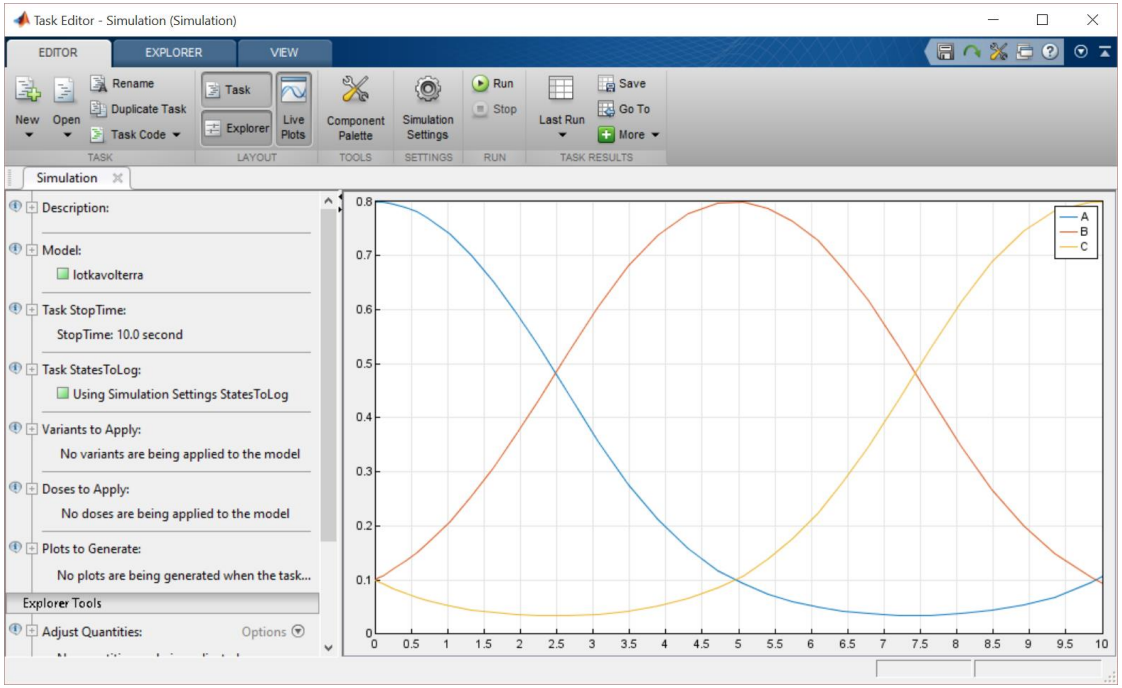
Software Tools

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



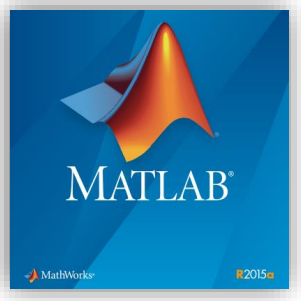
Probabilistic Model Checkers

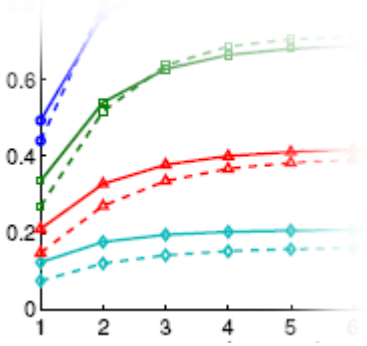
- Verifying the correctness of molecular systems

- 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

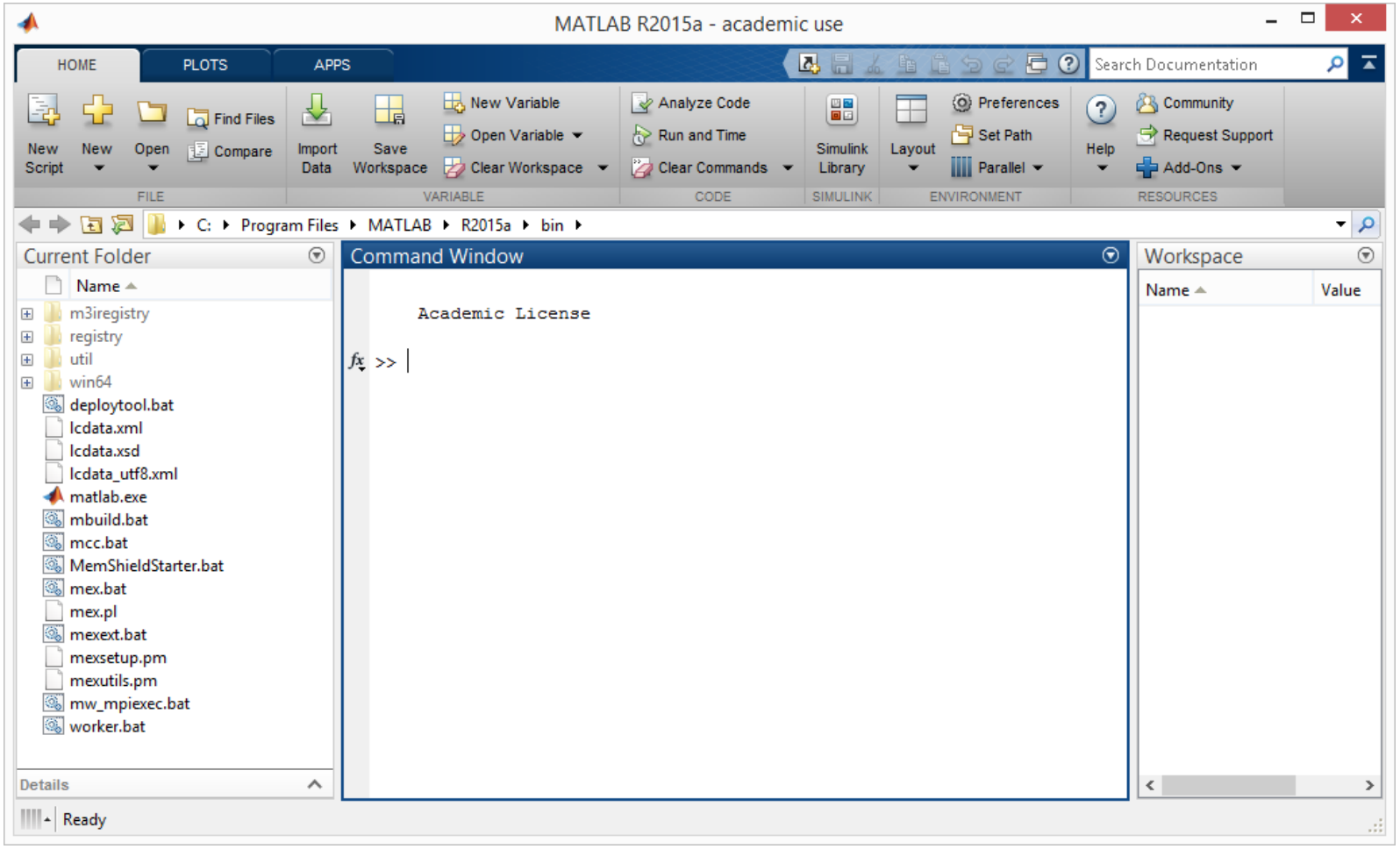
- Suppose we want a CRN to do the following:
  - Remove all the A molecules from the system

