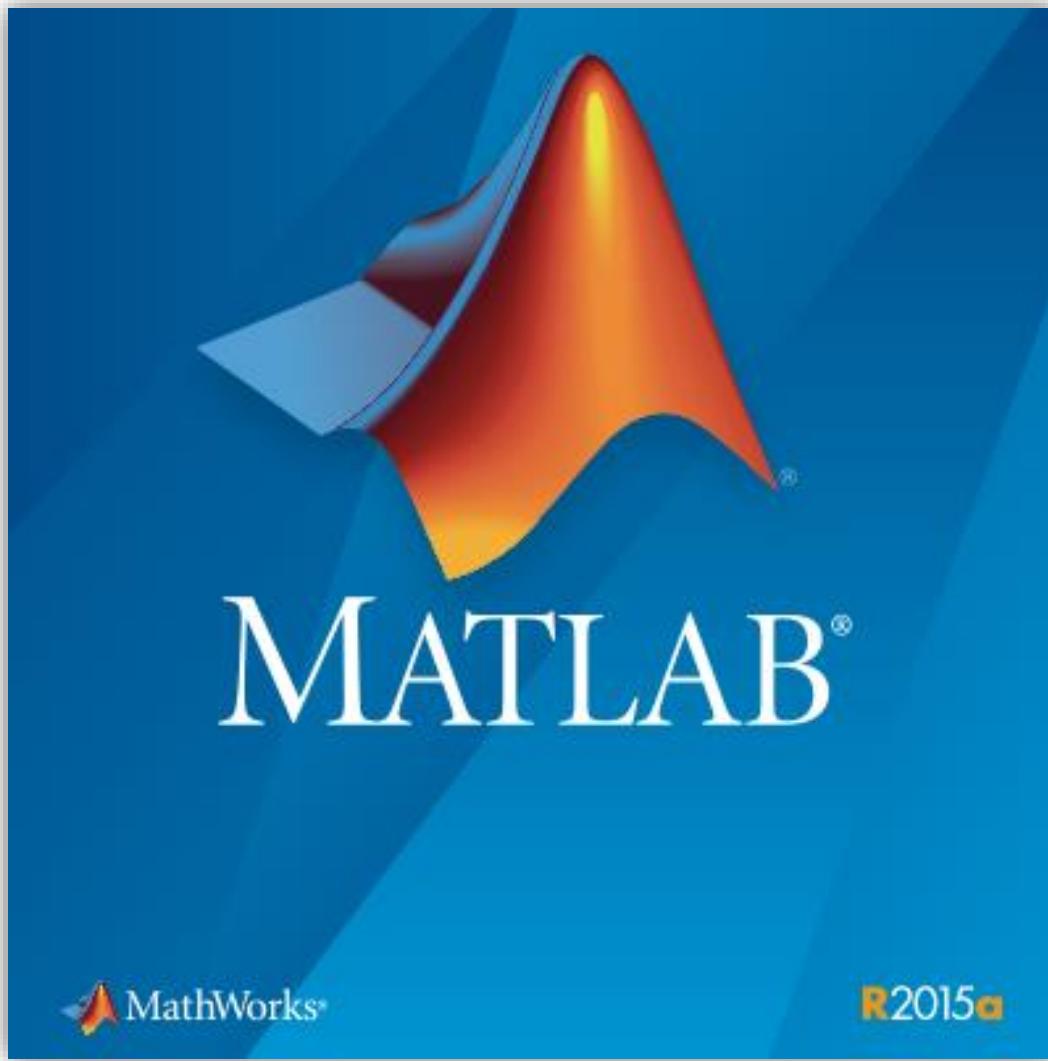


Software Tools

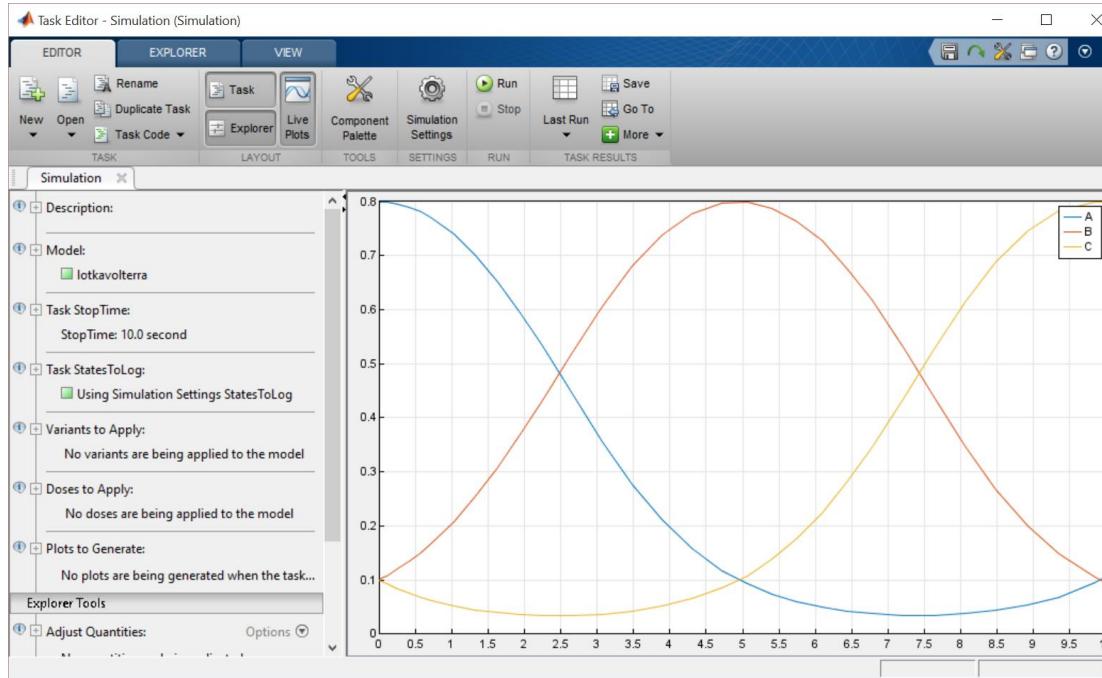
Molecular Programming tutorial
ASE 205

Robyn Lutz, Titus Klinge, James Lathrop, Jack Lutz



MATLAB (Simbiology)

- Designing models of CRNs, simulating them, and exporting them to other tools

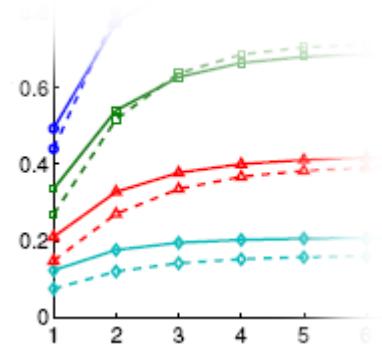
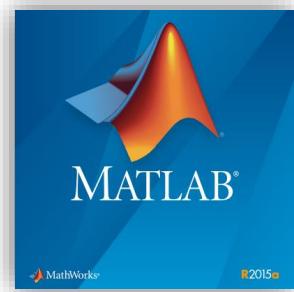


Probabilistic Model Checkers

- Verifying the correctness of molecular systems
- PRISM
 - Very extensive language for requirements
 - Supports many types of models
 - Graphical user interface
- SMART
 - Native support for Petri nets (nice for CRNs)
 - Powerful language features
 - Command line

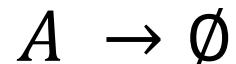
Outline

- MATLAB (Simbiology)
 - Tutorial – designing and simulating CRNs
 - Several CRN examples
- Afternoon break
- Probabilistic Model Checking
 - Overview of PMC
 - PRISM tutorial
 - SMART tutorial



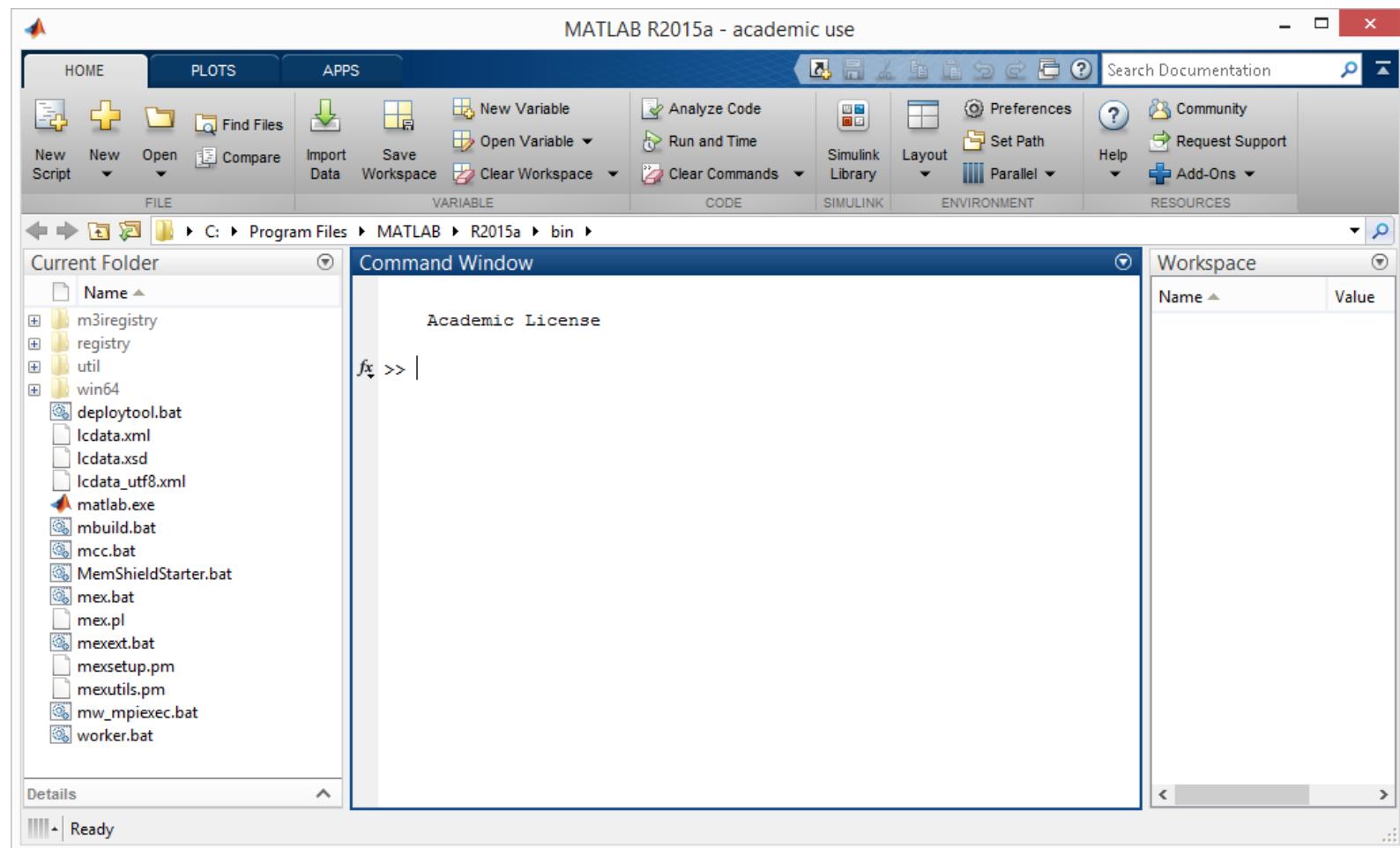
A working example

- Suppose we want a CRN to do the following:
 - Remove all the A molecules from the system
- Here's an idea:

