External Functions

LibraryStrings

In pure B there are only two built-in operators on strings: equality $=$ and inequality $\neq$. This library provides several string manipulation functions, and assumes that STRINGS are sequences of unicode characters (in UTF-8 encoding). You can obtain the definitions below by putting the following into your DEFINITIONS clause:

```
DEFINITIONS "LibraryStrings.def"
```

The file `LibraryStrings.def` is bundled with ProB and can be found in the `stdlib` folder. You can also include the machine `LibraryStrings.mch` instead of the definition file; the machine defines some of the functions below as proper B functions (i.e., functions for which you can compute the domain and use constructs such as relational image).

In [29]:
```
::load
MACHINE Jupyter_LibraryStrings
DEFINITIONS "LibraryStrings.def"
END
```

Out[29]:
```
Loaded machine: Jupyter_LibraryStrings
```

STRING_APPEND

This external function takes two strings and concatenates them.

Type: $STRING \times STRING \rightarrow STRING$

In [2]:
```
STRING_APPEND("abc","abc")
```

Out[2]:
```
"abcabc"
```

In [3]:
```
STRING_APPEND("abc","")
```

Out[3]:
```
"abc"
```

STRING_LENGTH

This external function takes a string and returns the length.

Type: $STRING \rightarrow INTEGER$

In [4]:
```
STRING_LENGTH("abc")
```

Out[4]:
```
3
```

In [5]:
```
1 STRING_LENGTH("")
```

Out[5]: 0

STRING_SPLIT

This external function takes two strings and separates the first string according to the separator specified by the second string.

Type: $STRING \times STRING \to seq(STRING)$

In [6]:
```
1 STRING_SPLIT("filename.ext",".")
```

Out[6]: {(1↦"filename"),(2↦"ext")}

In [7]:
```
1 STRING_SPLIT("filename.ext","/")
```

Out[7]: {(1↦"filename.ext")}

In [8]:
```
1 STRING_SPLIT("usr/local/lib","/")
```

Out[8]: ["usr","local","lib"]

In [9]:
```
1 STRING_SPLIT("",".")
```

Out[9]: {(1↦"")}

I am not sure the following result makes sense, maybe a sequence of all characters is more appropriate?

In [10]:
```
1 STRING_SPLIT("usr/local/lib","")
```

Out[10]: {(1↦"usr/local/lib")}

In [11]:
```
1 STRING_SPLIT("usr/local/lib","cal")
```

Out[11]: {(1↦"usr/lo"),(2↦"/lib")}

STRING_JOIN

This external function takes a sequence of strings and a separator string and joins the strings together inserting the separators as often as needed. It is the inverse of the `STRING_SPLIT` function.

Type: $seq(STRING) \times STRING \to STRING$

In [12]:
```
1 STRING_JOIN(["usr","local","lib"],"/")
```

Out[12]: "usr/local/lib"

In [13]:
```
1 STRING_JOIN(["usr/lo","/lib"],"cal")
```

Out[13]: "usr/local/lib"

In [14]:
```
1 STRING_JOIN(["usr/local/lib"],"")
```

Out[14]: "usr/local/lib"

STRING_CHARS

This external function takes a strings splits it into a sequence of the individual characters. Each character is represented by a string.

Type: $STRING \rightarrow seq(STRING)$

In [15]:
```
1 STRING_CHARS("")
```

Out[15]: ∅

In [16]:
```
1 STRING_CHARS("abc")
```

Out[16]: ["a","b","c"]

In [17]:
```
1 STRING_JOIN(STRING_CHARS("abc"),".")
```

Out[17]: "a.b.c"

STRING_CODES

This external function takes a strings splits it into a sequence of the individual characters. Each character is represented by a natural number (the ASCII or Unicode representation of the character).

Type: $STRING \rightarrow seq(INTEGER)$

In [18]:
```
1 STRING_CODES("")
```

Out[18]: ∅

In [19]:
```
1 STRING_CODES("AZ az 09")
```

Out[19]: [65,90,32,97,122,32,48,57]

STRING_IS_INT

This external predicate takes a string and is true if the string represents an integer.