- Here's an idea:

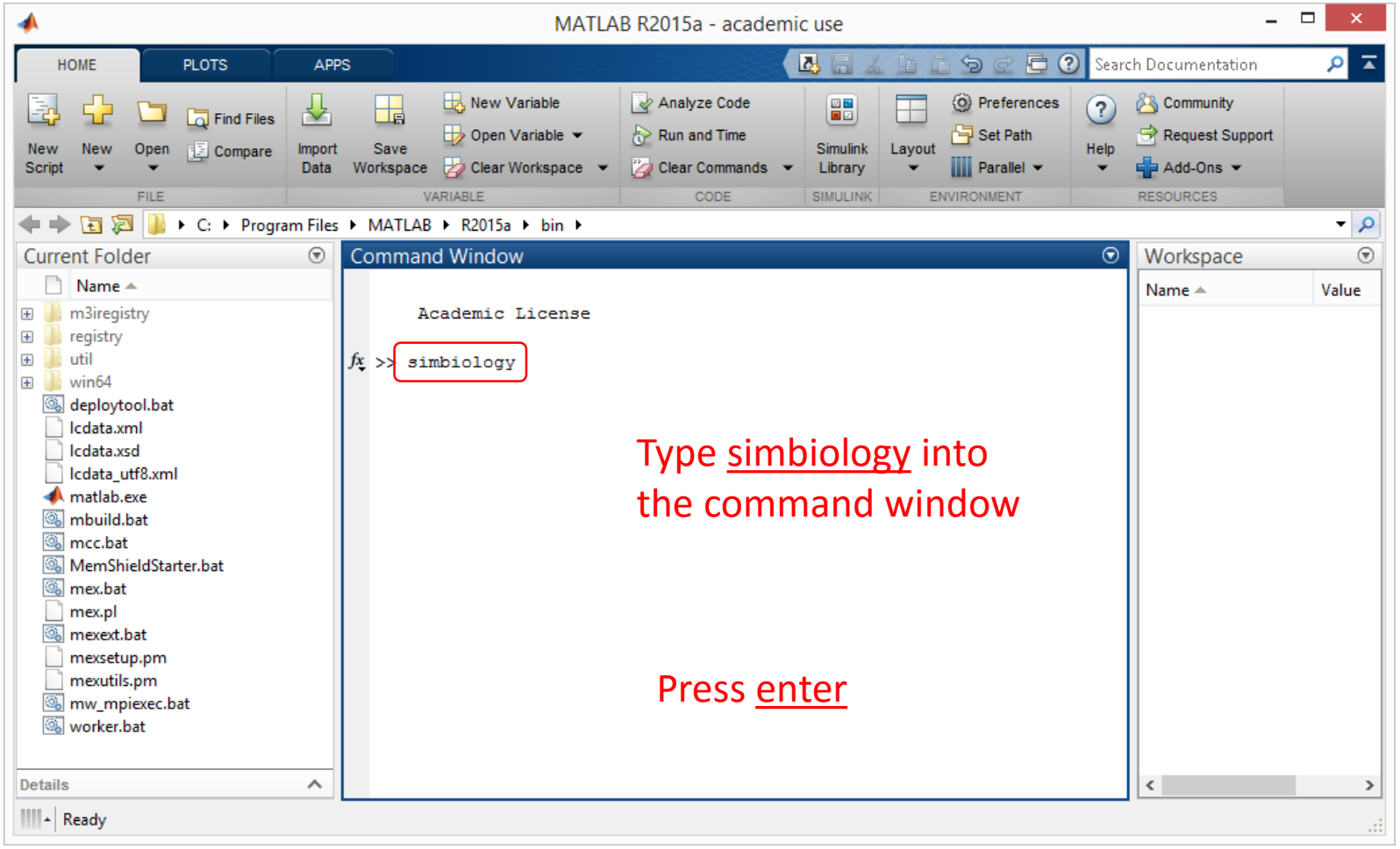
$$A \rightarrow \emptyset$$

- Want to build and simulate this CRN with MATLAB

