

LOGICAL MODELS OF PROBLEMS AND COMPUTATIONS

Theory and Software



Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria



Logical Models of Problems and Computations

What is the purpose of logical modeling?

- Precisely describe the problem to be solved.
 - Clarification of mind, resolution of ambiguities.
 - Specification of program to be developed.
- Software-supported analysis of the problem and its solution.
 - Validation of specification.
 - Validation/verification of solution.
 - Interactive/automatic provers and model checkers.
- Automatic computation of solution respectively simulation of execution.
 - Logical solvers (SMT: Satisfiability Modulo Theories).
 - Perhaps: rapid prototyping of a later manually written program.

To profit from software, we need computer-understandable models.

1. Specifying Problems

2. The RISC Algorithm Language (RISCAL)

3. Modeling Computations

4. The Temporal Logic of Actions (TLA)

Specifying Problems

- A (computational) problem:

Input: $x_1 \in T_1, \dots, x_n \in T_n$ where I_x

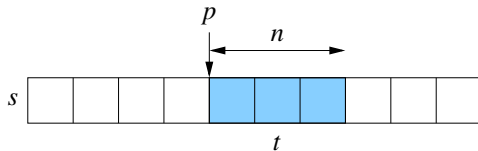
Output: $y_1 \in U_1, \dots, y_m \in U_m$ where $O_{x,y}$

- **Input variables** x_1, \dots, x_n .
 - With types T_1, \dots, T_n .
- **Input condition (precondition)** I_x .
 - A formula whose free variables occur in x_1, \dots, x_n .
- **Output variables** y_1, \dots, y_m .
 - With types U_1, \dots, U_m .
- **Output condition (postcondition)** $O_{x,y}$.
 - A formula whose free variables occur in $x_1, \dots, x_n, y_1, \dots, y_m$.

Formulas refer to functions and predicates that characterize the problem domain.

Example

Extract from a finite sequence s a subsequence of length n starting at position p .



Input: $s \in T^*$, $n \in \mathbb{N}$, $p \in \mathbb{N}$ where

$$n + p \leq \text{length}(s)$$

Output: $t \in T^*$ where

$$\text{length}(t) = n \wedge$$

$$\forall i \in \mathbb{N}. i < n \Rightarrow t[i] = s[i + p]$$

The resulting sequence must have appropriate length and contents.

Implementing Problem Specifications

- The specification demands a function $f: T_1 \times \dots \times T_n \rightarrow U_1 \times \dots \times U_m$ such that

$$\forall x_1 \in T_1, \dots, x_n \in T_n. I_x \Rightarrow \text{let } (y_1, \dots, y_m) = f(x_1, \dots, x_n) \text{ in } O_{x,y}$$

- For all arguments x_1, \dots, x_n that satisfy the input condition,
 - the result (y_1, \dots, y_m) of f satisfies the output condition.
- The specification itself already implicitly defines such a function:

$$f(x_1, \dots, x_n) := \text{choose } y_1 \in U_1, \dots, y_m \in U_m. O_{x,y}$$

- An implicit function definition (whose result is arbitrary, if no values satisfy O).
- An actual implementation must provide an explicitly defined function.
 - Right-side of definition is a term that describes a constructive computation.

The ultimate goal of computer science/mathematics is to provide explicit definitions of functions (i.e., programs) that implement problem specifications.

Function Definitions

- An (explicit) function definition

$$f : T_1 \times \dots \times T_n \rightarrow T$$

$$f(x_1, \dots, x_n) := t_x$$

- Special case $n = 0$: a constant definition $c : T, c := t$.

- Function constant f of arity n .
- Type signature $T_1 \times \dots \times T_n \rightarrow T$.
- Parameters x_1, \dots, x_n (variables).
- Body t_x (a term whose free variables occur in x_1, \dots, x_n).

We thus know $\forall x_1 \in T_1, \dots, x_n \in T_n. f(x_1, \dots, x_n) = t_x$.

Examples

- **Definition:** Let x and y be natural numbers. Then the square sum of x and y is the sum of the squares of x and y .

$$\begin{aligned} \text{squaresum} &: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ \text{squaresum}(x, y) &:= x^2 + y^2 \end{aligned}$$

- **Definition:** Let x and y be natural numbers. Then the squared sum of x and y is the square of z where z is the sum of x and y .

$$\begin{aligned} \text{sumsquared} &: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ \text{sumsquared}(x, y) &:= \text{let } z = x + y \text{ in } z^2 \end{aligned}$$

- **Definition:** Let n be a natural number. Then the square sum set of n is the set of the square sums of all numbers x and y from 1 to n .

$$\begin{aligned} \text{squaresumset} &: \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N}) \\ \text{squaresumset}(n) &:= \{ \text{squaresum}(x, y) \mid x, y \in \mathbb{N} \wedge 1 \leq x \leq n \wedge 1 \leq y \leq n \} \end{aligned}$$

Predicate Definitions

- An (explicit) predicate definition

$$p \subseteq T_1 \times \dots \times T_n$$

$$p(x_1, \dots, x_n) :\Leftrightarrow F_x$$

- Predicate constant p of arity n .
- Type signature $T_1 \times \dots \times T_n$.
- Parameters x_1, \dots, x_n (variables).
- Body F_x (a formula whose free variables occur in x_1, \dots, x_n).

We thus know $\forall x_1 \in T_1, \dots, x_n \in T_n. p(x_1, \dots, x_n) \Leftrightarrow F_x$.

Examples

- **Definition:** Let x, y be natural numbers. Then x divides y (written as $x|y$) if $x \cdot z = y$ for some natural number z .

$$\begin{aligned} \mid &\subseteq \mathbb{N} \times \mathbb{N} \\ x|y &:\Leftrightarrow \exists z \in \mathbb{N}. x \cdot z = y \end{aligned}$$

- **Definition:** Let x be a natural number. Then x is prime if x is at least two and the only divisors of x are one and x itself.

$$\begin{aligned} isprime &\subseteq \mathbb{N} \\ isprime(x) &:\Leftrightarrow x \geq 2 \wedge \forall y \in \mathbb{N}. y|x \Rightarrow y = 1 \vee y = x \end{aligned}$$

- **Definition:** Let p, n be a natural numbers. Then p is a prime factor of n , if p is prime and divides n .

$$\begin{aligned} isprimefactor &\subseteq \mathbb{N} \times \mathbb{N} \\ isprimefactor(p, n) &:\Leftrightarrow isprime(p) \wedge p|n \end{aligned}$$

Implicit Definitions

- An implicit function definition

$$f : T_1 \times \dots \times T_n \rightarrow T$$

$$f(x_1, \dots, x_n) := \text{choose } y \in T. F_{x,y}$$

- Function constant f of arity n .
- Type signature $T_1 \times \dots \times T_n \rightarrow T$.
- Parameters x_1, \dots, x_n (variables).
- Result variable y .
- Result condition $F_{x,y}$ (a formula whose free variables occur in x_1, \dots, x_n, y).

