jtrl

User Manual

Mark Richters
University of Bremen, Germany
Database Systems Group
email: mr@informatik.uni-bremen.de

Contents

1 Introduction

This manual provides information on using the `jtrl` system for developing and validating models of database applications. The object specification language TROLL *light* is used for the description of conceptual database models. With `jtrl` a model can be validated by direct execution (also called *animation*) of its specification.

The process of developing a database application can be defined roughly as follows.

1. Requirements analysis: collect and describe user demands on the system to be developed.
2. Conceptual modeling: build an abstract model of the system.
3. Validating: check the model w.r.t. consistency, completeness, etc.
4. Implementation: translate the model into an executable system.

The `jtrl` system is designed to support steps 2 and 3. Extensive Information on how TROLL *light* can be used to describe conceptual models can be found in [Her95, GH95, HCG94, GCH93, CGH92]. This manual only contains minimal information necessary to write TROLL *light* specifications for use with `jtrl`. More information about the underlying concepts of `jtrl` can be found in [RG97a, RG97b]. The rest of this manual describes how to use `jtrl` for validating specifications.

2 Running `jtrl`

Once you have a TROLL *light* specification, you can use `jtrl` to execute it and check whether it suits your requirements. `jtrl` reads the file containing your specification and compiles it into an abstract syntax tree. This internal representation is then used to control object manipulation and to evaluate queries. For user interaction, `jtrl` currently provides three different mechanisms:

- Interactive mode: basic commands and requests can be issued in textual form (see Sect. 2.3).
- Web interface: a web browser can be used to connect to `jtrl` and allows visual browsing and manipulation of objects (see Sect. 2.4).
- External tools: the graph visualization tool *daVinci* [FW94, DaV97] can be used to display and navigate objects (see Sect. 2.5). Note that *daVinci* is not strictly required for using `jtrl`.

2.1 Setting Permanent Options

Before starting `jtrl` you should check the file `jtrl.properties` in the top level directory of the distribution and make sure the settings are correct for your site. Table 1 lists the most important settings.

Property	Default value	Meaning	See Sec.
jtrl.www.server.port	8080	port for web server	2.4
jtrl.davinci.path	`/usr/local/bin/daVinci`	path for *daVinci*	2.5
jtrl.netscape.path	`/usr/local/bin/netscape`	path for netscape	2.4

Table 1: Main Properties for adapting `jtrl` to a local site

These options only need to be set, if you are going to use the related feature. For example, if you do not plan to use *daVinci* you may leave the default settings untouched.

2.2 Invoking `jtrl`

The `jtrl` program is invoked by a shell command. Arguments and options control the behavior of the system at startup time. The general syntax is as follows[1].

```
$ jtrl [options] spec_file [cmd_file]
```

You can get a list of available options by calling `jtrl` with option `-h`. The `spec_file` is the file containing your specification. In general, the extension `.trl` is used for files containing TROLL *light* specifications. The optional argument `cmd_file` will be described in Section 2.6.

2.2.1 Checking Specifications

Similar to the process of programming, the process of writing a specification often means eliminating a lot of syntax errors, typos, etc. before a specification is ready to use. Therefore, it is convenient to have a syntax checker that helps in finding errors. You can use `jtrl` to perform just a syntax (and some plausibility) checks on a specification. The option `-c` only calls the compiler, reports any errors and then exits `jtrl`.

```
$ jtrl -c Company.trl
```

[1]In the following, we assume that you have changed your PATH environment variable to include the top level directory of the `jtrl` distribution.

The file `Company.trl` contains a TROLL *light* specification describing a very simple car rental company [RG97b]. It can be found in the examples directory of the distribution. Note that there will be no output if the specification is found to be syntactically correct. If you want some progress information use the `-v` switch. This option works with all other options, too, and will give some progress information.

```
$ jtrl -c -v Company.trl
loading properties from: /home/dbs/share/jtrl/jtrl.properties
compiling specification...
no errors.
```

2.3 Interactive Mode

If you do not specify any options, the default usage mode after compiling a specification is *interactive mode.* That means `jtrl` will present a prompt and wait for user commands.

```
$ jtrl Company.trl
jtrl version 0.79, Copyright (C) 1997 Mark Richters
jtrl>
```

Commands usually consist of a single word sometimes followed by a list of arguments. Available commands are explained in the following sections.

2.3.1 Exiting `jtrl`

You can stop `jtrl` and get back to the shell at any time by entering the exit command.

```
jtrl> exit
$
```

2.3.2 Getting help

The `help` command lists all available commands in interactive mode.

```
jtrl> help
Available commands:

! EVENT       compile and execute EVENT call
? QUERY       compile and evaluate QUERY expression
exit          same as quit
```

```
help         print available commands
obj OID      print state of object with OID
quit         exit jtrl
read FILE    read FILE and process each line as command
step FILE    read FILE and process each line as command
             in single step mode
start        start web animation server
stop         stop web animation server
tpl NAME     print template NAME
version      print version of jtrl
set [OPT=on|off] set option to on or off. Without argument,
             print current settings. Options are:
               verbose:     print verbose messages
               trace_parse: trace parsing
               dump_ast:    dump AST after parsing
               dump_trl:    dump TROLL light code
               trace:       print trace messages
```

2.3.3 Executing Event Calls

Event calls are used to create and manipulate object states. They are called by using their specified name and a list of arguments. The command for event calls is a single exclamation mark followed by a list of event names supplied with arguments. A set of events can be issued simultaneously by giving a comma–separated list of event calls.

