Appendix B

The Tcl Interface of UV

The overall structure of the UV System has been described in section 6 as consisting of a kernel providing the functionality of the UV workspace and of a graphical user interface making the core functions of the kernel accessible in a convenient fashion. This separation is possible due to the use of an interface layer between kernel and user interface: the kernel makes its operations and data structures available through a number of Tcl commands, the user interface is built as a collection of Tcl/Tk scripts that rely solely on the commands provided by the kernel.

This appendix serves as a reference section for the Tcl commands implemented in the current version of the UV system (2.3.3). Each command is described in its own entry with information about command syntax, parameters and return values, and general usage description.

For displaying the command syntax, the following conventions are used: `typewriter font` is used to show text that is to be typed literally, *italics* denote variables or options that need to be replaced with some suitable text. Moreover the common meta-symbols [] for optional occurrence, {} for grouping, | for separating alternatives, and $^+$ for indicating one or more occurrences of the preceding item are used.

There are also a few additional commands and command options that will

be made available in future releases of the UV system. Such commands include `uv_symbol` for accessing symbol table information about identifiers, `uv_order` for accessing and modifying the OBDD variable ordering of variable encodings, and finally `uv_regexp` for accessing information about regular expressions used as hints in generalized leads-to properties.

B.1 uv_check

Invoke the model checker on a property.

Synopsis

`uv_check` *n* [*inv*]

n ID number of property to check,

inv option indicating which invariant is to be used invariant to use for checking; one of `-type`, `-current`, and `-strongest`.

Result

A two-element list of the form *invoke status* is returned, where *invoke* indicates, whether the checker was actually invoked:

`0` if checker was not invoked (e.g. because checking result could be determined from invariants alone),

`1` if checker was invoked,

and *status* indicates the checking status of the property:

`ok` if property has been proved correct,

`?` if checking status is not known yet,

`fail` if property has been proved incorrect.

Description

The model checker is invoked on the property with ID `#prop`*n* using the invariant indicated by *inv* as follows:

`-type` use type invariant

`-current`	use current invariant (default)
`-strongest`	use strongest invariant

Special Considerations

The UV workspace must have been initialized prior to executing `uv_check`.

B.2 uv_expr

Access information about an expression.

Synopsis

`uv_expr` *n* [`-context` | `-def` | `-sat` | `-size` | `-type` | `-value`]

n ID number of expression to access information about

Result

An exception is raised if *n* is not an integer or if there is no expression in the workspace with ID *n*. Otherwise a value is returned depending on the selected option:

no option	The empty string is returned.
`-context`	The context of the expression is returned as a list of scope names denoting the path from the global scope (corresponding to the empty list) to the scope of the expression
`-def`	The source text defining the expression is returned.
`-sat`	An exception is raised if the expression is not of type boolean or is not satisfiable; otherwise a satisfying variable assignment is returned in the form of a list of scope assignment lists. Each scope assignment list is ordered from outmost (global) scope to innermost (local) scope, and each scope list consists of the scope name ({} for the global scope) followed by lists of two elements, the first being a variable name and the second being its assigned value.

For instance, the satisfying variable assignment for the expression `x=3 /\ b` where `x` is a program variable of program `P` and `b` is a global variable, is represented as {{} {b true}} {P {x 3}}.

`-size` The number of OBDD nodes used in internally representing the expression is returned.

`-type` The type of the expression is returned.

`-value` The value of the expression is returned: if the expression is a constant, that constant value is returned; otherwise all variables the expression depends on are displayed in a list of scope lists, where scope lists are ordered from outmost (global) scope to innermost (local) scope, and where each scope list consists of the scope name ({} for the global scope) followed by all variables of that scope the expression depends on.

For instance, the value of an expression of program `P` depending on the program variable `x` and the global variable `b` is represented as {{} b} {P x}.

Description

Information about an expression in the UV workspace is returned. An expression is referred to by the ID number that is part of the expression identifier generated by the parser or by certain expression setting commands at the time the expression was entered into the workspace. Without an option `uv_expr` can be used to check whether an expression with a given ID number is present in the UV workspace. Specific information about an expression can be obtained by using one of the listed options as shown above.

Special Considerations

The UV workspace must have been initialized prior to executing `uv_expr`.

B.3 uv_info

Access information about the UV system and the UV workspace.

Synopsis

`uv_info {-flags | -memory | -version}`

Result

Depending on the chosen option the following information is returned:

`-flags` A list of compile-time flags is returned, indicating how the UV system was built. Currently the only supported flag is the `DEBUG` flag indicating whether the UV system was compiled in debug mode.