We thus know $\forall x_1 \in T_1, \dots, x_n \in T_n. (\exists y \in T. F_{x,y}) \Rightarrow \text{let } y = f(x_1, \dots, x_n) \text{ in } F_{x,y}$.

Examples

- **Definition:** A root of x is some y such that y squared is x (if such a y exists).

$$\begin{aligned} aRoot &: \mathbb{R} \rightarrow \mathbb{R} \\ aRoot(x) &:= \text{choose } y \in \mathbb{R}. y^2 = x \end{aligned}$$

- **Definition:** The root of $x \geq 0$ is that y such that the square of y is x and $y \geq 0$.

$$\begin{aligned} theRoot &: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0} \\ theRoot(x) &:= \text{choose } y \in \mathbb{R}_{\geq 0}. y^2 = x \wedge y \geq 0 \end{aligned}$$

- **Definition:** The quotient q of m and $n \neq 0$ is such that $m = n \cdot q + r$ for some $r < n$.

$$\begin{aligned} quotient &: \mathbb{N} \times \mathbb{N} \setminus \{0\} \rightarrow \mathbb{N} \\ quotient(m, n) &:= \text{choose } q \in \mathbb{N}. \exists r \in \mathbb{N}. m = n \cdot q + r \wedge r < n \end{aligned}$$

- **Definition:** The $gcd(x, y)$ of x, y (not both 0), is the greatest number dividing x and y .

$$\begin{aligned} gcd &: (\mathbb{N} \times \mathbb{N}) \setminus \{(0, 0)\} \rightarrow \mathbb{N} \\ gcd(x, y) &:= \text{choose } z \in \mathbb{N}. z|x \wedge z|y \wedge \forall z' \in \mathbb{N}. z'|x \wedge z'|y \Rightarrow z' \leq z \end{aligned}$$

Function result need not be uniquely defined (may be even arbitrary).

Predicates versus Functions

A predicate gives rise to functions in two ways.

- A **predicate**:

$$\text{isprimefactor} \subseteq \mathbb{N} \times \mathbb{N}$$

$$\text{isprimefactor}(p, n) :\Leftrightarrow \text{isprime}(p) \wedge p|n$$

- An **implicitly defined function**:

$$\text{someprimefactor} : \mathbb{N} \rightarrow \mathbb{N}$$

$$\text{someprimefactor}(n) := \text{choose } p \in \mathbb{N}. \text{isprimefactor}(p, n)$$

- An **explicitly defined function** whose result is a set:

$$\text{allprimefactors} : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N})$$

$$\text{allprimefactors}(n) := \{p \mid p \in \mathbb{N} \wedge \text{isprimefactor}(p, n)\}$$

The preferred style of definition is a matter of taste and purpose.

The Adequacy of Specifications

Given a specification

Input: x where P_x **Output:** y where $Q_{x,y}$

we may ask the following questions:

- **Is precondition satisfiable?** $(\exists x. P_x)$
 - Otherwise no input is allowed.
- **Is precondition not trivial?** $(\exists x. \neg P_x)$
 - Otherwise every input is allowed, why then the precondition?
- **Is postcondition always satisfiable?** $(\forall x. P_x \Rightarrow \exists y. Q_{x,y})$
 - Otherwise no implementation is legal.
- **Is postcondition not always trivial?** $(\exists x, y. P_x \wedge \neg Q_{x,y})$
 - Otherwise every implementation is legal.
- **Is result unique?** $(\forall x, y_1, y_2. P_x \wedge Q_{x,y_1} \wedge Q_{x,y_2} \Rightarrow y_1 = y_2)$
 - Whether this is required, depends on our expectations.

Example: The Problem of Integer Division

Input: $m \in \mathbb{N}, n \in \mathbb{N}$ **Output:** $q \in \mathbb{N}, r \in \mathbb{N}$ where $m = n \cdot q + r$

- The postcondition is always satisfiable but not trivial.
 - For $m = 13, n = 5$, e.g., $q = 2, r = 3$ is legal but $q = 2, r = 4$ is not.
- But the result is not unique.
 - For $m = 13, n = 5$, both $q = 2, r = 3$ and $q = 1, r = 8$ are legal.

Input: $m \in \mathbb{N}, n \in \mathbb{N}$ **Output:** $q \in \mathbb{N}, r \in \mathbb{N}$ where $m = n \cdot q + r \wedge r < n$

- Now the postcondition is not always satisfiable.
 - For $m = 13, n = 0$, no output is legal.

Input: $m \in \mathbb{N}, n \in \mathbb{N}$ where $n \neq 0$ **Output:** $q \in \mathbb{N}, r \in \mathbb{N}$ where $m = n \cdot q + r \wedge r < n$

- The precondition is not trivial but satisfiable.
 - $m = 13, n = 0$ is not legal but $m = 13, n = 5$ is.
- The postcondition is always satisfiable and result is unique.
 - For $m = 13, n = 5$, only $q = 2, r = 3$ is legal.

Example: The Problem of Linear Search

Given a finite integer sequence a and an integer x , determine the smallest position p at which x occurs in a ($p = -1$, if x does not occur in a).

Example: $a = [2, 3, 5, 7, 5, 11]$, $x = 5 \leadsto p = 2$

Input: $a \in \mathbb{Z}^*, x \in \mathbb{Z}$

Output: $p \in \mathbb{N} \cup \{-1\}$ where

let $n = \text{length}(a)$ in

if $\exists p \in \mathbb{N}. \underline{p < n \wedge a[p] = x}$

then $\underline{p < n \wedge a[p] = x} \wedge (\forall q \in \mathbb{N}. \underline{q < n \wedge a[q] = x} \Rightarrow p \leq q)$

else $p = -1$

All inputs are legal; a result with the specified property always exists and is uniquely determined.

Example: The Problem of Binary Search

Given a finite integer sequence a sorted in ascending order and an integer x , determine some position p at which x occurs in a ($p = -1$, if x does not occur in a).