Type: $STRING.$

In [20]:
```
1 STRING_IS_INT("1204")
```

Out[20]: TRUE

In [21]:
```
1 STRING_IS_INT("-1204")
```

Out[21]: TRUE

In [22]:
```
1 STRING_IS_INT(" - 1204")
```

Out[22]: TRUE

In [23]:
```
1 STRING_IS_INT("1.1")
```

Out[23]: FALSE

In [24]:
```
1 STRING_IS_INT("1.0")
```

Out[24]: FALSE

In [25]:
```
1 STRING_IS_INT("a")
```

Out[25]: FALSE

In [26]:
```
1 STRING_IS_INT("10000000000000000000000000")
```

Out[26]: TRUE

In [27]:
```
1 STRING_IS_INT("-00001")
```

Out[27]: TRUE

In [28]:
```
1 STRING_IS_INT("00002")
```

Out[28]: TRUE

STRING_TO_INT

This external function takes a string and converts it into an integer. An error is raised if this cannot be done. It is safer to first check with `STRING_IS_INT` whether the conversion can be done.

Type: $STRING \rightarrow INTEGER$

In [29]:
```
1 STRING_TO_INT("1024")
```

Out[29]: 1024

In [30]:
```
1 STRING_TO_INT(" - 00001")
```

Out[30]: -1

INT_TO_STRING

This external function converts an integer to a string representation.

Type: $INTEGER \rightarrow STRING$.

```
In [31]:  1  INT_TO_STRING(1024)
```

```
Out[31]: "1024"
```

```
In [32]:  1  INT_TO_STRING(-1024)
```

```
Out[32]: "-1024"
```

```
In [33]:  1  INT_TO_STRING(STRING_TO_INT(" - 00001"))
```

```
Out[33]: "-1"
```

```
In [34]:  1  STRING_TO_INT(INT_TO_STRING(-1))=-1
```

```
Out[34]: TRUE
```

DEC_STRING_TO_INT

This external function takes a decimal string (with optional decimal places) and converts it to an integer with the given precision (rounding if required).

Type: $STRING \times INTEGER \rightarrow INTEGER$

```
In [35]:  1  DEC_STRING_TO_INT("1024",0)
```

```
Out[35]: 1024
```

```
In [36]:  1  DEC_STRING_TO_INT("1024",2)
```

```
Out[36]: 102400
```

```
In [37]:  1  DEC_STRING_TO_INT("1024",-1)
```

```
Out[37]: 102
```

```
In [38]:  1  DEC_STRING_TO_INT("1025",-1)
```

```
Out[38]: 103
```

```
In [39]:  1  DEC_STRING_TO_INT(" -1025",-1)
```

```
Out[39]: -103
```

```
In [40]:  1  DEC_STRING_TO_INT("1024.234",2)
```

```
Out[40]: 102423
```

In [41]:
```
DEC_STRING_TO_INT("1024",100)
```
Out[41]:
```
1024000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000
```

In [42]:
```
DEC_STRING_TO_INT("10000000000000000000000000000000000",-32)=100
```
Out[42]:
```
TRUE
```

INT_TO_DEC_STRING

This external function converts an integer to a decimal string representation with the precision provided by the second argument.

Type: $INTEGER \times INTEGER \rightarrow STRING$

In [43]:
```
INT_TO_DEC_STRING(1204,2)
```
Out[43]:
```
"12.04"
```

In [44]:
```
INT_TO_DEC_STRING(-1204,3)
```
Out[44]:
```
"-1.204"
```

In [45]:
```
INT_TO_DEC_STRING(0,2)
```
Out[45]:
```
"0.00"
```

In [46]:
```
INT_TO_DEC_STRING(1204,-2)
```
Out[46]:
```
"120400"
```

In [47]:
```
INT_TO_DEC_STRING(-10,3)
```
Out[47]:
```
"-0.010"
```

INT_TO_HEX_STRING

This external function converts an integer to a hexadecimal string representation.

Type: $INTEGER \rightarrow STRING.$

In [3]:
```
INT_TO_HEX_STRING(254)
```
Out[3]:
```
"fe"
```

In [4]:
```
INT_TO_HEX_STRING(0)
```
Out[4]:
```
"0"
```

In [5]:
```
1 INT_TO_HEX_STRING(-254)
```

Out[5]: "-fe"

In [7]:
```
1 INT_TO_HEX_STRING(2**100-1)
```

Out[7]: "fffffffffffffffffffffffff"

TO_STRING

This external function converts a B data value to a string representation.