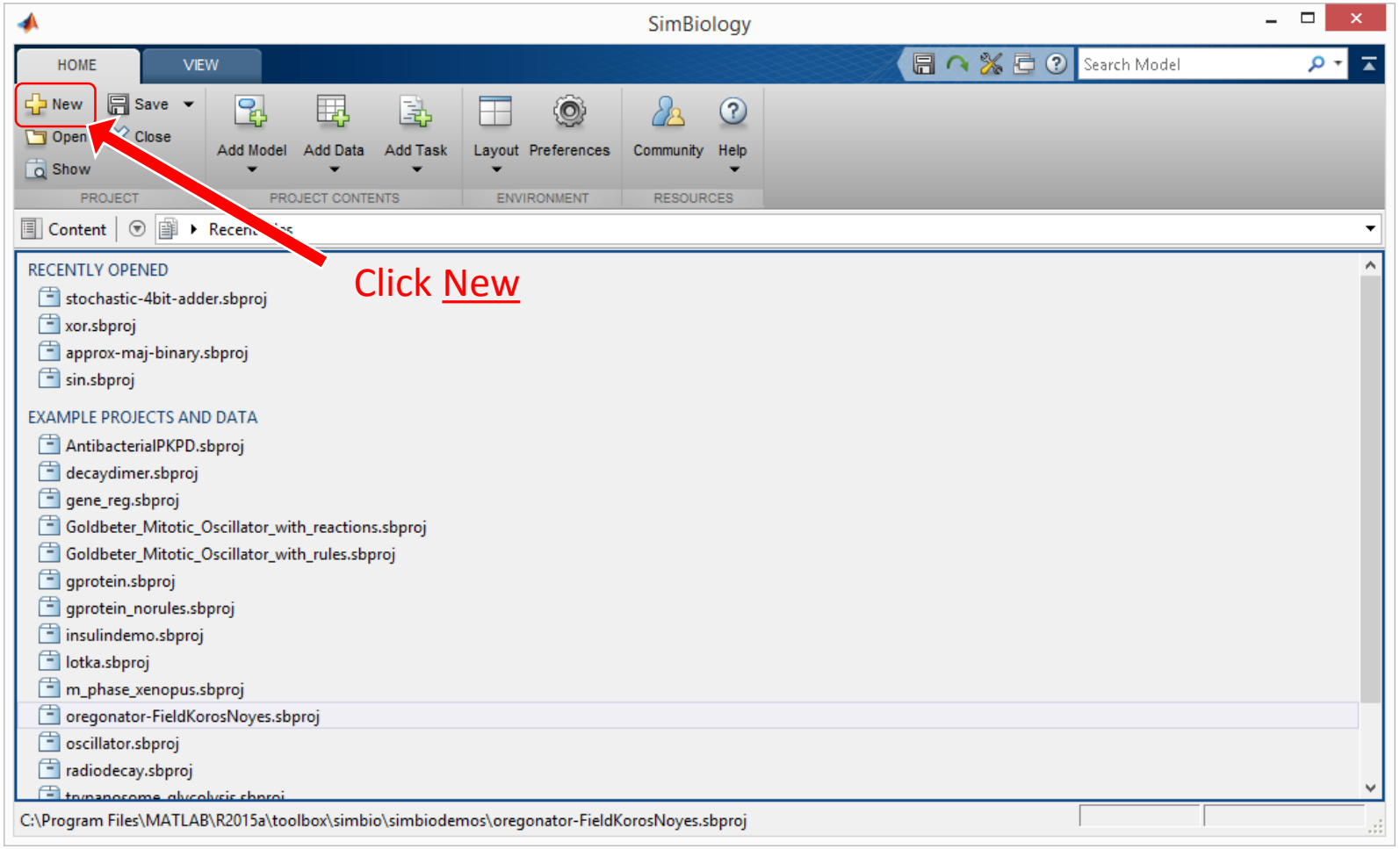
MATLAB



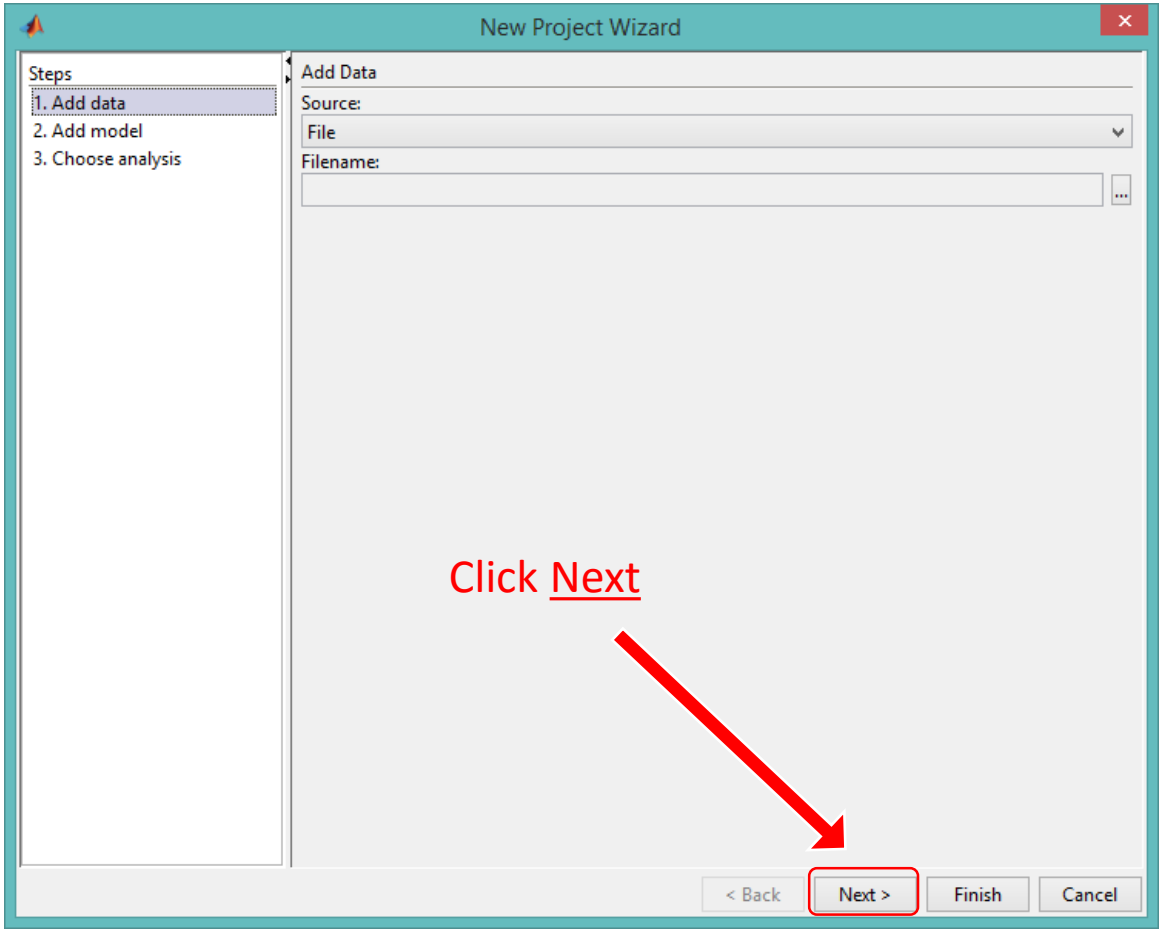
MATLAB



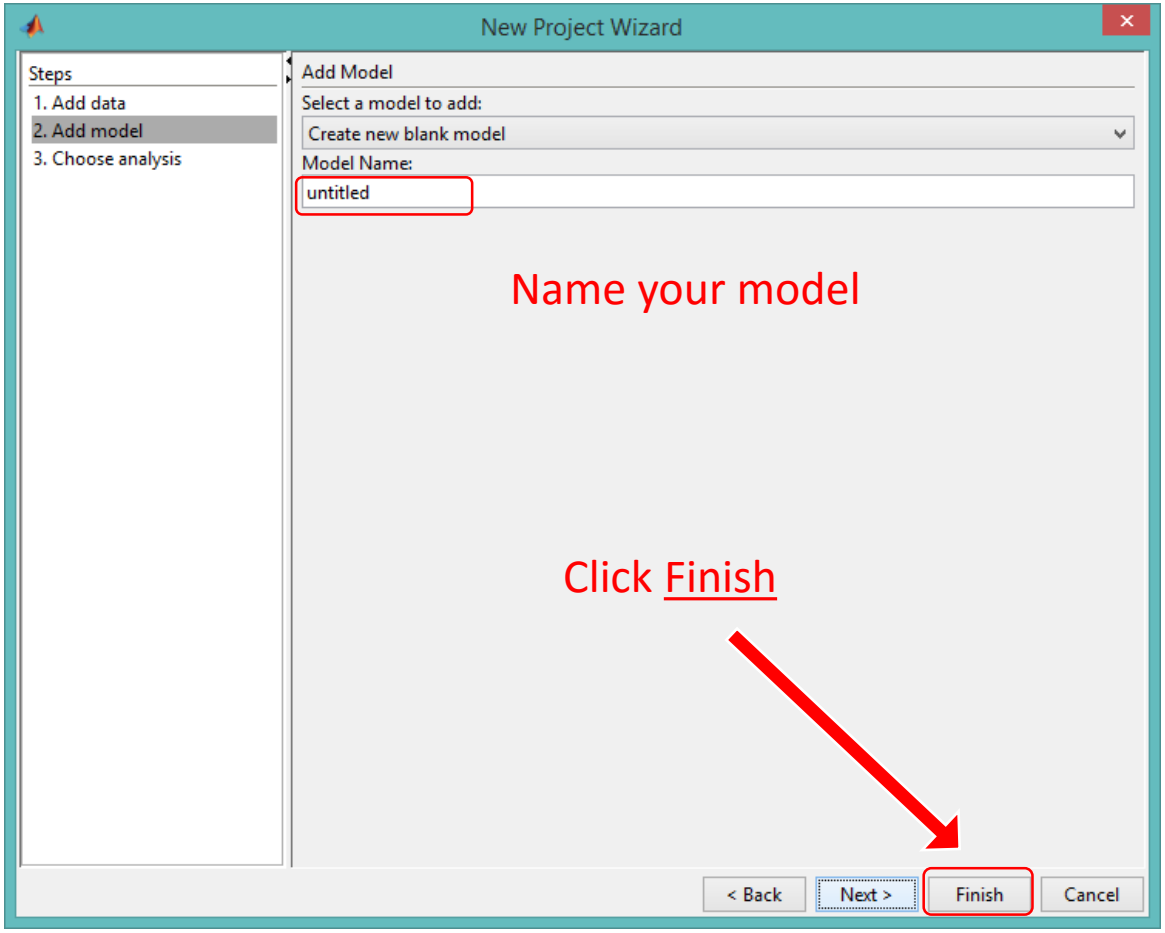
MATLAB Simbiology