Example: $a = [2, 3, 5, 5, 5, 7, 11]$, $x = 5 \leadsto p \in \{2, 3, 4\}$

Input: $a \in \mathbb{Z}^*$, $x \in \mathbb{Z}$ where

let $n = \text{length}(a)$ in $\forall k \in \mathbb{N}. k < n - 1 \Rightarrow a[k] \leq a[k + 1]$

Output: $p \in \mathbb{N} \cup \{-1\}$ where

if $\exists p \in \mathbb{N}. \underline{p < n \wedge a[p] = x}$

then $\underline{p < n \wedge a[p] = x}$

else $p = -1$

Not all inputs are legal; for every legal input, a result with the specified property exists but may not be unique.

Example: The Problem of Sorting

Given a finite integer sequence a , determine that permutation b of a that is sorted in ascending order.

Example: $a = [5, 3, 7, 2, 3] \rightsquigarrow b = [2, 3, 3, 5, 7]$

Input: $a \in \mathbb{Z}^*$

Output: $b \in \mathbb{Z}^*$ where

let $n = \text{length}(a)$ in

$\text{length}(b) = n \wedge (\forall k \in \mathbb{N}. k < n - 1 \Rightarrow b[k] \leq b[k + 1]) \wedge$

$\exists p \in \mathbb{N}^*. \text{length}(p) = n \wedge$

$(\forall k \in \mathbb{N}. k < n \Rightarrow p[k] < n) \wedge$

$(\forall k_1 \in \mathbb{N}, k_2 \in \mathbb{N}. k_1 < n \wedge k_2 < n \wedge k_1 \neq k_2 \Rightarrow p[k_1] \neq p[k_2]) \wedge$

$(\forall k \in \mathbb{N}. k < n \Rightarrow a[k] = b[p[k]])$

All inputs are legal; the specified result exists and is uniquely determined.

1. Specifying Problems

2. The RISC Algorithm Language (RISCAL)

3. Modeling Computations

4. The Temporal Logic of Actions (TLA)

The RISC Algorithm Language (RISCAL)

- **A system for formally modeling mathematical theories and algorithms.**
 - Research Institute for Symbolic Computation (RISC), 2016–.
 - <http://www.risc.jku.at/research/formal/software/RISCAL>
 - Implemented in Java with SWT library for the GUI.
 - Tested under Linux only; freely available as open source (GPL3).
- **A language for the defining mathematical theories and algorithms.**
 - A static type system with only finite types (of parameterized sizes).
 - Predicates, explicitly (also recursively) and implicitly def.d functions.
 - Theorems (universally quantified predicates expected to be true).
 - Procedures (also recursively defined).
 - Pre- and post-conditions, invariants, termination measures.
- **A framework for evaluating/executing all definitions.**
 - Model checking: predicates, functions, theorems, procedures, annotations may be evaluated/executed for all possible inputs.
 - All paths of a non-deterministic execution may be elaborated.
 - The execution/evaluation may be visualized.

The RISC Algorithm Language (RISCAL)

RISCAL divide.txt &

The screenshot displays the RISC Algorithm Language (RISCAL) IDE. The main window shows a file named `robots0.txt` containing a program for robot collision avoidance. The program defines a system of robots moving in a square without colliding, using a safety property and an initial state condition. The analysis window on the right shows the results of the analysis, including the translation of the program to a logic solver, the execution of the analysis, and the visualization of the results.

```
1 // -----
2 // a simple system of robots moving in a square without colliding
3 // -----
4
5 val R:N; // number of robots
6 val P:N; // number of positions
7
8 axiom notzero = R ≥ 1 ∧ P ≥ 1;
9
10 type Robot = N[R-1];
11 type Position = N[P-1];
12 enumtype Direction = Stop | Left | Right | Up | Down;
13 type Positions = Array[R,Position];
14
15 // the desired safety property of the system:
16 // no two robots are at the same position
17 pred noCollision(x:Positions, y:Positions) =
18   ∀ r1:Robot, r2:Robot with r1 < r2. x[r1] ≠ x[r2] ∧ y[r1] ≠ y[r2];
19
20 // the initial state condition of the system:
21 // robots are at different positions
22 pred init(x:Positions, y:Positions) =
23   noCollision(x, y);
24
25 // any robot different from r may move to position xr, yr
26 pred anyOtherAt(x:Positions, y:Positions, r:Robot, xr:Position, yr:Position) =
27   ∃ r0:Robot with r0 ≠ r. xr = x[r0] ∧ yr = y[r0];
28
29 // the relation between the current system state and the new direction d of robot r
30 pred nextDir(x:Positions, y:Positions, r:Robot, d:Direction) =
31   (d = Direction!Stop)
32   ∨ (d = Direction!Left ∧ x[r] > 0 ∧ ¬anyOtherAt(x, y, r, x[r]-1, y[r]))
33   ∨ (d = Direction!Right ∧ x[r] < P-1 ∧ ¬anyOtherAt(x, y, r, x[r]+1, y[r]))
34   ∨ (d = Direction!Up ∧ y[r] > 0 ∧ ¬anyOtherAt(x, y, r, x[r], y[r]-1))
35   ∨ (d = Direction!Down ∧ y[r] < P-1 ∧ ¬anyOtherAt(x, y, r, x[r], y[r]+1))
36
37
38 // the new x-positions if robot r moves into direction dr
39 fun moveX(x:Positions, r:Robot, dr: Direction):Positions =
40   match dr with
41   {
42     Left -> x with [r] = x[r]-1;
43     Right -> x with [r] = x[r]+1;
```

Analysis

Translation: ☒ Nondeterminism Default Value: 0 Other Values:

Execution: ☒ Silent Inputs: Per Mille: Branches:

Visualization: ☐ Trace ☐ Tree Width: 4580X Height: 600





Parallelism: ☒ Multi-Threaded Threads: 4 ☐ Distributed Servers:

Operation:

Using R=2.
Using P=5.
Computing the truth value of notzero...
Using N=2.
Type checking and translation completed.
Executing invIsStrongEnough(Array[Z],Array[Z]) with all 625 inputs.
Execution completed for ALL inputs (101 ms, 625 checked, 0 inadmissible).
Executing invHoldsInitially(Array[Z],Array[Z]) with all 625 inputs.
Execution completed for ALL inputs (82 ms, 625 checked, 0 inadmissible).
Executing invIsPreserved(Array[Z],Array[Z]) with all 625 inputs.
PARALLEL execution with 4 threads (output disabled).
365 inputs (288 checked, 0 inadmissible, 0 ignored, 77 open)...
625 inputs (572 checked, 0 inadmissible, 0 ignored, 53 open)...
Execution completed for ALL inputs (4409 ms, 625 checked, 0 inadmissible).