In the following example the event `incTime` is called with the integer argument 19970202[2].

```
jtrl> !incTime(19970202)
-> INFO: event 'incTime' in Company object 0
-> INFO: state transition in Company object 0 from
         'SetTime' to 'AddCustomerOrBooking'
```

The execution of an event may lead to complex state transitions. `jtrl` gives some information about what happens as a response. These messages may be of category *information*, *warning* or *error*, the latter case usually indicating some condition prohibiting a state transition. In the example above, the event call proceeded without errors in a company object with internal oid 0 changing its local behavior state.

The next example shows an invalid event call. The object's life cycle does not allow the event `deliverCar` in the current state.

```
jtrl> !deliverCar(0, "FO-123")
-> INFO: event 'deliverCar' in Company object 0
```

[2] Line breaks have been inserted to improve the formatting of the output

```
-> ERROR: event ‘deliverCar’ not allowed in state
          ‘AddCustomerOrBooking’ of Company object 0.
          Expected one of: ‘incTime’, ‘addCustomer’, ‘addBooking’
```

The target object of an event can be specified by using any expression (with the root object as context) resulting in an object reference. An optional target expression follows an event after the keyword `IN`. In the previous examples, a target expression was omitted which means that the root object is used as a default target (equivalent to using `IN SELF`).

```
jtrl> !updateAddress("foo") IN Customers(0)
-> INFO: event ‘updateAddress’ in Customer object 1
-> INFO: state transition in Customer object 1
   from ‘CustomerUpdate’ to ‘CustomerUpdate’
jtrl> ? Address(Customers(0))
-> "foo" : string
```

The event `updateAddress` changes the address of a customer object which is a subobject of the company root object. Subsequently, a query term (described in the next section) is used to validate the effect of the event.

2.3.4 Query Processing

The state of an object base can be queried using TROLL *light* expressions. A query is written as a question mark followed by an expression. The result of evaluating the expression is printed on the next line.

```
jtrl> ? 3+4
-> 7 : int
```

Syntax errors may result in an appropriate error message.

```
jtrl> ? "a"+2
error: query:1:5: operation ‘+’ not defined on ‘string x int’
```

More interesting is the use of select terms for complex SQL–like queries. In contrast to SQL, the next examples shows that the range of a query (the `FROM` part) can be any set–valued expression.

```
jtrl> ? SELECT x FROM x IN y, y IN {{1,2}, {3,4}} WHERE TRUE
-> bag(4,3,2,1) : BAG(int)
```

In a way, you can even simulate a relational database giving sets of tuples for each relation. The next query uses a nested select term and a join on a common attribute of two "tables" to determine the bag of employees currently working on a project.

```
SELECT Name(emp)
FROM   emp IN {[Name:"Wayne", Ssn:111], [Name:"Smith", Ssn:222]}
WHERE  NOT EMPTY(SELECT works_on
                 FROM   works_on IN {[PName:"ProjectX", Ssn:121],
                                     [PName:"ProjectY", Ssn:222]}
                 WHERE Ssn(emp) = Ssn(works_on))
```

Note that you have to enter the whole expression on a single line.

```
jtrl> ? SELECT Name(emp) FROM emp IN ...
-> bag("Smith") : BAG(string)
```

More examples can be found in file `Query.trl` in the examples directory of the distribution.

In order to get information about object states you also can use attribute and subobject operations. The following expression queries the `Cars` subobject in the specification of a Car Rental Company (see file `Company.trl`).

```
jtrl> ? Cars
-> map([CarNo:"VW-234",Car:@6],[CarNo:"FO-123",Car:@5],
   [CarNo:"DB-777",Car:@7]) : MAP(CarNo : string, Car : car)
```

In the result, object references are printed as `@n` where `n` is the object identifier (oid) of the referenced object. The following select term uses the `Cars` subobject as range and selects attributes of the contained car objects.

```
jtrl> ? SELECT Brand(Car(c)),CarNo(c) FROM c IN Cars WHERE TRUE
-> bag([:"Mercedes",CarNo:"DB-777"],[:"VW",CarNo:"VW-234"],
   [:"Ford",CarNo:"FO-123"]) : BAG(TUPLE( : string, CarNo : string))
```

2.4 The Web Interface

An easy way to explore TROLL *light* objects is to use the web interface of `jtrl`. In this mode `jtrl` acts as a basic web server and responds to requests issued by a web browser. Thus, information about objects is presented in form of HTML–documents. The dynamically generated documents already contain links that facilitate navigation and querying.