Type: $\tau \rightarrow STRING$.

In [48]:
```
1 TO_STRING(1024)
```

Out[48]: "1024"

In [49]:
```
1 TO_STRING("1024")
```

Out[49]: "1024"

In [50]:
```
1 TO_STRING({2,3,5})
```

Out[50]: "{2,3,5}"

In [51]:
```
1 TO_STRING((TRUE,3,{11|->rec(a:22,b:33)}))
```

Out[51]: "((TRUE|->3)|->{(11|->rec(a:22,b:33))})"

FORMAT_TO_STRING

This external function takes a format string and a B sequence of values and generates an output string, where the values have been inserted into the format string in place of the `~w` placeholders.

- the length of sequence must correspond to the number of `~w` in the format string.
- the format string follows the conventions of SICStus Prolog. E.g., one can use `~n` for newlines.

Type: $(STRING * seq(\tau)) \rightarrow STRING$

In [52]:
```
1 FORMAT_TO_STRING("two to the power ten = ~w",[2**10])
```

Out[52]: "two to the power ten = 1024"

In [53]:
```
1 FORMAT_TO_STRING("My two sets are ~w and ~w",[1..2,2..1])
```

Out[53]: "My two sets are {1,2} and {}"

Format Strings

Various external functions and predicates work with format strings. ProB uses the conventions of the SICStus Prolog format string.

- `~n` inserts a newline into the generated output
- `~Nn` where N is a number: it inserts N newlines into the output
- `~w` inserts the next argument into the generated output
- `~i` consumes the next argument but ignores it; i.e., nothing is inserted into the output
- `~~` inserts the tilde symbol into the generated output
- `~N` inserts a newline if not at the beginning of the line

SICStus Prolog also uses a few other formatting codes, such as `~@` , `~p` ,... which should not be used.

STRINGIFY

This external function converts a B expression to a string representation of the expression, not the value. It can be used to obtain the name of variables. Warning: ProB may simplify and rewrite expressions (you can turn this off by setting the OPTIMIZE_AST preference to false).

Type: $\tau \rightarrow STRING$.

In [30]:
```
1 STRINGIFY(dom({1|->2}))
```

Out[30]:
```
"dom({1 |-> 2})"
```

Compare this with the result of TO_STRING:

In [31]:
```
1 TO_STRING(dom({1|->2}))
```

Out[31]:
```
"{1}"
```

In [34]:
```
1 :table rec(stringify:STRINGIFY("abc"),tostring:TO_STRING("abc"))
```

Out[34]:

stringify	tostring
"\"abc\""	"abc"

Choose Operator

You can obtain access to the definitions below by putting the following into your DEFINITIONS clause: `DEFINITIONS "CHOOSE.def"`

Choose

This external function takes a set and returns an element of the set. This is a proper mathematical function, i.e., it will always return the same value given the same argument. It is also known as Hilbert's operator.

The operator raises an error when it is called with an empty set. Also, it is not guaranteed to work for infinite sets.

Type: $POW(T) \rightarrow T$.

In [54]:
```
::load
MACHINE Jupyter_CHOOSE
DEFINITIONS "CHOOSE.def"
END
```

Out[54]: `Loaded machine: Jupyter_CHOOSE`

In [55]:
```
CHOOSE(1..3)
```

Out[55]: 1

In [56]:
```
CHOOSE({1,2,3})
```

Out[56]: 1

In [57]:
```
CHOOSE({"a","b","c"})
```

Out[57]: "a"

In [58]:
```
CHOOSE(NATURAL)
```

Out[58]: 0

In [59]:
```
CHOOSE(INTEGER)
```

Out[59]: 0

The operator is useful for writing WHILE loops or recursive functions which manipulate sets. The following example defines a recursive summation function using the CHOOSE operator.

```
MACHINE RecursiveSigmaCHOOSEv3
DEFINITIONS
  "Choose.def"
ABSTRACT_CONSTANTS sigma
PROPERTIES
  sigma: POW(INTEGER) <-> INTEGER &
  sigma = %x.(x:POW(INTEGER) |
              IF x={} THEN 0 ELSE
                LET c BE c=CHOOSE(x) IN c+sigma(x-{c}) END
              END
              )
ASSERTIONS
 sigma({3,5,7}) = 15;
END
```

Sorting Sets

You can obtain access to the definitions below by putting the following into your DEFINITIONS clause: `DEFINITIONS "SORT.def"`

Alternatively you can use the following if you use ProB prior to version 1.7.1: `DEFINITIONS`

```
SORT(X) == [];
EXTERNAL_FUNCTION_SORT(T) == (POW(T)-->seq(T));
```

This external function SORT takes a set and translates it into a B sequence. It uses ProB's internal order for sorting the elements. It will not work for infinite sets. Type: $POW(\tau) \rightarrow seq(\tau)$.