`-memory` Information about the current memory usage is returned as a list of five lists containing (in that order) information about the OBDD heap, the OBDD hash table, the ITE cache, the AEN cache, and the function cache. If a certain cache is not enabled (cf. the `uv_option` command), the corresponding list is empty. The information in each list is structured as follows:

OBDD heap: total number of OBDD nodes, number of used OBDD nodes, number of allocated OBDD nodes;

OBDD hash table: total number of slots, number of free slots, number of entries, maximum, average, and variance of length of entry chains, number of find operations, percentage of hits, average chaining length;

ITE cache, AEN cache, function cache: total number of slots, number of free slots, number of find operations, percentage of hits.

`-version` the revision number of the UV system presented in the form $n.m.p$ where n is the major revision number, m is the minor revision number, and p is the patch level.

Description

Information about the UV system and about the UV workspace is returned.

Special Considerations

The UV workspace must have been initialized prior to executing `uv_info` with the `-memory` option.

B.4 uv_init

Initialize the UV workspace.

Synopsis

`uv_init` [`-mem` n]

n Natural number in the range from 6 to 24, specifying the amount of memory allocated (see below). The default value is 16.

Result

none, or message string if workspace has already been initialized

Description

The UV workspace is initialized by allocating a OBDD heap of 2^n nodes, a OBDD hash table of 2^{n-2} entries, and proportionally sized cache tables. The initial global symbol table is set up as well.

Special Considerations

`UV_init` needs to be called once at the beginning of a session. Subsequent calls have no effect.

B.5 uv_option

Set system options or return their current status.

Synopsis

`uv_option` {`-bdd` | `-reduce`} [*value*]

value the value to which the specified option is set.

Result

If no *value* parameter is given, the current value of the specified option is returned.

Description

Without a *value* parameter the current status of the specified system option is returned. If a *value* parameter is given, then the specified option is set to the indicated value. The values for the different options and their meaning are as follows:

The `-bdd` option enables certain OBDD-level caches; its value is the sum of any of the following:

`1`	use a general function cache (default),
`2`	use a separate AEN cache.

The `-reduce` option controls the use of the reduction operation on OBDDs; its value is the sum of any of the following:

`1`	use during strongest invariant computations (default),
`2`	use during wlt-computations.

B.6 uv_parse

Parse input string according to UV language grammar and update UV workspace accordingly.

Synopsis

`uv_parse` *var* $\{string\}^+$

var	Tcl variable receiving list of parse results
string	input string

Result

No value is returned if input is parsed successfully, otherwise a list of the form *pos len msg* is returned, where *pos* is the character offset into the concatenation of input strings of the first character of the offending token, *len* is the number of characters of the offending token, and *message* is an error message string describing the nature of the parse error. (The first character of the input has offset 1.)

The Tcl variable *var* contains a list of result items, one for each successfully parsed input unit, of the following form:

`#expr`*n*	for expressions: expression ID,
`#prog`*n*	for programs: programs ID,
`#prop`*n*	for properties: property ID,
symbollist	for declarations: list of declared symbols,

where the numbers *n* are uniquely assigned for each input unit in each category.

Description

The parser for the UV language is invoked on the concatenation of all string arguments. The input string can represent any number of input units like declarations, expressions, programs, or properties. Each successfully parsed unit is compiled into the UV workspace, and a suitable result item identifying the unit is appended to the Tcl variable *var* (see results above).

Parsing is terminated upon the first encountered error, in which case some error information is returned. Possible errors include syntax errors, type errors, and compilation errors (e.g. exhausted resources).

Special Considerations

The UV workspace must have been initialized prior to executing `uv_parse`.

B.7 uv_prog

Access information about a program.

Synopsis

```
uv_prog  n
uv_prog  n  -def
uv_prog  n  -invariant  [inv]
uv_prog  n  -name
uv_prog  n  -statement  [s]
```

n — the ID number of the program to be accessed

inv — an option indicating which invariant of the program is to be accessed; one of `type`, `current`, or `strongest`

s — the number of the statement the label of which is to be returned (counting form 0).

Result

An exception is raised if *n* is not an integer or if there is no program in the workspace with ID number *n*. Otherwise a value is returned depending on the selected option:

no option — The empty string is returned.

`-def` — The source text defining the program is returned.

`-invariant` — Without a *inv* parameter a characterization of the invariant of the program is returned in the form of a two element list, each element being one of `type`, `current`, or `strongest`. The first element is the weakest characterization of the invariant or the program, the second is its strongest characterization (`type` is weaker than `current`, which is weaker than `strongest`).

For instance, the result `type strongest` means that the strongest invariant has been computed and is the same as the type invariant.

If the *inv* parameter is given, the indicated invariant is entered into the workspace as an expression and an expression identifier of the form `#expr`n is returned, where n is the unique ID number which the new expression can be referenced by. An exception is raised if the strongest invariant is requested without having been computed.