In order to turn on the web interface use the switch `-w` on the command line.

```
$ jtrl -w -v Company.trl
loading properties from: /home/dbs/share/jtrl/jtrl.properties
jtrl version 0.79, Copyright (C) 1997 Mark Richters
compiling specification...
starting web server on port 8080
web server waiting for connections...
jtrl>
```

During startup time `jtrl` reports the port number it uses for listening for HTTP connections. This number will default to the value given in the properties file (see Sect. 2.1) unless more than one `jtrl`-process is running simultaneously on the same host in which case the port number will be incremented to the next available port. So, in most cases you can connect to `jtrl` by entering the following URL in your browser:

```
http://localhost:8080
```

Of course, you have to substitute *localhost* by the name of the host running `jtrl`, if browser and `jtrl` are running on different machines.

2.5 Using *daVinci*

The representation of information in hypertext documents is mainly text oriented. In order to give some graphical overview of object structure and behavior state, `jtrl` utilizes a separate tool for graph visualization called *daVinci* [FW94]. The *daVinci* system is also available from the computer science department of University of Bremen. We do not discuss installation of *daVinci* here, since extensive online information is available [DaV97]. Once *daVinci* has been installed, you only need to set one variable in the properties file (see Section 2.1) to make it work with `jtrl`.

Calling `jtrl` with option `-davinci` will start the *daVinci* program. `jtrl` will then try to establish a connection with *daVinci* for interprocess communication.

```
$ jtrl -davinci -v -w Company.trl
loading properties from: /home/mr/java/jtrl/examples/../jtrl.properties
jtrl version 0.79, Copyright (C) 1997 Mark Richters
compiling specification...
launching daVinci and waiting for connection (max. 60 seconds)...
connected to daVinci
starting web server on port 8080
web server waiting for connections...
jtrl>
```

The graph visualization system can be used in combination with the web interface. Figure 1 gives an example of how a session using `jtrl` with a web browser and *daVinci* may look like. The top left window displays a graph with objects represented as nodes. Relationships between objects such as subobjects and object valued attributes used as references are shown as edges. If all programs (`jtrl`, browser, and *daVinci*) are running on the same host, it is possible to select an object in a browser document by just clicking on the object in the *daVinci* window.

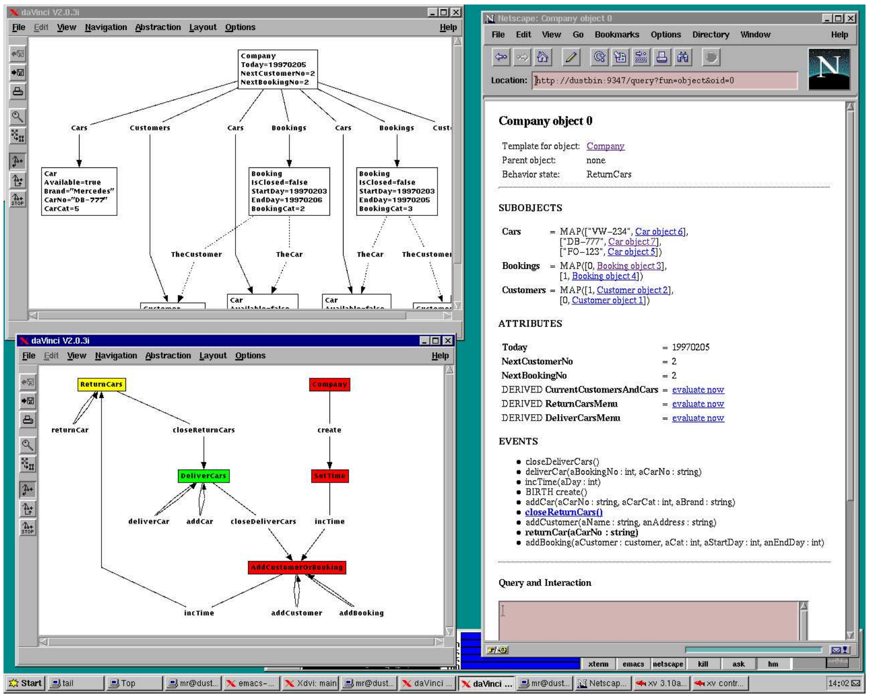

Figure 1: Using Web browser and *daVinci* with `jtrl`

The lower left window displays a finite state machine of the currently selected object. A node represents a single state with edges showing transitions to other states. Colors are used to indicate the currently active state (yellow), the set of reachable states in the next step (green), and the set of unreachable states (red).

2.6 Using a Command File

Sometimes it is tedious to call the same events again and again after making small changes to a specification and restarting `jtrl`. Command files provide a simple solution to this problem. A command file may contain any number of commands that are valid in interactive mode (see Sec. 2.3). The name of a command file is specified at start time. The file will be read sequentially and each command will be executed in turn.