Using RISCAL

See also the (printed/online) “Tutorial and Reference Manual”.

- Press button  (or <Ctrl>-s) to save specification.
 - Automatically processes (parses and type-checks) specification.
 - Press button  to re-process specification.
- Choose values for undefined constants in specification.
 - Natural number for `val const: N`.
 - Default Value: used if no other value is specified.
 - Other Values: specific values for individual constants.
- Select Operation from menu and then press button .
 - Executes operation for chosen constant values and all possible inputs.
 - Option Silent: result of operation is not printed.
 - Option Nondeterminism: all execution paths are taken.
 - Option Multi-threaded: multiple threads execute different inputs.
 - Press button  to abort execution.

Typing Mathematical Symbols

ASCII String	Unicode Character
Int	\mathbb{Z}
Nat	\mathbb{N}
:=	$:=$
true	\top
false	\perp
~	\neg
/\	\wedge
\/	\vee
=>	\Rightarrow
<=>	\Leftrightarrow
forall	\forall
exists	\exists
sum	\sum
product	\prod

ASCII String	Unicode Character
~=	\neq
<=	\leq
>=	\geq
*	\cdot
times	\times
{}	\emptyset
intersect	\cap
union	\cup
Intersect	\bigcap
Union	\bigcup
isin	\in
subsetq	\subseteq
<<	\langle
>>	\rangle

Type the ASCII string and press <Ctrl>-# to get the Unicode character.

Example: Quotient and Remainder

Given naturals n and m , compute the quotient q and remainder r of n divided by m .

```
// the type of natural numbers less than equal N
val N: N;
type Num = N[N];
```

```
// the precondition of the computation
pred pre(n:Num, m:Num)  $\Leftrightarrow$  m  $\neq$  0;
```

```
// the postcondition, first formulation
pred post1(n:Num, m:Num, q:Num, r:Num)  $\Leftrightarrow$ 
  n = m·q + r  $\wedge$ 
   $\forall$ q0:Num, r0:Num.
    n = m·q0 + r0  $\Rightarrow$  r  $\leq$  r0;
```

```
// the postcondition, second formulation
pred post2(n:Num, m:Num, q:Num, r:Num)  $\Leftrightarrow$ 
  n = m·q + r  $\wedge$  r < m;
```


Example: Quotient and Remainder

```
// for all inputs that satisfy the precondition
// both formulations are equivalent:
//  $\forall n:\text{Num}, m:\text{Num}, q:\text{Num}, r:\text{Num}.$ 
//  $\text{pre}(n, m) \Rightarrow (\text{post1}(n, m, q, r) \Leftrightarrow \text{post2}(n, m, q, r));$ 
theorem postEquiv(n:Num, m:Num, q:Num, r:Num)
  requires pre(n, m);
 $\Leftrightarrow \text{post1}(n, m, q, r) \Leftrightarrow \text{post2}(n, m, q, r);$ 

// we will thus use the simpler formulation from now on
pred post(n:Num, m:Num, q:Num, r:Num)  $\Leftrightarrow \text{post2}(n, m, q, r);$ 
```

Check equivalence for all values that satisfy the precondition.

Example: Quotient and Remainder

Choose e.g. $N = 5$.

■ Switch option Silent off:

Executing `postEquiv($\mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z}$)` with all 1296 inputs.

Ignoring inadmissible inputs...

Run 6 of deterministic function `postEquiv(0,1,0,0)`:

Result (0 ms): true

Run 7 of deterministic function `postEquiv(1,1,0,0)`:

Result (0 ms): true

...

Run 1295 of deterministic function `postEquiv(5,5,5,5)`:

Result (0 ms): true

Execution completed for ALL inputs (6314 ms, 1080 checked, 216 inadmissible).

■ Switch option Silent on:

Executing `postEquiv($\mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z}$)` with all 1296 inputs.

Execution completed for ALL inputs (244 ms, 1080 checked, 216 inadmissible).

If theorem is false for some input, an error message is displayed.

Example: Quotient and Remainder

Drop precondition from theorem.

```
theorem postEquiv(n:Num, m:Num, q:Num, r:Num)  $\Leftrightarrow$   
  // requires pre(n, m);  
  post1(n, m, q, r)  $\Leftrightarrow$  post2(n, m, q, r);
```

Executing postEquiv($\mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z}$) with all 1296 inputs.

Run 0 of deterministic function postEquiv(0,0,0,0):

ERROR in execution of postEquiv(0,0,0,0): evaluation of

postEquiv

at line 25 in file divide.txt:

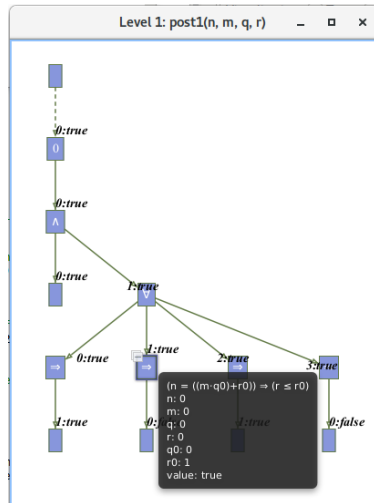
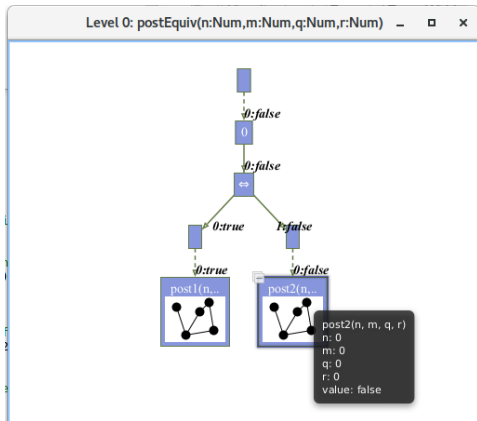
theorem is not true

ERROR encountered in execution.

For $n = 0, m = 0, q = 0, r = 0$, the modified theorem is not true.

Visualizing the Formula Evaluation

Select $N = 1$ and visualization option “Tree”.



Investigate the (pruned) evaluation tree to determine how the truth value of a

Example: Quotient and Remainder

Switch option “Nondeterminism” on.

```
// 1. investigate whether the specified input/output combinations are as desired
fun quotremFun(n:Num, m:Num): Tuple[Num,Num]
  requires pre(n, m);
  ensures post(n, m, result.1, result.2);
= choose q:Num, r:Num with post(n, m, q, r);
```

Executing quotremFun(\mathbb{Z} , \mathbb{Z}) with all 36 inputs.