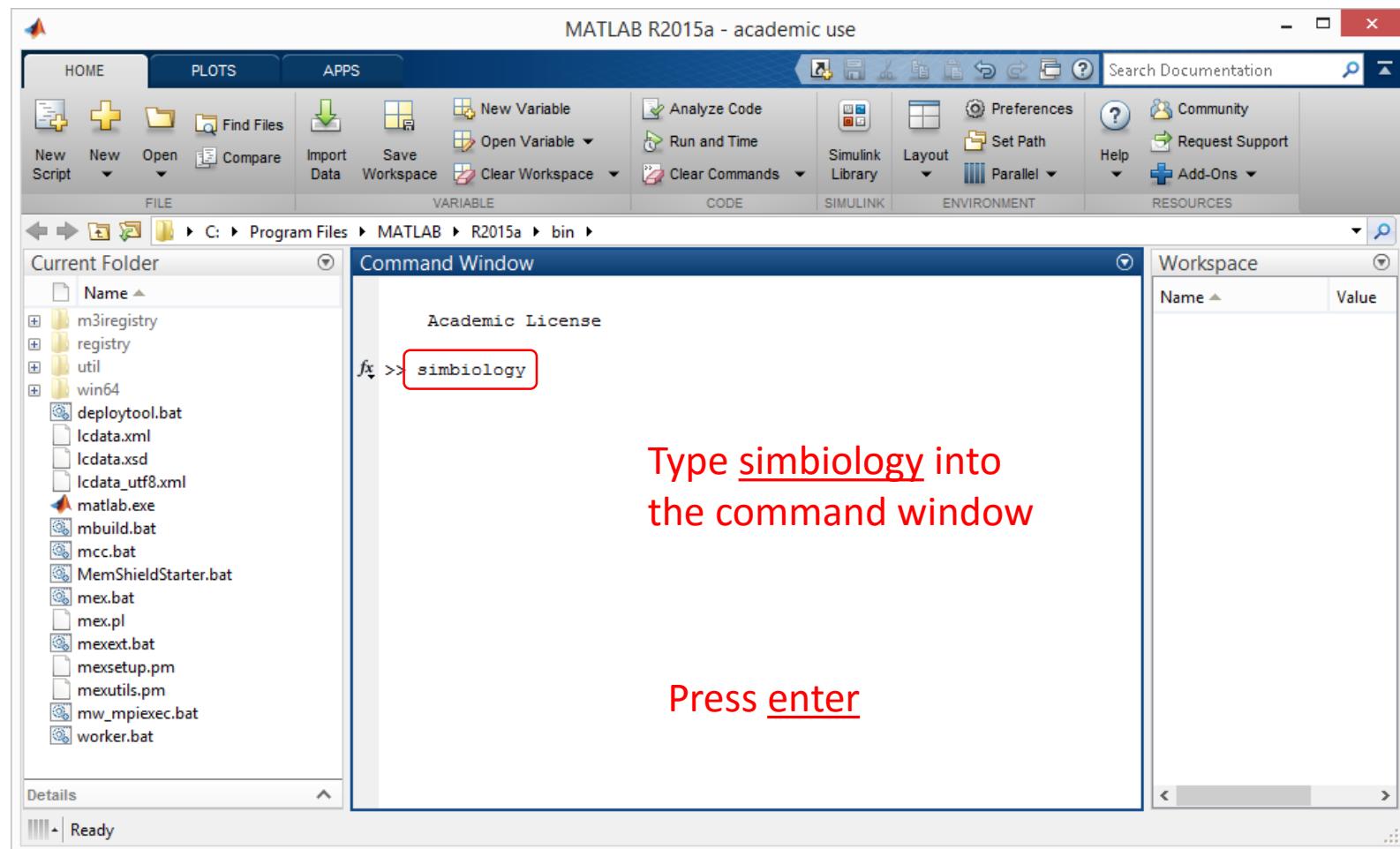


- Want to build and simulate this CRN with MATLAB

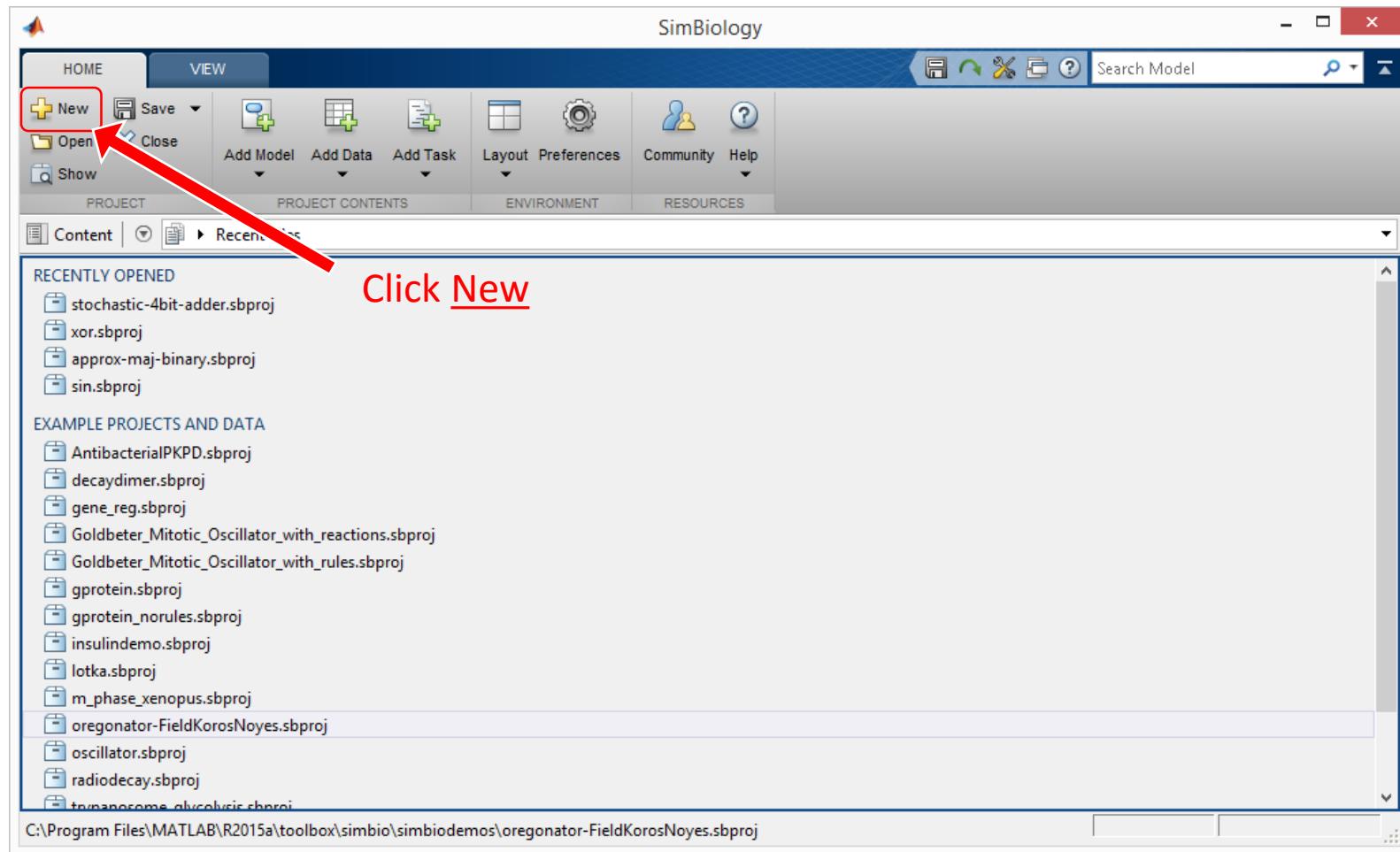
MATLAB



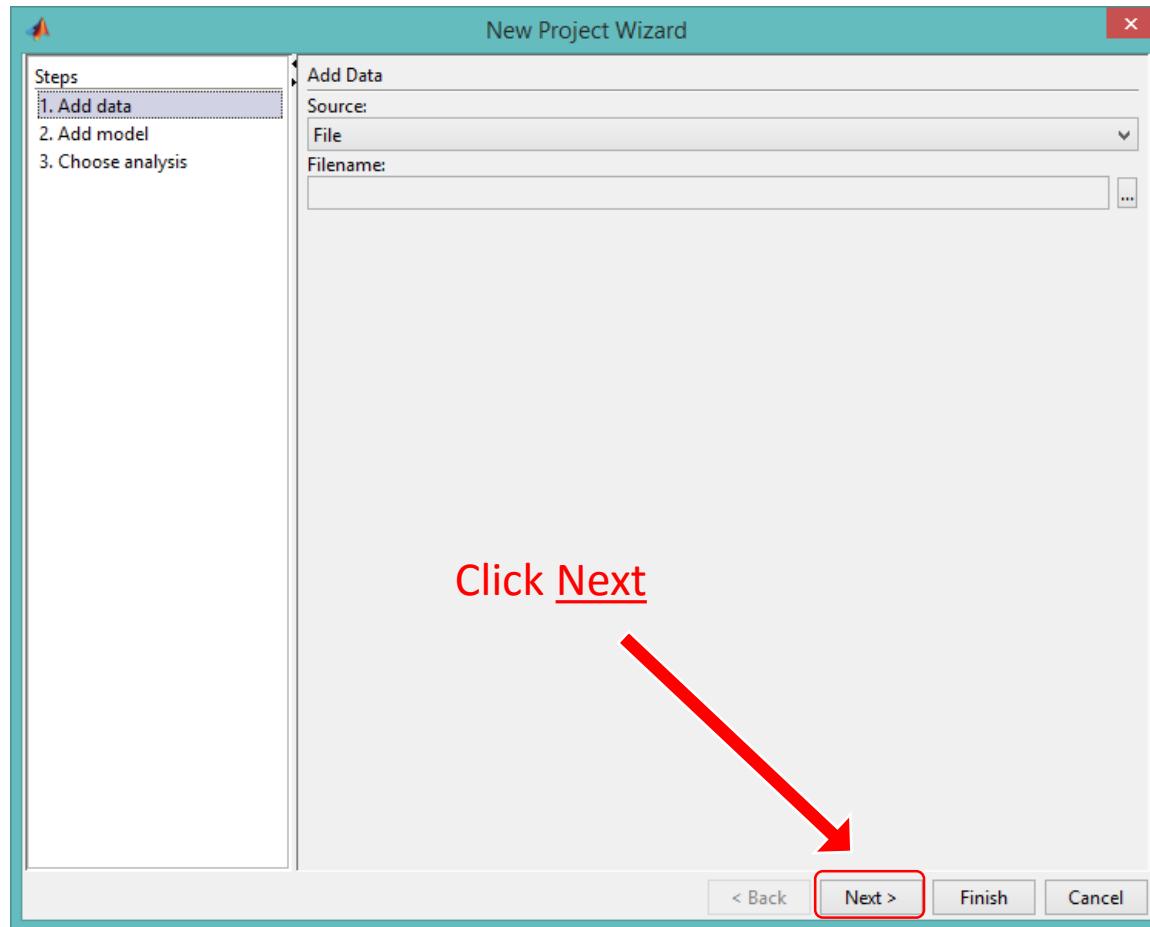
MATLAB



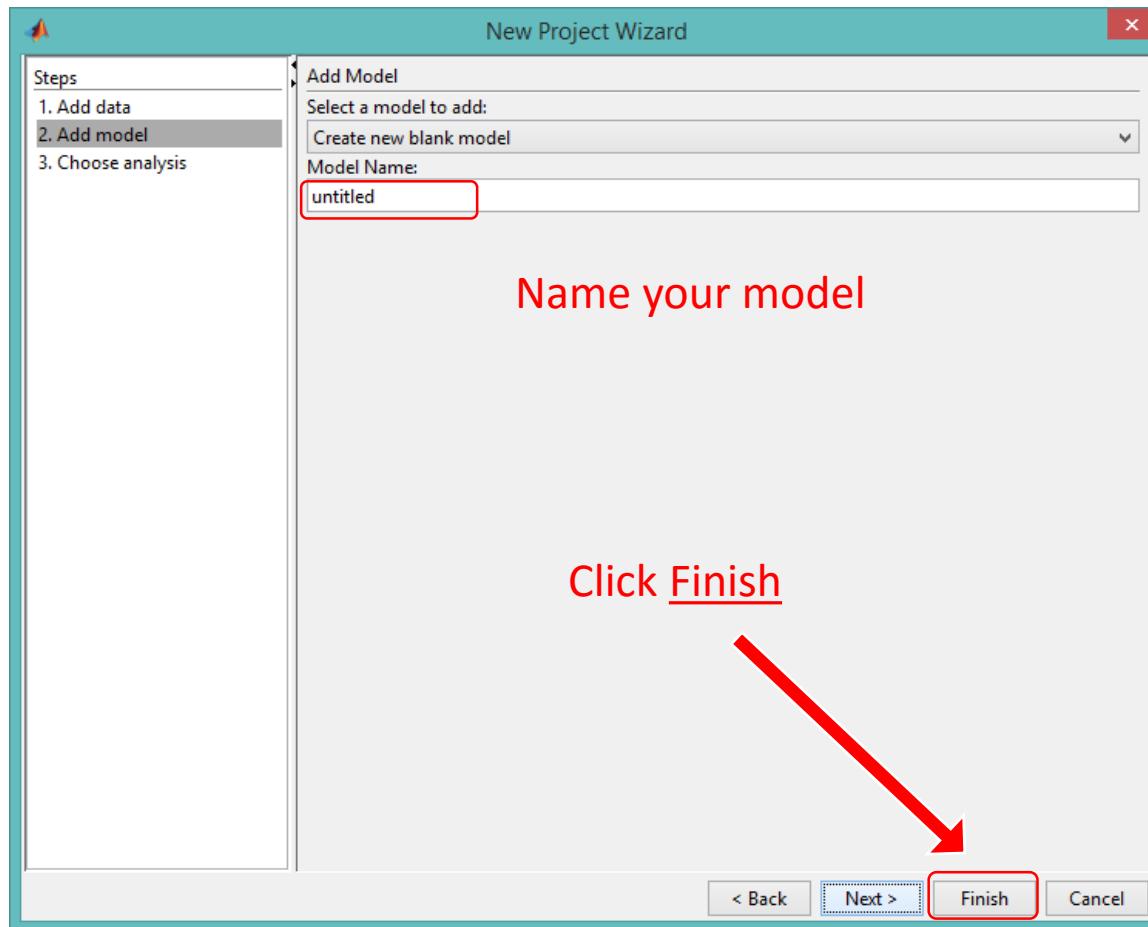
MATLAB Simbiology



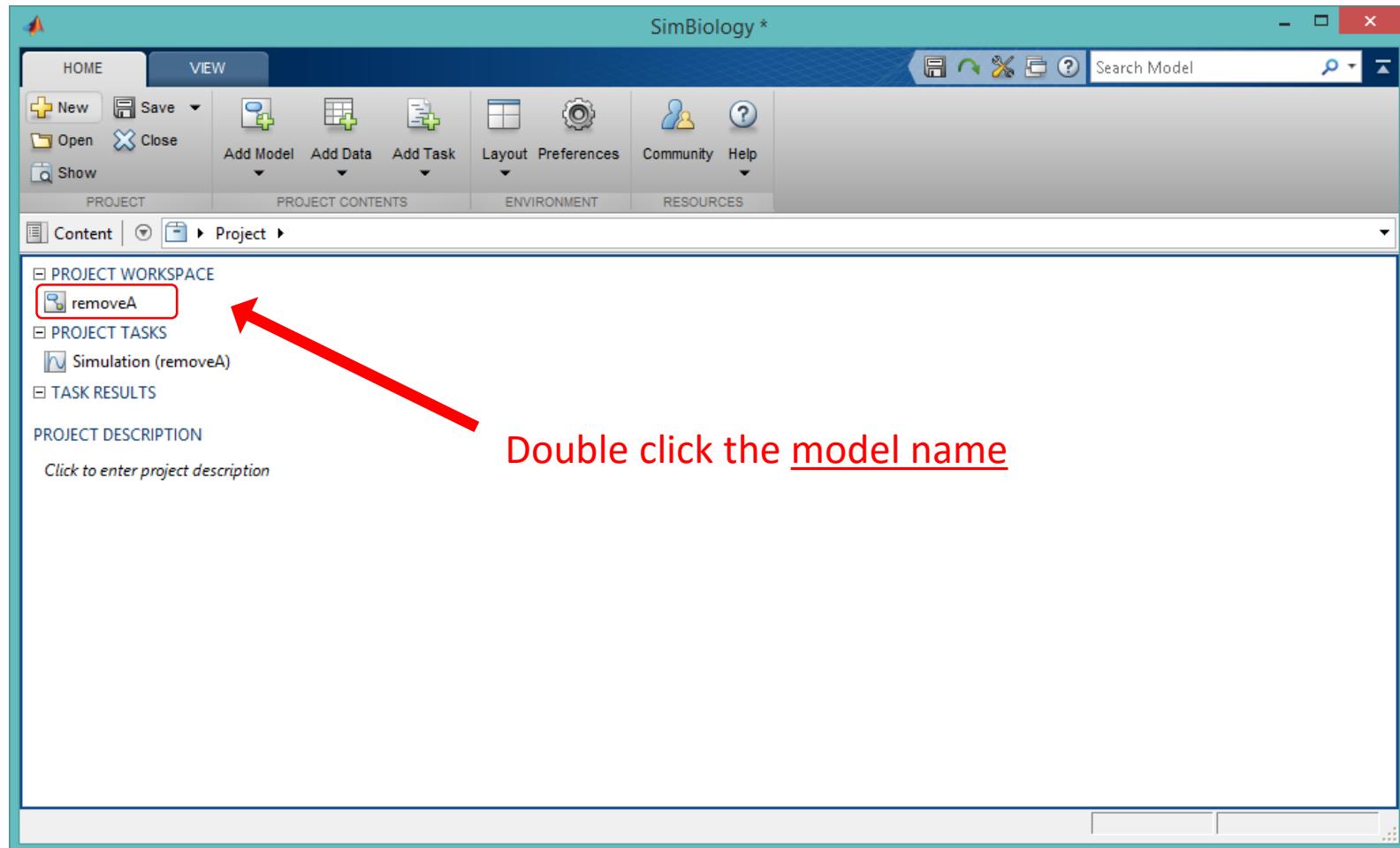
MATLAB Simbiology



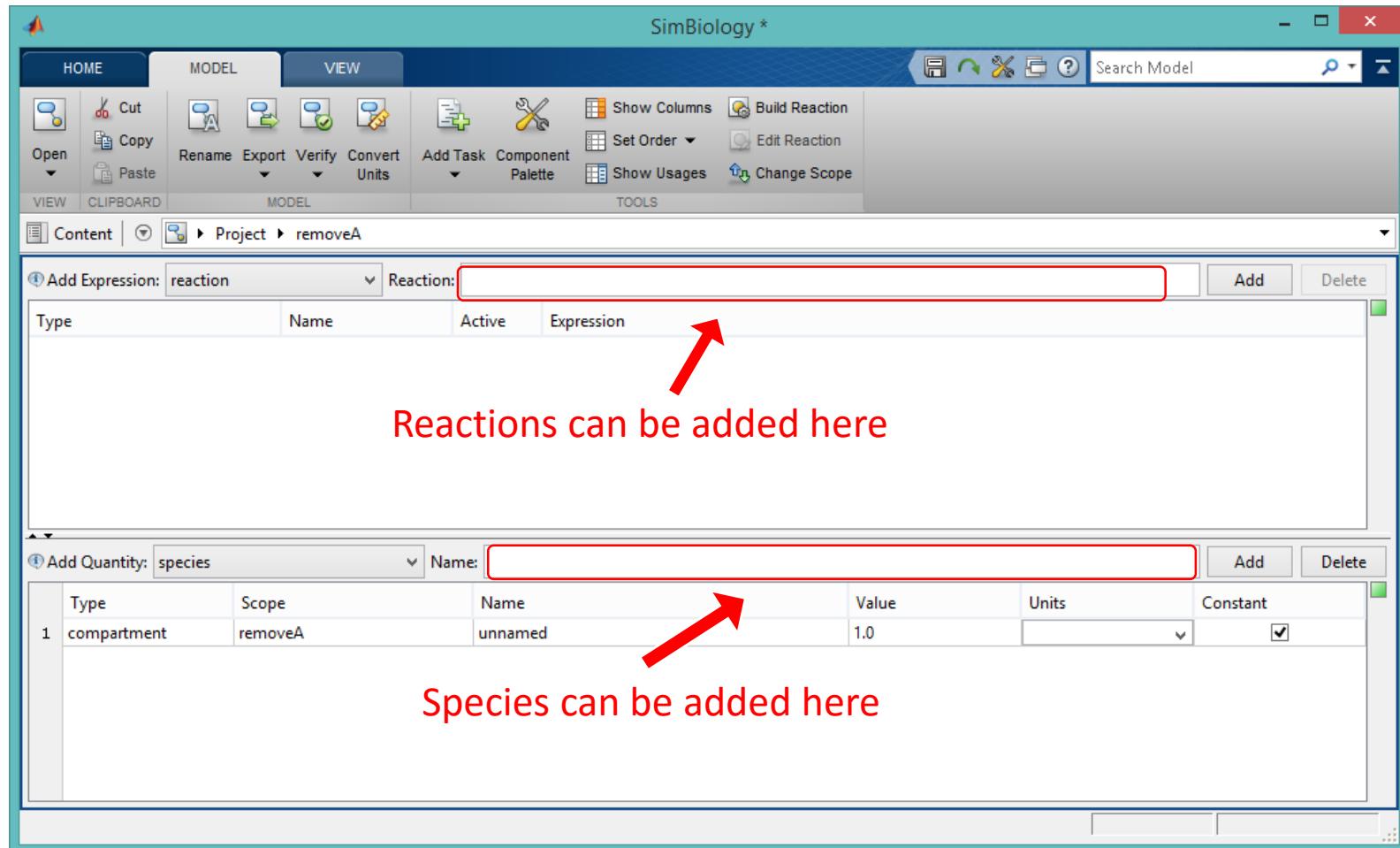
MATLAB Simbiology



MATLAB Simbiology



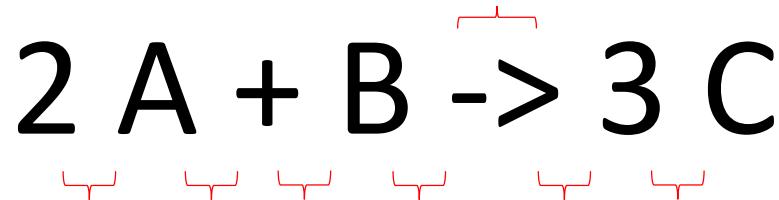
MATLAB Simbiology



MATLAB Simbiology

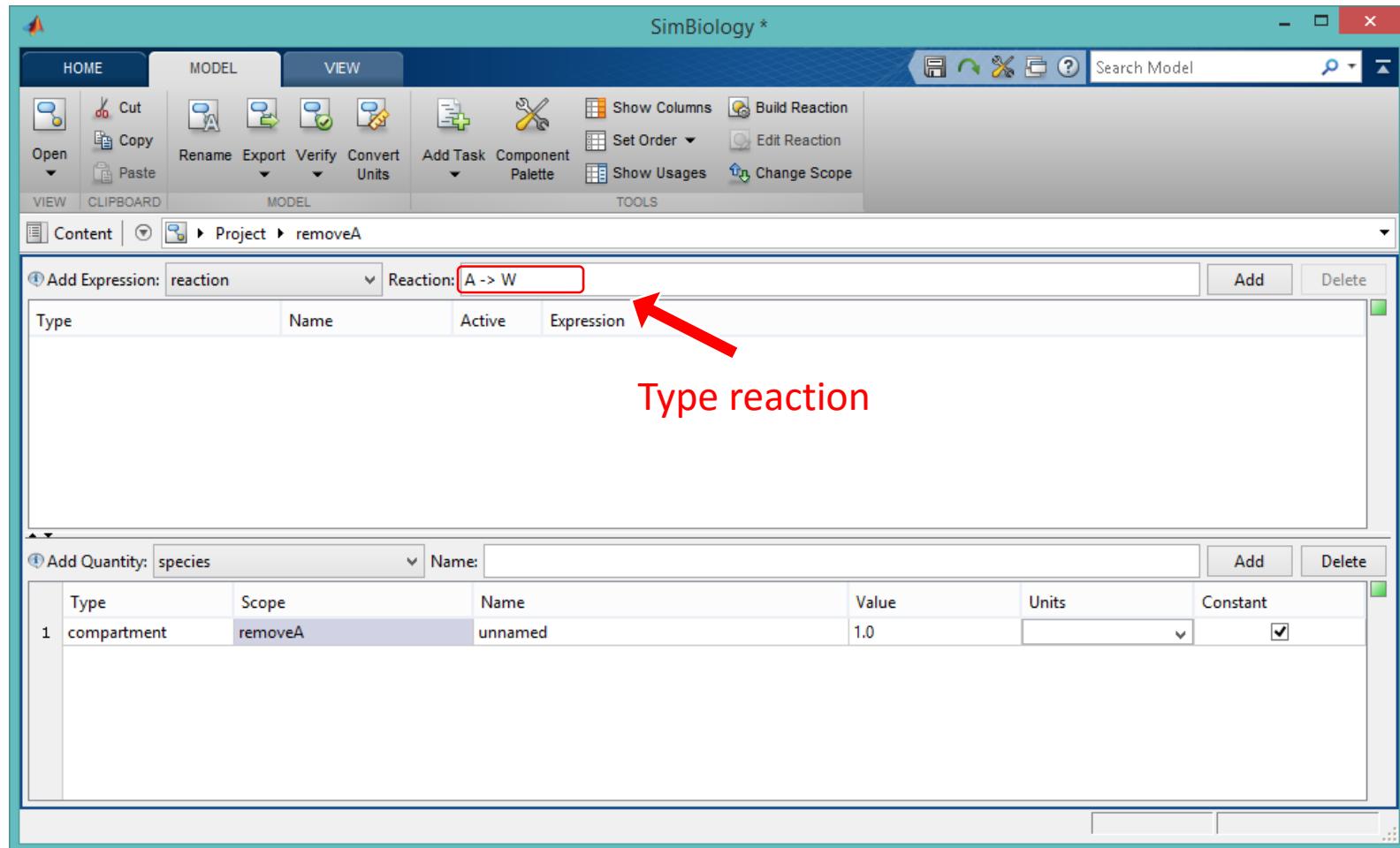
- Reactions in Simbiology have the following format

Note the arrow is two characters!



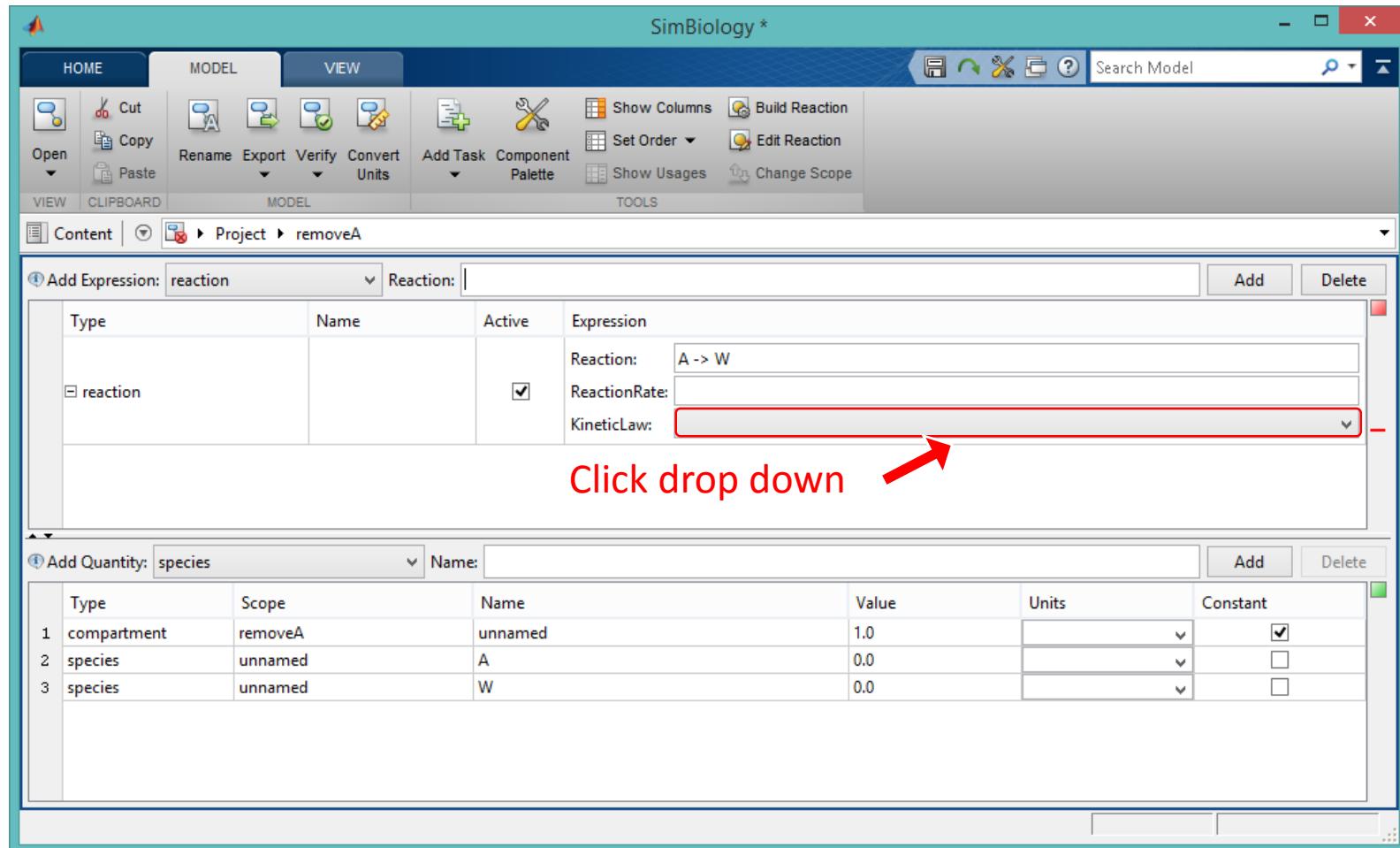
Note the spaces!

MATLAB Simbiology

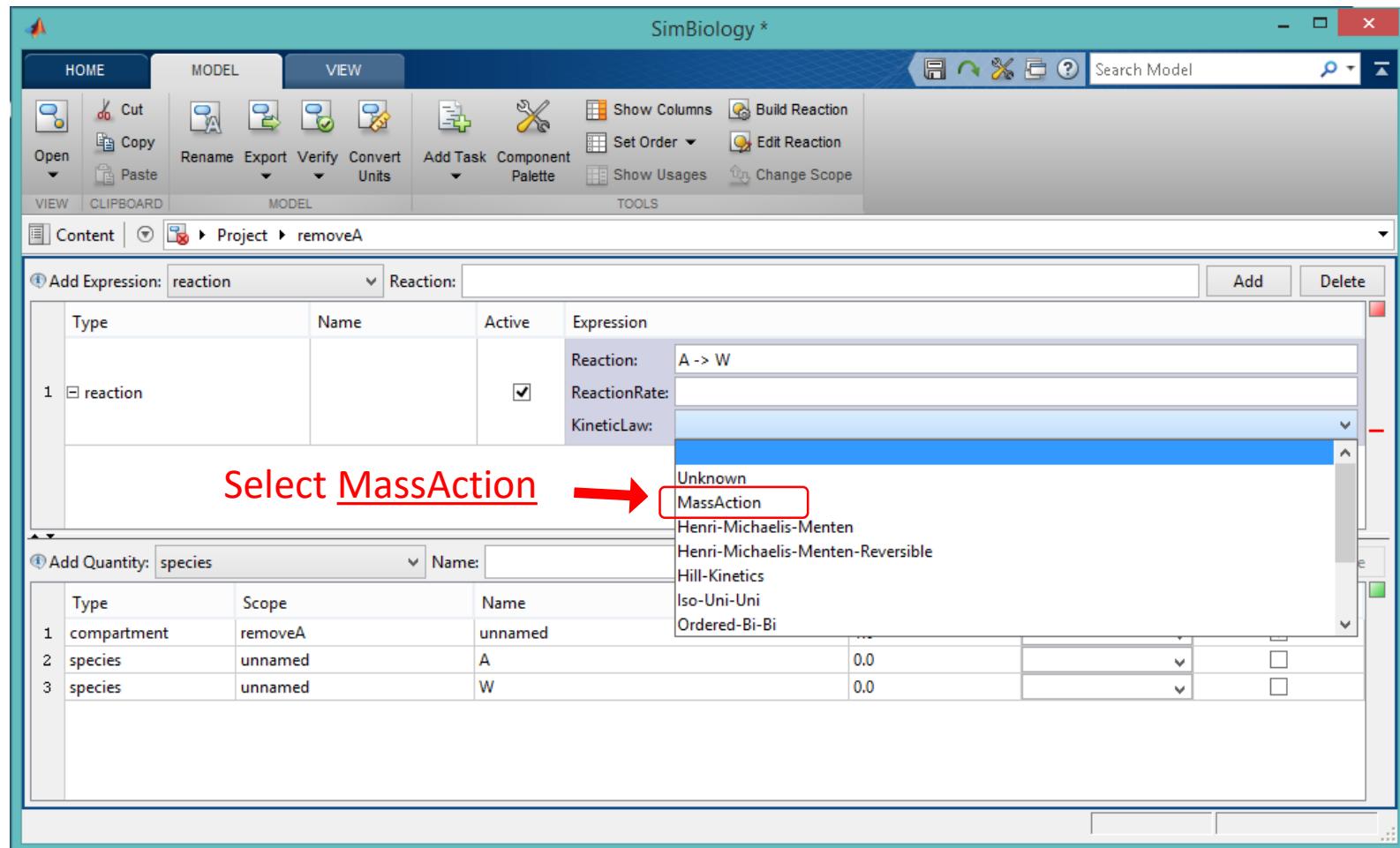


Type reaction

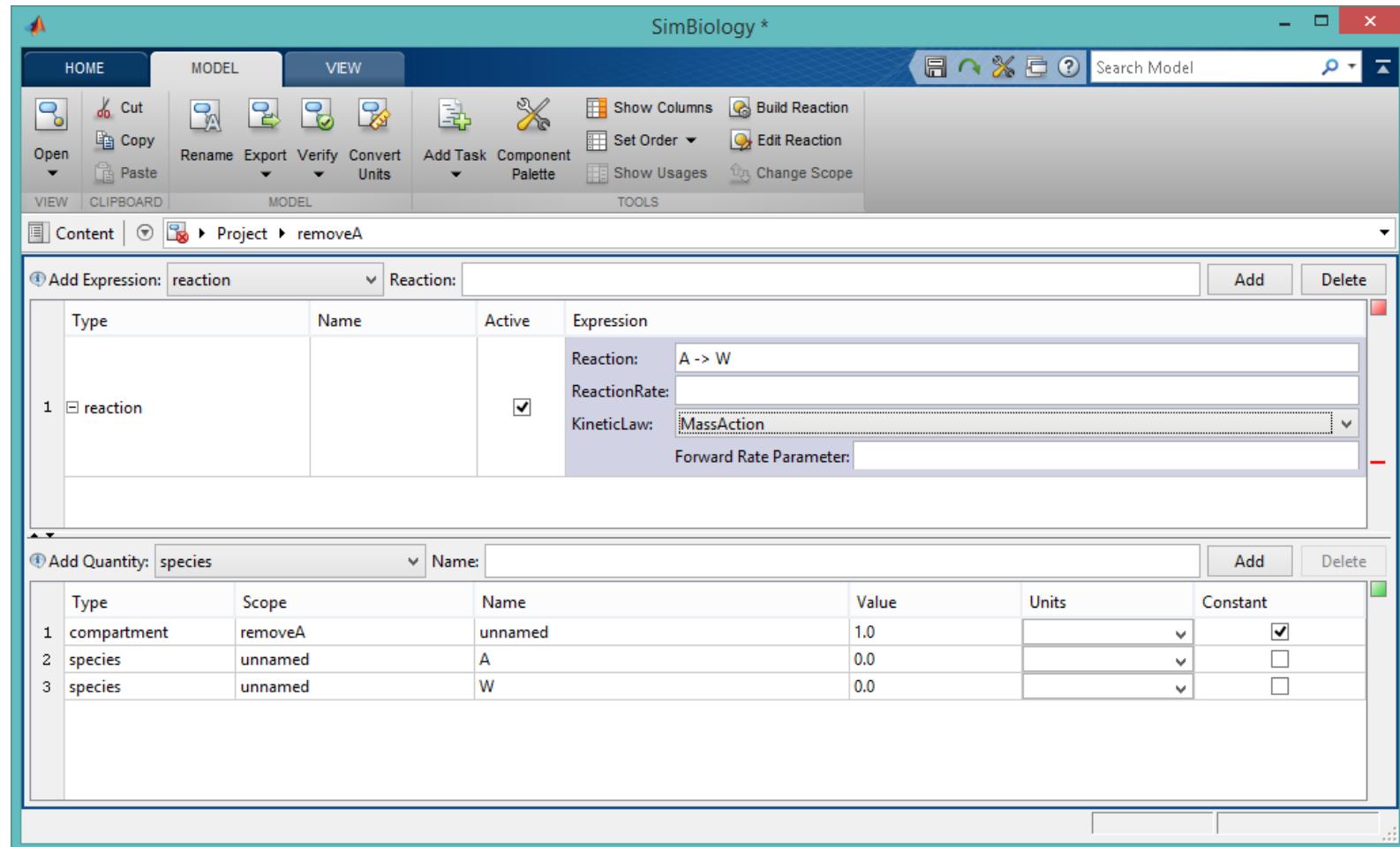
MATLAB Simbiology



MATLAB Simbiology



MATLAB Simbiology



MATLAB Simbiology

The screenshot shows the MATLAB SimBiology interface. The top menu bar includes HOME, MODEL (selected), and VIEW. The MODEL tab has buttons for Open, Cut, Copy, Paste, Rename, Export, Verify, Convert Units, Add Task, Component Palette, Show Columns, Set Order, Show Usages, Build Reaction, Edit Reaction, and Change Scope. Below the menu is a toolbar with icons for Open, Cut, Copy, Paste, Rename, Export, Verify, Convert Units, Add Task, Component Palette, Show Columns, Set Order, Show Usages, Build Reaction, Edit Reaction, and Change Scope. The main workspace shows two tables:

Reaction Table:

Type	Name	Active	Expression
1 reaction		<input checked="" type="checkbox"/>	 Reaction: A → W ReactionRate: k*A KineticLaw: MassAction Forward Rate Parameter: k

A red arrow points to the "Forward Rate Parameter: k" field.

Species Table:

Type	Scope	Name	Value	Units	Constant
1 compartment	removeA	unnamed	1.0		<input checked="" type="checkbox"/>
2 species	unnamed	A	0.0		<input type="checkbox"/>
3 species	unnamed	W	0.0		<input type="checkbox"/>

Text Overlay:

Type a name for a rate constant
and press Enter

MATLAB Simbiology

The screenshot shows the MATLAB SimBiology interface with the following details:

Reaction Table (Top):

Type	Name	Active	Expression
reaction		<input checked="" type="checkbox"/>	 Reaction: A → W ReactionRate: k*A KineticLaw: MassAction Forward Rate Parameter: k

Species Table (Bottom):

Type	Scope	Name	Value	Units	Constant
compartment	removeA	unnamed	1.0		<input checked="" type="checkbox"/>
species	unnamed	A	0.0		<input type="checkbox"/>
species	unnamed	W	0.0		<input type="checkbox"/>
parameter	A → W	k	1.0		<input checked="" type="checkbox"/>

Text Overlay: Change concentration of A to 10

A red arrow points to the "Value" column for the species "A" in the bottom table.

MATLAB Simbiology

The screenshot shows the MATLAB SimBiology interface. The top menu bar includes HOME, MODEL, and VIEW tabs. Under the MODEL tab, there are buttons for Rename, Export, Verify, Convert Units, Add Task, Component Palette, Show Columns, Build Reaction, Set Order, Edit Reaction, Show Usages, and Change Scope. The main workspace displays two tables: a 'Reaction' table and a 'Species' table.

Reaction Table:

	Type	Name	Active	Expression
1	reaction		<input checked="" type="checkbox"/>	 Reaction: A -> W ReactionRate: k*A KineticLaw: MassAction Forward Rate Parameter: k

Species Table:

	Type	Scope	Name	Value	Units	Constant
1	compartment	removeA	unnamed	1.0		<input checked="" type="checkbox"/>
2	species	removeA	A	10.0		<input type="checkbox"/>
3	species	removeA	W	0.0		<input type="checkbox"/>
4	parameter	A -> W	k	1.0		<input checked="" type="checkbox"/>

A red arrow points to the 'Constant' column for species 'W', which is currently unchecked. Red text above the table reads: "Change W to be a constant by checking the box".

MATLAB Simbiology

The screenshot shows the MATLAB SimBiology interface. At the top is a menu bar with HOME, MODEL, and VIEW tabs. Below the menu is a toolbar with various icons for file operations like Open, Cut, Copy, Paste, and model-specific functions like Rename, Export, Verify, Convert Units, Add Task, Component Palette, Show Columns, Set Order, Edit Reaction, Show Usages, and Change Scope. A search bar labeled "Search Model" is also present.

The main workspace displays two tables. The first table, titled "Content", lists a single reaction named "reaction". The second table, titled "species", lists four entries: compartment, species, species, and parameter. A red arrow points to the "Content" button in the toolbar, and a red box highlights the "Content" tab in the workspace.