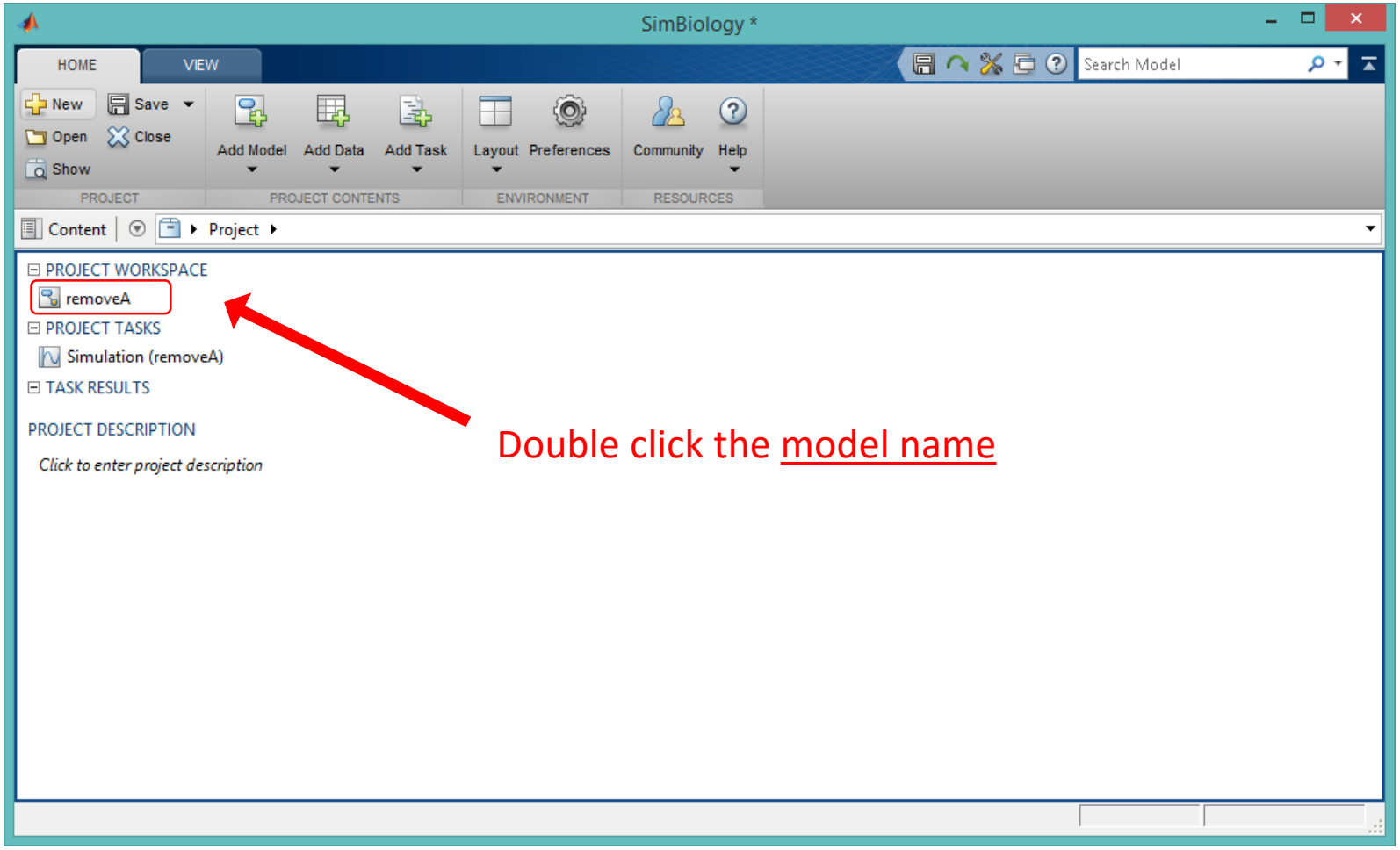
MATLAB Simbiology



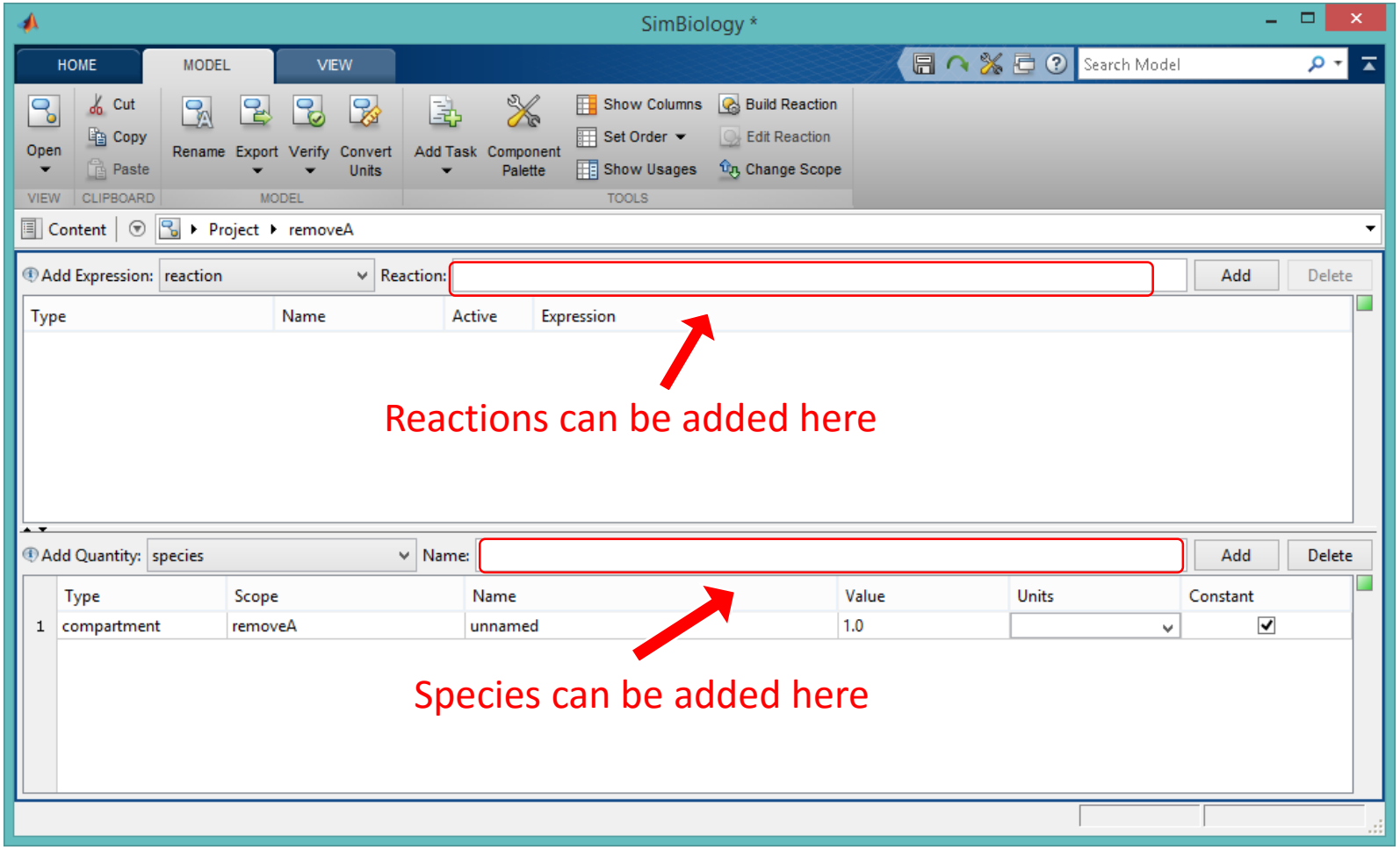
MATLAB Simbiology



MATLAB Simbiology



MATLAB Simbiology