In [2]:
```
::load
MACHINE Jupyter_SORT
DEFINITIONS "SORT.def"
END
```

Out[2]: `Loaded machine: Jupyter_SORT`

In [61]:
```
SORT(1..3)
```

Out[61]: `[1,2,3]`

In [62]:
```
SORT({3*3,3+3,3**3})
```

Out[62]: `[6,9,27]`

In [63]:
```
SORT({"ab","aa","a","b","10","1","2","11"})
```

Out[63]: `["1","10","11","2","a","aa","ab","b"]`

In [64]:
```
1 SORT({("a"|->1),("b"|->0),("a"|->0)})
```

Out[64]:
```
[("a"↦0),("a"↦1),("b"↦0)]
```

A related external function is LEQ_SYM_BREAK which allows one to compare values of arbitrary type. Calls to this external function are automatically inserted by ProB for symmetry breaking of quantifiers. It should currently not be used for sets or sequences.

The SORT.def file also contains a definition for the SQUASH operator which takes a sequence with gaps and completes it into a proper sequence:

In [6]:
```
1 SQUASH({0|->"a",100|->"c",1001 |->"d",4|->"b", 44|->"c"})
```

Out[6]:
```
["a","b","c","c","d"]
```

LibraryMeta

This library provides various meta information about ProB and the current model. You can obtain the definitions below by putting the following into your DEFINITIONS clause:

```
DEFINITIONS "LibraryMeta.def"
```

The file `LibraryMeta.def` is also bundled with ProB and can be found in the `stdlib` folder.

In [1]:
```
1 ::load
2 MACHINE Jupyter_LibraryMeta
3 DEFINITIONS "LibraryMeta.def"
4 END
```

Out[1]:
```
Loaded machine: Jupyter_LibraryMeta
```

PROB_INFO_STR

This external function provides access to various information strings about ProB. Type: $STRING \rightarrow STRING$.

In [66]:
```
1 PROB_INFO_STR("prob-version")
```

Out[66]:
```
"1.8.2-beta2"
```

In [67]:
```
1 PROB_INFO_STR("prob-revision")
```

Out[67]:
```
"ce702ba99f667cb03de8ed41ab58ba72db9112c3"
```

In [68]:
```
1 PROB_INFO_STR("prob-last-changed-date")
```

Out[68]:
```
"Fri Aug 10 17:40:37 2018 +0200"
```

In [69]:
```
1 PROB_INFO_STR("java-version")
```

Out[69]: "1.8.0_172-b11"

In [70]:
```
1 PROB_INFO_STR("java-command-path")
```

Out[70]: "/Library/Java/JavaVirtualMachines/jdk1.8.0_172.jdk/Contents/Home/bin/java"

In [71]:
```
1 PROB_INFO_STR("current-time")
```

Out[71]: "13/8/2018 - 14h34 49s"

Another command is PROB_INFO_STR("parser-version") which does not work within Jupyter.

PROB_STATISTICS

This external function provides access to various statistics in the form of integers about ProB.
Type: $STRING \rightarrow INTEGER$

In [72]:
```
1 PROB_STATISTICS("prolog-memory-bytes-used")
```

Out[72]: 150940944

In [73]:
```
1 PROB_STATISTICS("states")
```

Out[73]: 1

In [74]:
```
1 PROB_STATISTICS("transitions")
```

Out[74]: 0

In [75]:
```
1 PROB_STATISTICS("processed-states")
```

Out[75]: 0

In [76]:
```
1 PROB_STATISTICS("current-state-id")
```

Out[76]: -1

In [77]:
```
1 PROB_STATISTICS("now-timestamp")
```

Out[77]: 1534163689

In [78]:
```
1 PROB_STATISTICS("prolog-runtime")
```

Out[78]: 1660

In [79]:
```
1 PROB_STATISTICS("prolog-walltime")
```

Out[79]: 2890

Other possible information fields are prolog-memory-bytes-free, prolog-global-stack-bytes-used, prolog-local-stack-bytes-used, prolog-global-stack-bytes-free, prolog-local-stack-bytes-free, prolog-trail-bytes-used, prolog-choice-bytes-used, prolog-atoms-bytes-used, prolog-atoms-nb-used, prolog-gc-count, prolog-gc-time.

