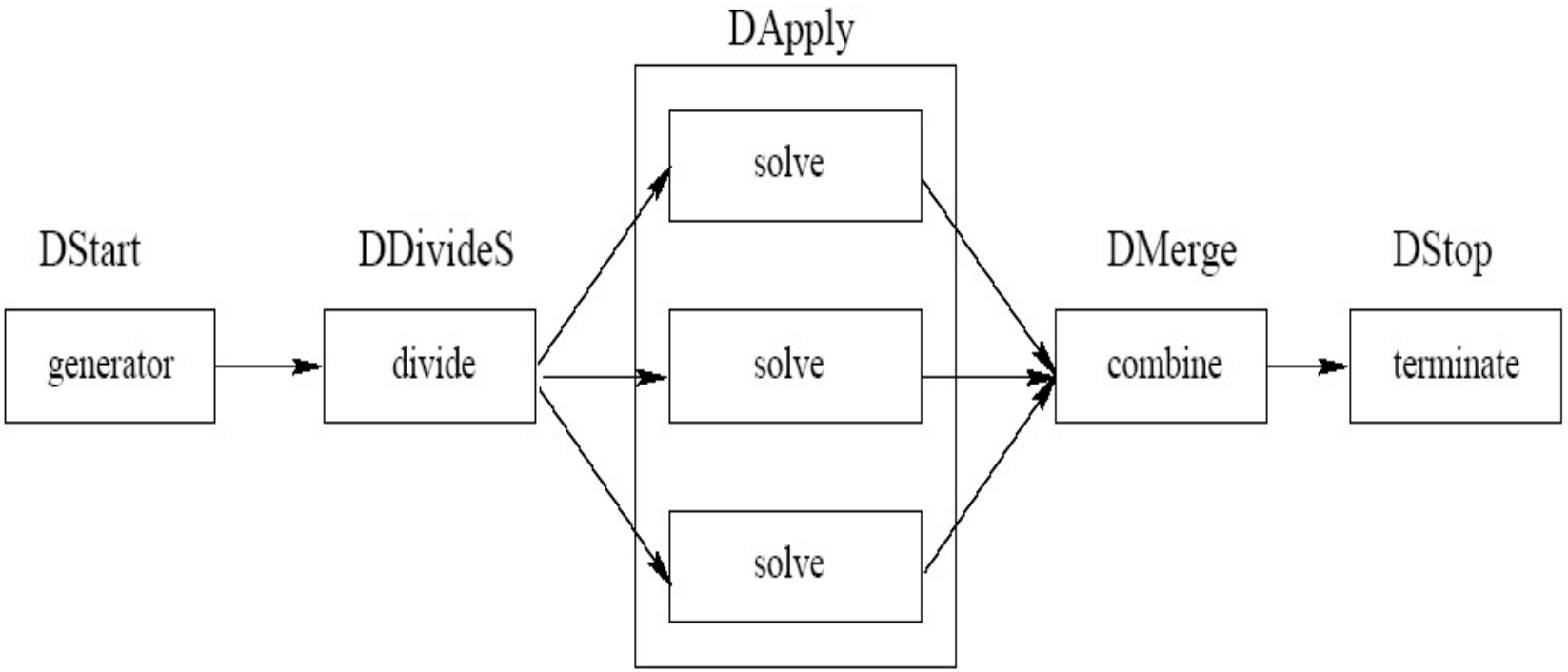
- A Clean-like language with distributed computations over a cluster
- Preserves the functional style
- Tasks are pure functional computations
- The dataflow is controlled by the coordination language elements
- Common algorithmic skeletons can be written elegantly using the newly defined language elements
- A D-Clean program consists of a start expression = composition of coordination structures parameterized by Clean expressions

Execution model

- The distributed computation is described using D-Clean primitives
- The primitives are compiled into D-Box expressions
- D-Box primitives are generating Clean computation nodes
- The nodes are communicating via channels
- The middleware will distribute the nodes over the cluster, where they will be started as individual Clean programs

Small example

```
DistrStart = (DStop terminate) o (DMerge mergesort) o
             (DApply quicksort) o (DDivideS divide 3) o
             (DStart [1..100000])
```



Why executable version?

- the well-formedness of the semantics can be checked by the Clean compiler
- the semantics can be executed
- properties of the semantics can be tested automatically
- visualization can be generated showing the parallelism automatically

BNF

Instead of an ugly BNF description:

⟨DISTART_RULE⟩ :=="DistrStart" "=" ⟨DEXPR⟩
⟨DEXPR⟩ :=⟨DPRIMITIVE⟩ | ⟨SCHEME_NAME⟩ { ⟨act_param⟩ }*
| ⟨DEXPR⟩ ⟨DEXPR⟩
⟨DPRIMITIVE⟩ :=⟨DStart_USE⟩ | ⟨DStop_USE⟩ | ⟨DMap_USE⟩ |
⟨DDivideS_USE⟩ | ⟨DMerge_USE⟩
⟨SCHEME_DEF⟩ :="SCHEME" ⟨SCHEME_NAME⟩ { ⟨formal_param⟩}*
”=” { ⟨DEXPR⟩ }+
⟨SCHEME_NAME⟩ :=={⟨UpCaseLetter⟩ }+

we provide Clean datatypes and executable programs.