MATLAB Simbiology

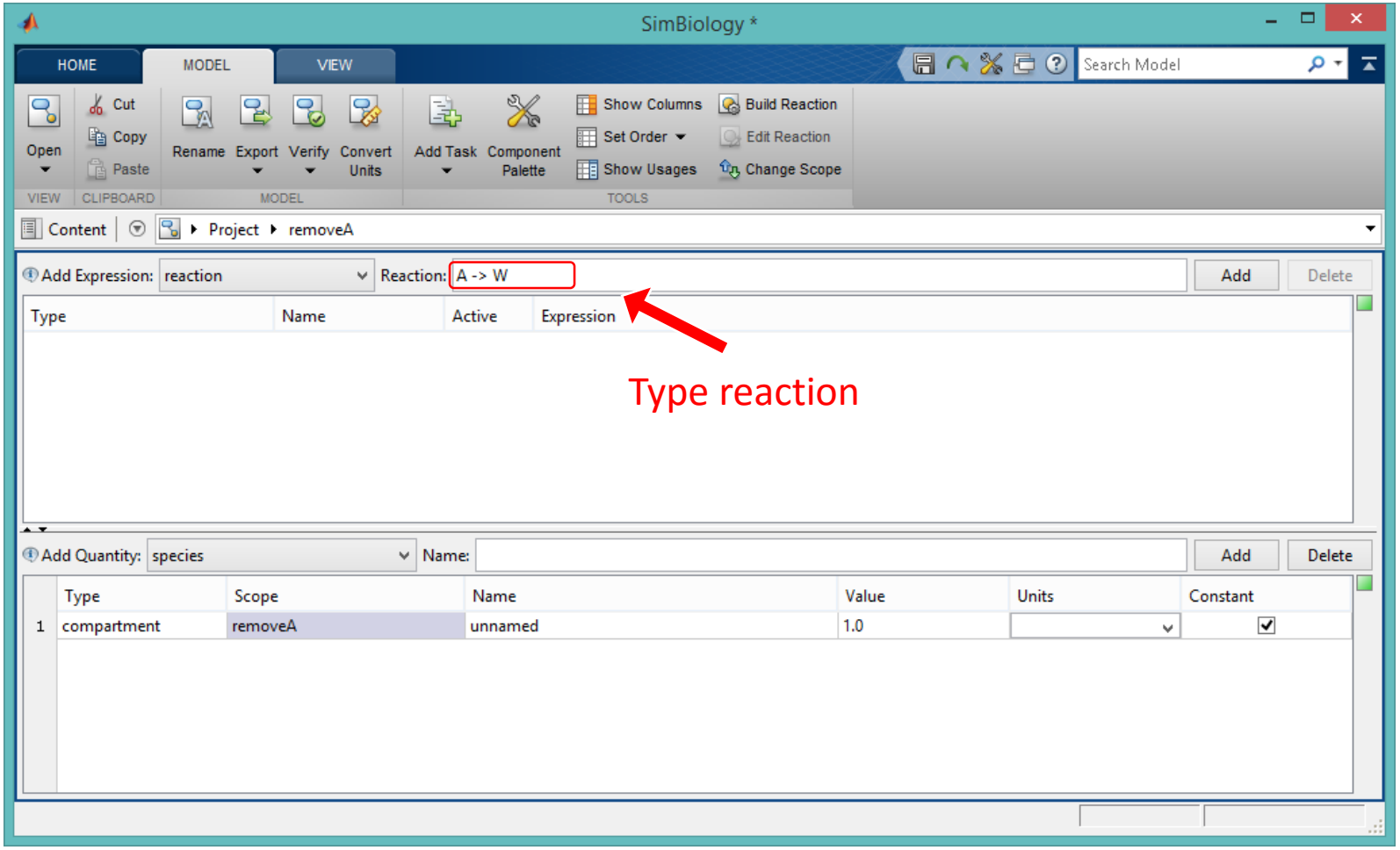
- Reactions in Simbiology have the following format

Note the arrow is two characters!