Ignoring inadmissible inputs...

Branch 0:6 of nondeterministic function quotremFun(0,1):

Result (0 ms): [0,0]

...

Branch 1:35 of nondeterministic function quotremFun(5,5):

No more results (14 ms).

Execution completed for ALL inputs (413 ms, 30 checked, 6 inadmissible).

First validation by inspecting the values determined by output condition
(nondeterminism may produce for some inputs multiple outputs).

Example: Quotient and Remainder

```
// 2. check that some but not all inputs are allowed  
theorem someInput()  $\Leftrightarrow \exists n:\text{Num}, m:\text{Num}. \text{pre}(n, m);$   
theorem notEveryInput()  $\Leftrightarrow \exists n:\text{Num}, m:\text{Num}. \neg \text{pre}(n, m);$ 
```

Executing someInput().

Execution completed (0 ms).

Executing notEveryInput().

Execution completed (0 ms).

A very rough validation of the input condition.

Example: Quotient and Remainder

```
// 3. check whether for all inputs that satisfy the precondition
// there are some outputs that satisfy the postcondition
theorem someOutput(n:Num, m:Num)
  requires pre(n, m);
 $\Leftrightarrow \exists q:\text{Num}, r:\text{Num}. \text{post}(n, m, q, r);$ 
```

```
// 4. check that not every output satisfies the postcondition
theorem notEveryOutput(n:Num, m:Num)
  requires pre(n, m);
 $\Leftrightarrow \exists q:\text{Num}, r:\text{Num}. \neg \text{post}(n, m, q, r);$ 
```

Executing someOutput(\mathbb{Z}, \mathbb{Z}) with all 36 inputs.

Execution completed for ALL inputs (5 ms, 30 checked, 6 inadmissible).

Executing notEveryOutput(\mathbb{Z}, \mathbb{Z}) with all 36 inputs.

Execution completed for ALL inputs (5 ms, 30 checked, 6 inadmissible).

A very rough validation of the output condition.

Example: Quotient and Remainder

```
// 5. check that the output is uniquely defined
// (optional, need not generally be the case)
theorem uniqueOutput(n:Num, m:Num)
  requires pre(n, m);
  ⇔
  ∀q:Num, r:Num. post(n, m, q, r) ⇒
  ∀q0:Num, r0:Num. post(n, m, q0, r0) ⇒
    q = q0 ∧ r = r0;
```

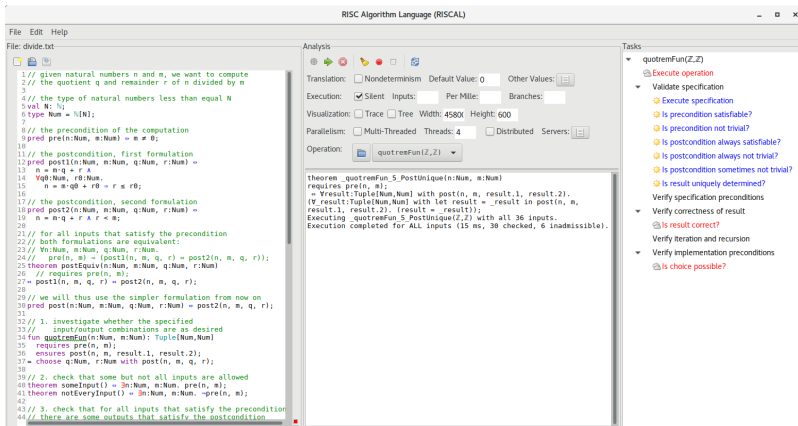
Executing uniqueOutput(\mathbb{Z} , \mathbb{Z}) with all 36 inputs.

Execution completed for ALL inputs (18 ms, 30 checked, 6 inadmissible).

The output condition indeed determines the outputs uniquely.

Validating the Specification of an Operation

Select operation `quotRemFun` and press the button  “Show/Hide Tasks”.



Automatic generation of those formulas that validate a specification.

Example: Quotient and Remainder

Right-click to print definition of a formula, double-click to check it.

For every input, is postcondition true for only one output?

```
theorem _quotremFun_5_PostUnique(n:Num, m:Num)
requires pre(n, m);
  ⇔ ∀result:Tuple[Num,Num] with post(n, m, result.1, result.2).
    (∀_result:Tuple[Num,Num] with let result = _result in
      post(n, m, result.1, result.2). (result = _result));
```

Using N=5.

Type checking and translation completed.

Executing _quotremFun_5_PostUnique(\mathbb{Z} , \mathbb{Z}) with all 36 inputs.

Execution completed for ALL inputs (7 ms, 30 checked, 6 inadmissible).

The output is indeed uniquely defined by the output condition.

Example: Quotient and Remainder

```
// 6. check whether the algorithm satisfies the specification
proc quotRemProc(n:Num, m:Num): Tuple[Num,Num]
  requires pre(n, m);
  ensures let q=result.1, r=result.2 in post(n, m, q, r);
{
  var q: Num = 0;
  var r: Num = n;
  while r ≥ m do
  {
    r := r-m;
    q := q+1;
  }
  return ⟨q,r⟩;
}
```

Check whether the algorithm satisfies the specification.

Example: Quotient and Remainder

Executing `quotRemProc(\mathbb{Z}, \mathbb{Z})` with all 36 inputs.

Ignoring inadmissible inputs...

Run 6 of deterministic function `quotRemProc(0,1)`:

Result (0 ms): [0,0]

Run 7 of deterministic function `quotRemProc(1,1)`:

Result (0 ms): [1,0]

...

Run 32 of deterministic function `quotRemProc(2,5)`:

Result (0 ms): [0,2]

Run 33 of deterministic function `quotRemProc(3,5)`:

Result (0 ms): [0,3]

Run 34 of deterministic function `quotRemProc(4,5)`:

Result (0 ms): [0,4]

Run 35 of deterministic function `quotRemProc(5,5)`:

Result (1 ms): [1,0]

Execution completed for ALL inputs (161 ms, 30 checked, 6 inadmissible).

A verification of the algorithm by checking all possible executions.

Example: Quotient and Remainder

```
proc quotRemProc(n:Num, m:Num): Tuple[Num,Num]
  requires pre(n, m);
  ensures post(n, m, result.1, result.2);
{
  var q: Num = 0; var r: Num = n;
  while r > m do // error!
  {
    r := r-m; q := q+1;
  }
  return <q,r>;
}
```

Executing quotRemProc(\mathbb{Z}, \mathbb{Z}) with all 36 inputs.