`-name` The program name is returned.

`-statement` Without an s parameter the number of statements of the program is returned; otherwise the label of statement number s is returned.

Description

Information about a program in the UV workspace is returned. A program is referred to by the ID number that is part of the program identifier generated by the parser at the time the program was entered into the workspace. Without an option `uv_prog` can be used to check whether a program with a given ID number is present in the UV workspace. Specific information about a program can be obtained by using one of the listed options as shown above.

Special Considerations

The UV workspace must have been initialized prior to executing `uv_prog`.

B.8 uv_prop

Access information about a property or delete a property.

Synopsis

```
uv_prop  n
uv_prop  n  -arg  [a]
uv_prop  n  -def
uv_prop  n  -delete
uv_prop  n  -info  [key]
uv_prop  n  -infoexpr  key
uv_prop  n  -op
uv_prop  n  -prog
uv_prop  n  -status
```

n	the ID number of the property to be accessed
a	number indicating which argument is to be accessed (counting from 0),
key	key indicating which checking information is to be returned; one of `implication`, `invariant`, `iterations`, `safety`, `transition`, or `value`.

Result

An exception is raised if *n* is not an integer or if there is no property in the workspace with ID number *n*. Otherwise a value is returned depending on the selected option:

no option	The empty string is returned.
`-arg`	Without an *a* parameter the number of predicate arguments of the property is returned; otherwise argument number *a* (counting from 0) is entered into the workspace and a unique expression identifier is returned.

`-def` The source text defining the property is returned.

`-delete` The empty string is returned and the property is removed from the workspace.

`-info` Without a *key* parameter, a list of all key values for which there is checking information for the property is returned. If a *key* parameter is given, the following specific information about the checking status of the property is returned (if present) depending on the value of *key*:

`implication`: 1 if and only if the implication part of the property is true, 0 otherwise;

`invariant`: characterization of the checking invariant as a list of two items each having one of the values `type`, `current`, or `strongest`; cf. the `-invariant` option of the `uv_prog` command for a description of the format;

`iterations`: list of numbers of major and minor iterations;

`safety`: label of a violating statement (or empty string);

`transition`: label of a helpful statement (or empty string);

`value`: 1 if and only if there is value violating constant condition, 0 otherwise.

`-infoexpr` Depending on the *key* parameter the following expressions are entered into the workspace and an expression identifier is returned (provided expression information is available):

`invariant`: checking invariant,

`implication`: negation of implication check,

`safety`: negation of safety check of a violating statement (or empty string if there is no violation),

`transition`: empty string (no expression entered),

`value`: value for which constant condition is violated (or empty string if there is no violation),

`iterations`: negation of leads-to check.

Note that the negated check expressions for the keys `implication`, `iterations`, and `safety` characterize the states that do not satisfy the required property.

`-op` The operator of the property is returned.

`-prog` The program identifier of the program to which the property belongs is returned,

`-status` The checking status of the property is returned; it is one of `ok`, `?`, `fail`, or `new`.

Description

Information about a property in the UV workspace is returned. A property is referred to by the ID number that is part of the property identifier generated by the parser at the time the property was entered into the workspace. Without an option `uv_prop` can be used to check whether a property with a given ID number is present in the UV workspace. Specific information about a property can be obtained by using one of the listed options as shown above.

Special Considerations

The UV workspace must have been initialized prior to executing `uv_prop`.

B.9 `uv_si`

Compute strongest invariant of program.

Synopsis

`uv_si` *n* [*opt*]

n — ID number of program for which strongest invariant is to be computed,

opt — option indicating which operation is to be performed; one of `-clear`, `-forward`, `-frontier`, and `-square`.

Result

Return the number of iterations performed for computing the strongest invariant for options `-forward`, `-frontier`, and `square`, or the empty string for option `-clear`. Raise an exception if workspace does not contain a program with ID number *n*.

Description

Compute the strongest invariant of the program with ID number *n* in the UV workspace using an algorithm determined by the option *opt* as follows (if the strongest invariant has been computed previously, it is simply recalled from a special cache):

`-clear` — do not compute the strongest invariant but forget it in case it has been computed previously; useful for comparing strongest invariant computations using different algorithms,

`-forward` — use the standard forward chaining algorithm to explore the reachable state space,

`-frontier`	use a modification of the forward chaining algorithms in which only successors of states are considered that are at the frontier of the state space exploration, i.e.that have been added to the set of reachable states in the previous iteration (default),
`-square`	use an iterative squaring algorithm.

Special Considerations

The UV workspace must have been initialized prior to executing `uv_si`.