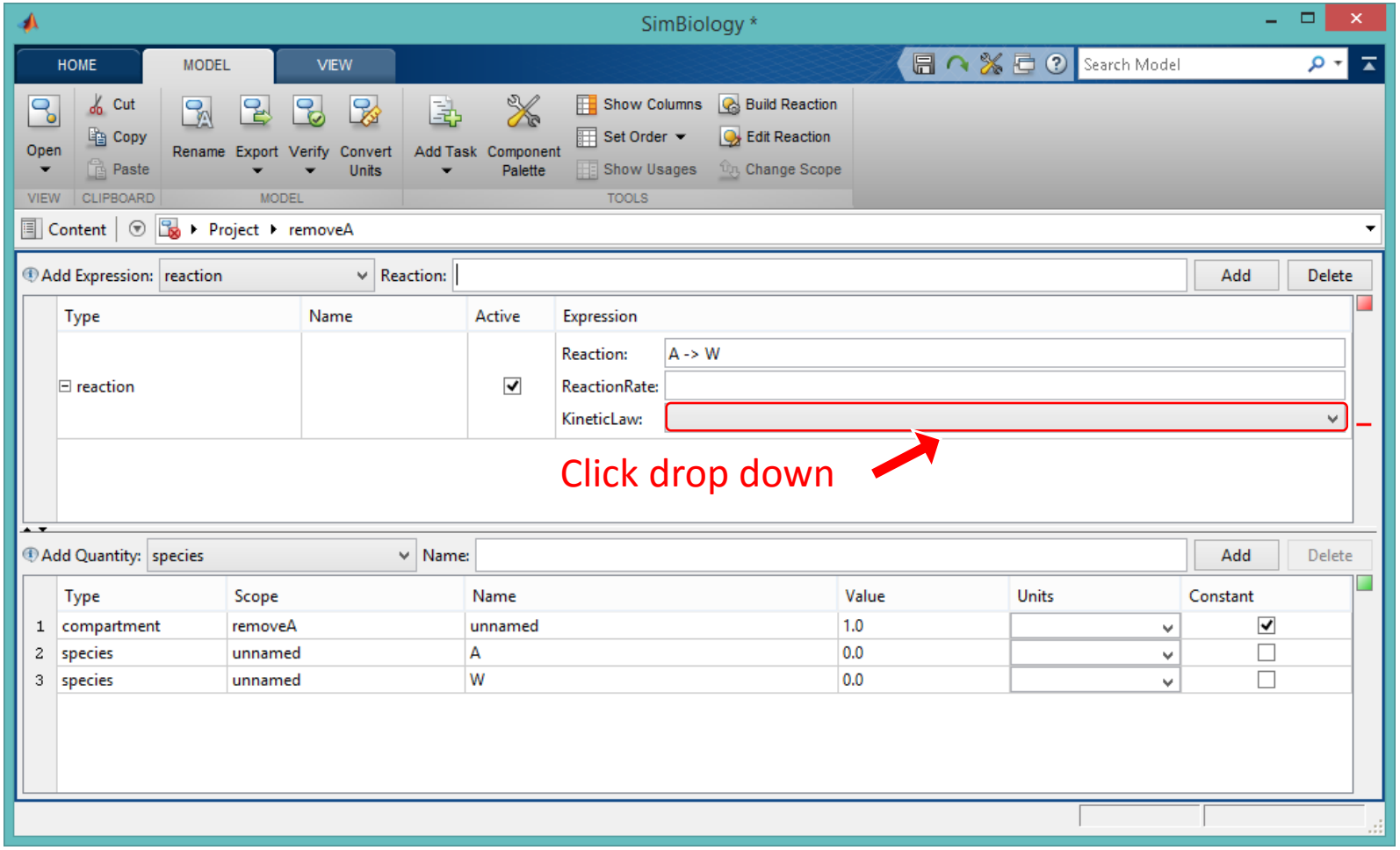
2 A + B -> 3 C

Note the spaces!

