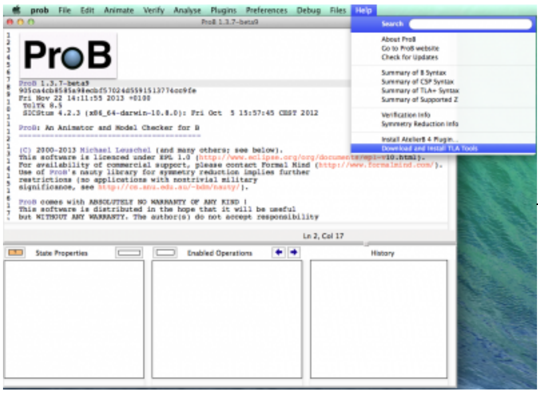
TLA

As of version 1.3.5, ProB supports TLA+ (http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html).

Contents

Using ProB for Animation and Model Checking of TLA+ specifications

The latest version of ProB (http://nightly.cobra.cs.uni-duesseldorf.de/tcl/) uses the translator TLA2B, which translates the non temporal part of a TLA+ (http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html) module to a B machine. To use ProB for TLA+ you have to download the TLA tools. They are released as an open source project, under the MIT License (http://research.microsoft.com/en-us/um/people/lamport/tla/license.html). In the ProB Tcl/Tk GUI you have to select the menu command "Download and Install TLA Tools" in the Help menu.



(/stups/prob/index.php/File:Download_TLA_Tools.png)

When you open a TLA+ module ProB generates the translated B machine in the same folder and loads it in the background. If there is a valid translation you can animate and model check the TLA+ specification. There are many working examples in the 'ProB/Examples/TLA+/'-directory.

There is also an iFM'2012 paper (http://www.stups.uni-duesseldorf.de/w/Special:Publication/HansenLeuschelTLA2012) that describes our approach and performs some comparison with TLC. Our online ProB Logic Calculator (/stups/prob/index.php/ProB_Logic_Calculator) now also supports TLA syntax and you can experiment with its predicate and expression evaluation capabilities.

TLA2B

The parser and semantic analyzer SANY (http://research.microsoft.com/en-us/um/people/lamport/tla/sany.html) serves as the front end of TLA2B. SANY was written by Jean-Charles Grégoire and David Jefferson and is also the front end of the model checker TLC (http://research.microsoft.com/en-us/um/people/lamport/tla/tlc.html). After parsing there is type checking phase, in which types of variables and constants are inferred. So there is no need to especially declare types in a invariant clause (in the manner of the B method). Moreover it checks if a TLA+ module is translatable (see Limitations of Translation).

To tell TLA2B the name of a specification of a TLA+ module you can use a configuration file, just like for TLC. The configuration file must have the same name as the name of the module and the filename extension 'cfg'. The configuration file parser is the same as for TLC so you can look up the syntax in the 'Specifying Systems'-book (http://research.microsoft.com/en-us/um/people/lamport/tla/book.html) (Leslie Lamport). If there is no configuration file available TLA2B looks for a TLA+ definition named 'Spec' or alternatively for a 'Init' and a 'Next' definition describing the initial state and the next state relation. Besides that in the configuration file you can give a constant a value but this is not mandatory, in contrast to TLC. Otherwise ProB lets you choose during the animation process a value for the constant which satisfy the assumptions under the ASSUME clause. TLA2B supports furthermore overriding of a constant or definition by another definition in the configuration file.