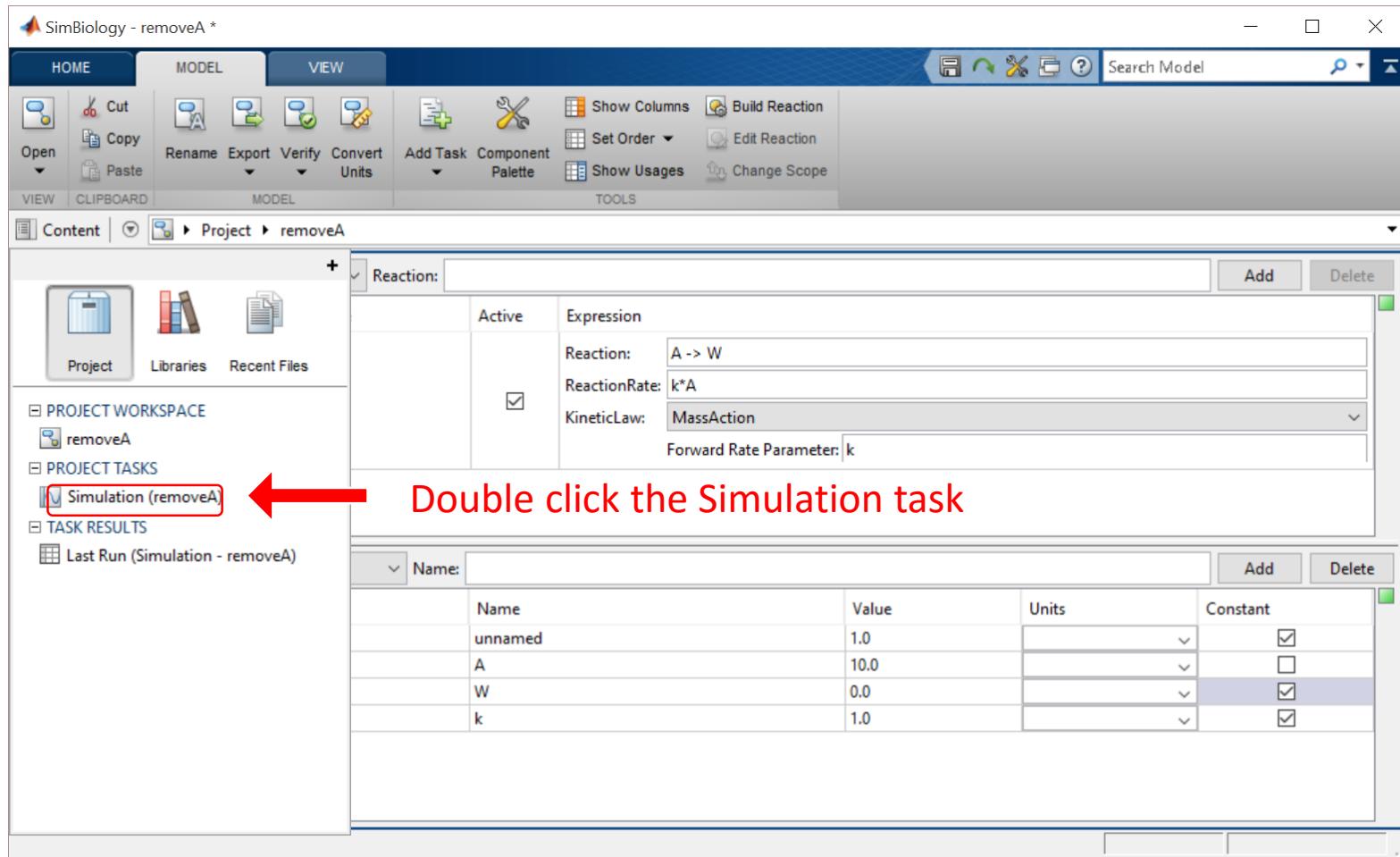
Reaction Table:

Type	Name	Active	Expression
1 reaction		<input checked="" type="checkbox"/>	 Reaction: A → W ReactionRate: k*A KineticLaw: MassAction Forward Rate Parameter: k

Species Table:

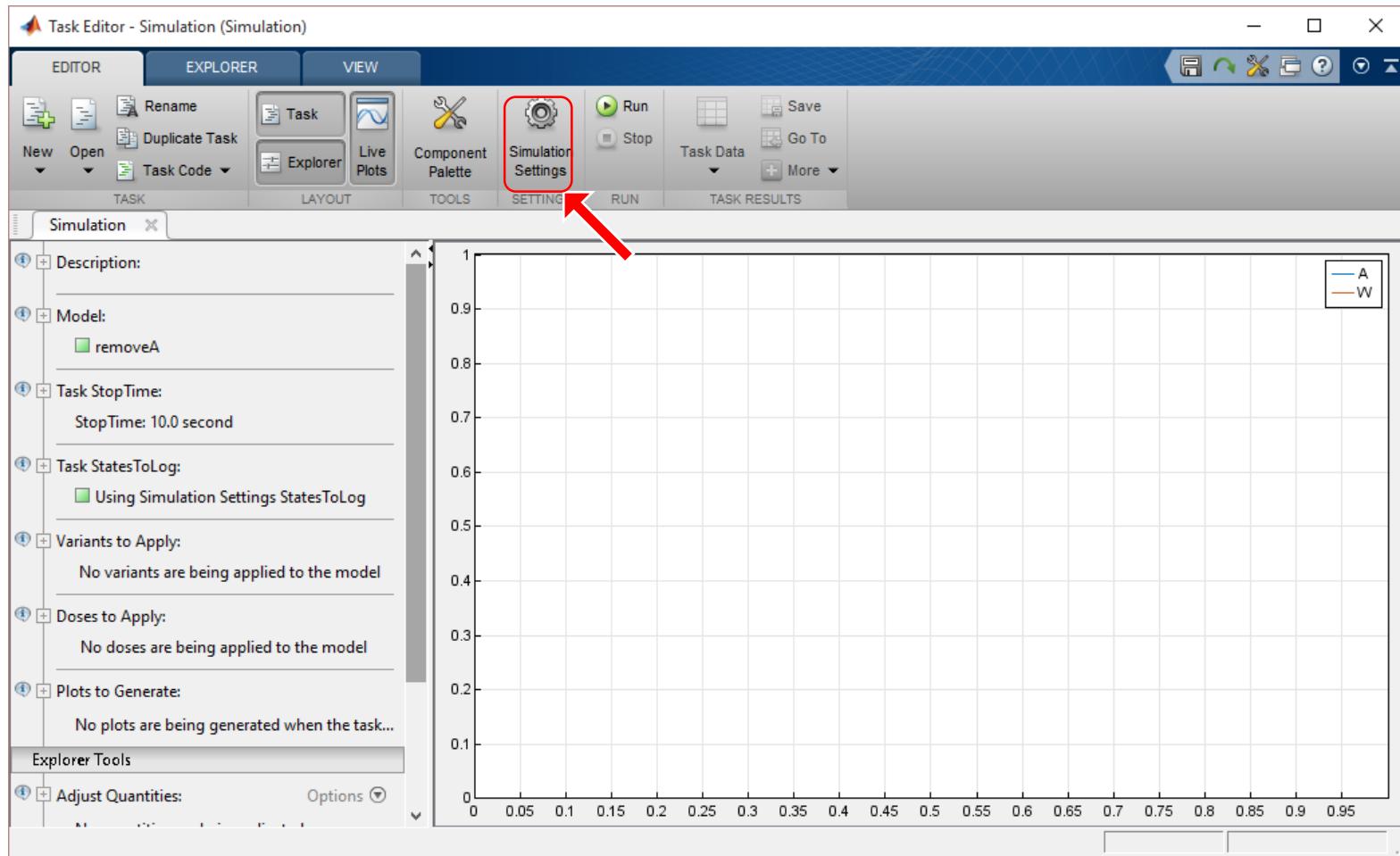
Type	Scope	Name	Value	Units	Constant
1 compartment	removeA	unnamed	1.0		<input checked="" type="checkbox"/>
2 species	unnamed	A	10.0		<input type="checkbox"/>
3 species	unnamed	W	0.0		<input checked="" type="checkbox"/>
4 parameter	A → W	k	1.0		<input checked="" type="checkbox"/>

MATLAB Simbiology

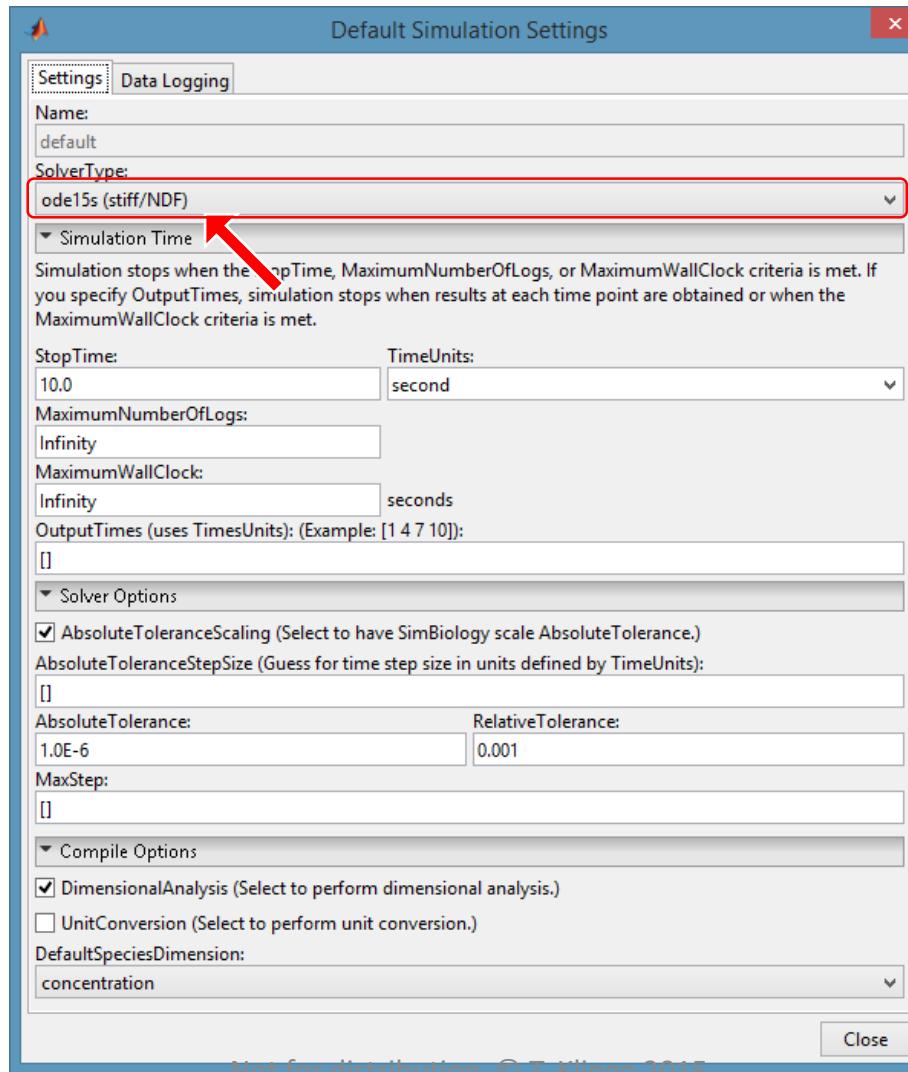


Double click the Simulation task

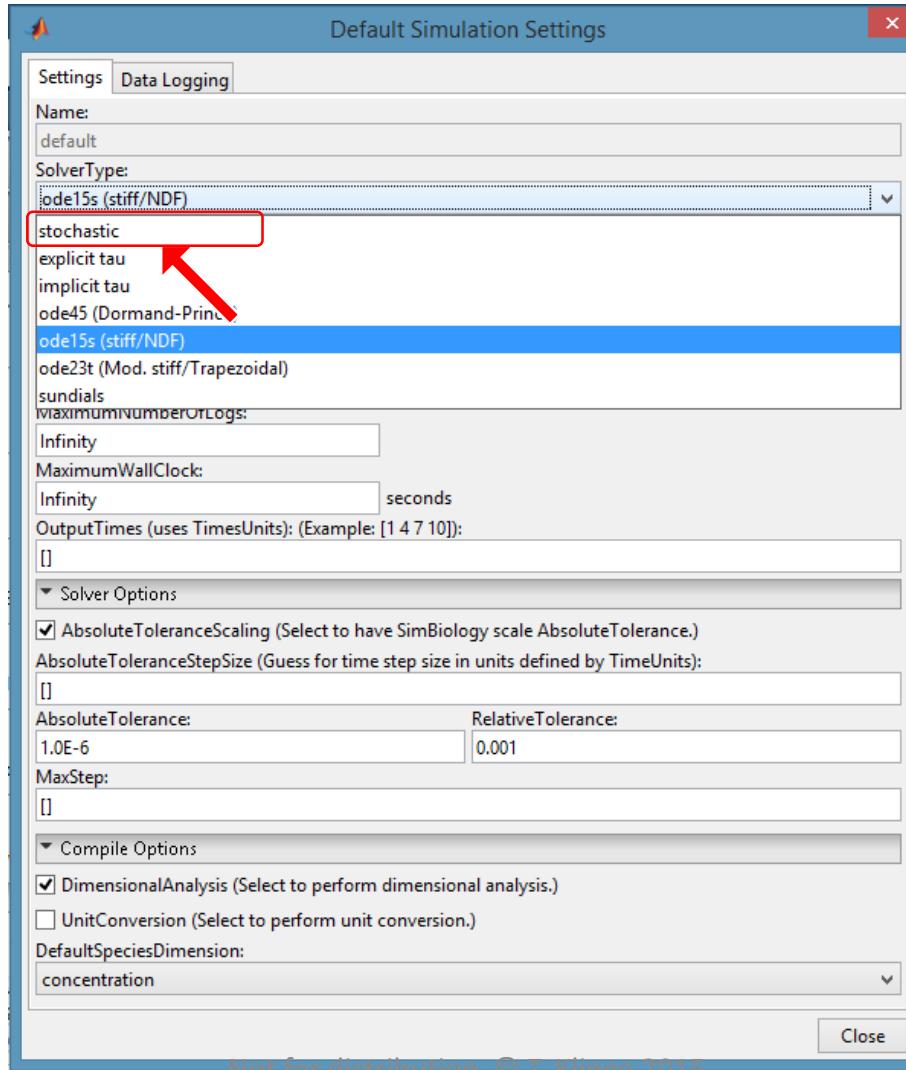
MATLAB Simbiology



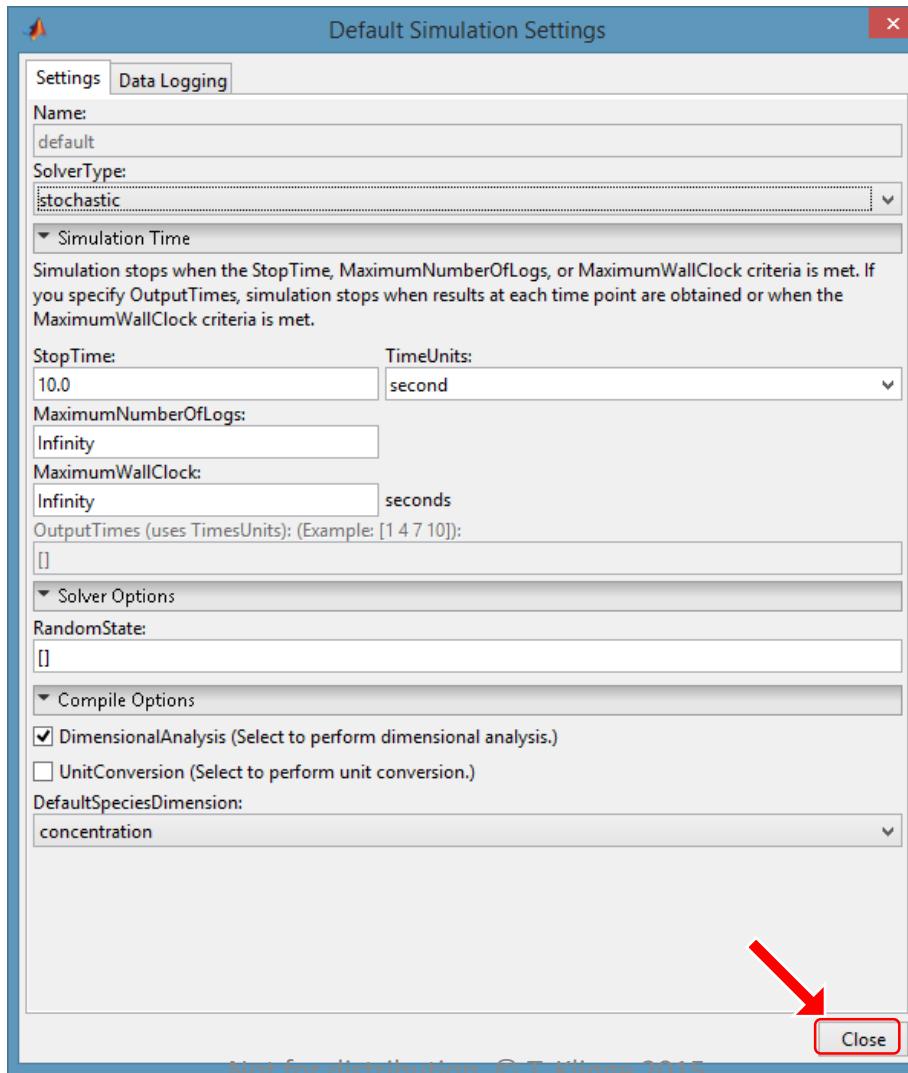
MATLAB Simbiology



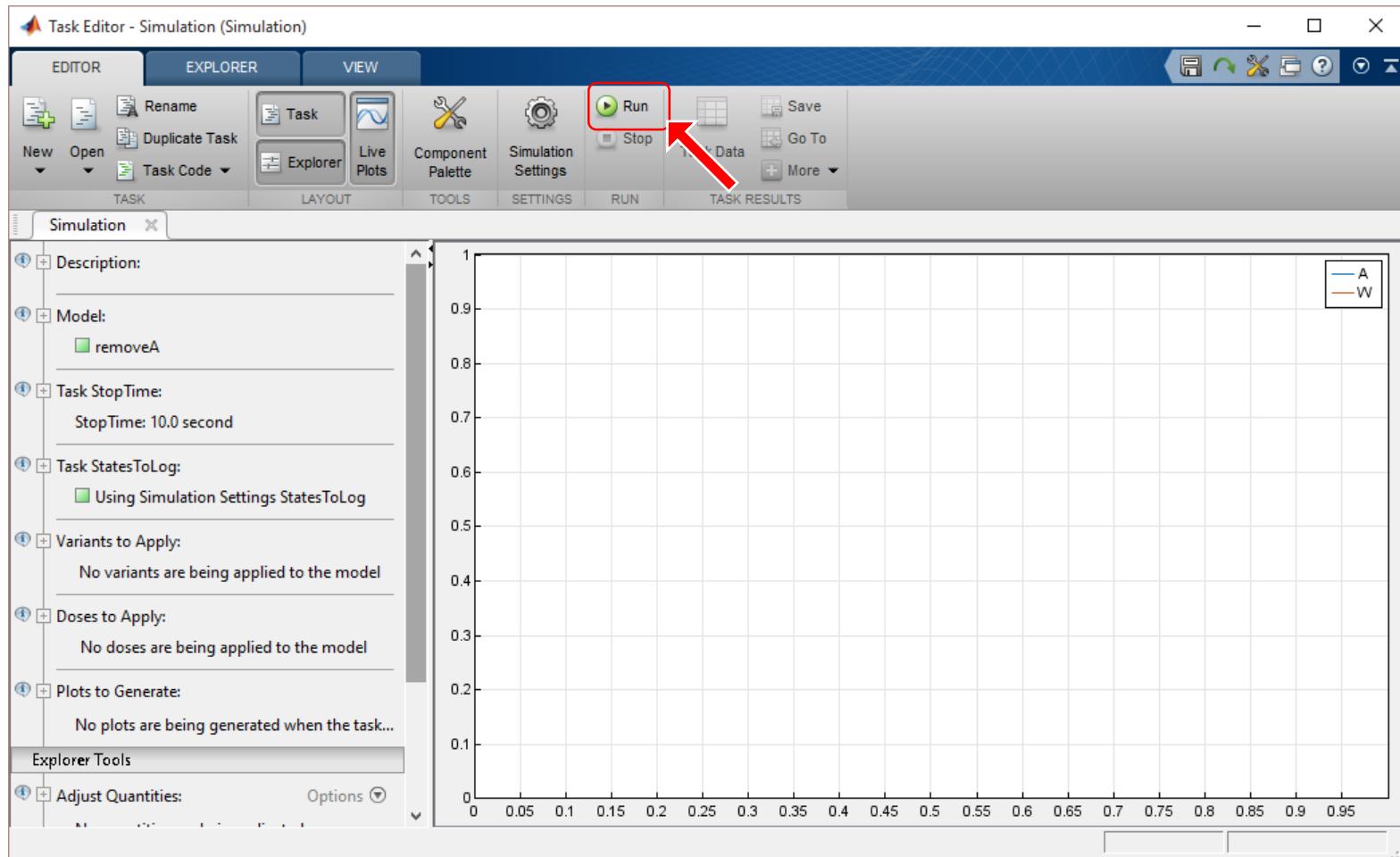
MATLAB Simbiology



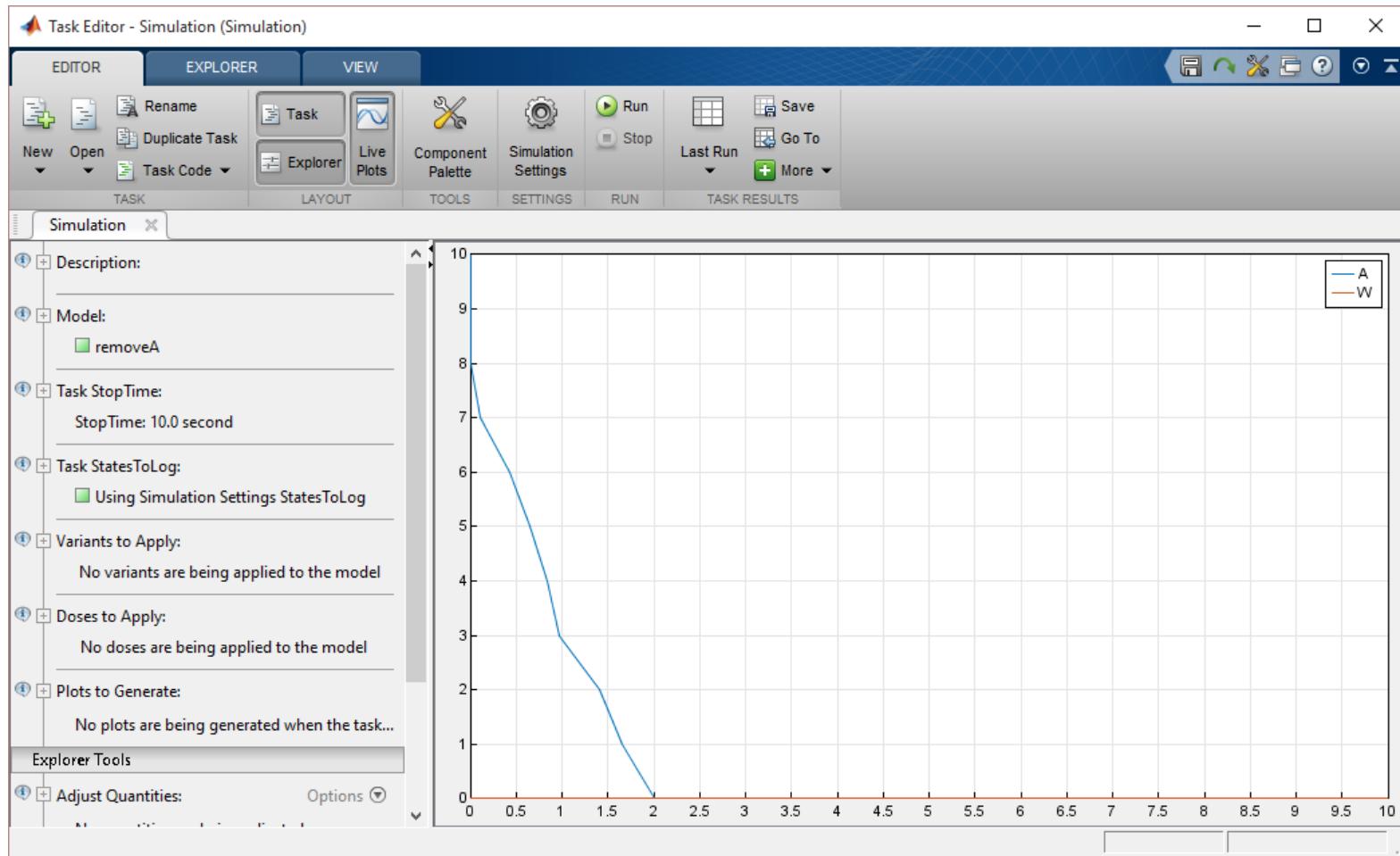
MATLAB Simbio



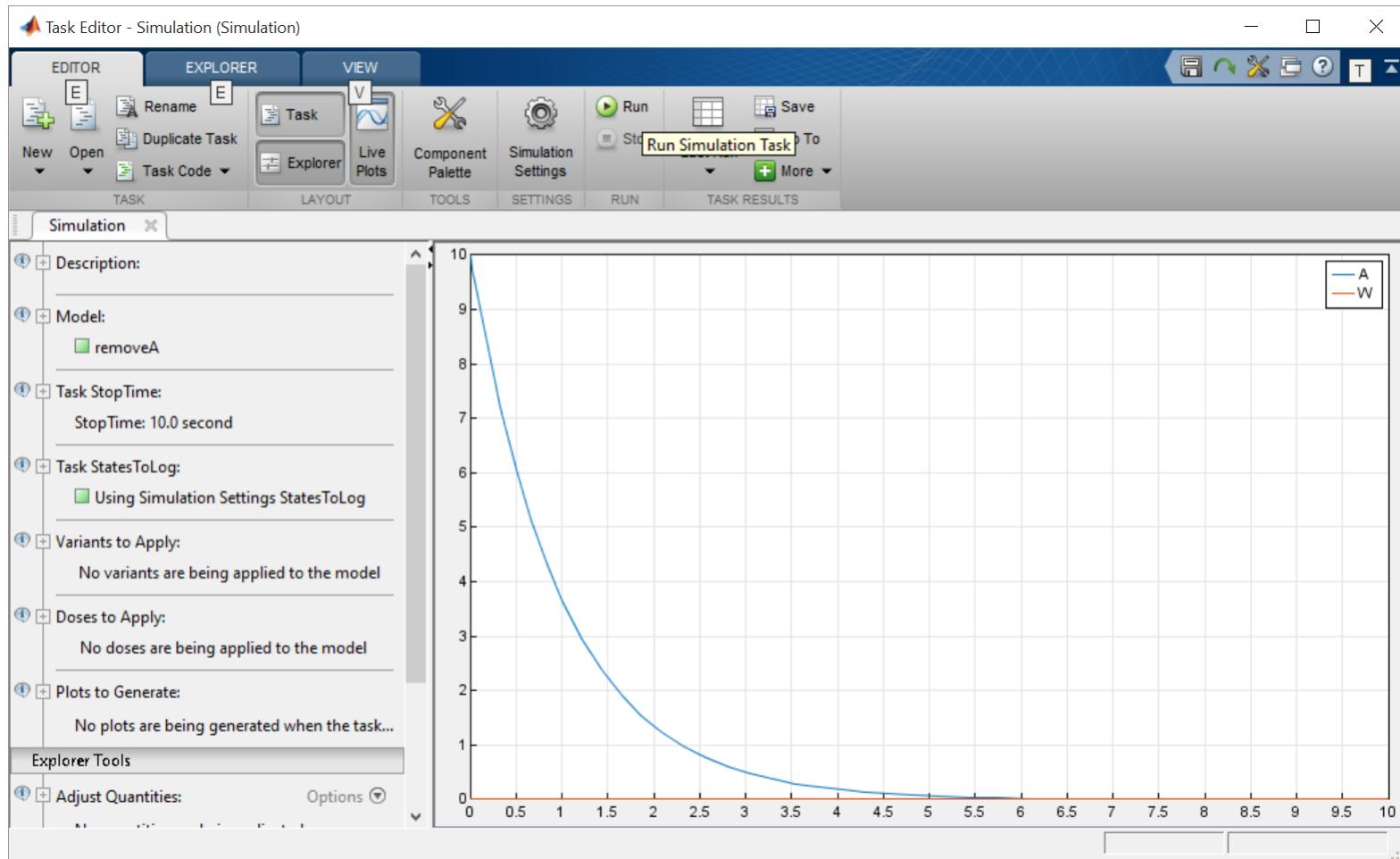
MATLAB Simbiology



MATLAB Simbiology



MATLAB Simbiology



CRN Exercise 1

- Input: n molecules of X
- Output: exactly $3n + 5$ molecules of Y

CRN Exercise 2

- Input: n molecules of X
- Output: Two output species: Y and Z
 - If n is even output one Y and zero Z
 - If n is odd output one Z and zero Y

CRN Exercise 3

- Suppose you have four species: X, S_1, S_2, S_3
- Initial state: $X = n, S_1 = 1, S_2 = S_3 = 0$
With $n \in \{0,1\}$
- Output: **NOT POSSIBLE WITH PROBABILITY 1**
 - If $n = 0$, output $S_1 = S_2 = 0$ and $S_3 = 1$
 - If $n = 1$, output $S_1 = S_3 = 0$ and $S_2 = 1$

CRN Exercise 3

- Suppose you have four species: X, S_1, S_2, S_3
- Initial state: $X = n, S_1 = 1, S_2 = S_3 = 0$
With $n \in \{0,1\}$
- Output: Fix $\epsilon > 0$
 - If $n = 0$, output $S_1 = S_2 = 0, S_3 = 1$
 - If $n = 1$, output $S_1 = S_3 = 0, S_2 = 1$ with $p \geq 1 - \epsilon$

CRN Exercise 4

- Input: n molecules of X
- Output: $2n$ molecules of X

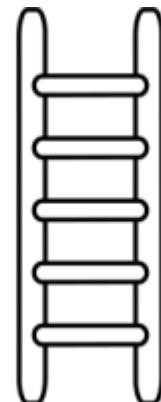
Also not possible with probability 1

CRN Exercise 4

- Input: n molecules of X
- Output: $2n$ molecules of X with $p \geq 1 - \epsilon$
 - Want sequential behavior like:
 1. $X + Y \rightarrow Xp + Y$
 2. $Y \rightarrow Z$ // after all Xs are converted
 3. $Xp + Z \rightarrow 2X + Z$

CRN Exercise 4

- Noticing when a species hits 0 (or just has low molecule counts / concentration) is hard
- In our watchdog timer we used what we call a ladder