PROJECT_STATISTICS

This external function provides access to various statistics in the form of integers about the current specification being processed, with all auxiliary files (i.e., project). Type: $STRING \rightarrow INTEGER$

```
In [80]:  1 PROJECT_STATISTICS("constants")
```

```
Out[80]: 0
```

```
In [81]:  1 PROJECT_STATISTICS("variables")
```

```
Out[81]: 0
```

```
In [82]:  1 PROJECT_STATISTICS("properties")
```

```
Out[82]: 0
```

```
In [83]:  1 PROJECT_STATISTICS("invariants")
```

```
Out[83]: 0
```

```
In [84]:  1 PROJECT_STATISTICS("operations")
```

```
Out[84]: 0
```

```
In [85]:  1 PROJECT_STATISTICS("static_assertions")
```

```
Out[85]: 0
```

```
In [86]:  1 PROJECT_STATISTICS("dynamic_assertions")
```

```
Out[86]: 0
```

PROJECT_INFO

This external function provides access to various information strings about the current specification being processed, with all auxiliary files (i.e., project). Type: $STRING \rightarrow POW(STRING)$

```
In [87]:  1 PROJECT_INFO("files")
```

```
Out[87]: {"(machine from Jupyter cell).mch","LibraryMeta.def"}
```

In [88]:
```
1 PROJECT_INFO("main-file")
```

Out[88]: {"(machine from Jupyter cell).mch"}

In [89]:
```
1 PROJECT_INFO("variables")
```

Out[89]: ∅

In [90]:
```
1 PROJECT_INFO("constants")
```

Out[90]: ∅

In [91]:
```
1 PROJECT_INFO("sets")
```

Out[91]: ∅

In [92]:
```
1 PROJECT_INFO("operations")
```

Out[92]: ∅

In [93]:
```
1 PROJECT_INFO("assertion_labels")
```

Out[93]: ∅

In [94]:
```
1 PROJECT_INFO("invariant_labels")
```

Out[94]: ∅

In [2]:
```
1 PROJECT_INFO("sha-hash")
```

Out[2]: {"5d45f08d7e5cf22716b8fd3dd54a29b4ba4c443c"}

MACHINE_INFO

This external function provides access to various information strings about B machines being processed. Type: $STRING \rightarrow STRING$.

In [10]:
```
1 MACHINE_INFO("Jupyter_LibraryMeta","TYPE")
```

Out[10]: "abstract_machine"

LibraryIO

This library provides various input/output facilities. It is probably most useful for debugging, but can also be used to write B machines which can read and write data. You can obtain the definitions below by putting the following into your DEFINITIONS clause:

```
DEFINITIONS "LibraryIO.def"
```

The file `LibraryIO.def` is also bundled with ProB and can be found in the `stdlib` folder.

LibraryXML

This library provides various functions to read and write XML data from file and strings. You can obtain the definitions below by putting the following into your DEFINITIONS clause:

```
DEFINITIONS "LibraryXML.def"
```

The file `LibraryXML.def` is also bundled with ProB and can be found in the `stdlib` folder.

Internal Data Type

An XML document is represented using the type seq(XML_ELement_Type), i.e., a sequence of XML elements, whose type is defined by the following (included in the LibraryXML.def file):

```
XML_ELement_Type ==
     struct(
       recId: NATURAL1,
       pId:NATURAL,
       element:STRING,
       attributes: STRING +-> STRING,
       meta: STRING +-> STRING
       );
```

Files and Strings

XML documents can either be stored in a file or in a B string.

In [1]:
```
::load
MACHINE Jupyter_LibraryXML
DEFINITIONS "LibraryXML.def"
END
```

Out[1]:
```
Loaded machine: Jupyter_LibraryXML
```

READ_XML_FROM_STRING

This external function takes an XML document string and converts into into the B format seq(XML_ELement_Type)}. Note that all strings in ProB are encoded using UTF-8, so no encoding argument has to be provided.