The following commands are listed in the file `Company-short.cmd`.

```
!incTime(19970202)
!addCustomer("Jackson", "CA")
!incTime(19970203), updateAddress("NY") IN Customers(0)
? Customers
```

Calling `jtrl` with this command file will result in the following output.

```
$ jtrl Company.trl Company-short.cmd
jtrl version 0.79, Copyright (C) 1997 Mark Richters
1: !incTime(19970202);
-> INFO: event ‘incTime’ in Company object 0
-> INFO: state transition in Company object 0
   from ‘SetTime’ to ‘AddCustomerOrBooking’
2: !addCustomer("Jackson", "CA");
-> INFO: event ‘addCustomer’ in Company object 0
-> INFO: birth event ‘createCustomer’ in
   subobject ‘Customers’ of Company object 0
-> INFO: state transition in Company object 0
   from ‘AddCustomerOrBooking’ to ‘AddCustomerOrBooking’
-> INFO: state transition in Customer object 1
   from ‘Customer’ to ‘CustomerUpdate’
3: !incTime(19970203), updateAddress("NY") IN Customers(0);
-> INFO: event ‘incTime’ in Company object 0
-> INFO: event ‘updateAddress’ in Customer object 1
-> INFO: state transition in Company object 0
   from ‘AddCustomerOrBooking’ to ‘ReturnCars’
-> INFO: state transition in Customer object 1
   from ‘CustomerUpdate’ to ‘CustomerUpdate’
4: ? Customers
-> map([CustomerNo:0,Customer:@1])
   : MAP(CustomerNo : int, Customer : customer)
```

By adding an `exit` command at the end, you can run `jtrl` completely in "batch mode", thus enabling the integration with other tools.

3 Writing TROLL *light* Specifications

(most parts still to be written)

3.1 Preamble

The `SPECIFICATION` keyword followed by an identifier indicates which template should be treated as top-level template. The identifier must be the name of an existing template definition in the current specification unit. An instance of a top-level template serves as a root object of an object community. The birth event for a root object is automatically performed by the animator. Therefore, the following restriction applies. The top-level template has an implicit birth event `create` with no arguments. No other birth events are allowed.

3.2 Built-in Operations and Predicates

In TROLL *light* all operations on data sorts are given as part of a data signature. The specification of these operations is outside the scope of TROLL *light*. The set of basic data sorts in `jtrl` currently includes `int`, `bool` and `string`. Many operations are defined to perform calculations, transformations, etc. on values of these sorts. Furthermore, sort expressions can be used to construct more complex sorts (e.g. sets, lists, bags, and tuples). Additional operations are available to manipulate complex data structures. This section lists all implemented operations that can be used in TROLL *light* specifications.

3.2.1 Basic Data Values

Operation:	`+`
Description:	Build sum of two integer values
Infix/Prefix:	Infix
Signature:	int × int → int
Usage:	i_1 `+` i_2
Returns:	Sum of i_1 and i_2
Example:	`3 + 4` → 7

Further operations on integers are subtraction (`-`), multiplication (`*`), division (`/`), and modulo (`%`).

Operation:	`<`
Description:	Compare two integer values
Infix/Prefix:	Infix
Signature:	int × int → bool
Usage:	i_1 `<` i_2
Returns:	true, if $i_1 < i_2$
Example:	`3 < 4` → true

The operators `<=`, `>`, and `>=` are defined analogously.

Operation:	`+`
Description:	Concatenate two strings
Infix/Prefix:	Infix
Signature:	string $\times$ string $\rightarrow$ string
Usage:	s_1 `+` s_2
Returns:	s_1 concatenated with s_2
Example:	`"ab" + "cde"` $\rightarrow$ "abcde"

Comparison of strings can be done using the operators `<`, `<=`, `>`, and `>=`.

3.2.2 Bags

Operation:	`MKBAG`
Description:	Create bag from enumeration of elements
Infix/Prefix:	Prefix
Signature:	$s_1 \times \cdots \times s_n \rightarrow \text{bag}(s_1, \ldots, s_n)$
Usage:	`MKBAG(`$s_1, \ldots, s_n$`)`
Returns:	bag containing elements $s_1, \ldots, s_n$
Example:	`MKBAG(2,2,4,5)` $\rightarrow \{\!\{2, 2, 4, 5\}\!\}$

Operation:	`BTS`
Description:	Convert bag to set
Infix/Prefix:	Prefix
Signature:	$\text{bag}(s) \rightarrow \text{set}(s)$
Usage:	`BTS(`b`)`
Returns:	A set containing the elements of b with duplicates eliminated
Example:	`BTS(1,2,2,3)` $\rightarrow \{1, 2, 3\}$

Operation:	`SINGLE`
Description:	Convert singleton bag to element
Infix/Prefix:	Prefix
Signature:	$\text{bag}(s) \rightarrow s$
Usage:	`SINGLE(`b`)`
Returns:	If b contains a single element e return e, otherwise return nil
Example:	`SINGLE(MKBAG(3))` $\rightarrow 3$

Operation:	`+`
Description:	Calculate union of two bags
Infix/Prefix:	Infix
Signature:	$\text{bag}(s) \times \text{bag}(s) \rightarrow \text{bag}(s)$
Usage:	b_1 `+` b_2
Returns:	union of b_1 and b_2
Example:	`MKBAG(2,2,3) + MKBAG(1,2,3)` $\rightarrow \{\!\{1, 2, 2, 2, 3, 3\}\!\}$