ERROR in execution of quotRemProc(1,1): evaluation of

ensures let q = result.1, r = result.2 in post(n, m, q, r);

at line 65 in file divide.txt:

postcondition is violated by result [0,1]

ERROR encountered in execution.

A falsification of an incorrect algorithm.

Example: Sorting an Array

```
val N:Nat; val M:Nat;
type nat = Nat[M]; type array = Array[N,nat]; type index = Nat[N-1];

proc sort(a:array): array
  ensures  $\forall i:\text{nat}. i < N-1 \Rightarrow \text{result}[i] \leq \text{result}[i+1]$ ;
  ensures  $\exists p:\text{Array}[N,\text{index}]. (\forall i:\text{index}, j:\text{index}. i \neq j \Rightarrow p[i] \neq p[j]) \wedge$ 
     $(\forall i:\text{index}. a[i] = \text{result}[p[i]])$ ;
{
  var b:array = a;
  for var i:Nat[N]:=1; i<N; i:=i+1 do {
    var x:nat := b[i];
    var j:Int[-1,N] := i-1;
    while j  $\geq$  0  $\wedge$  b[j] > x do {
      b[j+1] := b[j];
      j := j-1;
    }
    b[j+1] := x;
  }
  return b;
}
```

Example: Sorting an Array

Using N=5.

Using M=5.

Type checking and translation completed.

Executing sort(Array[\mathbb{Z}]) with all 7776 inputs.

1223 inputs (1223 checked, 0 inadmissible, 0 ignored)...

2026 inputs (2026 checked, 0 inadmissible, 0 ignored)...

...

5792 inputs (5792 checked, 0 inadmissible, 0 ignored)...

6118 inputs (6118 checked, 0 inadmissible, 0 ignored)...

6500 inputs (6500 checked, 0 inadmissible, 0 ignored)...

6788 inputs (6788 checked, 0 inadmissible, 0 ignored)...

7070 inputs (7070 checked, 0 inadmissible, 0 ignored)...

7354 inputs (7354 checked, 0 inadmissible, 0 ignored)...

7634 inputs (7634 checked, 0 inadmissible, 0 ignored)...

Execution completed for ALL inputs (32606 ms, 7776 checked, 0 inadmissible).

Not all nondeterministic branches may have been considered.

Also this algorithm can be automatically checked.

Model Checking versus Proving

Two fundamental techniques for validation/verification.

- **Model checking:** processing a semantic model.
 - Fully automatic, no human interaction is required.
 - Completely possible only if the model is finite.
 - State space explosion: “finite” actually means “not too big”.
- **Proving:** constructing a logical deduction.
 - Assumes a sound deduction calculus.
 - Also possible if the model is infinite.
 - Complexity of deduction is independent of size of model.
 - Many properties can be automatically proved (automated reasoners); in general, however, interaction with a human is required (proof assistants).

While verifying the validity of a conjecture generally requires deduction, its invalidity can be often quickly established by checking.

1. Specifying Problems

2. The RISC Algorithm Language (RISCAL)

3. Modeling Computations

4. The Temporal Logic of Actions (TLA)

Computational Systems

Programs are just special cases of “(computational) systems”.

■ Computational System

- One or more active components.
- Deterministic or nondeterministic behavior.
- May or may not terminate.

■ Safety

- “Nothing bad will ever happen.”
- Partial correctness of programs: for every admissible input, if the program terminates, its output does not violate the output condition.

■ Liveness

- “Something good will eventually happen.”
- Termination of programs: for every input, the program eventually terminates.

General goal is to establish the safety and liveness of computational systems.

Transition Systems

Any computational system can be modelled as a **transition system** $T = (S, I, R)$.

- **State space** $S = S_1 \times \dots \times S_n$: the set of all possible system states.
 - Determined by the possible values of system variables x_1, \dots, x_n with values from (finite or infinite) domains S_1, \dots, S_n .
- **Initial states** $I \subseteq S$: the possible starts of the execution of the system.
 - Typically defined by an a predicate I_x on the system variables x_1, \dots, x_n .
- **Transition relation** $R \subseteq S \times S$: the possible execution steps.
 - Typically defined by a predicate $R_{x,x'}$ between the **prestate** values x and the **poststate** values x' of the program variables.

Nondeterminism: for some prestate x there may be multiple poststates x' .

Example

System $C = (S, I, R)$ with counters x and y which may be independently incremented.

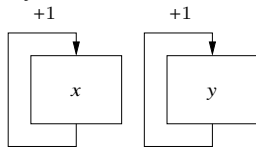
$$S := \mathbb{Z} \times \mathbb{Z}$$

$$I(x, y) :\Leftrightarrow x = y \wedge y \geq 0$$

$$R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$$

$$(x' = x + 1 \wedge y' = y) \vee$$

$$(x' = x \wedge y' = y + 1)$$



- Infinitely many starting states.

$$[x = 0, y = 0], [x = 1, y = 1], [x = 2, y = 2], \dots$$

- In each state two possibilities.

$$[x = 2, y = 3] \rightarrow [x = 3, y = 3]$$

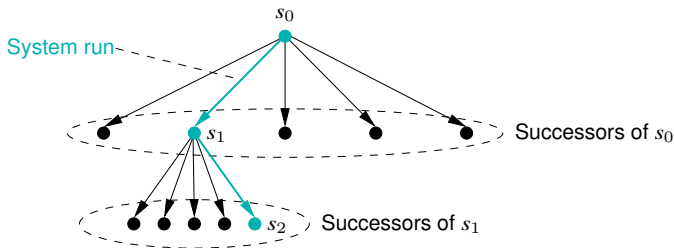
$$\rightarrow [x = 2, y = 4]$$

A nondeterministic system.

System Runs

Transition system $T = (S, I, R)$.

- **System run:** (finite or infinite) sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$ of states in S .
 - s_0 is initial: $I(s_0)$.
 - $s_i \rightarrow s_{i+1}$ ist a transition: $R(s_i, s_{i+1})$.
 - If run stops in s_n , then s_n has no successor: $\neg R(s_n, s')$, for all $s' \in S$.



System runs can be understood as paths in a directed graph.

Example

System $C = (S, I, R)$.

$$S := \mathbb{Z} \times \mathbb{Z}$$

$$I(x, y) :\Leftrightarrow x = y \wedge y \geq 0$$

$$R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$$

$$(x' = x + 1 \wedge y' = y) \vee$$

$$(x' = x \wedge y' = y + 1)$$

■ **Safety:** $\Box(x \geq 0 \wedge y \geq 0)$

- Both x and y never become negative.
- True, because every system run has this property.