Supported TLA+ syntax

```
Logic
-----
 P /\ Q                        conjunction
 P \/ Q                        disjunction
 ~ or \lnot or \neg            negation
 =>                            implication
 <=> or \equiv                 equivalence
 TRUE
 FALSE
 BOOLEAN                       set containing TRUE and FALSE
 \A x \in S : P                universal quantification
 \E x \in S : P                existential quantification

Equality:
------
 e = f                         equality
 e # f or e /= f               inequality

Sets
------
 {d, e}                        set consisting of elements d, e
 {x \in S : P}                 set of elements x in S satisfying p
 {e : x \in S}                 set of elements e such that x in S
 e \in S                       element of
 e \notin S                    not element of
 S \cup T or S \union T        set union
 S \cap T or S \intersect      set intersection
 S \subseteq T                 equality or subset of
 S \ t                         set difference
 SUBSET S                      set of subsets of S
 UNION S                       union of all elements of S

Functions
------
 f[e]                          function application
 DOMAIN f                      domain of function f
 [x \in S |-> e]               function f such that f[x] = e for x in S
 [S -> T]                      Set of functions f with f[x] in T for x in S
 [f EXCEPT ![e] = d]           the function equal to f except f[e] = d

Records
-------
 r.id                          the id-field of record r
 [id_1|->e_1,...,id_n|->e_n]  construct a record with given field names and values
 [id_1:S_1,...,id_n:S_n]       set of records with given fields and field types
 [r EXCEPT !.id = e]           the record equal to r except r.id = e

Strings and Numbers
-------------------
 "abc"                         a string
 STRING                        set of a strings
 123                           a number

Miscellaneous constructs
------------------------
IF P THEN e_1 ELSE e_2
CASE P_1 -> e_1 [] ... [] P_n ->e_n
CASE P_1 -> e_1 [] ... [] P_n ->e_n [] OTHER -> e
LET d_1 == e_1 ... d_n == e_n IN e

Action Operators
----------------
v'                             prime operator (only variables are able to be primed)
UNCHANGED v                    v'=v
UNCHANGED <<v_1, v_2>>         v_1'=v_1 /\ v_2=v_2

Supported standard modules
--------------------------
Naturals
--------
x + y                          addition
x - y                          difference
x * y                          multiplication
x \div y                       division
x % y                          remainder of division
x ^ y                          exponentiation
x > y                          greater than
```

```
x < y                          less than
x \geq y                       greater than or equal
x \leq y                       less than or equal
x .. y                         set of numbers from x to y
Nat                            set of natural numbers

Integers
--------
-x                             unary minus
Int                            set of integers

Sequences
---------
SubSeq(s,m,n)                  subsequence of s from component m to n
Append(s, e)                   appending e to sequence s
Len(s)                         length of sequence s
Seq(s)                         set of sequences
s_1 \o s_2 or s_1 \circ s_2   concatenation of s_1 and s_2
Head(s)
Tail(s)

FiniteSets
----------
Cardinality(S)
IsFiniteSet(S)                 (ProB can only handle certain infinite sets as argument)



typical structure of a TLA+ module
--------------------------

---- MODULE m ----
EXTENDS m_1, m_2
CONSTANTS c_1, c_2
ASSUME c_1 = ...
VARIABLES v_1, v_2
foo == ...
Init == ...
Next == ...
Spec == ...
====================
```

Temporal formulas and unused definitions are ignored by TLA2B (they are also ignored by the type inference algorithm).

Limitations of the translation

- due to the strict type system of the B method there are several restrictions to TLA+ modules.
  - the elements of a set must have the same type (domain and range of a function are sets)
  - TLA+ tuples are translated as sequences in B, hence all components of the tuple must have the same type
- TLA2B do not support 2nd-order operators, i.e. operators that take a operator with arguments as argument (e.g.: foo(bar(_),p))

TLA+ Actions

TLA2B divides the next state relation into different actions if a disjunction occurs. IF a existential quantification occurs TLA2B searches for further actions in the predicate of the quantification and adds the bounded variables as arguments to these actions. IF a definition call occurs and the definition has no arguments TLA2B goes into the definition searching for further actions. The displayed actions by ProB are not necessarily identical with the actions determined by TLC.

Understanding the type checker

Corresponding B types to TLA+ data values (let type(e) be the type of the expression e):

```
TLA+ data                               B Type
--------------------------------------------------
number e.g. 123                         INTEGER
string e.g. "abc"                       STRING
bool value e.g. TRUE                    BOOL
set e.g. {e,f}                          POW(type(e)), type(e) = type(f)
function e.g. [x \in S |-> e]           POW(type(x)*type(e)), type(S) = POW(type(x))
sequence e.g. <<a,b>>                   POW(INTEGER*type(a)), type(a) = type(b)
record e.g. [id_1|->e_1,...,id_n|->e_n] struct(id_1:type(e_1),...,id_n:type(e_n))
model value                             ENUM
 (only definable in config file)

Nat                                     POW(INTEGER)
Int                                     POW(INTEGER)
STRING                                  POW(STRING)
BOOLEAN                                 POW(BOOL)
SUBSET S                                POW(type(S))
```

You can only compare data values with the same type.