Operation:	`-`
Description:	Calculate difference of two bags
Infix/Prefix:	Infix
Signature:	bag(s) $\times$ bag(s) $\rightarrow$ bag(s)
Usage:	b_1 `-` b_2
Returns:	difference of b_1 and b_2
Example:	`MKBAG(2,2,3) - MKBAG(1,2,3)` $\rightarrow \{\!\{2\}\!\}$

Operation:	`IN`
Description:	Membership test
Infix/Prefix:	Infix
Signature:	bag(s) $\times$ s
Usage:	e `IN` b
Returns:	true, if the bag b contains element e
Example:	`2 IN MKBAG(1,2,2,3)` $\rightarrow$ true

Operation:	`CNT`
Description:	Count elements in bag
Infix/Prefix:	Prefix
Signature:	bag(s) $\rightarrow$ int
Usage:	`CNT(`b`)`
Returns:	Number of elements in b
Example:	`CNT(MKBAG(1,2,2,3))` $\rightarrow 4$

3.2.3 Lists

Operation:	`MKLIST`
Description:	Create list from enumeration of elements
Infix/Prefix:	Prefix
Signature:	$s_1 \times \cdots \times s_n \rightarrow$ list$(s_1, \ldots, s_n)$
Usage:	`MKLIST(`$s_1, \ldots, s_n$`)`
Returns:	list containing elements $s_1, \ldots, s_n$
Example:	`MKLIST(2,2,4,5)` $\rightarrow < 2, 2, 4, 5 >$

Operation:	`LTS`
Description:	Convert list to set
Infix/Prefix:	Prefix
Signature:	list(s) $\rightarrow$ set(s)
Usage:	`LTS(`l`)`
Returns:	A set containing the elements of the argument list l with duplicates eliminated
Example:	`LTS(MKLIST(3,1,2,3))` $\rightarrow \{1, 2, 3\}$

Operation:	`+`
Description:	Concatenate two lists
Infix/Prefix:	Infix
Signature:	list $\times$ list $\rightarrow$ list
Usage:	l_1 `+` l_2
Returns:	l_1 concatenated with l_2
Example:	`MKLIST(3,2,1) + MKLIST(4,5)` $\rightarrow < 3, 2, 1, 4, 5 >$

Operation:	DEL
Description:	Delete an element from a list
Infix/Prefix:	Prefix
Signature:	list(s) $\times$ int $\rightarrow$ list(s)
Usage:	DEL(l, n)
Returns:	l with element at position n removed
Example:	DEL(MKLIST(4,5,6),2) $\rightarrow < 4, 6 >$

Operation:	DEL
Description:	Delete a set of elements from a list
Infix/Prefix:	Prefix
Signature:	list(s) $\times$ set(int) $\rightarrow$ list(s)
Usage:	DEL(l, s)
Returns:	l with all elements at positions given in s removed
Example:	DEL(MKLIST(4,5,6),{1,2} $\rightarrow < 6 >$

Operation:	SEL
Description:	Select an element from a list
Infix/Prefix:	Prefix
Signature:	list(s) $\times$ int $\rightarrow s$
Usage:	SEL(l, n)
Returns:	element of l at position n
Example:	SEL(MKLIST(4,5,6),2) $\rightarrow 5$

Operation:	SUB
Description:	Select part of a list
Infix/Prefix:	Prefix
Signature:	list(s) $\times$ int $\times$ int $\rightarrow$ list(s)
Usage:	SUB(l, n_1, n_2)
Returns:	list containing elements of l from position n_1 to n_2
Example:	SUB(MKLIST(4,5,6),2,3) $\rightarrow < 5, 6 >$

Operation:	IN
Description:	Membership test
Infix/Prefix:	Infix
Signature:	list(s) $\times s$
Usage:	e IN l
Returns:	true, if the list l contains element e
Example:	2 IN MKLIST(1,2,2,3) $\rightarrow$ true

Operation:	CNT
Description:	Count elements in list
Infix/Prefix:	Prefix
Signature:	list(s) $\rightarrow$ int
Usage:	CNT(l)
Returns:	Number of elements in l
Example:	CNT(MKLIST(1,2,2,3)) $\rightarrow 4$

Operation:	`POS`
Description:	Search all occurences of an element in a list
Infix/Prefix:	Prefix
Signature:	list(s) $\times$ s $\rightarrow$ set(int)
Usage:	`POS(`l, e`)`
Returns:	set of positions in l containing elements equal to e
Example:	`POS(MKLIST(6,5,6,4),6)` $\rightarrow \{1, 3\}$

3.2.4 Sets

Operation:	`MKSET`
Description:	Create set from enumeration of elements
Infix/Prefix:	Prefix
Signature:	$s_1 \times \cdots \times s_n \rightarrow$ set$(s_1, \ldots, s_n)$
Usage:	`MKSET(`$s_1, \ldots, s_n$`)`
Returns:	set containing elements $s_1, \ldots, s_n$
Example:	`MKSET(2,4,5)` $\rightarrow \{2, 4, 5\}$

Operation:	`+`
Description:	Calculate union of two sets
Infix/Prefix:	Infix
Signature:	set(s) $\times$ set(s) $\rightarrow$ set(s)
Usage:	s_1 `+` s_2
Returns:	union of s_1 and s_2
Example:	`{2,4,5} + {3,5,6}` $\rightarrow \{2, 3, 4, 5, 6\}$