■ **Liveness:** $\Diamond x \geq 1$.

- Variable x eventually becomes greater equal 1.
- False, because this system run does not have this property.

$$[x = 0, y = 0] \rightarrow [x = 0, y = 1] \rightarrow [x = 0, y = 2] \rightarrow [x = 0, y = 3] \rightarrow \dots$$

Verifying Safety

We only consider the verification of a safety property.

- $M \models \Box F$.

- Verify that formula F is an **invariant** of system M .

- $M = (S, I, R)$.

- $I(s) :\Leftrightarrow \dots$

- $R(s, s') :\Leftrightarrow R_0(s, s') \vee R_1(s, s') \vee \dots \vee R_{n-1}(s, s')$.

- **Proof by induction.**

- $\forall s. I(s) \Rightarrow F(s)$.

- F holds in every initial state.

- $\forall s, s'. F(s) \wedge R(s, s') \Rightarrow F(s')$.

- Each transition preserves F .
 - Reduces to a number of subproofs:

$$F(s) \wedge R_0(s, s') \Rightarrow F(s')$$

...

$$F(s) \wedge R_{n-1}(s, s') \Rightarrow F(s')$$

1. Specifying Problems

2. The RISC Algorithm Language (RISCAL)

3. Modeling Computations

4. The Temporal Logic of Actions (TLA)

The Temporal Logic of Actions (TLA)

- **Leslie Lamport** (Microsoft Research since 2001).
 - ACM Turing Award 2013.
- **TLA model of a system:**

$$I_x \wedge \Box[R]_x \wedge \mathbf{WF}_x(A) \wedge \dots$$

- Initial condition I_x .
- Transition relation $[R]_x$:
 - $[R]_x \equiv (R \vee x = x')$
 - $x = x'$: stutter step (nothing changes).
- Fairness conditions:
 - Conjunction of formulas $\mathbf{WF}_x(A)$ and/or $\mathbf{SF}_x(A)$ for actions A .

<http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html>

Example

$$X \equiv \wedge x' = x + 1$$

$$\wedge y' = y$$

$$Y \equiv \wedge y' = y + 1$$

$$\wedge x' = x$$

$$S \equiv \wedge (x = 0) \wedge (y = 0)$$

$$\wedge \Box [X \vee Y]_{\langle x, y \rangle}$$

$$\wedge \mathbf{WF}_{\langle x, y \rangle}(X) \wedge \mathbf{WF}_{\langle x, y \rangle}(Y)$$

$$[x = 0, x = 0] \rightarrow [x = 1, y = 0] \rightarrow [x = 1, y = 0] \rightarrow [x = 1, y = 1] \rightarrow \dots$$

System is described in a structured way by the logical composition of actions.

TLA+

TLA is not just a logic.

- **TLA+:** A formal specification language based on TLA.

- Values from the theory of sets (no static type system).

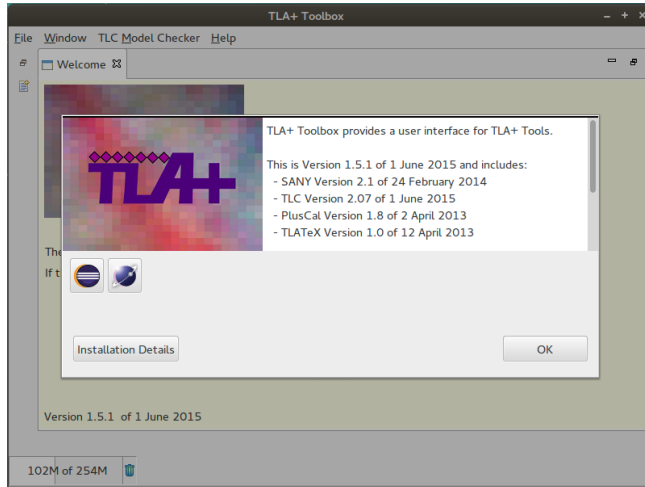
Chris Newcombe et al. *How Amazon Web Services Uses Formal Methods*.
Communications of the ACM, vol. 58 no. 4, pages 66-73, April 2015.
<https://doi.org/10.1145/2699417>

- **TLA+ Toolbox:** an IDE for various TLA tools.

- Writing and syntax checking of TLA+ specifications.
- Pretty printer for generation of \LaTeX documents.
- Translator from the algorithmic language PlusCal to TLA+.
- Simulation and model checking of TLA+-specifications.
- Derivation and checking of TLA+ proofs.

<http://research.microsoft.com/en-us/um/people/lamport/tla/tools.html>

TLA+ Toolbox



Example (Plain Text)

```
----- MODULE Counter -----  
EXTENDS Naturals  
VARIABLE x,y  
  
I == x = 0 /\ y = 0 (* the initial state condition *)  
  
X == /\ x' = x+1      (* increment x *)  
      /\ y' = y  
Y == /\ x' = x        (* increment y *)  
      /\ y' = y+1  
R == \/ X              (* increment x or y *)  
      \/ Y  
  
var == «x,y»           (* the system variables *)  
  
C == I /\ [] [R]_var /\ WF_var(X) /\ WF_var(Y) (* the whole specification *)  
  
NotNegative == [] (x >= 0 /\ y >= 0)             (* some properties *)  
BecomeOne    == <> (x = 1 /\ y = 1)  
=====
```

Example (L^AT_EX)

MODULE *Counter*

EXTENDS *Naturals*

VARIABLE x, y

the initial state condition

$I \triangleq x = 0 \wedge y = 0$

$X \triangleq \wedge x' = x + 1$ increment x

$\wedge y' = y$

$Y \triangleq \wedge x' = x$ increment y

$\wedge y' = y + 1$

$R \triangleq \vee X$ increment x or y

$\vee Y$

$var \triangleq \langle x, y \rangle$ the system variables

the whole specification

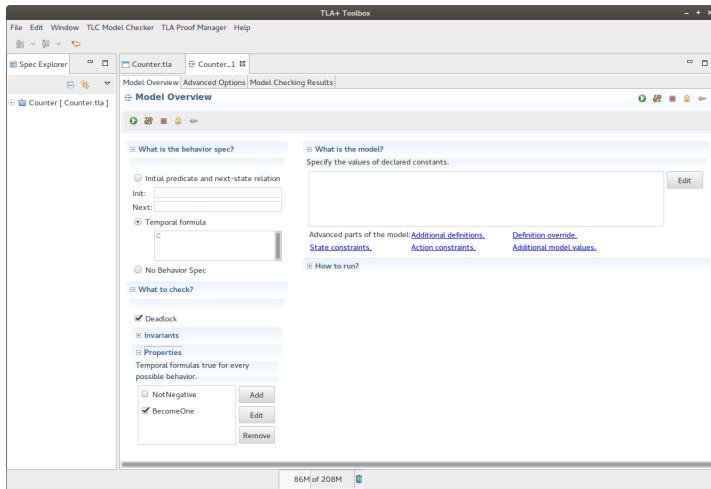
$C \triangleq I \wedge \Box[R]_{var} \wedge \text{WF}_{var}(X) \wedge \text{WF}_{var}(Y)$