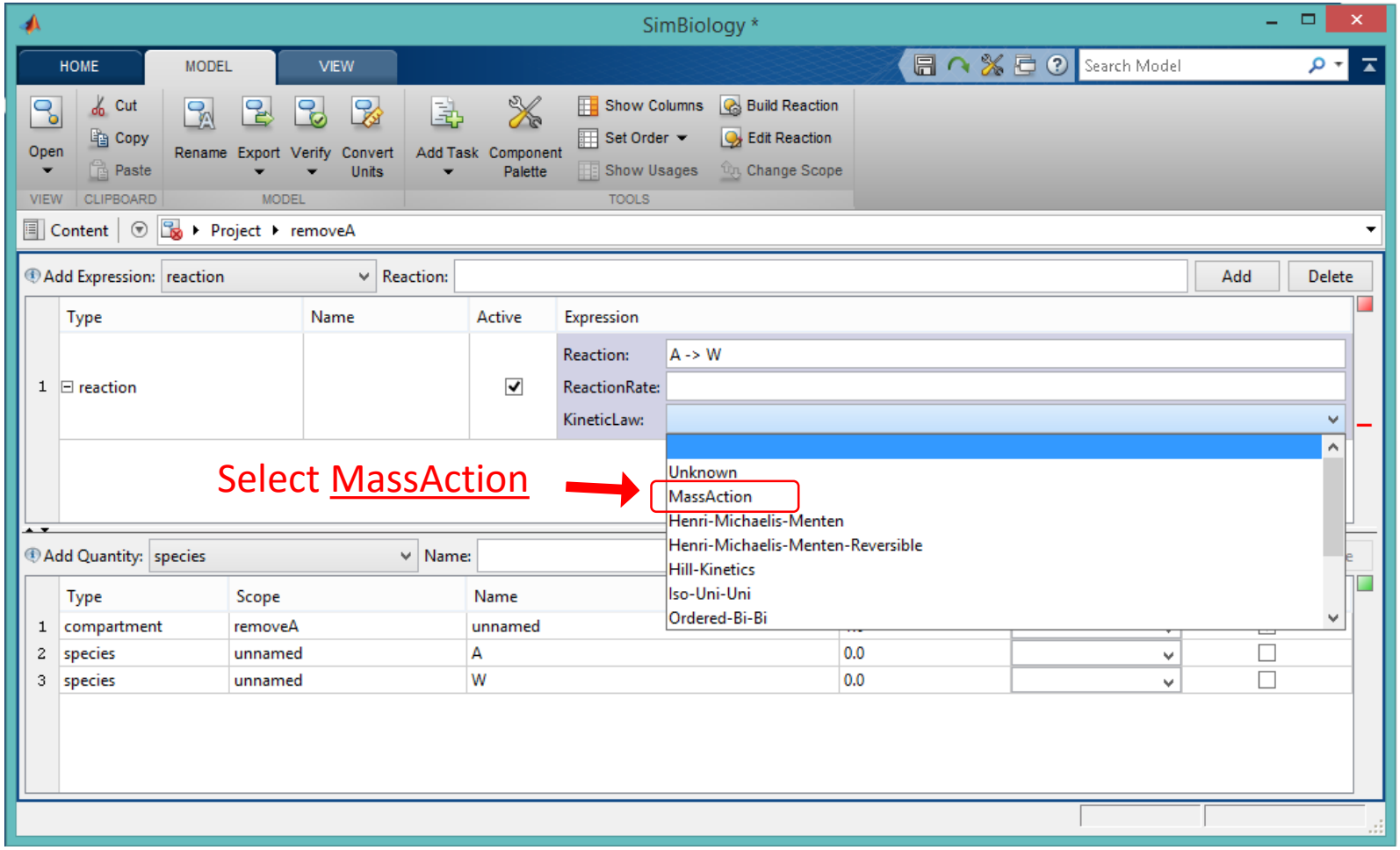
MATLAB Simbiology



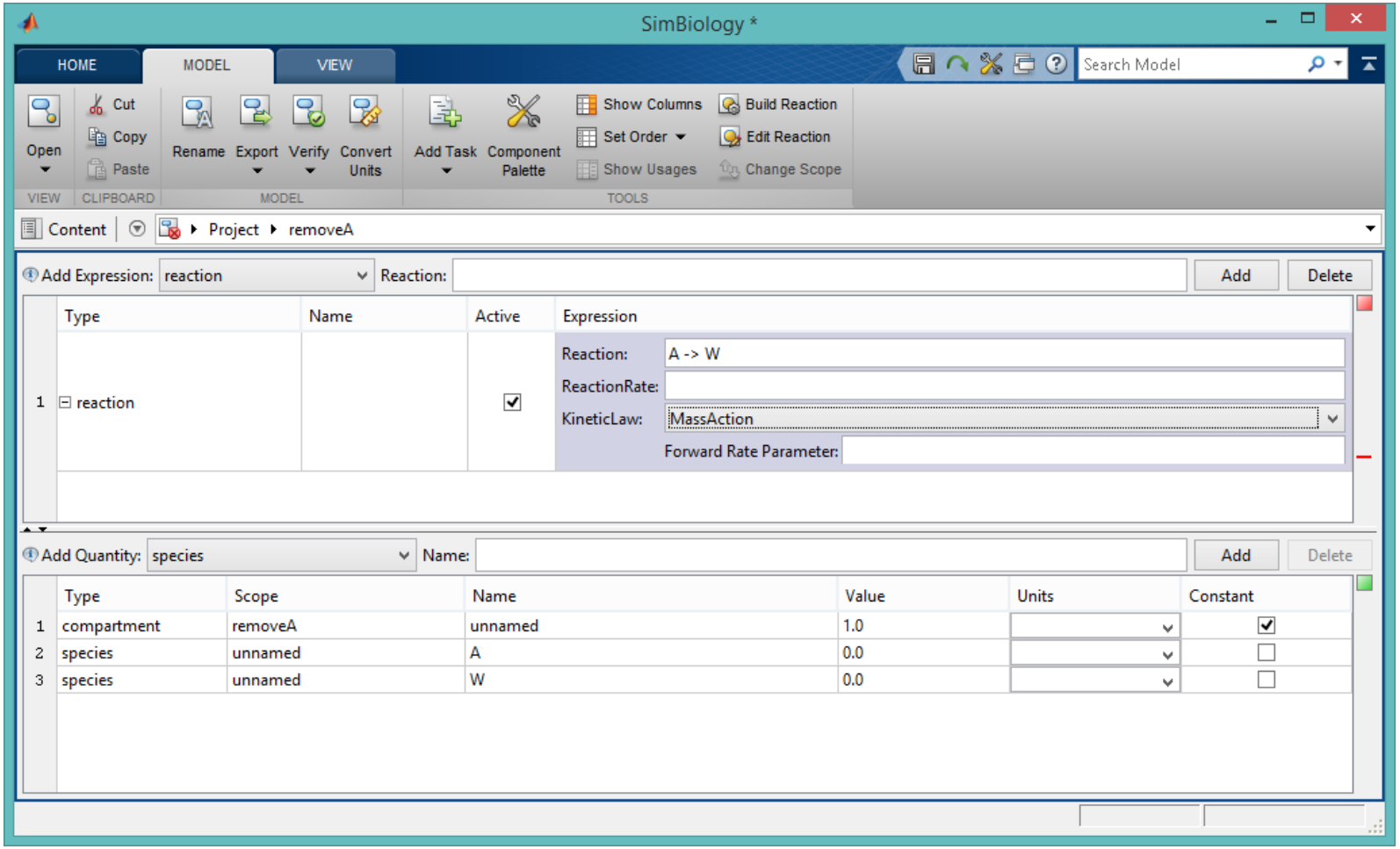
MATLAB Simbiology



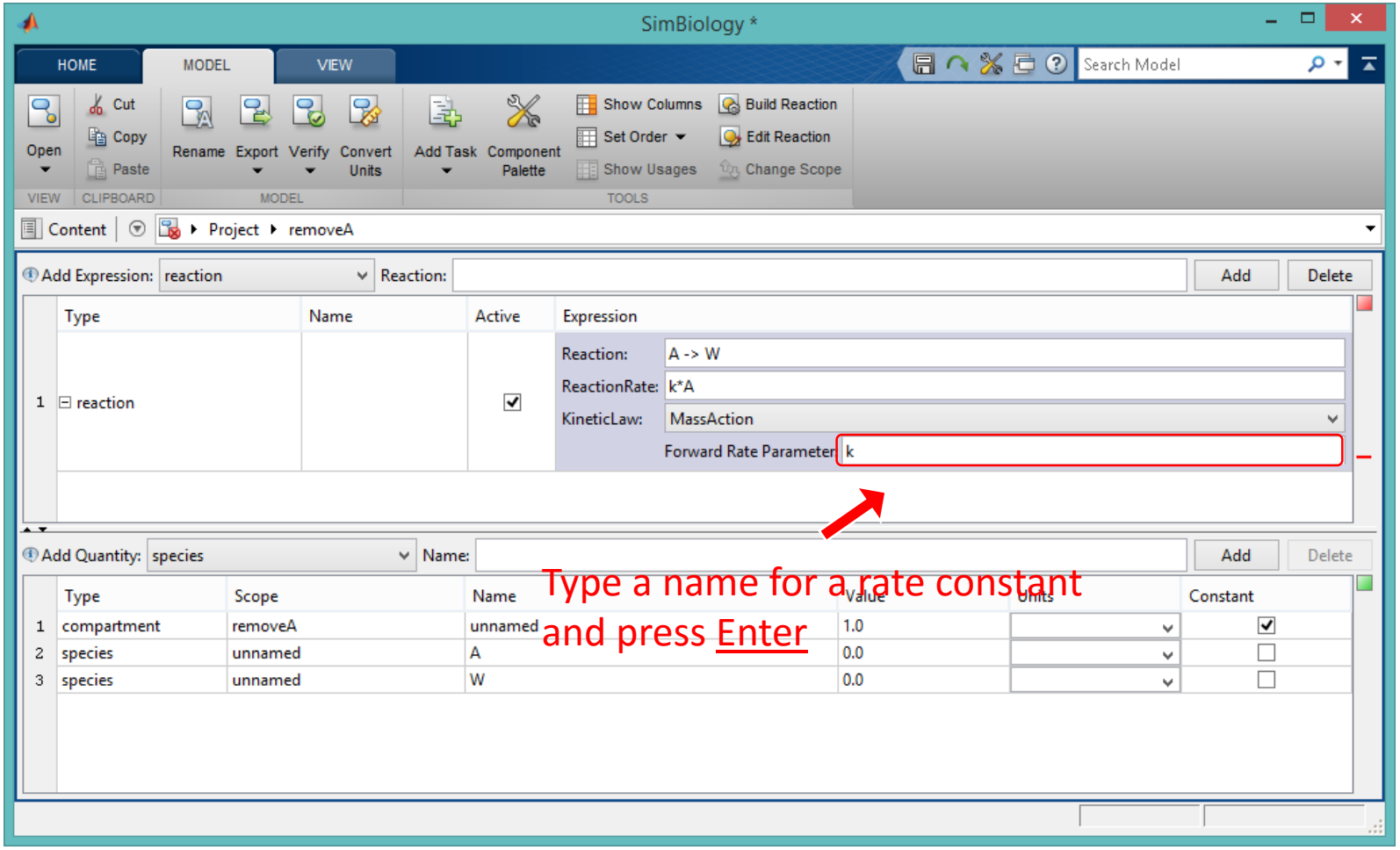