Afternoon Break

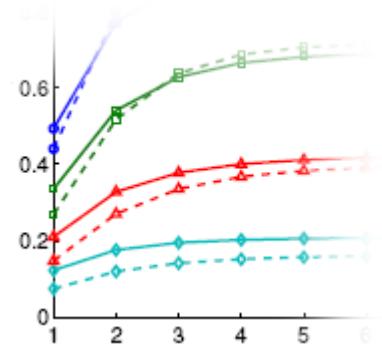
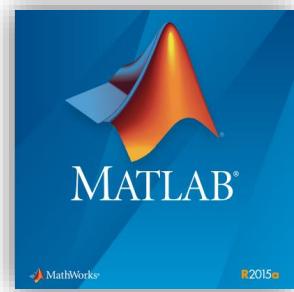
Software Tools

Molecular Programming tutorial
ASE 205

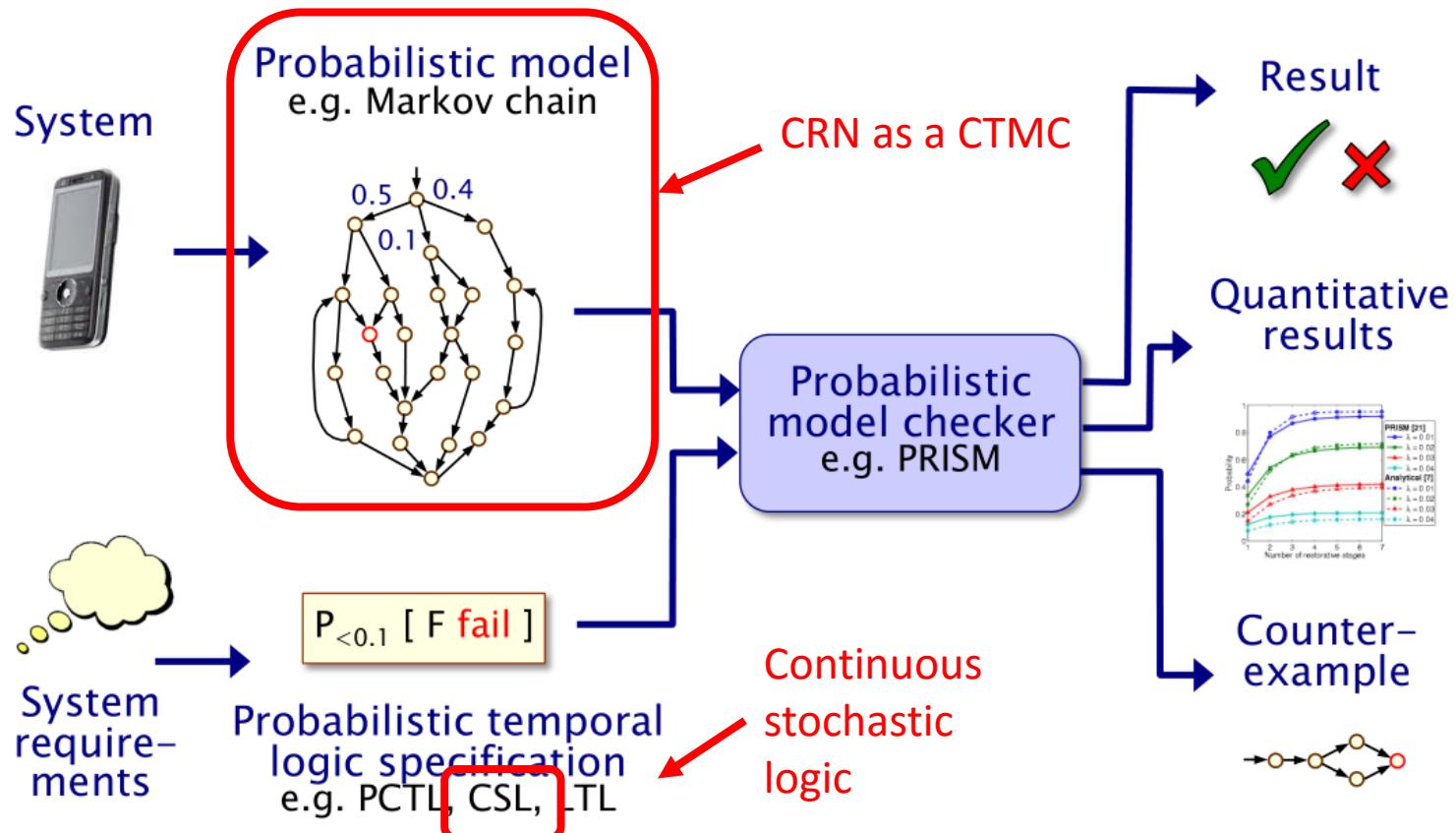
Jack Lutz, Robyn Lutz, James Lathrop, Titus Klinge

Outline

- MATLAB (Simbiology)
 - Tutorial – designing and simulating CRNs
 - Several CRN examples
- Afternoon break
- **Probabilistic Model Checking**
 - Overview of PMC
 - PRISM tutorial
 - SMART tutorial
 - (MATLAB integration with tools)



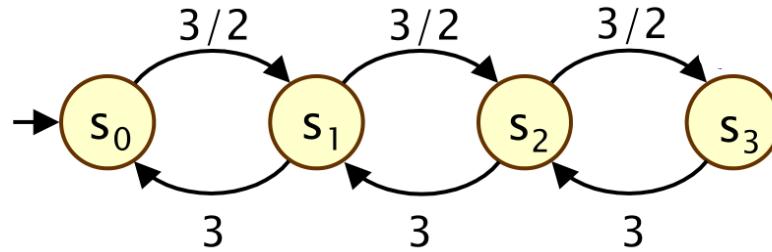
Probabilistic Model Checking



Source: Dave Parker lecture on PMC from PRISM website
<http://www.prismmodelchecker.org/lectures/pmc/>

Probabilistic Model Checking

- Continuous time Markov chains



<http://www.prismmodelchecker.org/lectures/pmc/>

- States encode the number of molecules present of each species
- Transitions are triggered by reactions firing and changing the number of molecules

Probabilistic Model Checking

- Continuous stochastic logic (CSL)
 - A temporal logic for specifying requirements over CTMCs

- State formula

$$\phi = \text{true} \mid a \mid \phi \vee \phi \mid \neg\phi \mid S[\phi] \mid P[\psi]$$

Steady state

Probability of a path
formula being satisfied

- Path formula

$$\psi = X \phi \mid F \phi \mid G \phi \mid \phi U \phi$$

Probabilistic Model Checking

- Path formula

$$\psi = X \phi \mid F \phi \mid G \phi \mid \phi U \phi$$

- $X \phi$

- ϕ is true in the next state of the path

- $F \phi$ (also written $\diamond \phi$)

- ϕ is true in some future state of the path

- $G \phi$ (also written $\square \phi$)

- ϕ is true in all future states of the path

- $\phi_1 U \phi_2$ - ϕ_1 is true until ϕ_2 becomes true

Probabilistic Model Checking

- Example

ACHIEVE: AlarmTripped if threshold for some time

$$\mathcal{P}_{\geq 1} \square [Th_H \implies \mathcal{P}_{\geq 1 - \eta_4} \diamondsuit_{\leq w_{th}} (A_{trip} \vee \neg Th_H)]$$

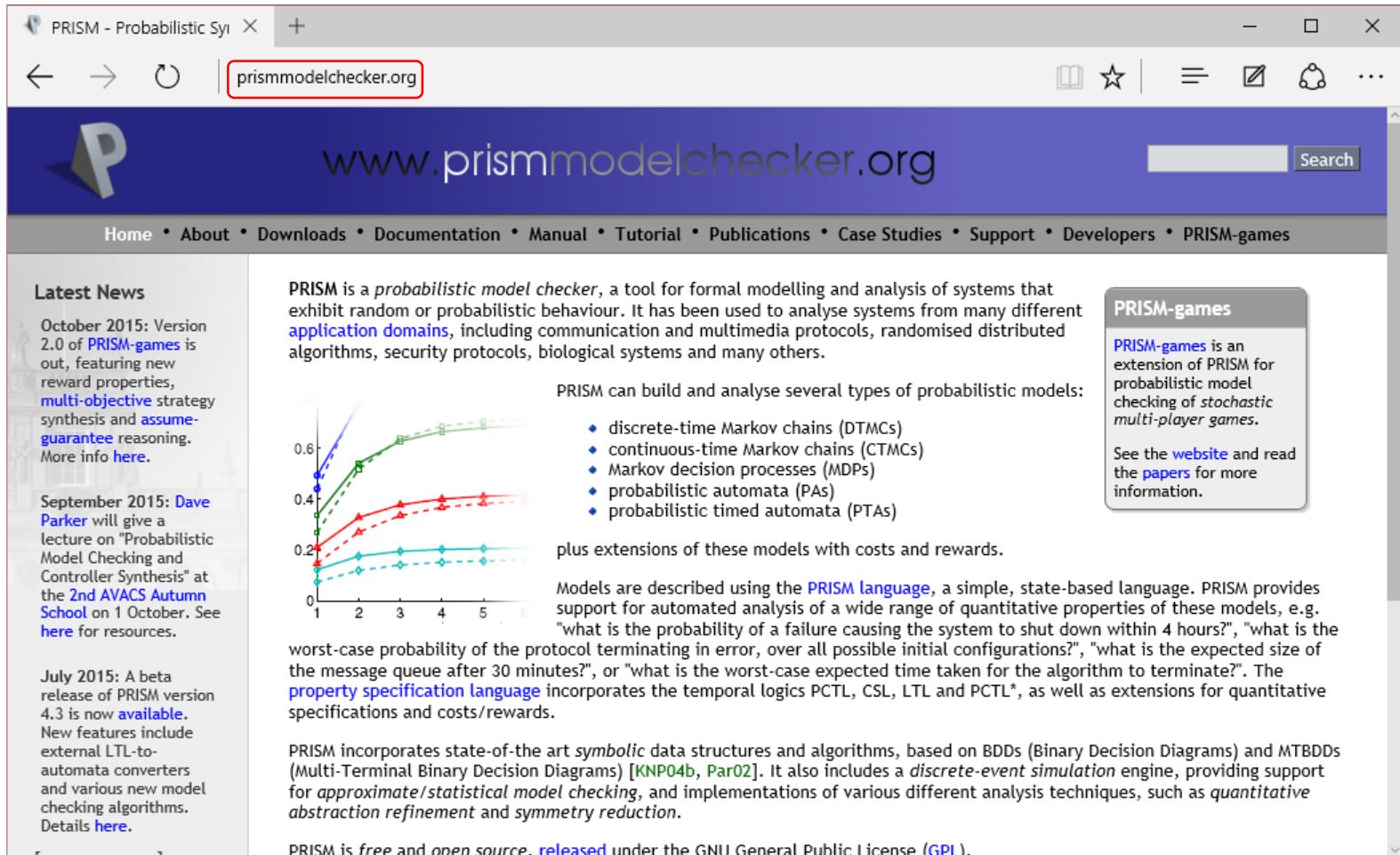
All meaningful paths
must always satisfy
the following...

If the heartbeat
threshold is
sufficiently high,
then...

With “high” probability,
within in a “short”
amount of time...

We either “trip” the
alarm OR drop below
sufficient threshold

PRISM: Model Checker



The screenshot shows a web browser window displaying the official PRISM Model Checker website at [www.prismmodelchecker.org](http://prismmodelchecker.org). The page features a large blue header with the PRISM logo and navigation links. Below the header, there's a "Latest News" section with three entries, a main content area with a graph, and a sidebar for "PRISM-games".

Latest News

- October 2015: Version 2.0 of [PRISM-games](#) is out, featuring new reward properties, multi-objective strategy synthesis and [assume-guarantee](#) reasoning. More info [here](#).
- September 2015: [Dave Parker](#) will give a lecture on "Probabilistic Model Checking and Controller Synthesis" at the [2nd AVACS Autumn School](#) on 1 October. See [here](#) for resources.
- July 2015: A beta release of PRISM version 4.3 is now [available](#). New features include external LTL-to-automata converters and various new model checking algorithms. Details [here](#).

PRISM-games

[PRISM-games](#) is an extension of PRISM for probabilistic model checking of stochastic multi-player games. See the [website](#) and read the [papers](#) for more information.

PRISM is a *probabilistic model checker*, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour. It has been used to analyse systems from many different [application domains](#), including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems and many others.

PRISM can build and analyse several types of probabilistic models:

- discrete-time Markov chains (DTMCs)
- continuous-time Markov chains (CTMCs)
- Markov decision processes (MDPs)
- probabilistic automata (PAs)
- probabilistic timed automata (PTAs)

plus extensions of these models with costs and rewards.

Models are described using the [PRISM language](#), a simple, state-based language. PRISM provides support for automated analysis of a wide range of quantitative properties of these models, e.g. "what is the probability of a failure causing the system to shut down within 4 hours?", "what is the worst-case probability of the protocol terminating in error, over all possible initial configurations?", "what is the expected size of the message queue after 30 minutes?", or "what is the worst-case expected time taken for the algorithm to terminate?". The [property specification language](#) incorporates the temporal logics PCTL, CSL, LTL and PCTL*, as well as extensions for quantitative specifications and costs/rewards.

PRISM incorporates state-of-the art *symbolic* data structures and algorithms, based on BDDs (Binary Decision Diagrams) and MTBDDs (Multi-Terminal Binary Decision Diagrams) [KNP04b, Par02]. It also includes a *discrete-event simulation* engine, providing support for *approximate/statistical model checking*, and implementations of various different analysis techniques, such as *quantitative abstraction refinement* and *symmetry reduction*.

PRISM is [free](#) and [open source](#), [released](#) under the [GNU General Public License \(GPL\)](#).

PRISM: Model Checker

The screenshot shows the SimBiology software interface for a project named "nextstate". A red arrow points to the "Project" dropdown in the top navigation bar.

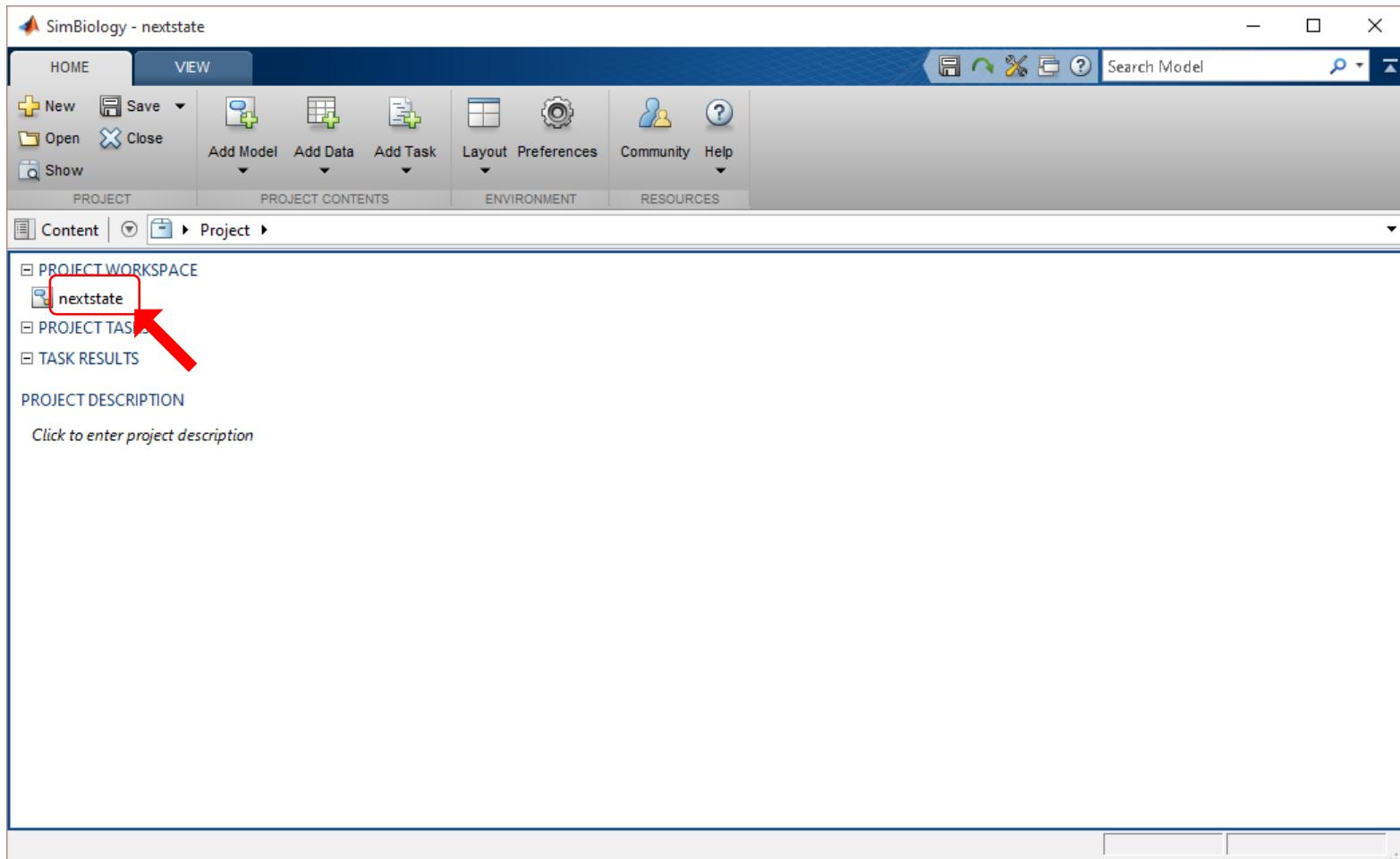
Reactions Table:

Type	Name	Active	Expression
reaction		<input checked="" type="checkbox"/>	$X + S1 \rightarrow X + S2, k1*X*S1$
reaction		<input checked="" type="checkbox"/>	$Y + S1 \rightarrow Y + S3, k2*Y*S1$

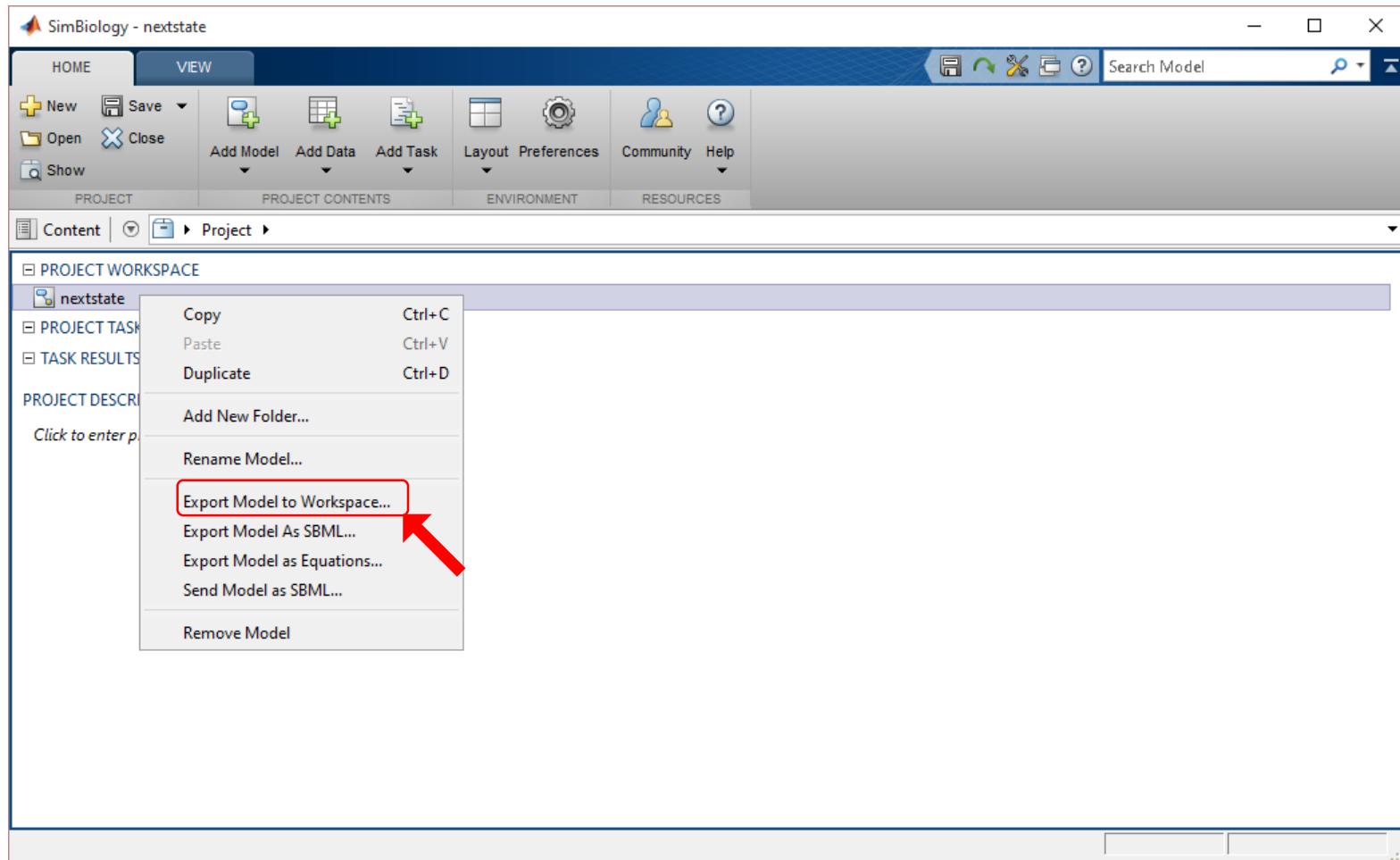
Species Table:

Type	Scope	Name	Value	Units	Constant
compartment	nextstate	unnamed	1.0		<input checked="" type="checkbox"/>
species	unnamed	X	1.0		<input type="checkbox"/>
species	unnamed	S1	1.0		<input type="checkbox"/>
species	unnamed	S2	0.0		<input type="checkbox"/>
species	unnamed	Y	1.0		<input type="checkbox"/>
species	unnamed	S3	0.0		<input type="checkbox"/>
parameter	$X + S1 \rightarrow X + S2$	k1	0.75		<input checked="" type="checkbox"/>