Operation:	`-`
Description:	Calculate difference of two sets
Infix/Prefix:	Infix
Signature:	set(s) $\times$ set(s) $\rightarrow$ set(s)
Usage:	s_1 `-` s_2
Returns:	difference of s_1 and s_2
Example:	`{2,4,5} - {3,5,6}` $\rightarrow \{2, 4\}$

Operation:	`IN`
Description:	Membership test
Infix/Prefix:	Infix
Signature:	set(s) $\times$ s
Usage:	e `IN` s
Returns:	true, if set s contains element e
Example:	`2 IN {1,2,3}` $\rightarrow$ true

Operation:	`CNT`
Description:	Count elements in set
Infix/Prefix:	Prefix
Signature:	set(s) $\rightarrow$ int
Usage:	`CNT(`s`)`
Returns:	Number of elements in s
Example:	`CNT({1,2,3})` $\rightarrow$ 3

Operation:	`MAX`
Description:	Find maximal element in set
Infix/Prefix:	Prefix
Signature:	$\mathrm{set}(s) \to s$ for $s \in \{\mathrm{int}, \mathrm{string}\}$
Usage:	`MAX(`s`)`
Returns:	maximum element in s
Example:	`MAX({3,5,1})` $\to 5$

Operation:	`MIN`
Description:	Find minimal element in set
Infix/Prefix:	Prefix
Signature:	$\mathrm{set}(s) \to s$ for $s \in \{\mathrm{int}, \mathrm{string}\}$
Usage:	`MIN(`s`)`
Returns:	minimum element in s
Example:	`MIN({3,5,1})` $\to 1$

3.2.5 Maps

Operation:	`MKMAP`
Description:	Create map from enumeration of pairs
Infix/Prefix:	Prefix
Signature:	$\mathrm{tuple}(s_1, s_2) \times \cdots \times \mathrm{tuple}(s_1, s_2) \to \mathrm{map}(s_1, s_2)$
Usage:	`MKMAP([`$l_s : s_1, l_d : d_1$`],...,[`$l_s : s_n, l_d : d_n$`]`
Returns:	map containing element pairs (s_i, d_i)
Example:	`MKMAP([l:"a",v:1],[l:"b",v:2])` $\to \{(''a'' \to 1), (''b'' \to 2)\}$

Operation:	`SEL`
Description:	Select an element from a map
Infix/Prefix:	Prefix
Signature:	$\mathrm{map}(s_1, s_2) \times s_1 \to s_2$
Usage:	`SEL(`m, e`)`
Returns:	element associated with e in map m
Example:	`SEL(MKMAP([l:"a",v:1],[l:"b",v:2]),"b")` $\to 2$

Operation:	`DEL`
Description:	Delete an element from a map
Infix/Prefix:	Prefix
Signature:	$\mathrm{map}(s_1, s_2) \times s_1 \to \mathrm{map}(s_1, s_2)$
Usage:	`DEL(`m, k`)`
Returns:	map m with pair (k, e) removed
Example:	`DEL(MKMAP([l:"a",v:1],[l:"b",v:2]),"b")` $\to \{(''a'' \to 1)\}$

Operation:	`UPDATE`
Description:	Update element of a map
Infix/Prefix:	Prefix
Signature:	map$(s_1, s_2) \times s_1 \times s_2 \rightarrow$ map(s_1, s_2)
Usage:	`UPDATE(`m, e_1, e_2`)`
Returns:	map with element associated with e_1 replaced by e_2
Example:	`UPDATE(MKMAP([l:"a",v:1],[l:"b",v:2]),"b",3)` $\rightarrow \{(''a'' \rightarrow 1), (''b'' \rightarrow 3)\}$

3.2.6 Miscellaneous

Operation:	`DEF`
Description:	Test expression for defined value
Infix/Prefix:	Prefix
Signature:	$s \rightarrow$ bool
Usage:	`DEF(`e`)`
Returns:	true, if e has a defined value
Example:	`DEF(3 + 4)` $\rightarrow$ true, `DEF(3 / 0)` $\rightarrow$ false

Operation:	`UNDEF`
Description:	Test expression for undefined value
Infix/Prefix:	Prefix
Signature:	$s \rightarrow$ bool
Usage:	`UNDEF(`e`)`
Returns:	true, if e has an undefined value
Example:	`UNDEF(3 + 4)` $\rightarrow$ false, `UNDEF(3 / 0)` $\rightarrow$ true

Test for equality (`=`, `<>`) is defined on every sort. The boolean operations `AND`, `OR`, `IMPLIES`, `NOT` also work as expected on boolean values.