some properties

$\text{NotNegative} \triangleq \Box(x \geq 0 \wedge y \geq 0)$

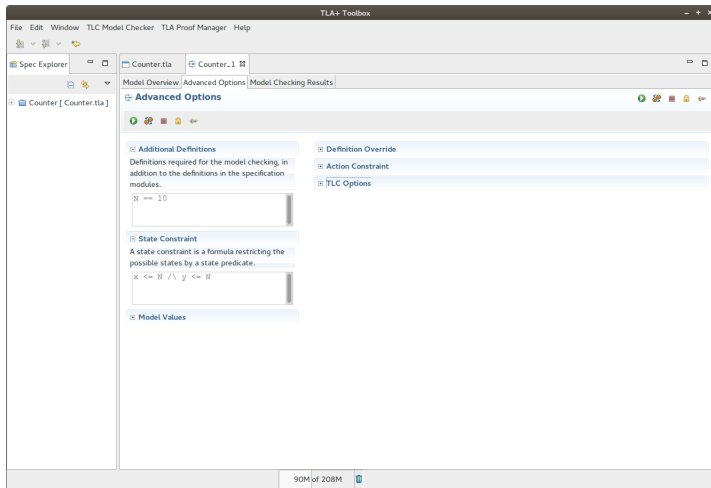
$\text{BecomeOne} \triangleq \Diamond(x = 1 \wedge y = 1)$

The TLC Model Checker



Select specification and properties to be checked.

The TLC Model Checker



If necessary, restrict state space to finite subset.

The TLC Model Checker

The screenshot displays the TLC Model Checker interface. The main window is titled "TLC+ Toolbox" and contains a menu bar (File, Edit, Window, TLC Model Checker, TLA Proof Manager, Help) and a toolbar. Below the toolbar is a "Spec Explorer" pane showing a tree view with "Counter [Counter.tla]". The main area is titled "Model Checking Results" and contains several tabs: "Model Overview", "Advanced Options", and "Model Checking Results". The "Model Checking Results" tab is active, showing a "General" section with the following information:

- Start time: Tue Jul 14 16:56:33 CEST 2015
- End time: Tue Jul 14 16:56:33 CEST 2015
- Last checkpoint time:
- Current status: Not running
- Errors detected: No errors
- Fingerprint collision probability: calculated: 8.0E-16, observed: 7.5E-17

Below the "General" section is a "Statistics" section. It includes a "State space progress (click column header for graph)" table and a "Coverage at 2015-07-14 16:56:33" table.

Time	Diamet	States Fox	Distinct Stats	Queue Size
2015-07-14 16:	21	243	121	0

Module	Location	Count
Counter	line 10, col 9 to line 10, col 1	121
Counter	line 11, col 9 to line 11, col 1	121
Counter	line 8, col 9 to line 8, col 16	121

Below the "Statistics" section are three expandable sections: "Evaluate Constant Expression", "User Output", and "Progress Output". At the bottom of the window, a status bar shows "103M of 247M" and "Spec Status : passed".

Check the selected properties.

The TLC Model Checker

TLA+ Toolbox

File Edit Window TLC Model Checker TLA Proof Manager Help

Spec ... Counter.tla Counter_1

Model Overview Advanced Options Model Checking Results

Model Overview 1_warning_detected

What is the behavior spec?

☐ Initial predicate and next-state relation

Init:

Next:

☒ Temporal formula

C

☐ No Behavior Spec

What to check?

☒ Deadlock

☐ Invariants

☐ Properties

Temporal formulas true for every possible behavior.

☐ NotNegative Add

☐ BecomeOne Edit

☒ $\Box(x+y \leq 5)$ Remove

What is the model?

Specify the values of declared constants.

Advanced parts of the model: [Additional definitions.](#) [Definition over State constraints.](#) [Action constraints.](#) [Additional mod](#)

How to run?

TLA Errors

Counter_1

Invariant BecomeOne is violated.

Error-Trace Exploration

Error-Trace

Name	Value
y	0
<Action line 18 State (num = 4)	
x	3
y	0
<Action line 18 State (num = 5)	
x	4
y	0
<Action line 18 State (num = 6)	
x	5
y	0

Select line in Error Trace to show its value here.

152M of 251M

Spec Status: **parsed**

In the error case a violating system run is displayed.

Example

```
----- MODULE Counter -----
```

```
EXTENDS Naturals, TLC
```

```
VARIABLE x,y
```

```
...
```

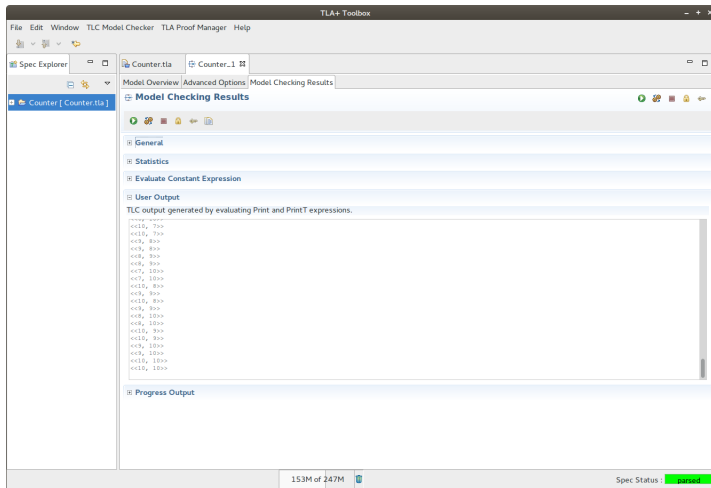
```
C == I /\ [] [R /\ PrintT(«x,y»)]_var /\ WF_var(X) /\ WF_var(Y)
```

```
...
```

```
=====
```

User output may help to validate the model.

The TLC Model Checker



The visited states are printed.