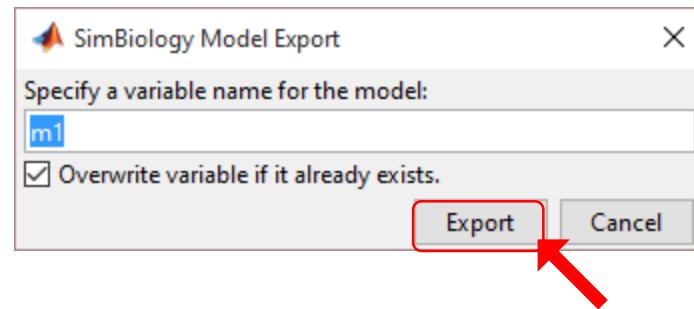
PRISM: Model Checker



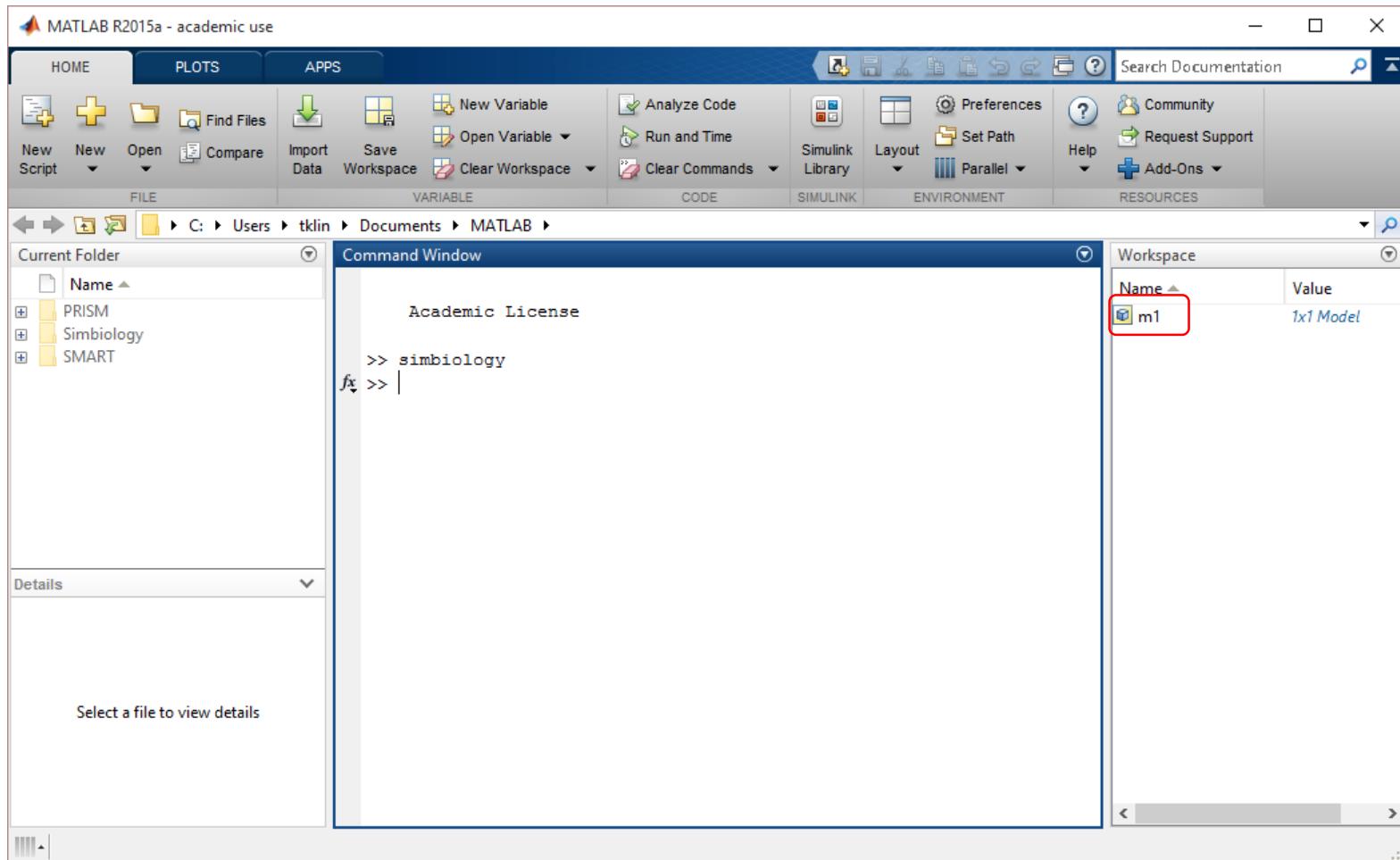
PRISM: Model Checker



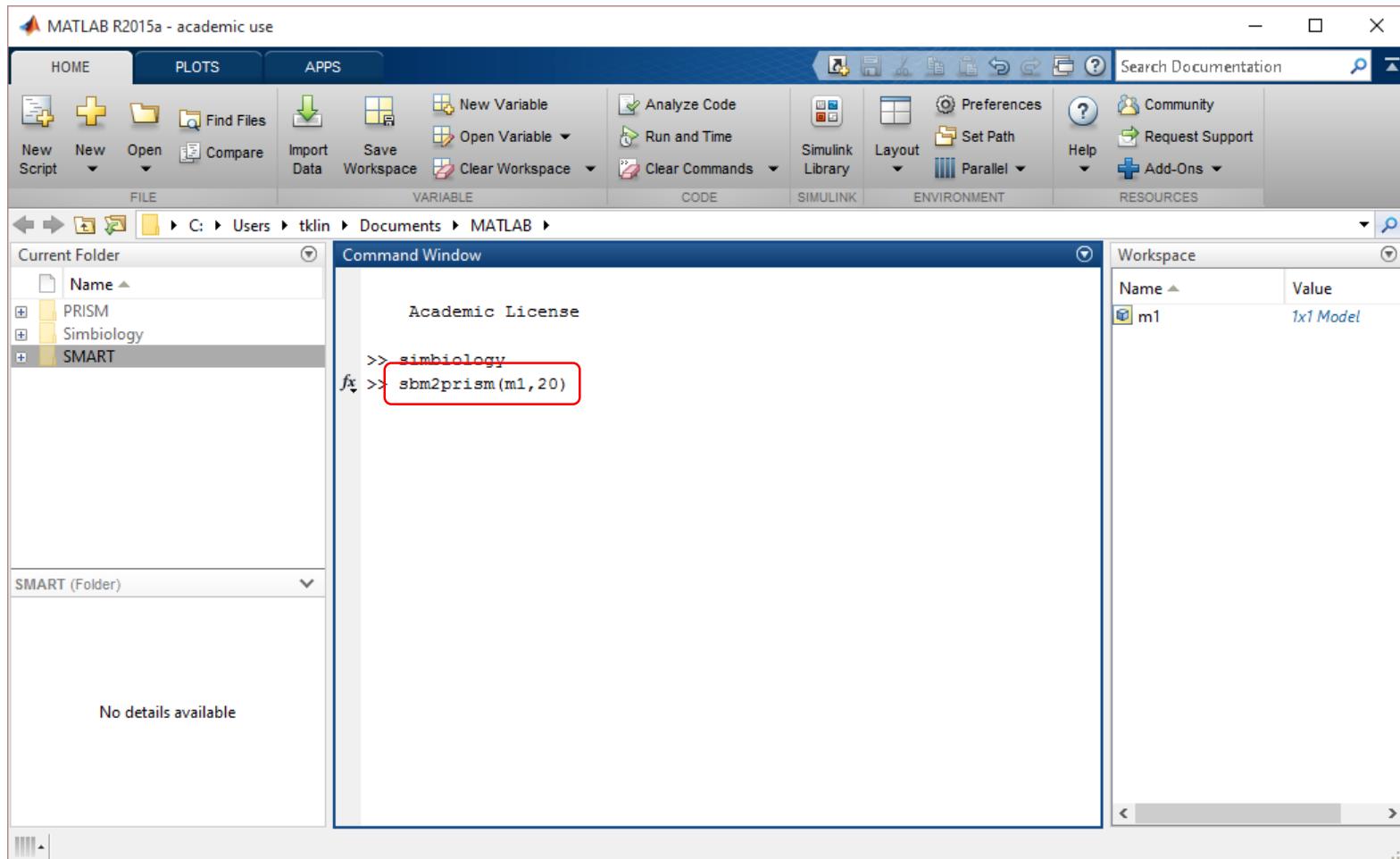
PRISM: Model Checker



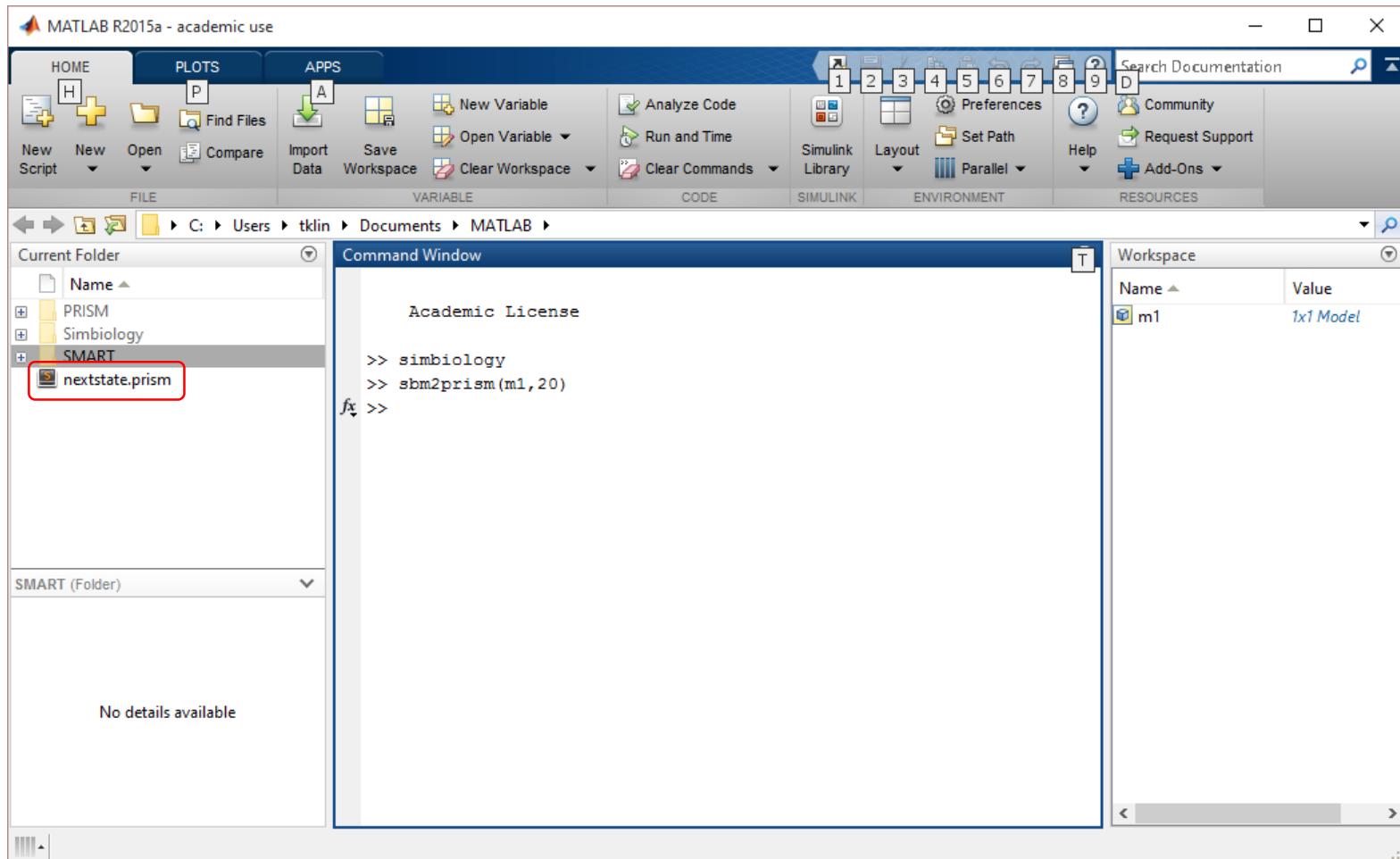
PRISM: Model Checker



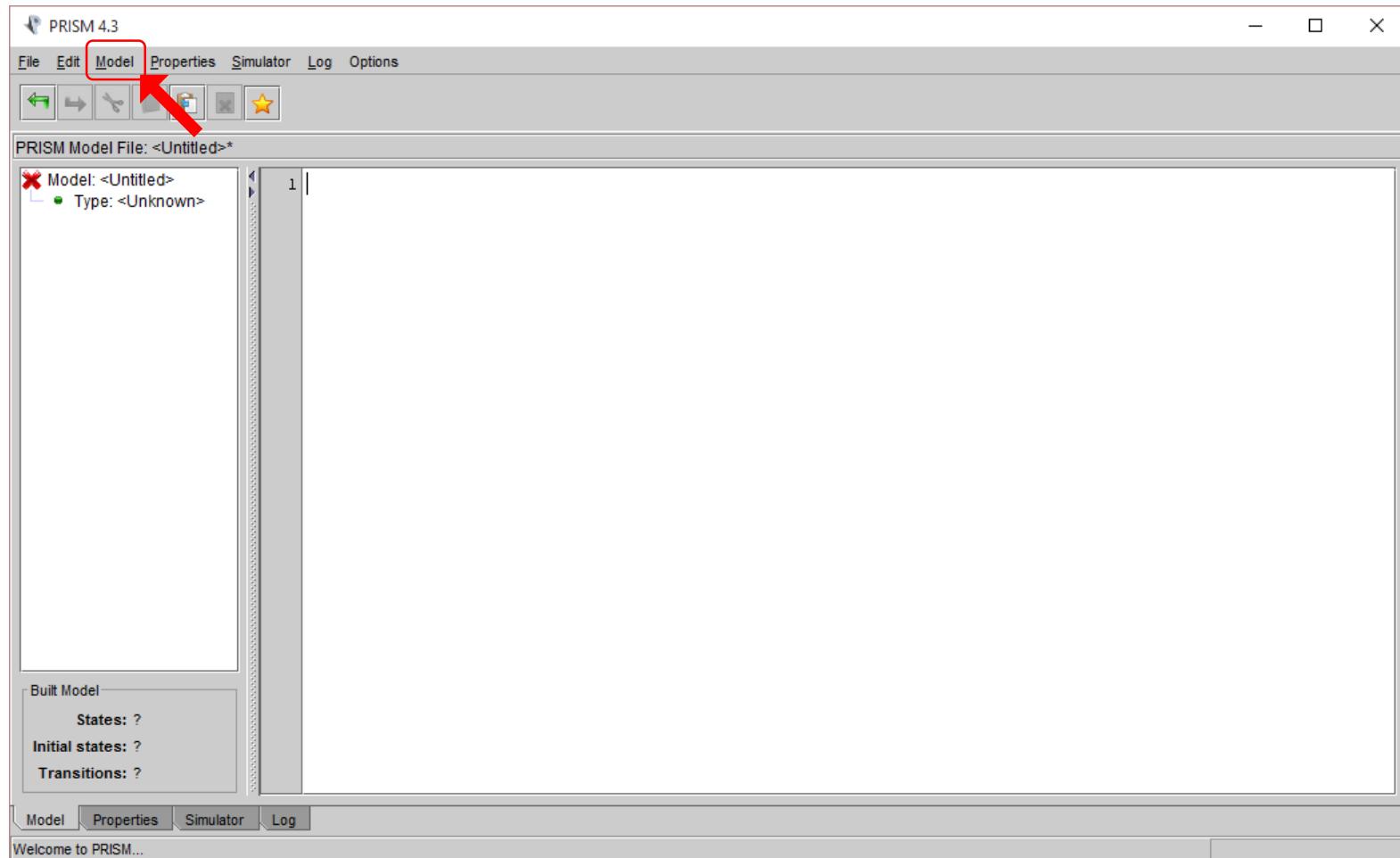
PRISM: Model Checker



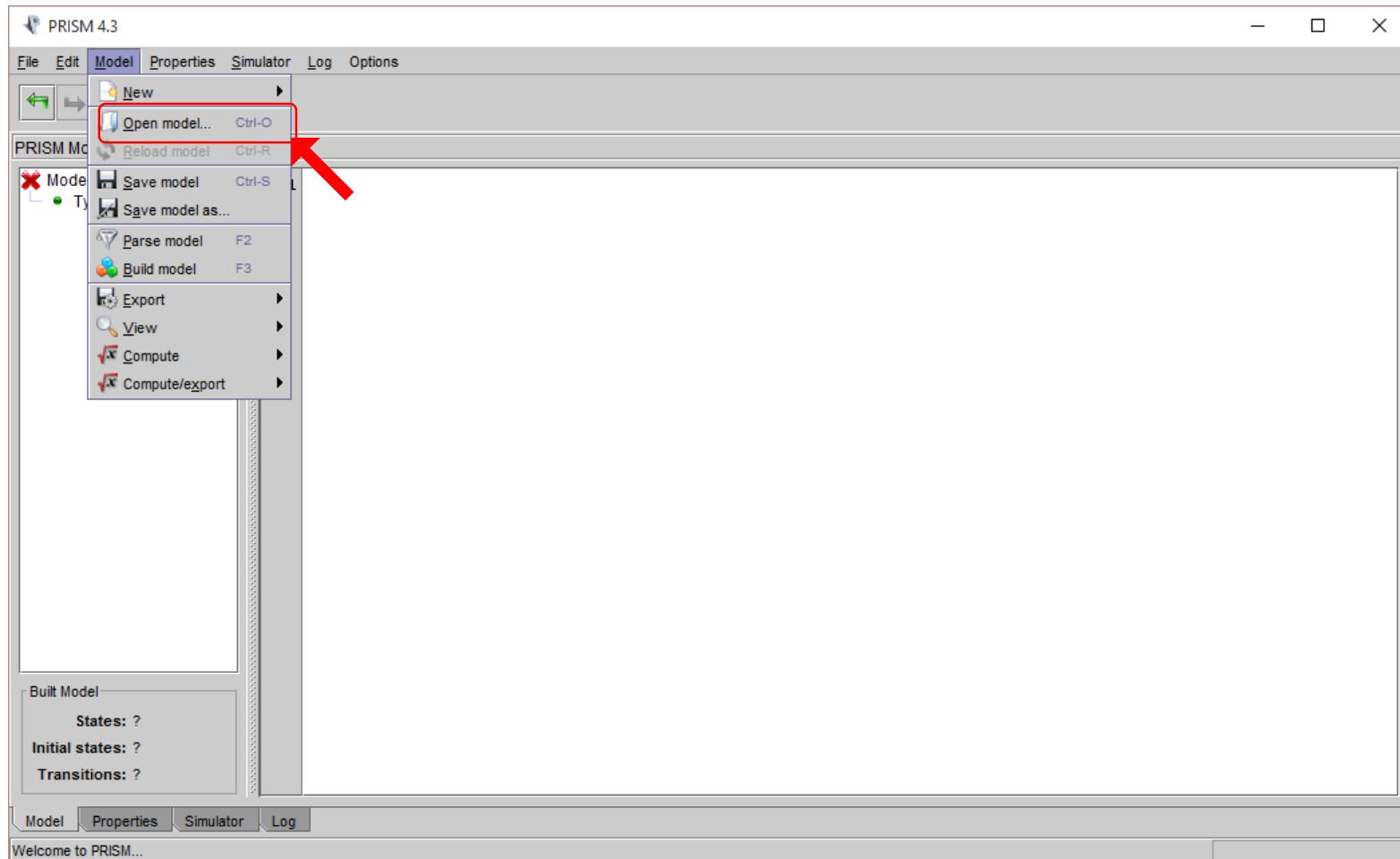
PRISM: Model Checker



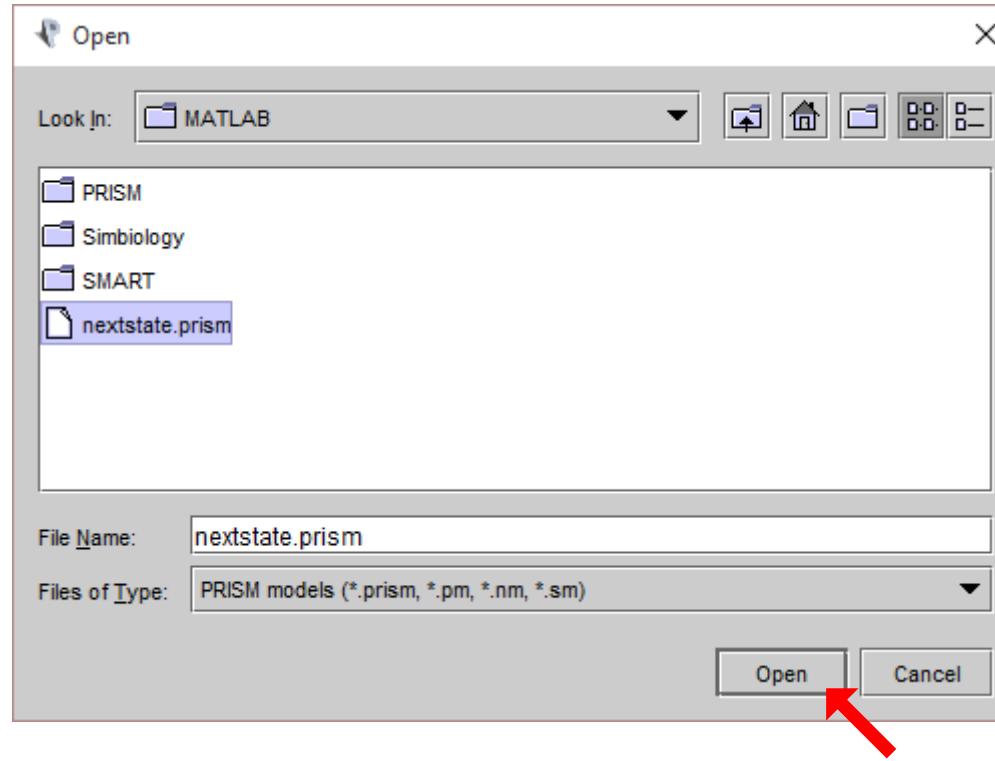
PRISM: Model Checker



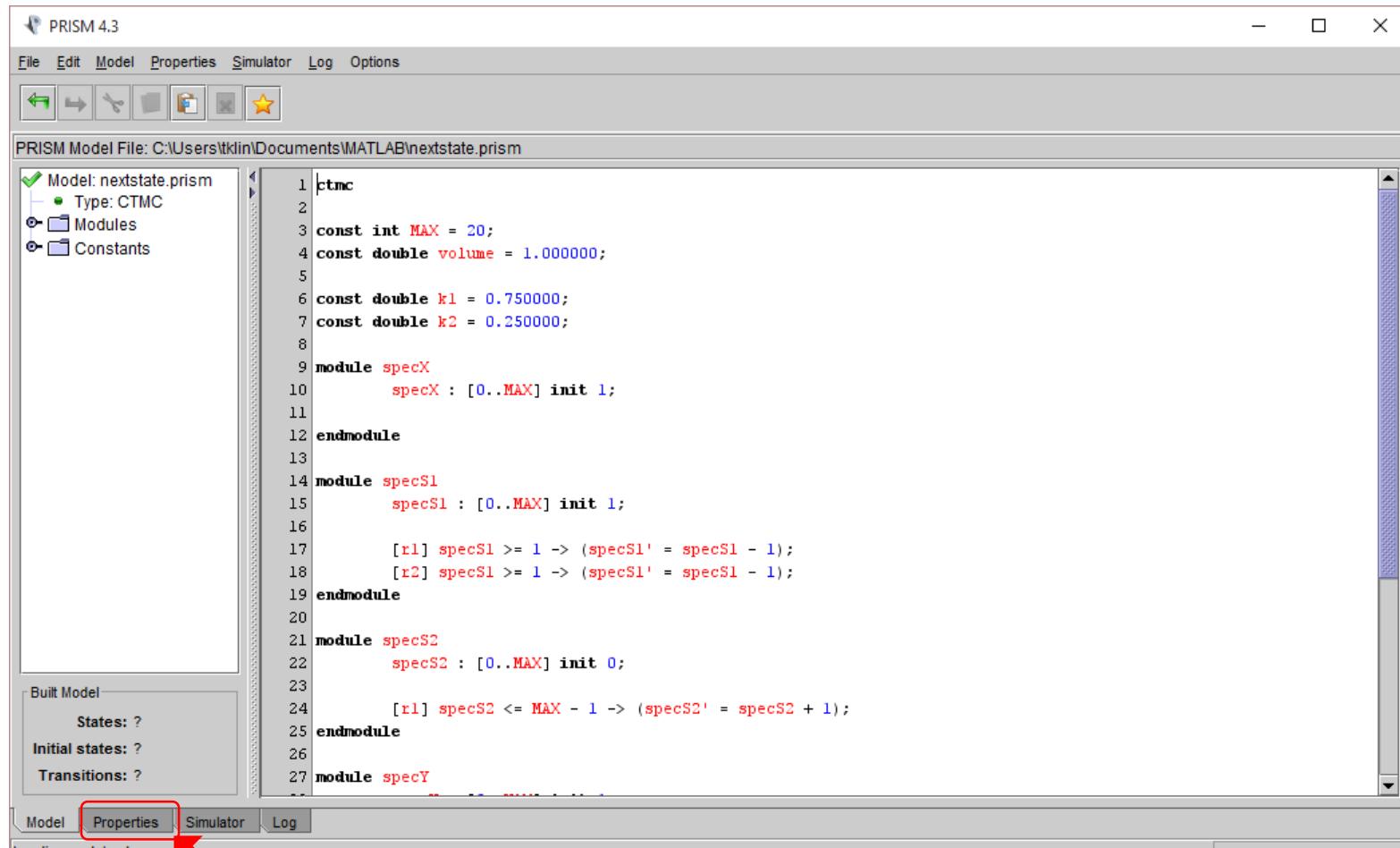
PRISM: Model Checker



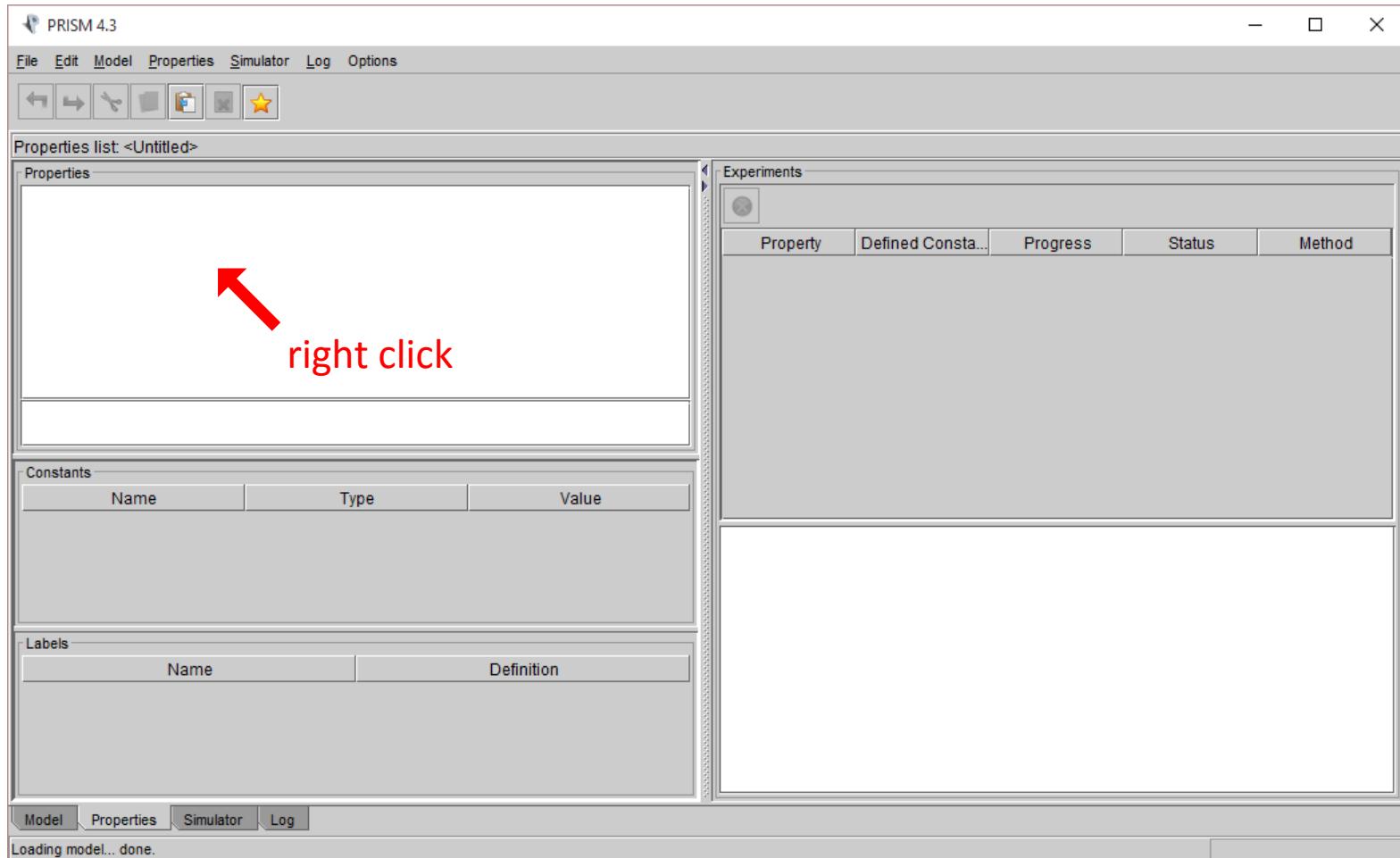
PRISM: Model Checker



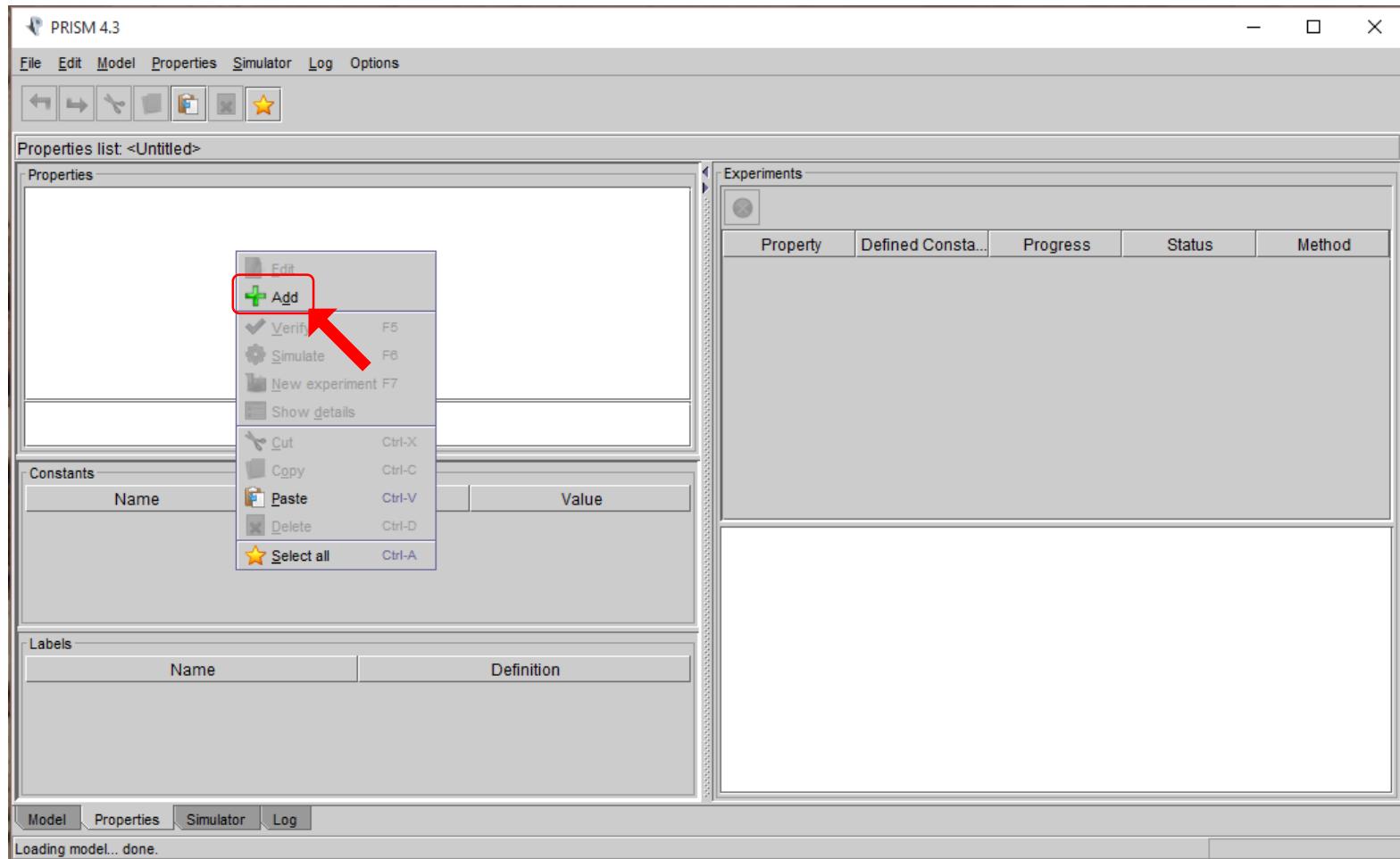
PRISM: Model Checker



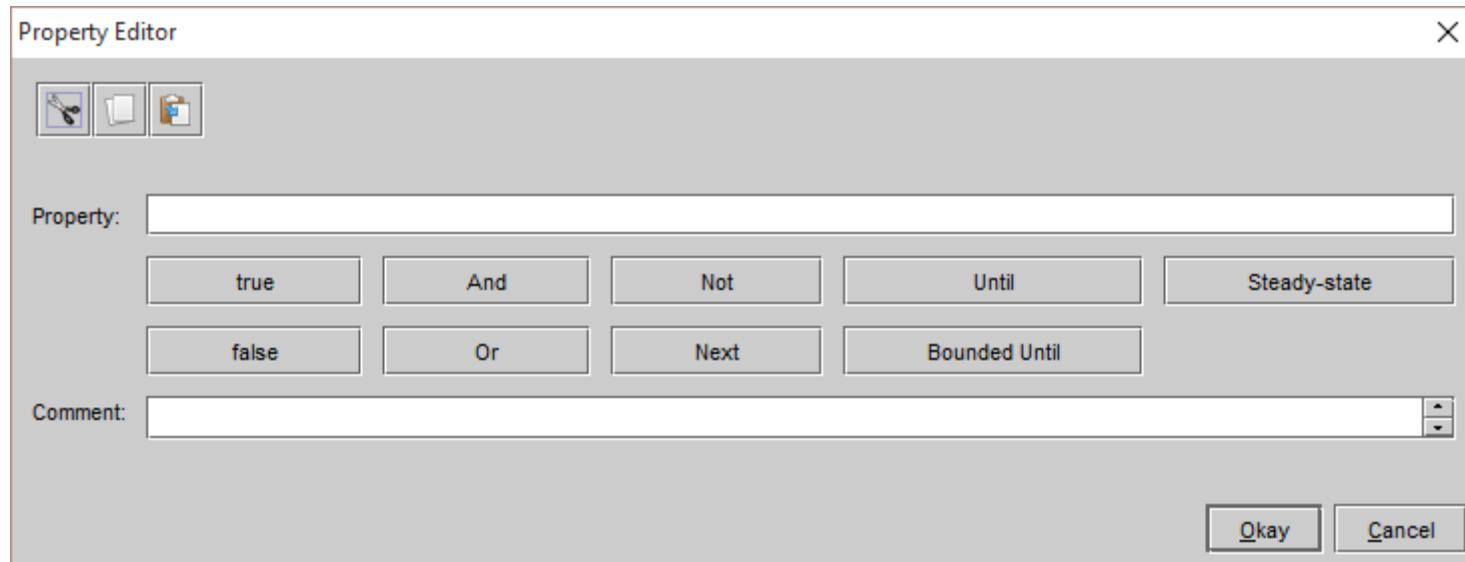
PRISM: Model Checker



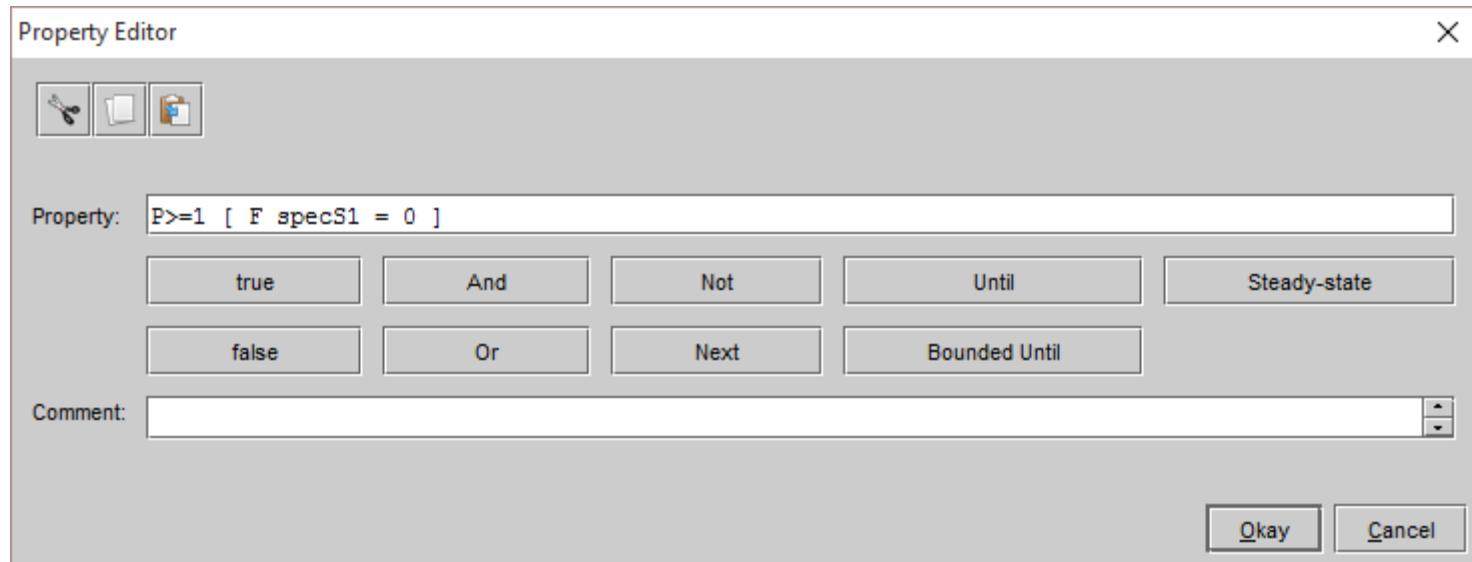
PRISM: Model Checker



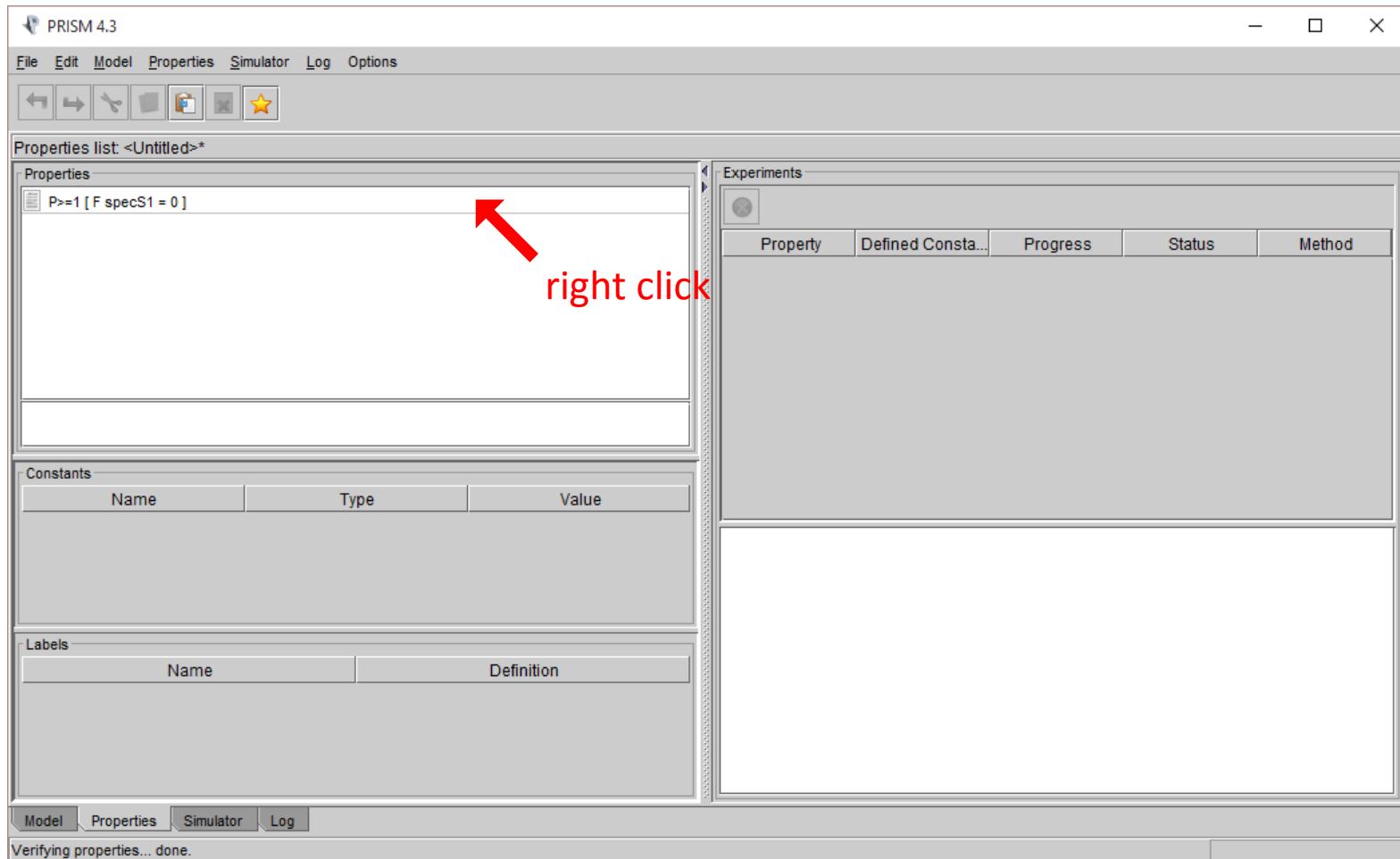
PRISM: Model Checker



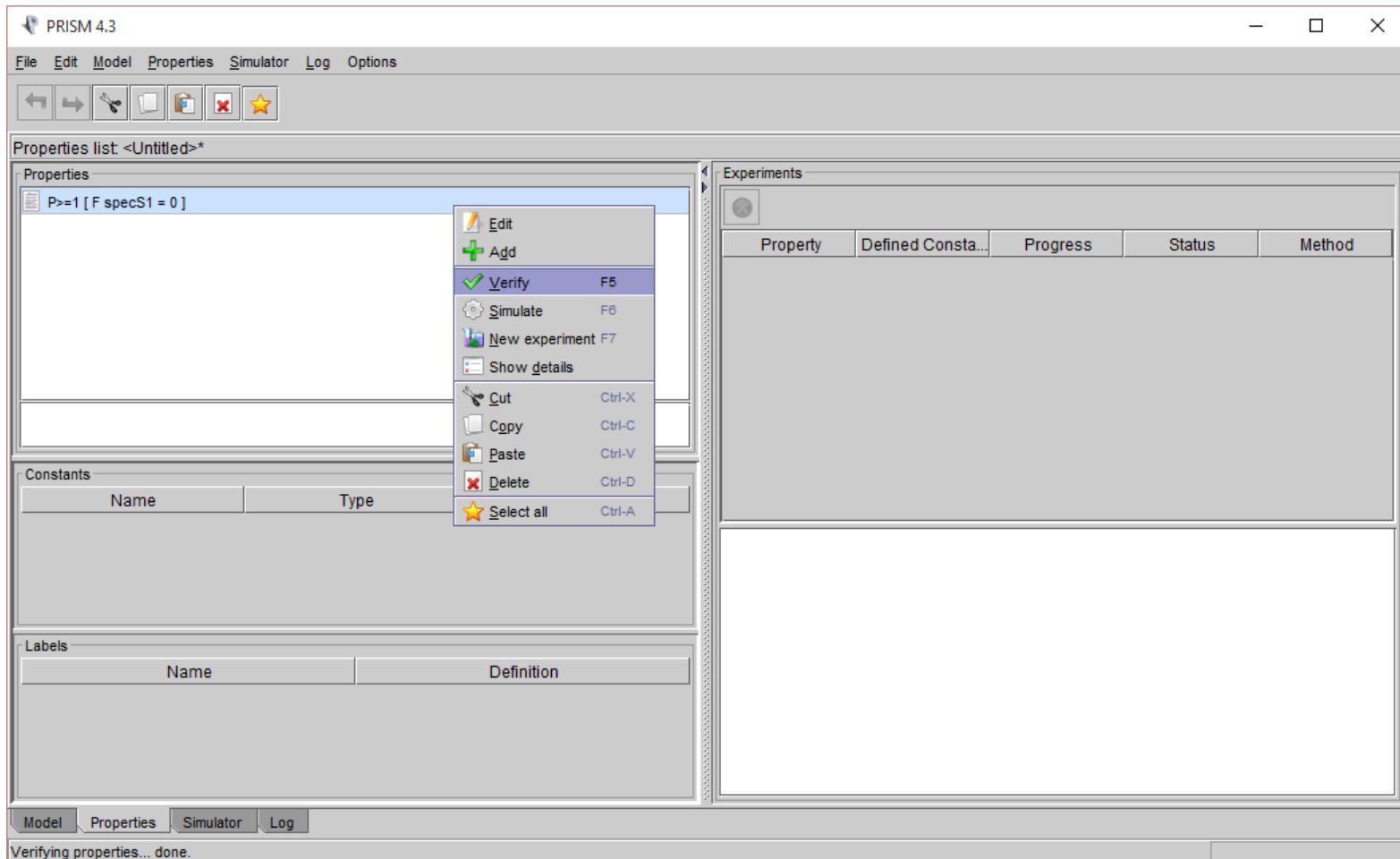
PRISM: Model Checker



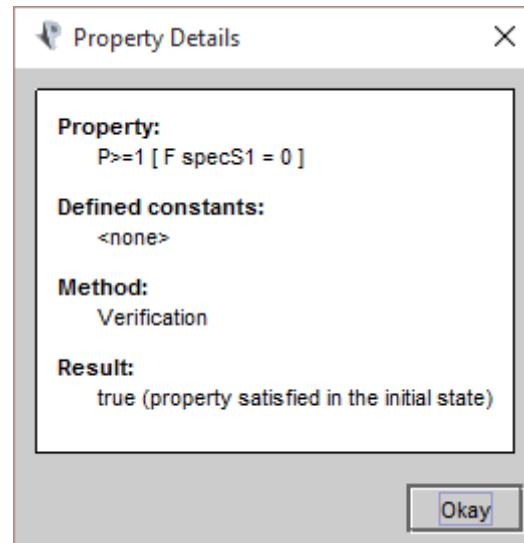
PRISM: Model Checker



PRISM: Model Checker



PRISM: Model Checker



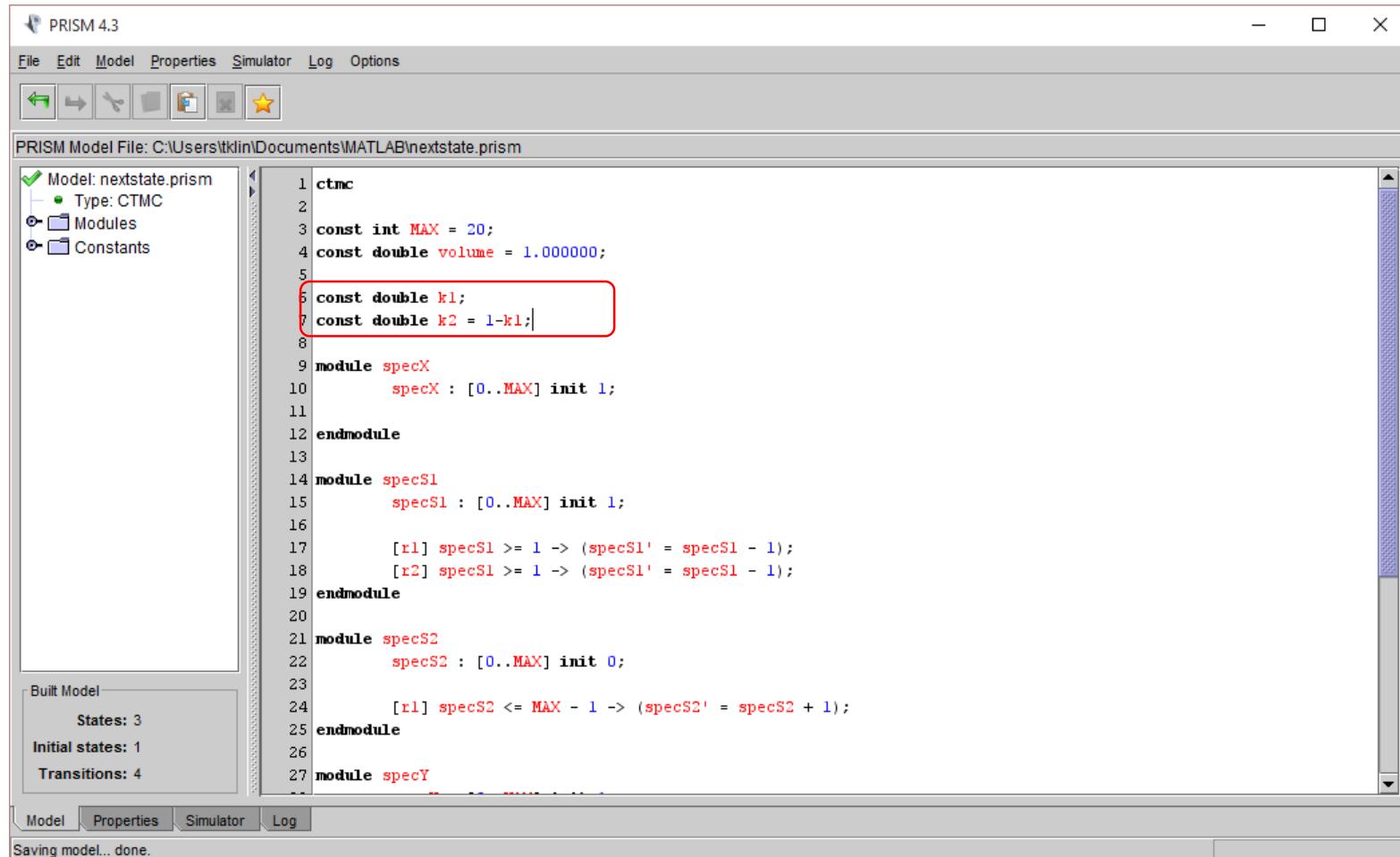
PRISM: Model Checker

The screenshot shows the PRISM 4.3 model checker interface. The window title is "PRISM 4.3". The menu bar includes File, Edit, Model, Properties, Simulator, Log, and Options. The toolbar contains icons for back, forward, search, and other functions. The main area displays a PRISM model file named "nextstate.prism" located at "C:\Users\tklin\Documents\MATLAB\nextstate.prism". The code editor highlights several lines of code in red:

```
1 ctmc
2
3 const int MAX = 20;
4 const double volume = 1.000000;
5
6 const double k1 = 0.750000;
7 const double k2 = 0.250000;
8
9 module specX
10     specX : [0..MAX] init 1;
11
12 endmodule
13
14 module specS1
15     specS1 : [0..MAX] init 1;
16
17     [r1] specS1 >= 1 -> (specS1' = specS1 - 1);
18     [r2] specS1 >= 1 -> (specS1' = specS1 - 1);
19 endmodule
20
21 module specS2
22     specS2 : [0..MAX] init 0;
23