Operation:	`STR`
Description:	Transform value of any sort into string
Infix/Prefix:	Prefix
Signature:	$s \rightarrow$ string
Usage:	`STR(`v`)`
Returns:	a string representation of v
Example:	`STR(3)` $\rightarrow$ "3", `STR({3})` $\rightarrow$ "{3}"

Operation:	`EMPTY`
Description:	Test emptiness of container values
Infix/Prefix:	Prefix
Signature:	$s_1(s_2) \rightarrow$ bool, $s_1 \in$ {set,bag,map,list}
Usage:	`EMPTY(`c`)`
Returns:	true, if c contains no elements
Example:	`EMPTY({3})` $\rightarrow$ false

Operation:	`PRJ`
Description:	Projection of tuple field
Infix/Prefix:	Prefix
Signature:	$tuple(s_1, \times, s_n) \times \text{string} \rightarrow s_i$
Usage:	`PRJ(`t, l`)`
Returns:	tuple element with label l
Example:	`PRJ([PName:"ProjectX", Ssn:121], "Ssn")` $\rightarrow$ 121

A Syntax of TROLL *light*

The following grammar defines the syntax of valid TROLL *light* specifications as currently implemented in `jtrl`. Note that there are some minor differences to the grammar given in [CGH92] (e.g. for interaction rules). Some of these differences are listed in Section B.

Comments may be given in any of following the three forms.

- `/* comment */` like in C
- `//` ... for line comments like in C^{++}
- `--` ... for line comments line in Ada

A.1 Syntax of Specifications

```
Specification    := "SPECIFICATION" Ident
                    TemplateSpec+

TemplateSpec     := "TEMPLATE" Ident
                    TemplateImport
                    TemplateSig
                    TemplateBody
                    "END" "TEMPLATE"

TemplateImport   := [ DataTypesSection ]
                    [ TemplatesSection ]

TemplateSig      := [ SubobjectsSection ]
                    [ AttributesSection ]
                    [ EventsSection ]

TemplateBody     := [ ConstraintsSection ]
                    [ ValuationSection ]
                    [ DerivationSection ]
                    [ InteractionSection ]
                    [ BehaviorSection ]

DataTypesSection := "DATA" "TYPES" NameList ";"
TemplatesSection := "TEMPLATES" NameList ";"

SubobjectsSection := "SUBOBJECTS" ( SubobjectDefinition ";" )+
SubobjectDefinition := Ident [ FormalParameterList ] ":" Domain

AttributesSection := "ATTRIBUTES" ( AttributeDefinition ";" )+
AttributeDefinition := [ "DERIVED" ] NameList ":" Domain
```

```
EventsSection    := "EVENTS" ( EventDefinition ";" )+
EventDefinition  := [ "BIRTH" | "DEATH" ] Ident [ FormalParameterList ]

ConstraintsSection := "CONSTRAINTS" ( ConstraintsDefinition )+
ConstraintsDefinition := Expression ";"

ValuationSection := "VALUATION" ( ValuationDefinition ";" )+
ValuationDefinition := OptionalPreCondition "[" EventDescription "]"
                    ValuationEffect { "," ValuationEffect }
ValuationEffect  := Ident "=" Expression

OptionalPreCondition := [ "{" Expression [ ":" Expression ] "}" ]

DerivationSection := "DERIVATION" ( DerivationDefinition ";" )+
DerivationDefinition := Ident "=" Expression

InteractionSection := "INTERACTION" ( InteractionDefinition ";" )+
InteractionDefinition := OptionalPreCondition
                    InteractionSource ">>" InteractionDestination
                    { "," InteractionDestination }
InteractionSource := EventDescription [ "IN" Expression ]
InteractionDestination := Ident [ Arguments ] [ "IN" Expression ]

BehaviorSection  := "BEHAVIOR" ( ProcessDefinition ";" )+
ProcessDefinition := "PROCESS" Ident "EQ" Process
Process          := "(" [ ProcessSequence { "|" ProcessSequence } ] ")"
ProcessSequence  := OptionalPreCondition
                    EventDescription "->" Ident

EventDescription := Ident

NameList         := Ident { "," Ident }

FormalParameterList := "(" FormalParameter { "," FormalParameter } ")"
FormalParameter  := Ident ":" Domain

Domain           := BaseDomain | MultiValuedDomain | CompositeDomain
BaseDomain       := Ident
MultiValuedDomain := SetDomain | BagDomain | ListDomain | MapDomain
SetDomain        := "SET" "(" Domain ")"
BagDomain        := "BAG" "(" Domain ")"
ListDomain       := "LIST" "(" Domain ")"
MapDomain        := "MAP" "(" Ident ":" Domain "," Ident ":" Domain ")"
CompositeDomain  := "TUPLE" FormalParameterList

/* Expressions */

Expression       := ConditionalImpliesExpression | SelectExpression
```

```
SelectExpression := "SELECT" ArgumentList
                    "FROM" DeclarationList
                    "WHERE" Expression

ConditionalImpliesExpression := ConditionalOrExpression
                    { "IMPLIES" ConditionalOrExpression }

ConditionalOrExpression := ConditionalAndExpression
                    { "OR" ConditionalAndExpression }

ConditionalAndExpression := EqualityExpression
                    { "AND" EqualityExpression }

EqualityExpression := RelationalExpression
                    [ ( "EQ" | "NE" | "IN" ) RelationalExpression ]

RelationalExpression := AdditiveExpression
                    [ ( "<" | ">" | t = "<=" | t = ">=" )
                      AdditiveExpression ]

AdditiveExpression := MultiplicativeExpression
                    { ( "+" | "-" ) MultiplicativeExpression }

MultiplicativeExpression := UnaryExpression
                    { ( "*" | "/" | "%" ) UnaryExpression }

UnaryExpression  :=   ( "NOT" UnaryExpression )
                    | PrimaryExpression
                    | IfExpression
                    | QuantifiedExpression

IfExpression     := "IF" Expression
                    "THEN" Expression
                    "ELSE" Expression
                    "FI"

QuantifiedExpression := ( "FORALL" | "EXISTS" )
                    "(" DeclarationList ")"
                    "(" Expression ")"

PrimaryExpression :=  Literal
                    | FunctionOrVariableExpression
                    | ComplexValueConstructor
                    | NilConstructor
                    | "(" Expression ")"

Literal          :=   INTEGER_LITERAL
                    | STRING_LITERAL
```

```
                      | BOOLEAN_LITERAL
                      | "SELF"
                      | TupleConstructor
                      | SetConstructor

FunctionOrVariableExpression := Ident [ Arguments ]

ComplexValueConstructor := "NEW" "(" MultiValuedDomain ")"
NilConstructor   := "NIL" "(" Domain ")"
TupleConstructor := "[" FieldDefinition { "," FieldDefinition } "]"
SetConstructor   := "{" Expression { "," Expression } "}"

FieldDefinition  := [ Ident ] ":" Expression

Arguments        := "(" ArgumentList ")"
ArgumentList     := Expression { "," Expression }

DeclarationList  := Declaration ( "," Declaration )*
Declaration      := Ident "IN" Expression

Ident := ID
```