Used types for executable semantics

```
// either a function or a composition of functions
:: DExpr a b =   F String (a → b)
               | D (DFun (Ch a) (Ch b))

// a DClean expression is a state transition function
:: DFun a b  :== a State → (b, State)

// single channel
:: Ch a
// multiple channel
:: MCh a :== [Ch a]

// wrapper
   DExec :: a (DFun a b) → (a, (b, State))
   DExec a expr  = (a, DStop (DStart a expr (0,[])))
```

D-Clean primitives

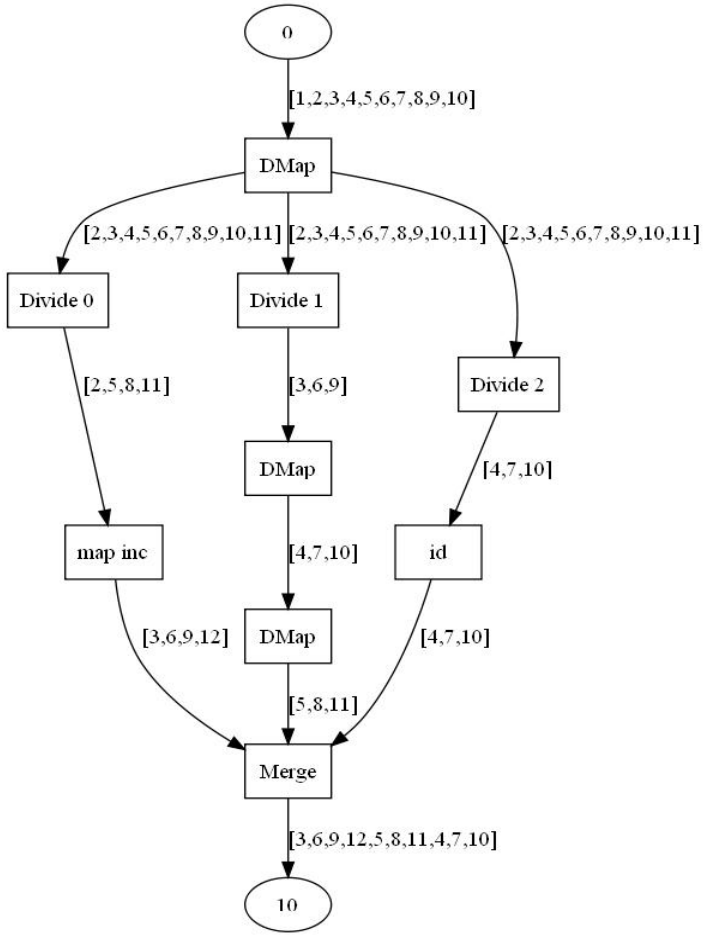
```
DApply   :: String (a → b)   → DFun (Ch a)   (Ch b)   | toString b

DDivideS :: (Int [a] → [[a]]) Int → DFun (Ch [a]) (MCh [a])
            | toString a
DDivideD :: (a → [b]) → DFun (Ch a)   (MCh b) | toString b

DMerge   :: ([b] → a) → DFun (MCh b) (Ch a)    | toString a
```

Farm

```
Start w = dumpGraph (DExec myval myflow) w
where
myval = [1..10]
myflow i
    = DMap inc (0,i)
  >>= DDivideD (divide 3)
  >>= DApplyN [F "map inc" (map inc),
       D (λi → DMap inc i >>= DMap inc),
       F "id" id]
  >>= DMerge flatten
```



D-Clean primitives

```
DApplyN :: [DExpr a b] → DFun (MCh a) (MCh b) | toString b
DApplyN funs = handleDExpr funs
where
handleDExpr :: [DExpr a b] (MCh a) State → (MCh b, State)
                    | toString b
handleDExpr []     [] state  = ([], state)
handleDExpr []     inflows state  = abort "run-time error"
handleDExpr [F name fun:funs]  [i:is] state
# (result,state) =  mkFunBox name fun i state
# (results,state) =  handleDExpr funs is state
= ([result:results],state)
handleDExpr [D dfun:funs] [i:is] state
# (result,state) =  dfun i state
# (results,state) =  handleDExpr funs is state
= ([result:results],state)
```

Utility primitives

```
DMap :: (a → b) → DFun (Ch [a]) (Ch [b]) | toString b

DReduce :: ([b] → a) → DFun (MCh [b]) (MCh a) | toString a

DProduce :: (a → [b]) → DFun (MCh a) (MCh [b]) | toString b

DFilter :: (a → Bool) → DFun (MCh [a]) (MCh [a]) | toString a
```

Code transformations

- optimization on the amount of communications on the channels
- change on the granularity of the computations

```
    DApplyN [F "f1" f1, .., F "fn" fn]
>>= DApplyN [F "g1" g1, .., F "gn" gn]
≡ DApplyN [F "fg1" (g1 o f1), .., F "fgn" (gn o fn)]
```

Properties

For all lists and appropriate functions f:

```
map f list == DDivideD (divide n) (0 ,list)
              >>= DApply1 (F "f" f)
              >>= DMerge id
```

To test the correctness of the transformations: for all channels c and appropriate functions f and g:

```
(DApply1 (F "f" f) >>= DApply1 (F "g" g)) c

== DApply1 (F "f" (g o f)) c
```

Conclusions and future work

- Executable semantics for the D-Clean primitives
- To mimic the functionality of the distributed programs
- Sanity check of the semantics
- Transformations and properties - decreasing the workload of the channels
- Visualizing parallelism
- Examples