24     [r1] specS2 <= MAX - 1 -> (specS2' = specS2 + 1);
25 endmodule
26
27 module specY
```

The left sidebar shows the model structure: "Model: nextstate.prism" (Type: CTMC), "Modules" (empty), and "Constants" (empty). The bottom-left panel shows the "Built Model" statistics: States: 3, Initial states: 1, Transitions: 4. The bottom status bar says "Verifying properties... done."

PRISM: Model Checker

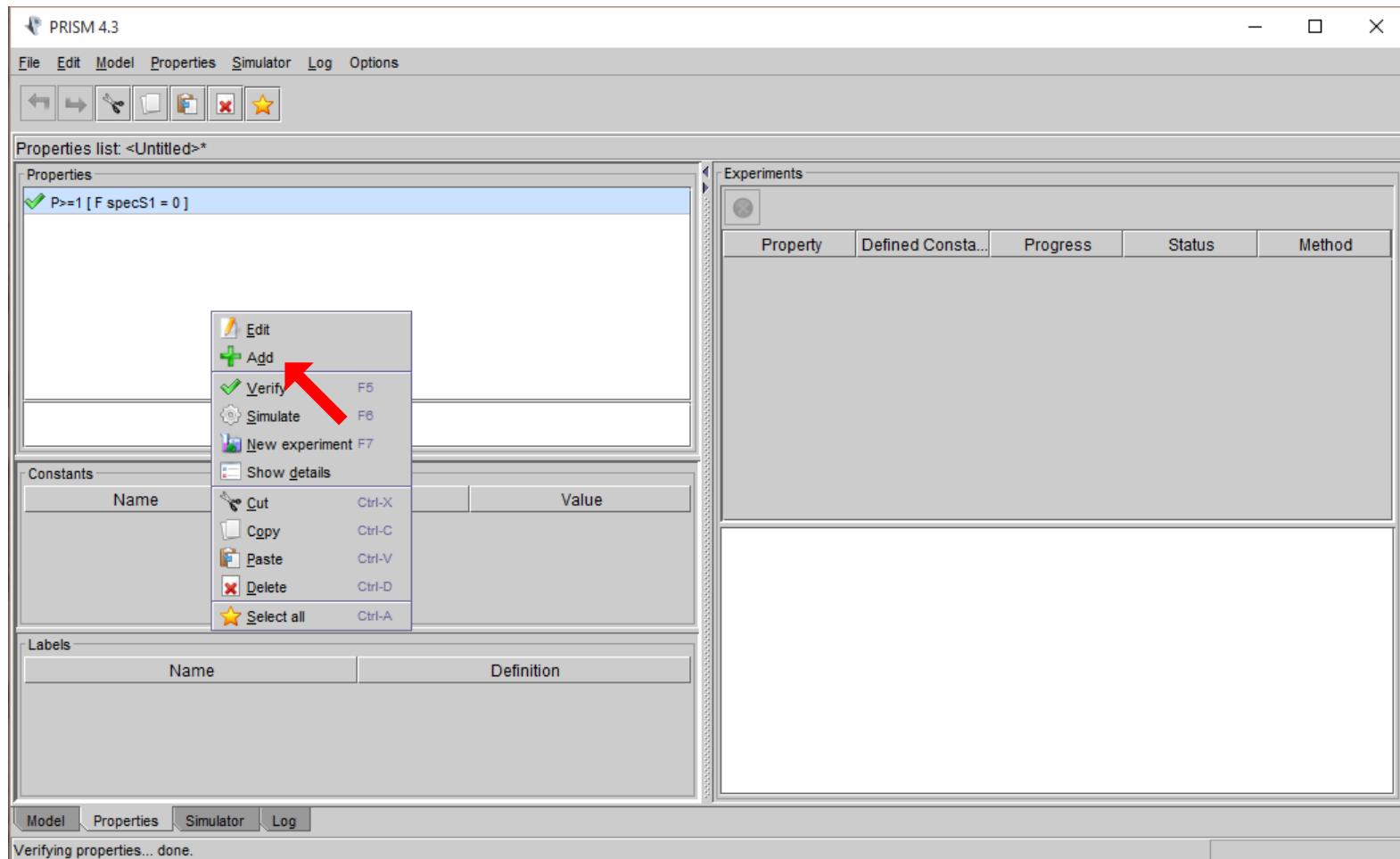


The screenshot shows the PRISM 4.3 graphical user interface. The main window title is "PRISM 4.3". The menu bar includes File, Edit, Model, Properties, Simulator, Log, and Options. Below the menu is a toolbar with icons for back, forward, search, and other functions. The central area displays a PRISM model file named "nextstate.prism" located at "C:\Users\tklin\Documents\MATLAB\nextstate.prism". The code editor shows the following PRISM code:

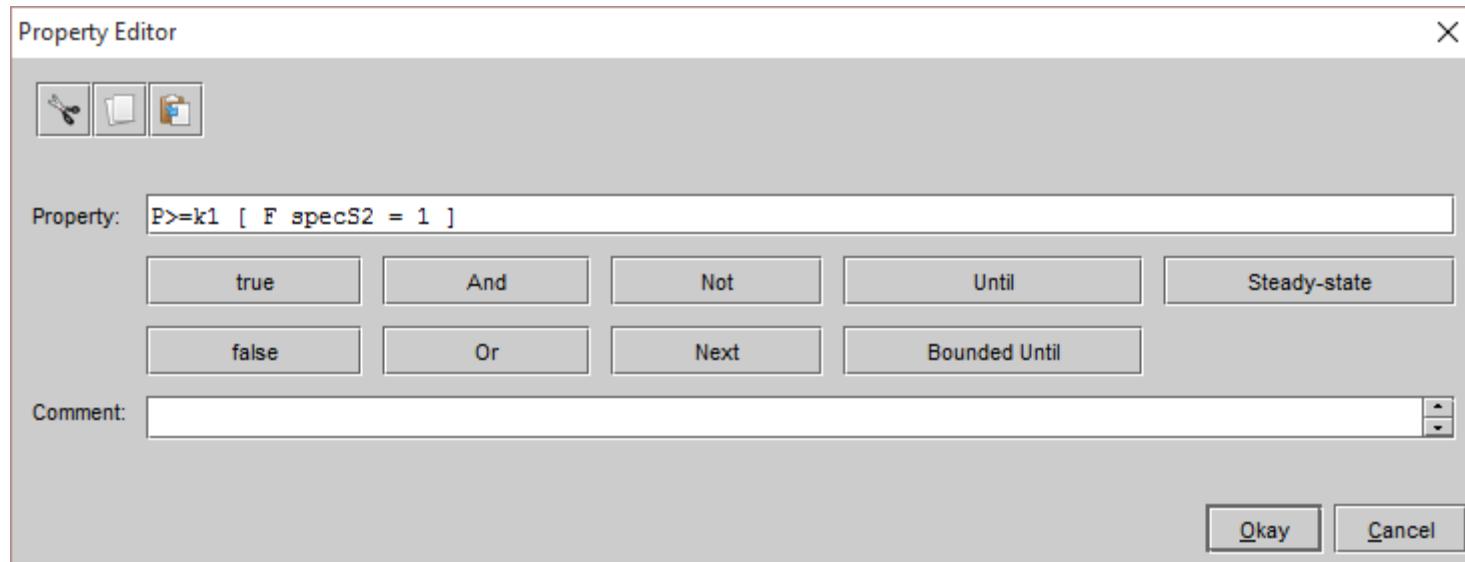
```
1 ctmc
2
3 const int MAX = 20;
4 const double volume = 1.000000;
5
6 const double k1;
7 const double k2 = 1-k1;
8
9 module specX
10     specX : [0..MAX] init 1;
11
12 endmodule
13
14 module specS1
15     specS1 : [0..MAX] init 1;
16
17     [r1] specS1 >= 1 -> (specS1' = specS1 - 1);
18     [r2] specS1 >= 1 -> (specS1' = specS1 - 1);
19 endmodule
20
21 module specS2
22     specS2 : [0..MAX] init 0;
23
24     [r1] specS2 <= MAX - 1 -> (specS2' = specS2 + 1);
25 endmodule
26
27 module specY
```

A red rectangular box highlights the declaration of `k1` and `k2`. On the left side of the interface, there is a tree view showing the model structure: "Model: nextstate.prism" (checked), "Type: CTMC", "Modules" (closed), and "Constants" (closed). At the bottom left, a "Built Model" section provides summary statistics: States: 3, Initial states: 1, Transitions: 4. The bottom navigation bar includes tabs for Model, Properties, Simulator, and Log, with "Model" currently selected. A status message at the bottom says "Saving model... done."