A.2 Syntax of Requests

For queries and event calls the general term *request* is used. A query may be any expression as defined in the previous section. To distinguish event calls from queries they have to be prefixed with an exclamation mark. The following grammar defines the syntax of requests.

```
Request          := ( Expression | EventRequestList ) ";"

EventRequestList := "!" EventRequest { "," EventRequest }
EventRequest     := Ident [ Arguments ] [ "IN" Expression ]
```

B Differences to Previously Published Syntax

This section lists some of the differences to syntax definitions given in some of the research papers on TROLL *light*. The reasons for a different syntax implemented in `jtrl` are almost always a matter of convenience and solutions necessary for practical work.

- A `SPECIFICATION`-clause must be used to declare the top level template.

- Parameterized subobjects have to be defined as `MAP`. There is no special syntax.
- Currently, basic data sorts are `String`, `Int`, and `Bool`.
- The order of source (destination) term and event call in interaction rules has changed. A term specifying an event location has to be written *after* an event, e.g. `changeName IN Author` instead of `Author.changeName`. Otherwise, the parser would need more lookahead to differentiate between expressions and event calls.

References

[CGH92] S. Conrad, M. Gogolla, and R. Herzig. TROLL light: A Core Language for Specifying Objects. Informatik-Bericht 92–02, TU Braunschweig, 1992.

[DaV97] DaVinci Homepage, `http://www.informatik.uni-bremen.de/~davinci`, 1997.

[FW94] M. Fröhlich and M. Werner. The Graph Visualization System daVinci - A User Interface for Applications. Technical Report 5/94, Universität Bremen, September 1994.

[GCH93] M. Gogolla, S. Conrad, and R. Herzig. Sketching Concepts and Computational Model of TROLL light. In A. Miola, editor, *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems (DISCO)*, pages 17–32. Springer, Berlin, LNCS 722, 1993.

[GH95] M. Gogolla and R. Herzig. An Algebraic Semantics for the Object Specification Language TROLL light. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification (WADT'94)*, pages 288–304. Springer, Berlin, LNCS 906, 1995.

[HCG94] R. Herzig, S. Conrad, and M. Gogolla. Compositional Description of Object Communities with TROLL light. In C. Chrisment, editor, *Proc. Basque Int. Workshop on Information Technology (BIWIT'94): Information Systems Design and Hypermedia*, pages 183–194. Cépaduès-Éditions, Toulouse, 1994.

[Her95] R. Herzig. *Zur Spezifikation von Objektgesellschaften mit TROLL light*. VDI-Verlag, Düsseldorf, Reihe 10 der Fortschritt-Berichte, Nr. 336, 1995. (Dissertation, Naturwissenschaftliche Fakultät, Technische Universität Braunschweig, 1994).

[RG97a] Mark Richters and Martin Gogolla. A Web-based Animator for Object Specifications in a Persistent Environment. In M. Dauchet and M. Bidoit, editors, *TAPSOFT '97: theory and practice of software development: 7th International Joint Conference CAAP/FASE, Lille, France, April 14–18, 1997: proceedings*, volume 1214 of *Lecture Notes in Computer Science*, pages 867–870, New York, NY, USA, 1997. Springer-Verlag.

[RG97b] Mark Richters and Martin Gogolla. A Web-based Animator for Validating Object Specifications. In Bipin C. Desai and Barry

Eaglestone, editors, *IDEAS'97: International Database Engineering & Applications Symposium, Montreal, Canada, August 25–27, 1997*, pages 211–219, 1997.