In [2]:
```
READ_XML_FROM_STRING('''
<?xml version="1.0" encoding="ASCII"?>
 <Data version= "0.1">
 <Tag1 elemID="ID1" attr1="value1" />
 </Data>
''')
```

Out[2]:
```
{(1↦rec(attributes∈{("version"↦"0.1")},element∈"Data",meta∈{("xmlLine
Number"↦"3")},pId∈0,recId∈1)),(2↦rec(attributes∈{("attr1"↦"value1"),(
"elemID"↦"ID1")},element∈"Tag1",meta∈{("xmlLineNumber"↦"4")},pId∈1,re
cId∈2))}
```

READ_XML

This external function can read in an XML document from file. In contrast to READ_XML_FROM_STRING it also takes a second argument specifying the encoding used. ProB cannot as of now detect the encoding from the XML header. In future this argument may be removed. Currently it can take these values: "auto","ISO-8859-1","ISO-8859-2","ISO-8859-15", "UTF-8","UTF-16","UTF-16LE","UTF-16BE","UTF-32","UTF-32LE","UTF-32BE", "ANSI_X3.4-1968", "windows 1252".

LibraryHash

This library provides various facilities to compute hash values for B values. You can obtain the definitions below by putting the following into your DEFINITIONS clause:

```
DEFINITIONS "LibraryHash.def"
```

The file `LibraryHash.def` is also bundled with ProB and can be found in the `stdlib` folder.

In [15]:
```
::load
MACHINE Jupyter_LibraryHash
DEFINITIONS "LibraryHash.def"
END
```

Out[15]:
```
Loaded machine: Jupyter_LibraryHash
```

HASH

This external function converts a B data value to an integer hash value. It uses the `term_hash` predicate of SICStus Prolog. It will generate an integer that can be efficiently handled by ProB, but may generate collisions.

Type: $\tau \rightarrow INTEGER$

In [16]:
```
HASH({1,2,4})
```

Out[16]:
```
92915201
```

In [17]:
```
HASH({1,2,5})
```

Out[17]:
```
191034877
```

In [20]:
```
i<: 1..7 & j<:1..7 & i /= j & HASH(i)=HASH(j)
```

Out[20]:
```
FALSE
```

SHA_HASH

This external function converts a B data value to a SHA hash value represented as a sequence of bytes. It is unlikely to generate a collision.

Type: $\tau \rightarrow INTEGER$.

```
In [21]:  1  SHA_HASH({1,2,4})
```

```
Out[21]: [37,168,75,91,175,1,8,58,13,207,7,42,222,208,212,29,243,31,27,154]
```

```
In [22]:  1  SHA_HASH({1,2,5})
```

```
Out[22]: [149,81,45,24,177,25,74,30,204,7,143,202,136,116,148,247,6,221,245,52]
```

```
In [23]:  1  i<: 1..7 & j<:1..7 & i /= j & SHA_HASH(i)=SHA_HASH(j)
```

```
Out[23]: FALSE
```

SHA_HASH_HEX

This external function converts a B data value to a SHA hash value represented as a hexadecimal string. It is unlikely to generate a collision.

Type: $\tau \rightarrow STRING$.

```
In [24]:  1  SHA_HASH_HEX({1,2,4})
```

```
Out[24]: "25a84b5baf01083a0dcf072aded0d41df31f1b9a"
```

```
In [25]:  1  SHA_HASH_HEX({1,2,5})
```

```
Out[25]: "95512d18b1194a1ecc078fca887494f706ddf534"
```

```
In [26]:  1  SHA_HASH_HEX({x|x<:1..8 & card(x)=2})
```

```
Out[26]: "6bd1d8beefa14ea131285d11bbf8580c5f31fe78"
```

```
In [27]:  1  SHA_HASH_HEX(0)
```

```
Out[27]: "068948b4d423a0db5fd1574edad799005fc456e0"
```

```
In [28]:  1  SHA_HASH_HEX(SHA_HASH_HEX(0))
```

```
Out[28]: "55b9c89f79362578c3641774db978b5455be5bfd"
```

```
In [ ]:  1
```