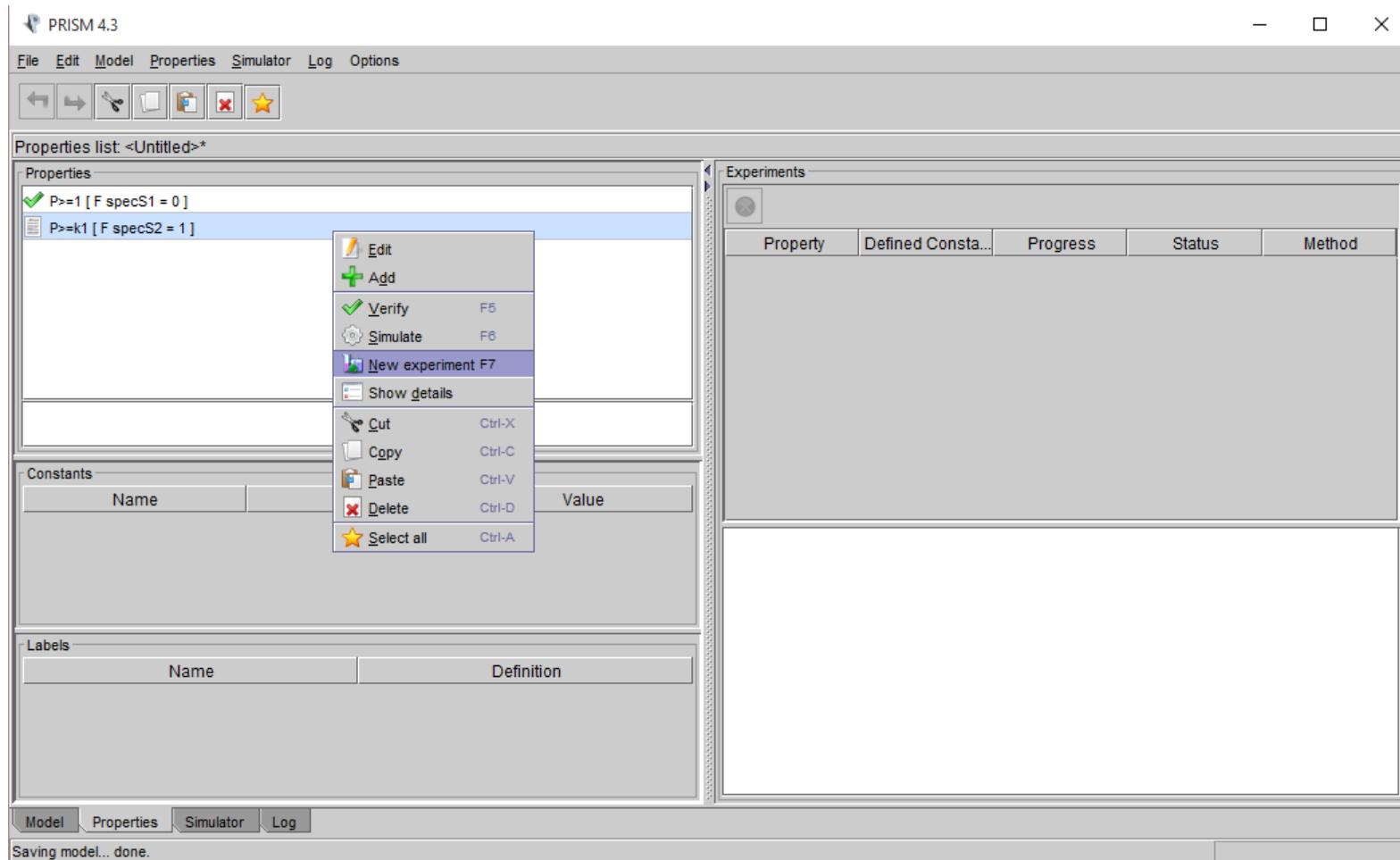
PRISM: Model Checker



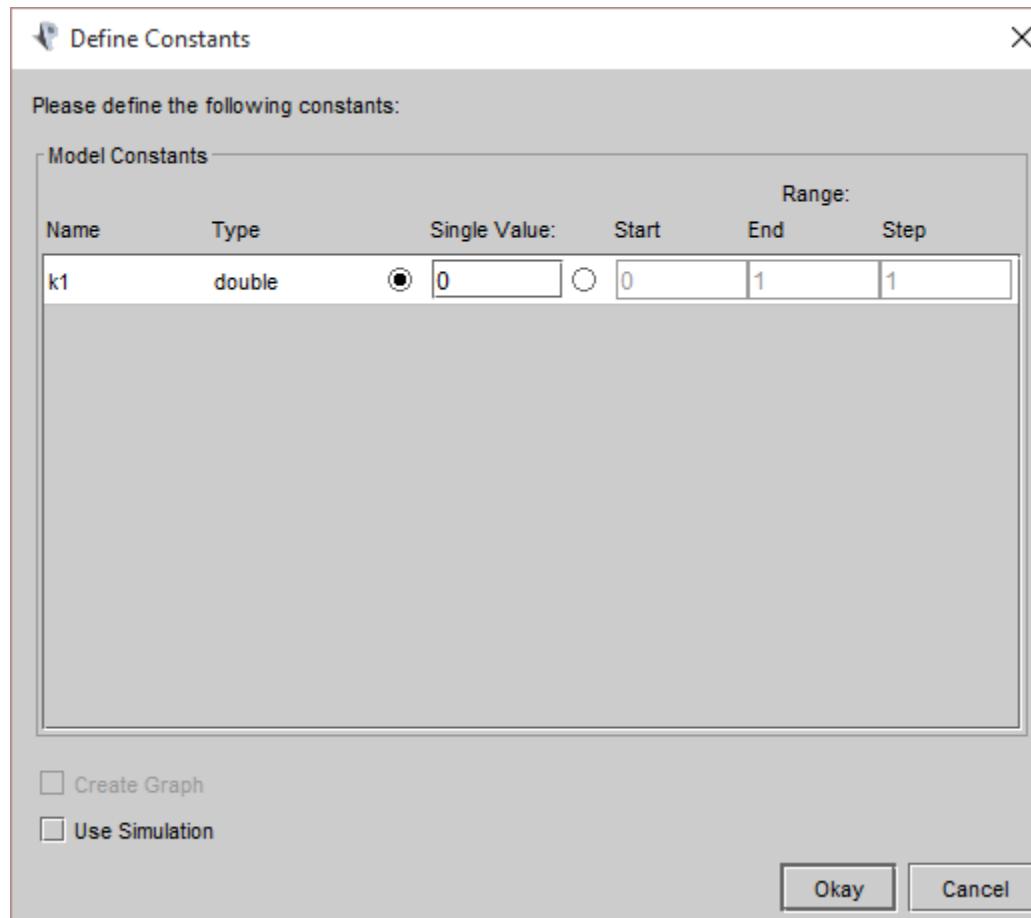
PRISM: Model Checker



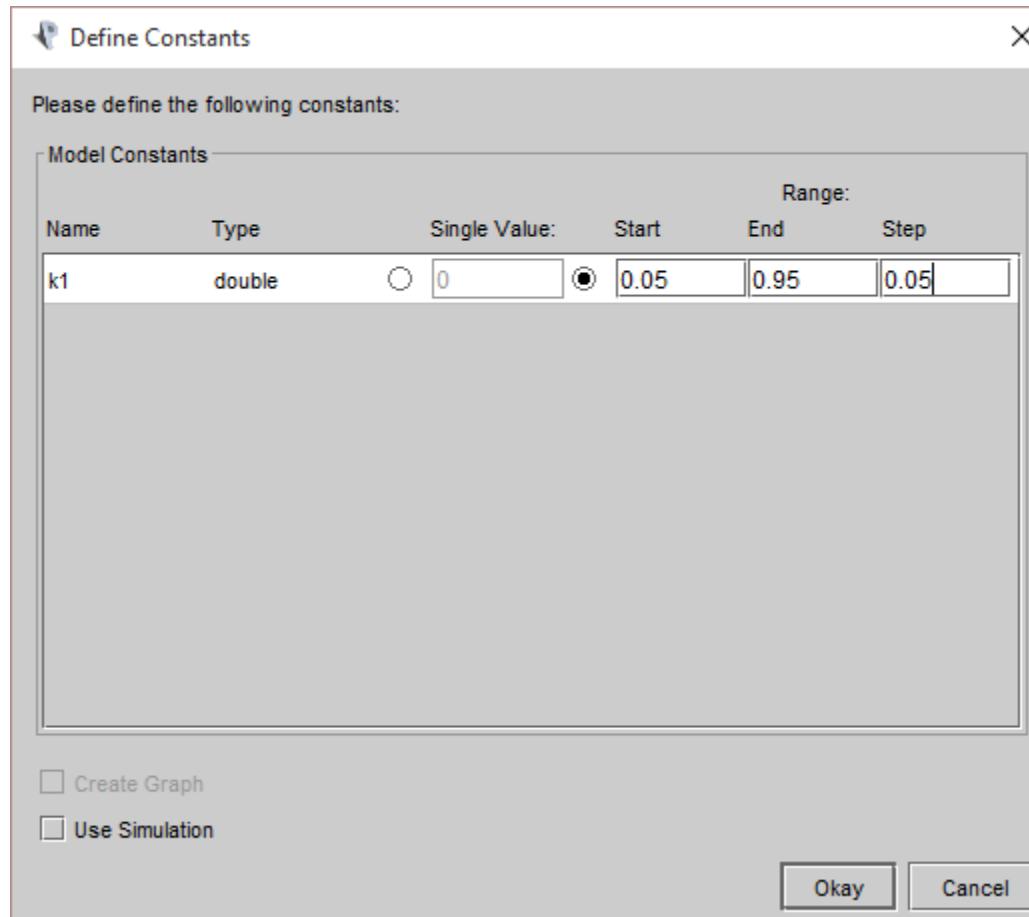
PRISM: Model Checker



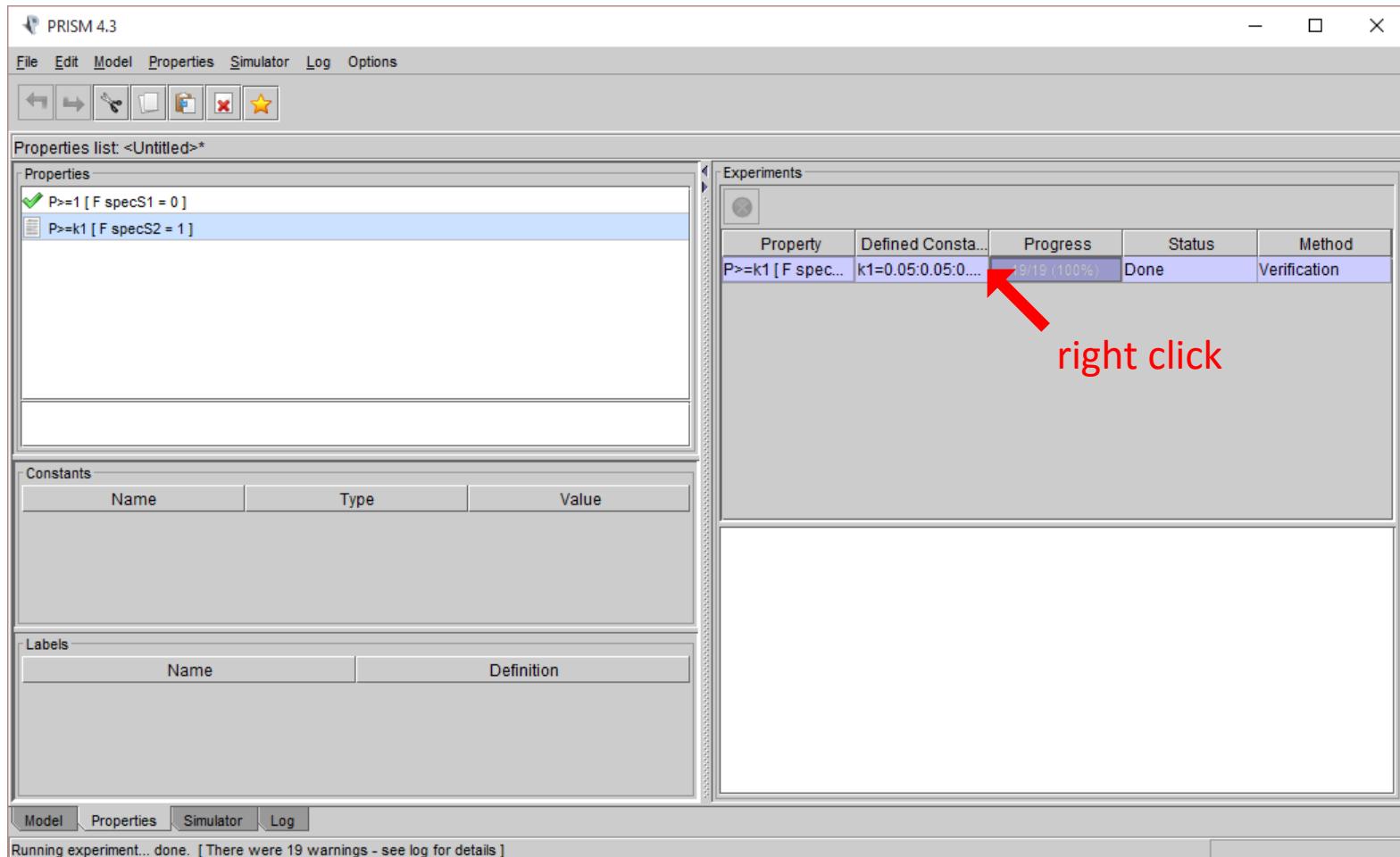
PRISM: Model Checker



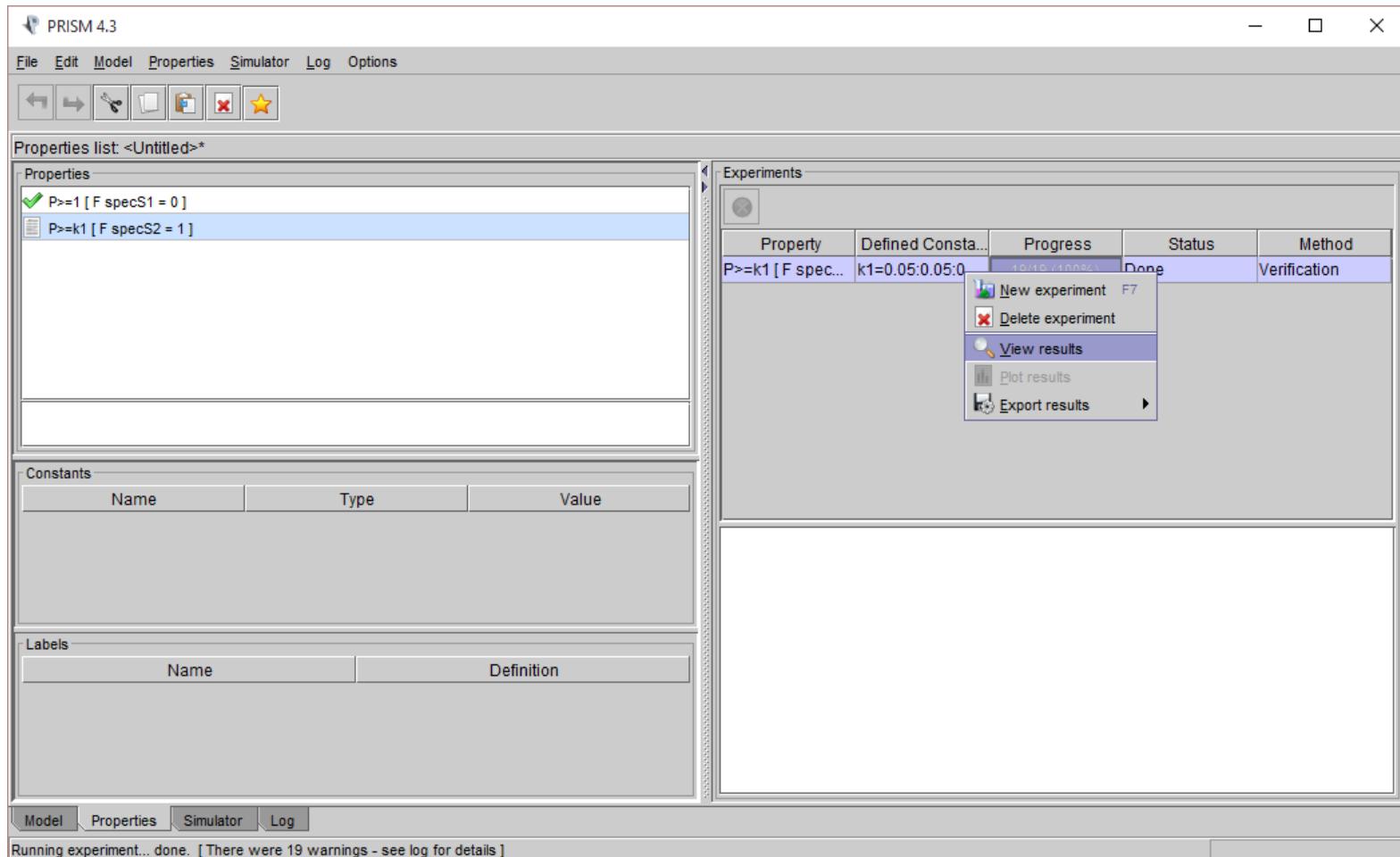
PRISM: Model Checker



PRISM: Model Checker



PRISM: Model Checker



PRISM: Model Checker

Experiment Results

Results of ' $P \geq k1 [F specS2=1]$ ' for ' $k1=0.05:0.05:0.9500000000000001$ '

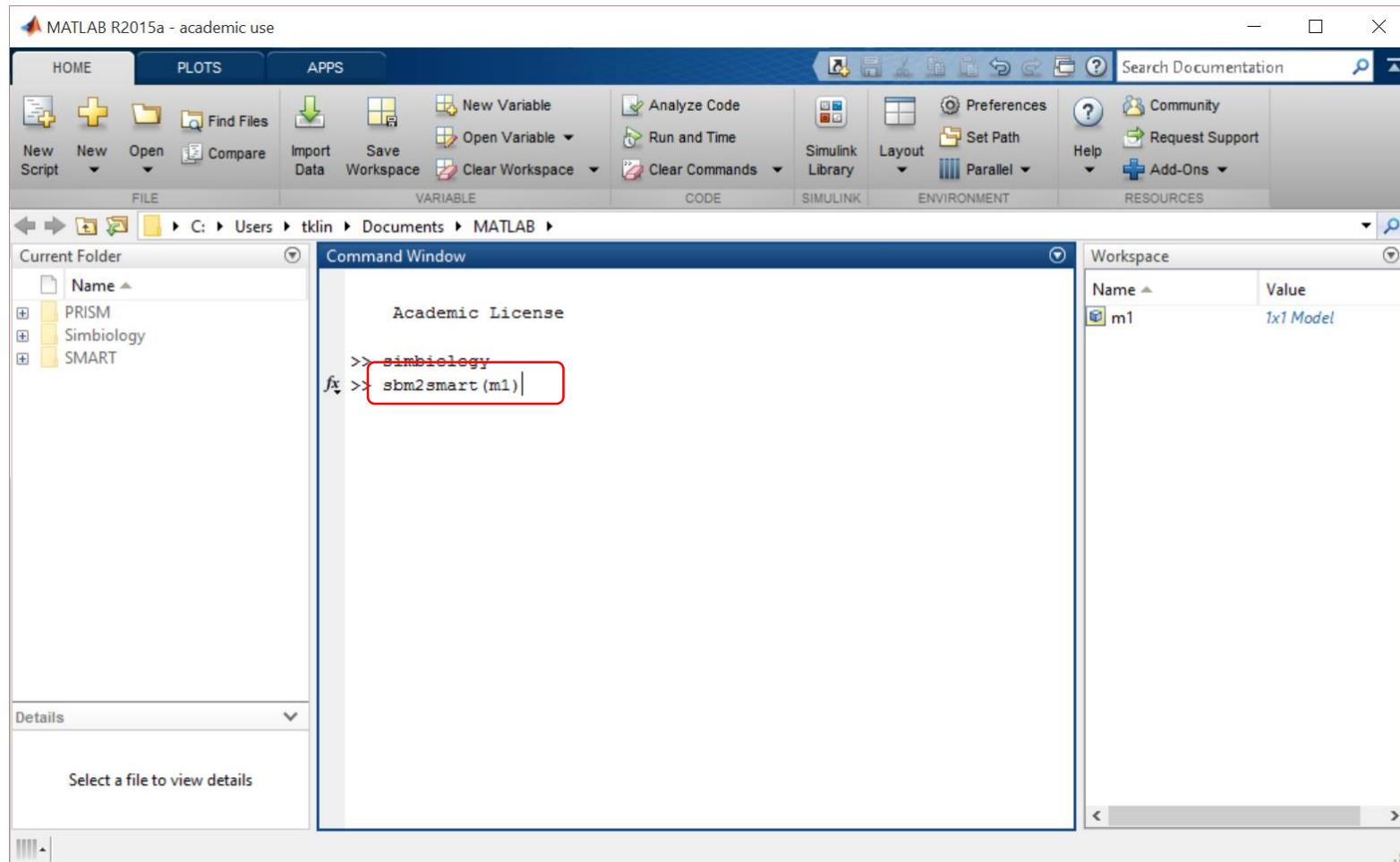
Results	
$k1$	Result
0.05	true
0.1	true
0.1500000000000002	true
0.2	true
0.25	true
0.3	true
0.3500000000000003	true
0.4	true
0.45	true
0.5	true
0.55	true
0.6000000000000001	true
0.6500000000000001	true

Close

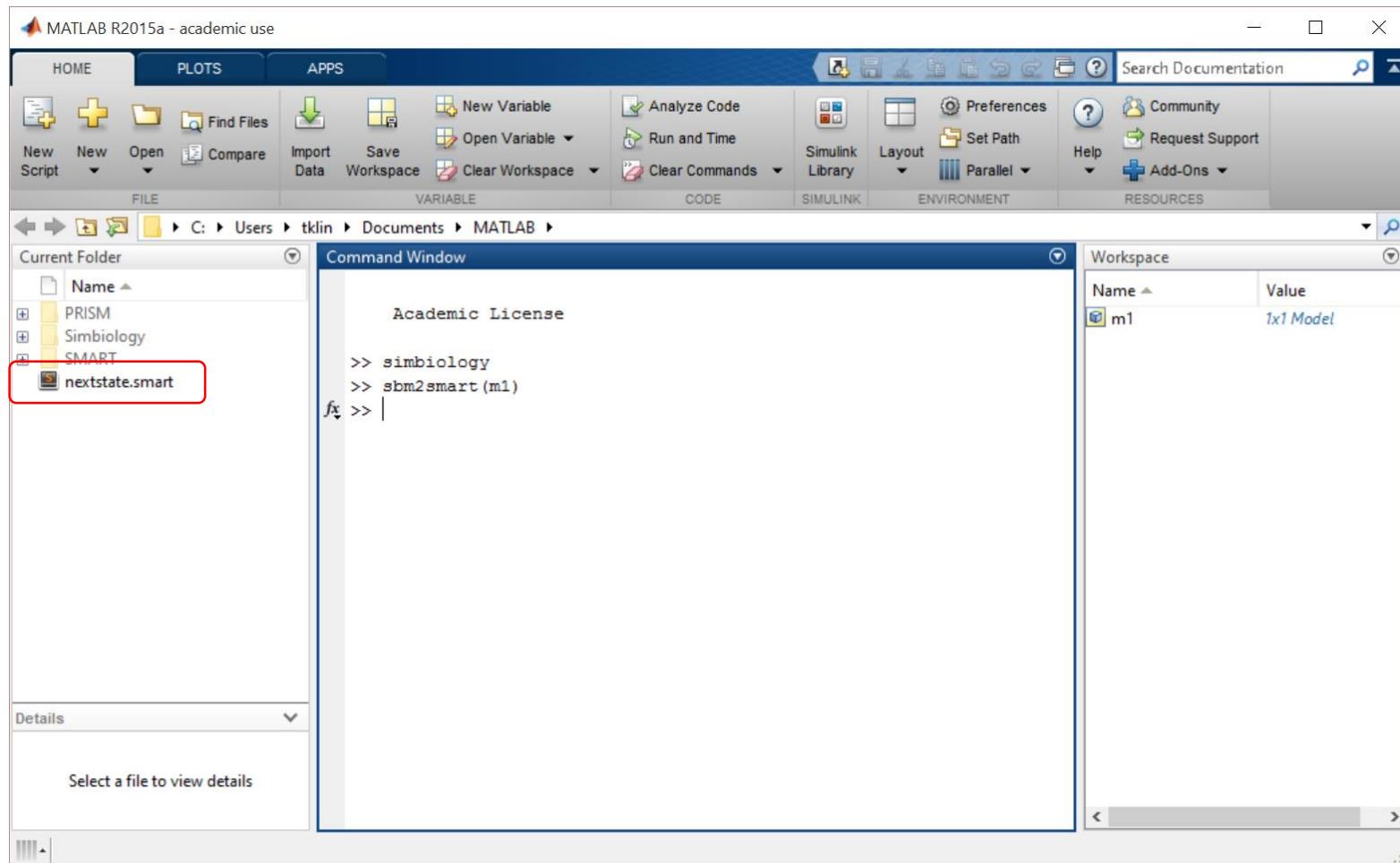
SMART: Model Checker

- Maintained by researchers at ISU
 - Andrew Miner
 - Gianfranco Ciardo
- Works great with Petri Nets
 - i.e. works great with CRNs
- Powerful programming language features
- Command line tool

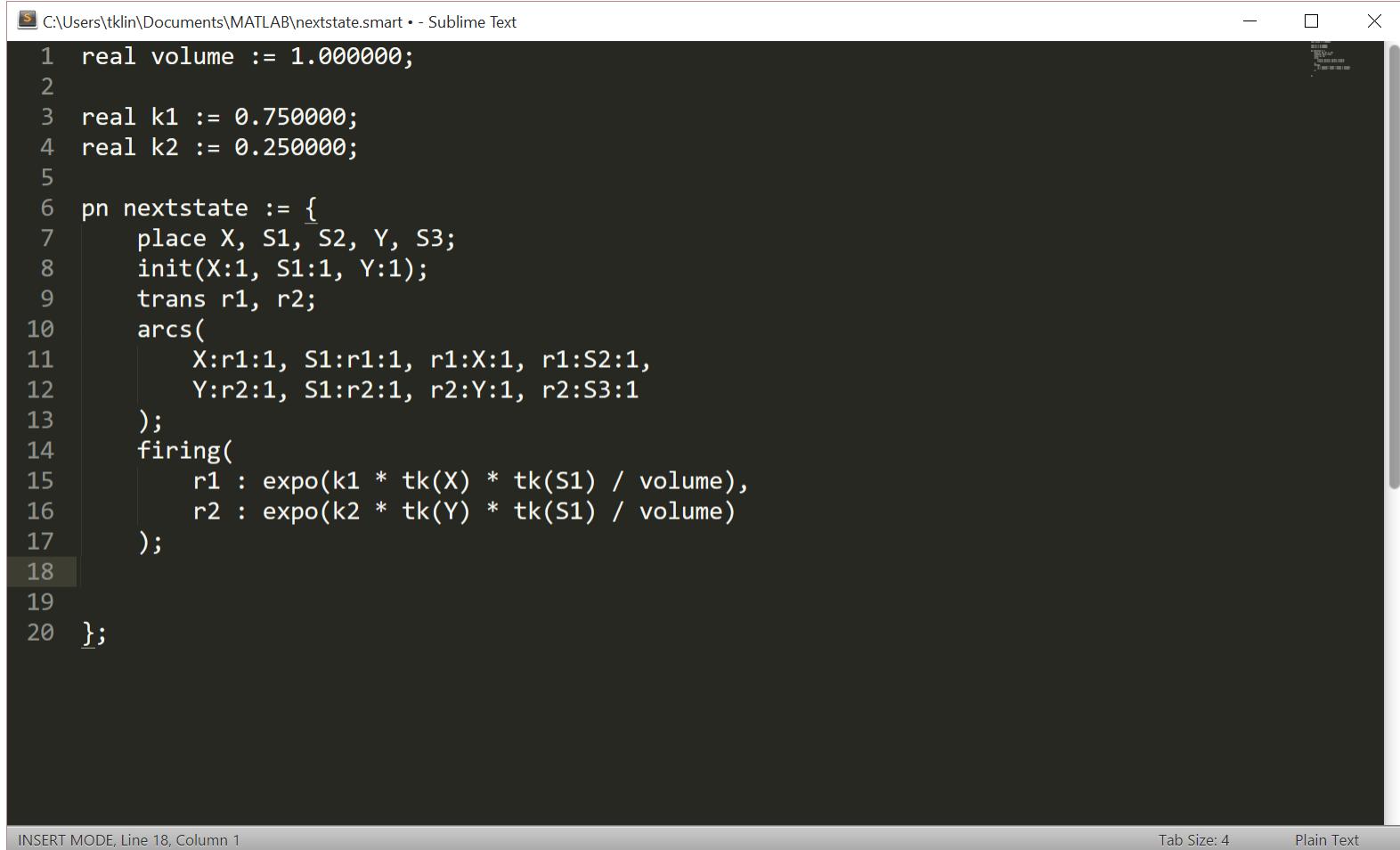
SMART: Model Checker



SMART: Model Checker



SMART: Model Checker

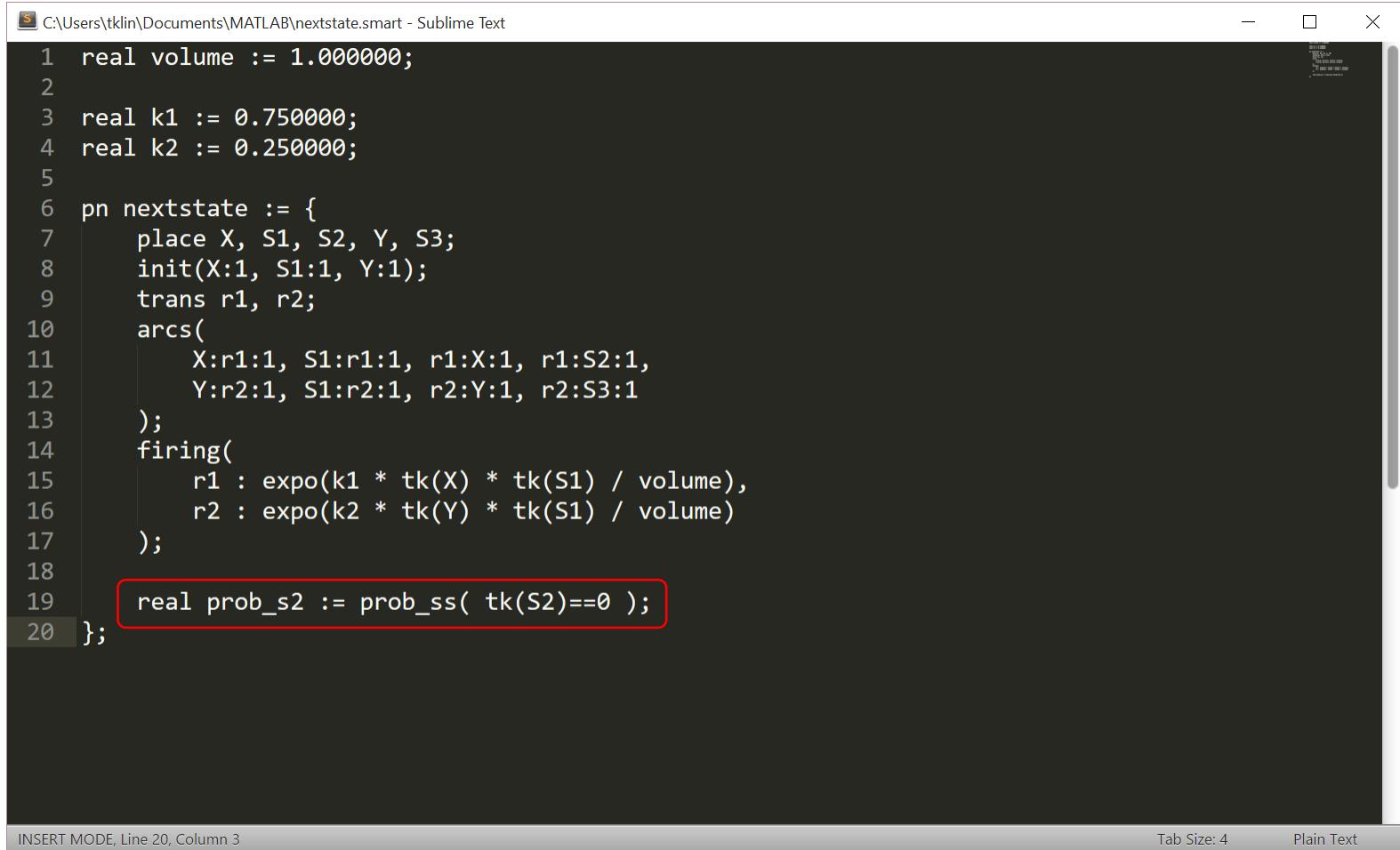


A screenshot of a Sublime Text editor window. The title bar reads "C:\Users\tklin\Documents\MATLAB\nextstate.smart - Sublime Text". The code in the editor is:

```
1 real volume := 1.000000;
2
3 real k1 := 0.750000;
4 real k2 := 0.250000;
5
6 pn nextstate := {
7     place X, S1, S2, Y, S3;
8     init(X:1, S1:1, Y:1);
9     trans r1, r2;
10    arcs(
11        X:r1:1, S1:r1:1, r1:X:1, r1:S2:1,
12        Y:r2:1, S1:r2:1, r2:Y:1, r2:S3:1
13    );
14    firing(
15        r1 : expo(k1 * tk(X) * tk(S1) / volume),
16        r2 : expo(k2 * tk(Y) * tk(S1) / volume)
17    );
18
19
20};
```

The status bar at the bottom left says "INSERT MODE, Line 18, Column 1". The status bar at the bottom right says "Tab Size: 4" and "Plain Text".

SMART: Model Checker



The screenshot shows a Sublime Text editor window with the following content:

```
C:\Users\tklin\Documents\MATLAB\nextstate.smart - Sublime Text
```

```
1 real volume := 1.000000;
2
3 real k1 := 0.750000;
4 real k2 := 0.250000;
5
6 pn nextstate := {
7     place X, S1, S2, Y, S3;
8     init(X:1, S1:1, Y:1);
9     trans r1, r2;
10    arcs(
11        X:r1:1, S1:r1:1, r1:X:1, r1:S2:1,
12        Y:r2:1, S1:r2:1, r2:Y:1, r2:S3:1
13    );
14    firing(
15        r1 : expo(k1 * tk(X) * tk(S1) / volume),
16        r2 : expo(k2 * tk(Y) * tk(S1) / volume)
17    );
18
19    real prob_s2 := prob_ss( tk(S2)==0 );
20};
```

The line `real prob_s2 := prob_ss(tk(S2)==0);` is highlighted with a red border.

Bottom status bar: INSERT MODE, Line 20, Column 3 | Tab Size: 4 | Plain Text

SMART: Model Checker

```
real prob_s2 := prob_ss( tk(S2)==0 );
```

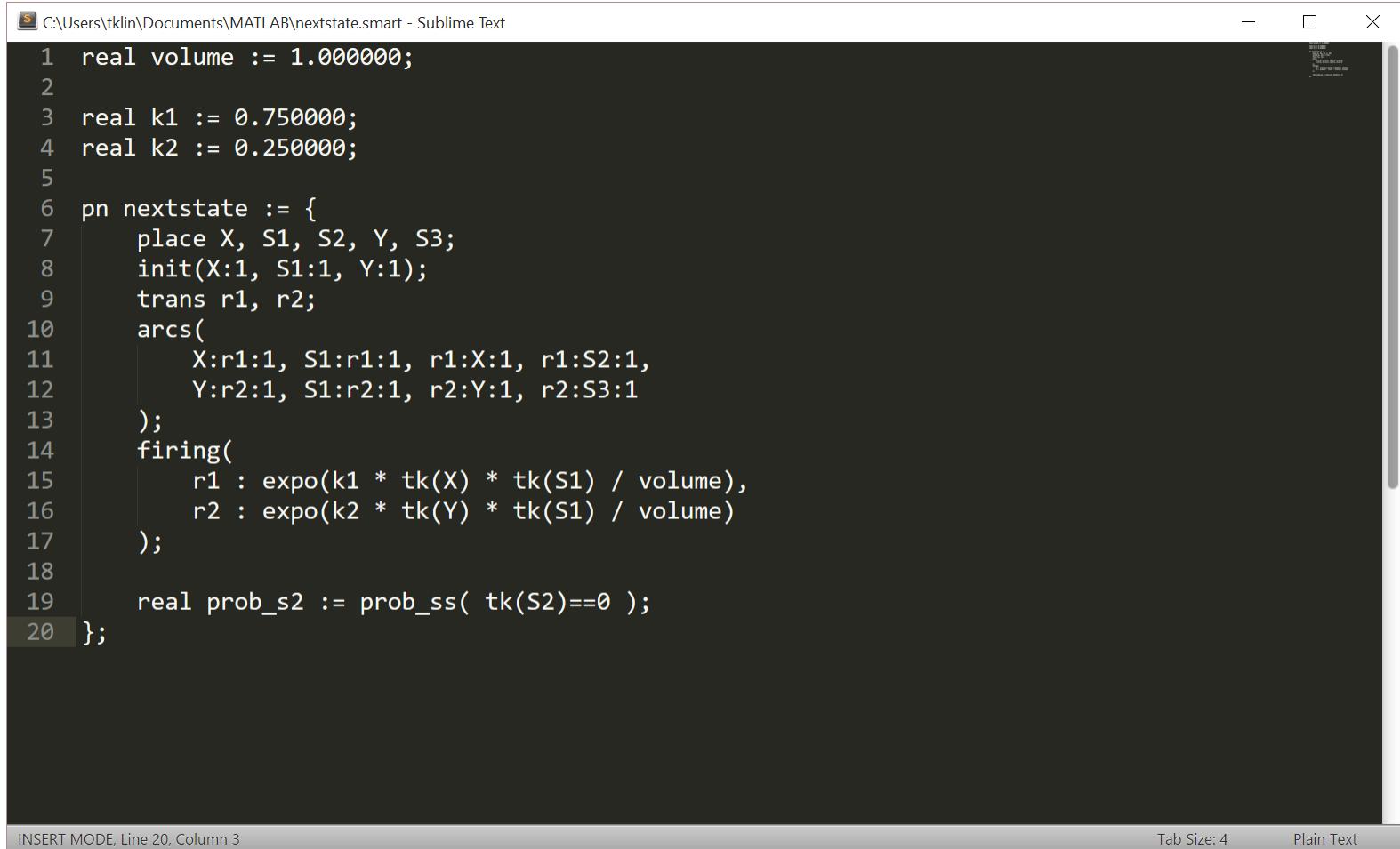


Just like the
S operator



Number of
tokens of S2

SMART: Model Checker

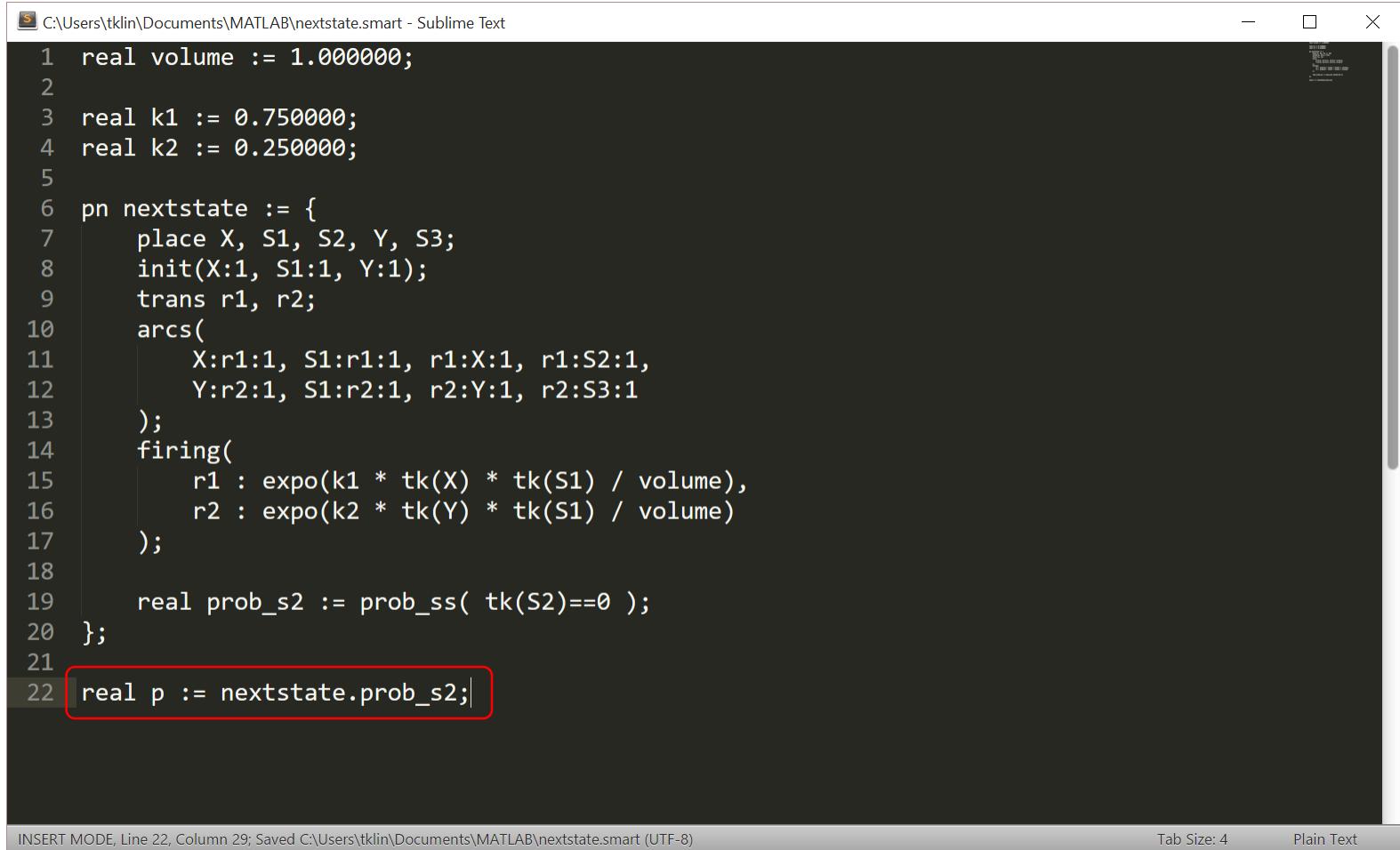


C:\Users\tklin\Documents\MATLAB\nextstate.smart - Sublime Text

```
1 real volume := 1.000000;
2
3 real k1 := 0.750000;
4 real k2 := 0.250000;
5
6 pn nextstate := {
7     place X, S1, S2, Y, S3;
8     init(X:1, S1:1, Y:1);
9     trans r1, r2;
10    arcs(
11        X:r1:1, S1:r1:1, r1:X:1, r1:S2:1,
12        Y:r2:1, S1:r2:1, r2:Y:1, r2:S3:1
13    );
14    firing(
15        r1 : expo(k1 * tk(X) * tk(S1) / volume),
16        r2 : expo(k2 * tk(Y) * tk(S1) / volume)
17    );
18
19    real prob_s2 := prob_ss( tk(S2)==0 );
20};
```

INSERT MODE, Line 20, Column 3 Tab Size: 4 Plain Text

SMART: Model Checker

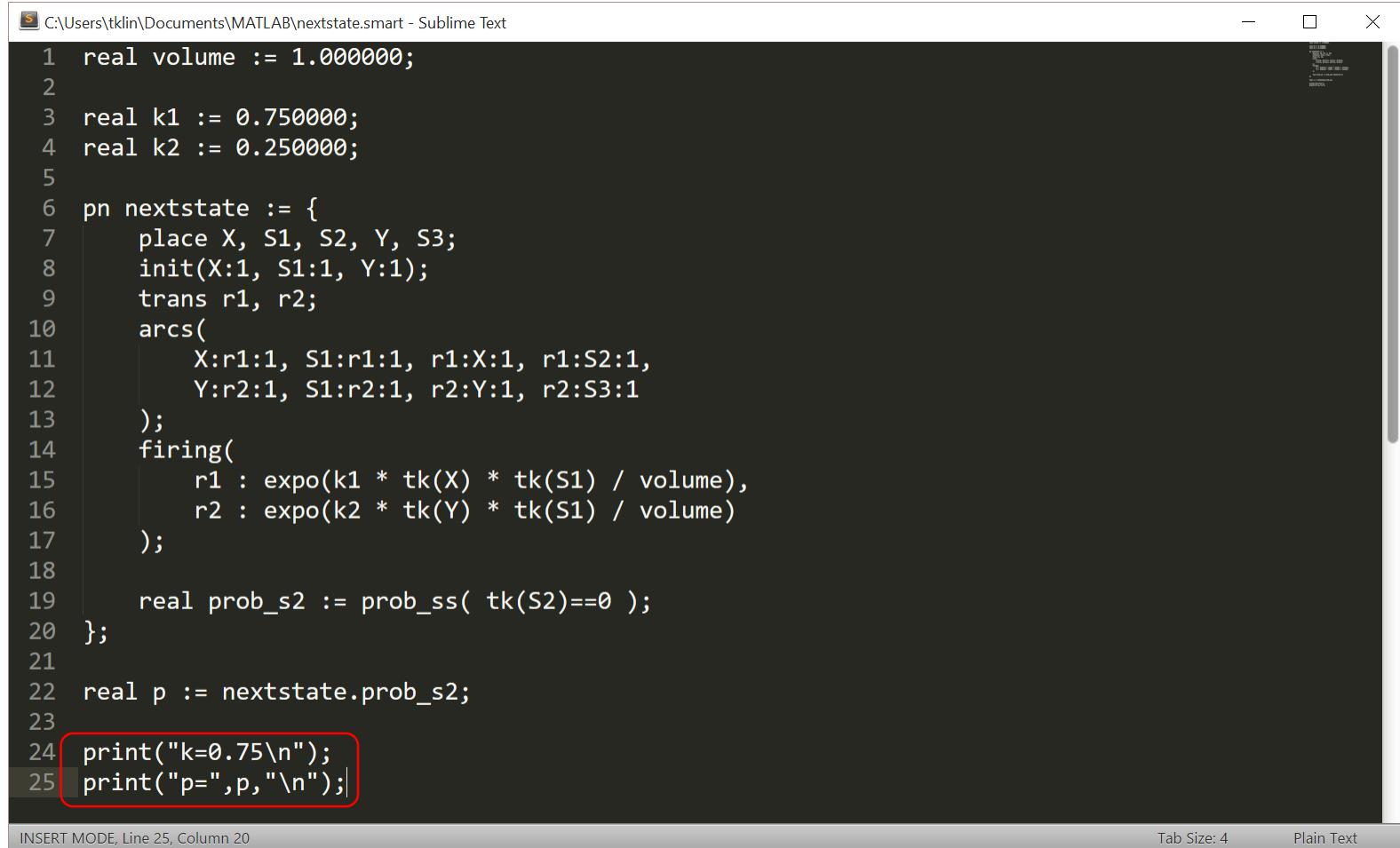


C:\Users\tklin\Documents\MATLAB\nextstate.smart - Sublime Text

```
1 real volume := 1.000000;
2
3 real k1 := 0.750000;
4 real k2 := 0.250000;
5
6 pn nextstate := {
7     place X, S1, S2, Y, S3;
8     init(X:1, S1:1, Y:1);
9     trans r1, r2;
10    arcs(
11        X:r1:1, S1:r1:1, r1:X:1, r1:S2:1,
12        Y:r2:1, S1:r2:1, r2:Y:1, r2:S3:1
13    );
14    firing(
15        r1 : expo(k1 * tk(X) * tk(S1) / volume),
16        r2 : expo(k2 * tk(Y) * tk(S1) / volume)
17    );
18
19    real prob_s2 := prob_ss( tk(S2)==0 );
20 };
21
22 real p := nextstate.prob_s2;
```

INSERT MODE, Line 22, Column 29; Saved C:\Users\tklin\Documents\MATLAB\nextstate.smart (UTF-8) Tab Size: 4 Plain Text

SMART: Model Checker



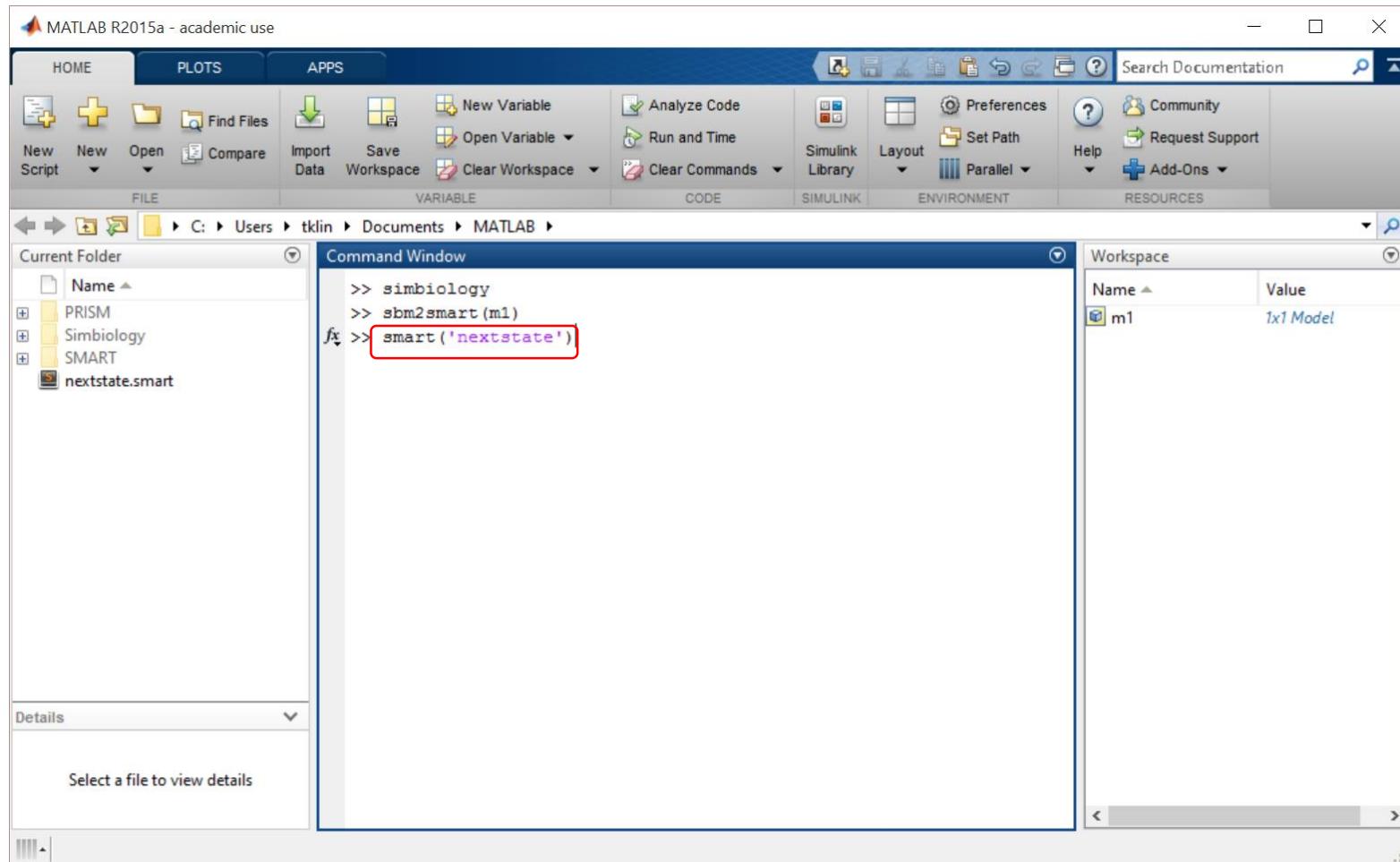
A screenshot of a Sublime Text editor window. The title bar reads "C:\Users\tklin\Documents\MATLAB\nextstate.smart - Sublime Text". The code in the editor is as follows:

```
1 real volume := 1.000000;
2
3 real k1 := 0.750000;
4 real k2 := 0.250000;
5
6 pn nextstate := {
7     place X, S1, S2, Y, S3;
8     init(X:1, S1:1, Y:1);
9     trans r1, r2;
10    arcs(
11        X:r1:1, S1:r1:1, r1:X:1, r1:S2:1,
12        Y:r2:1, S1:r2:1, r2:Y:1, r2:S3:1
13    );
14    firing(
15        r1 : expo(k1 * tk(X) * tk(S1) / volume),
16        r2 : expo(k2 * tk(Y) * tk(S1) / volume)
17    );
18
19    real prob_s2 := prob_ss( tk(S2)==0 );
20 };
21
22 real p := nextstate.prob_s2;
23
24 print("k=0.75\n");
25 print("p=",p,"\n");
```

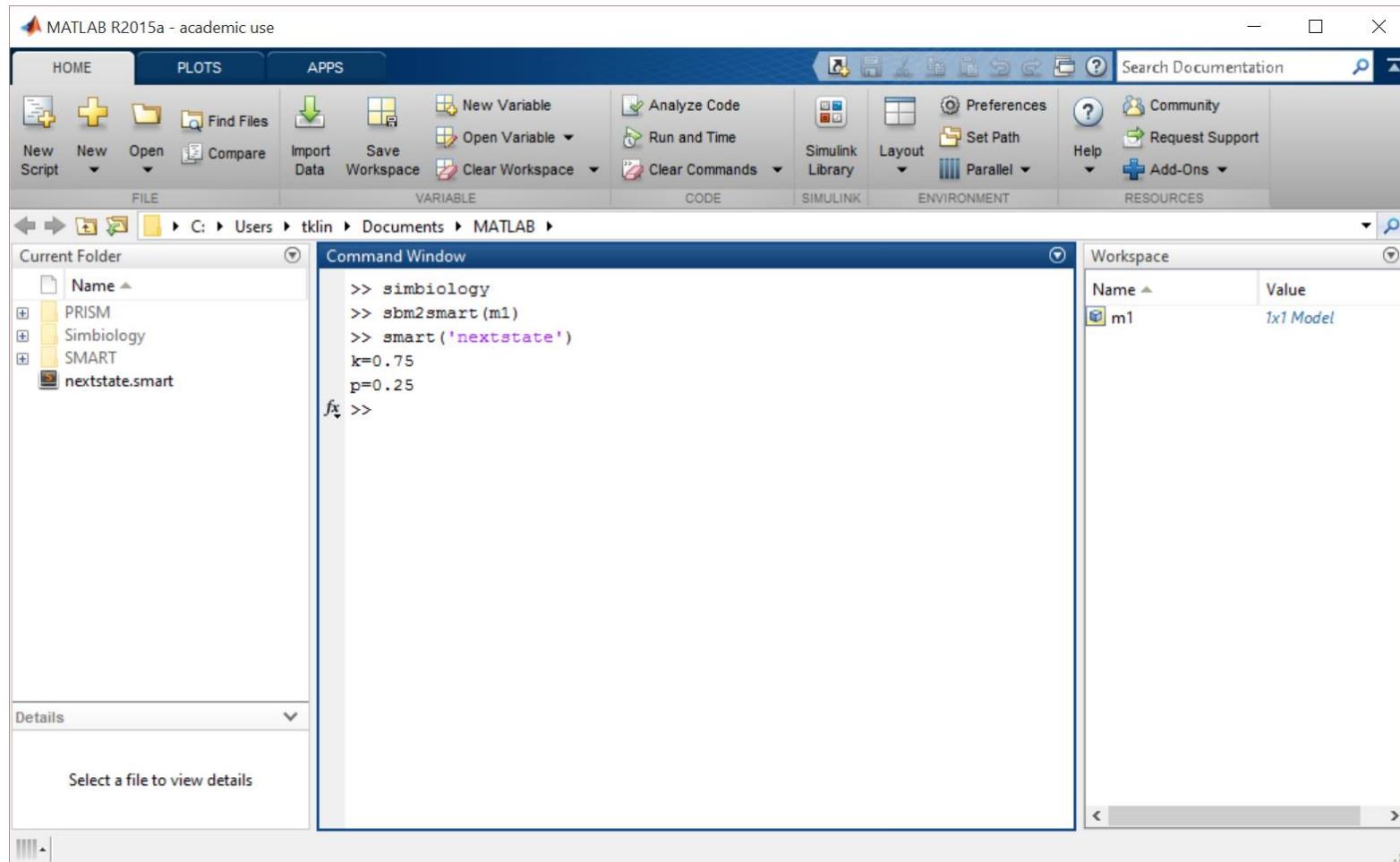
The lines from 24 to 25 are highlighted with a red rounded rectangle.

At the bottom of the editor, it says "INSERT MODE, Line 25, Column 20". On the right side, there are buttons for "Tab Size: 4" and "Plain Text".

SMART: Model Checker



SMART: Model Checker



SMART: Model Checker

- Notable functions of Petri nets in SMART
 - stateset **potential**(proc bool p)
 - proc int **tk**(place p)
 - ph real **phr_TF**(stateset p)
 - real **avg**(ph real x)
 - real **prob_at**(proc bool x, real t)
 - real **avg_at**(proc real x, real t)
 - real **prob_ss**(proc bool x)
 - real **avg_ss**(proc real x)