MATLAB Simbiology



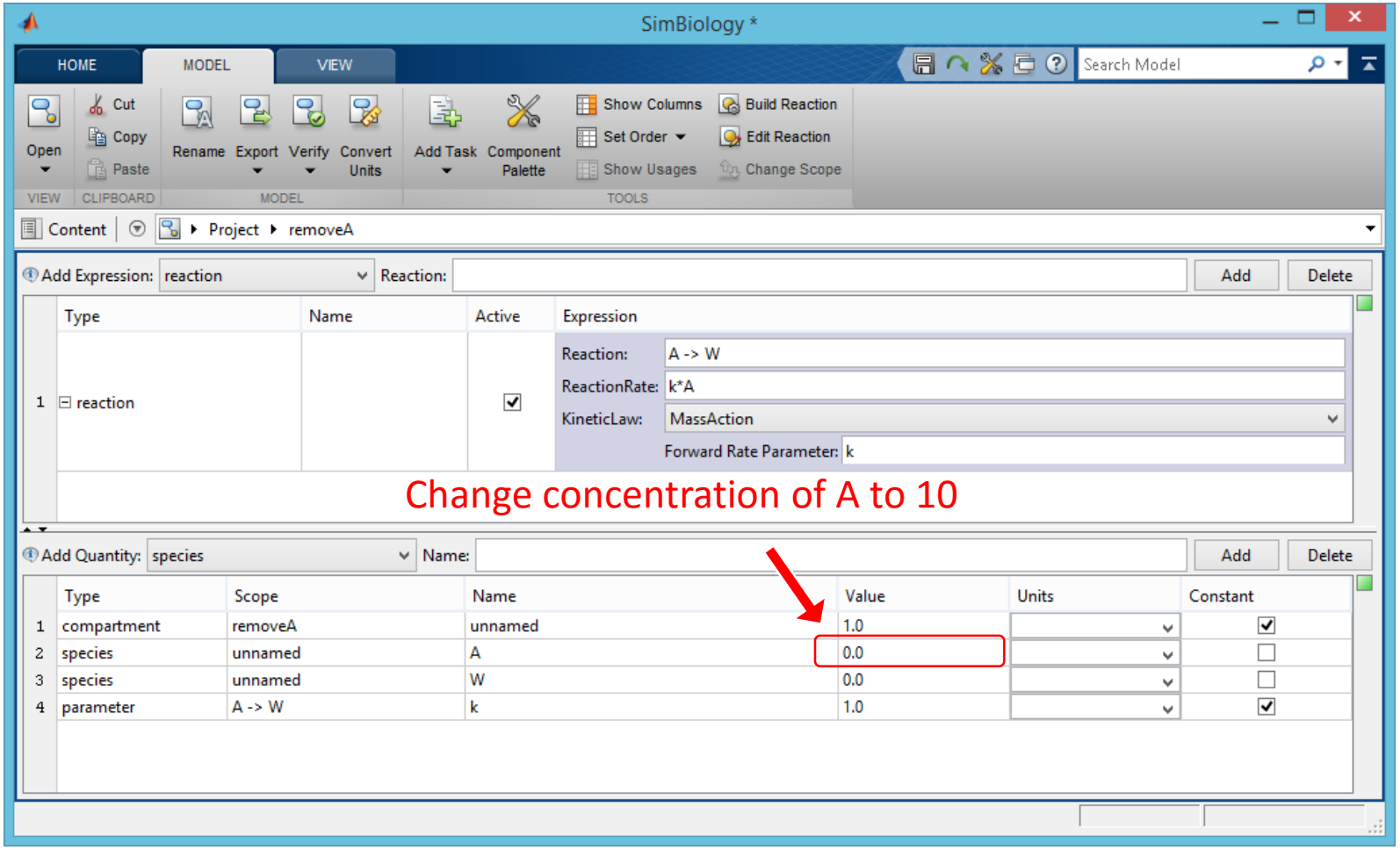
MATLAB Simbiology



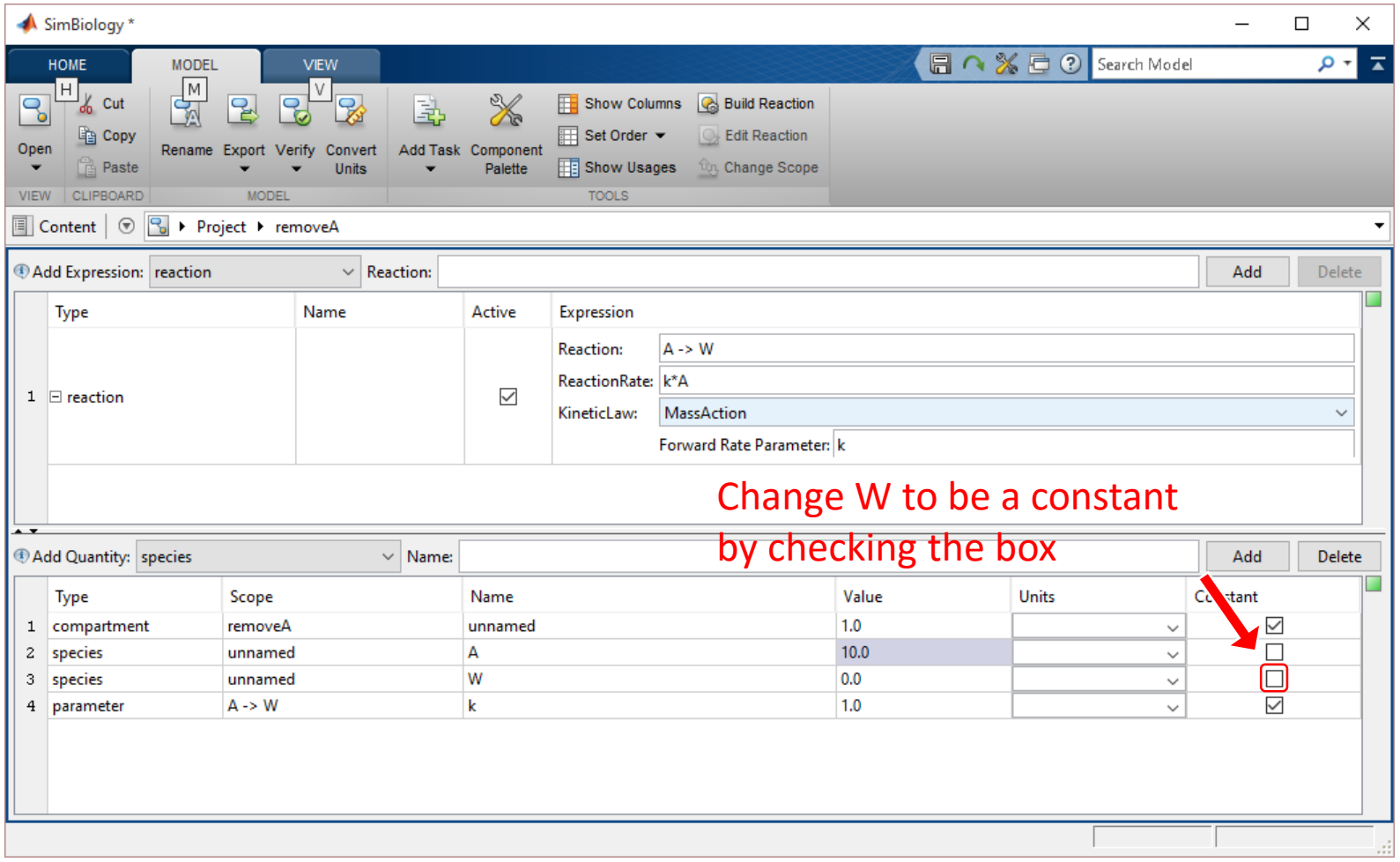
MATLAB Simbiology



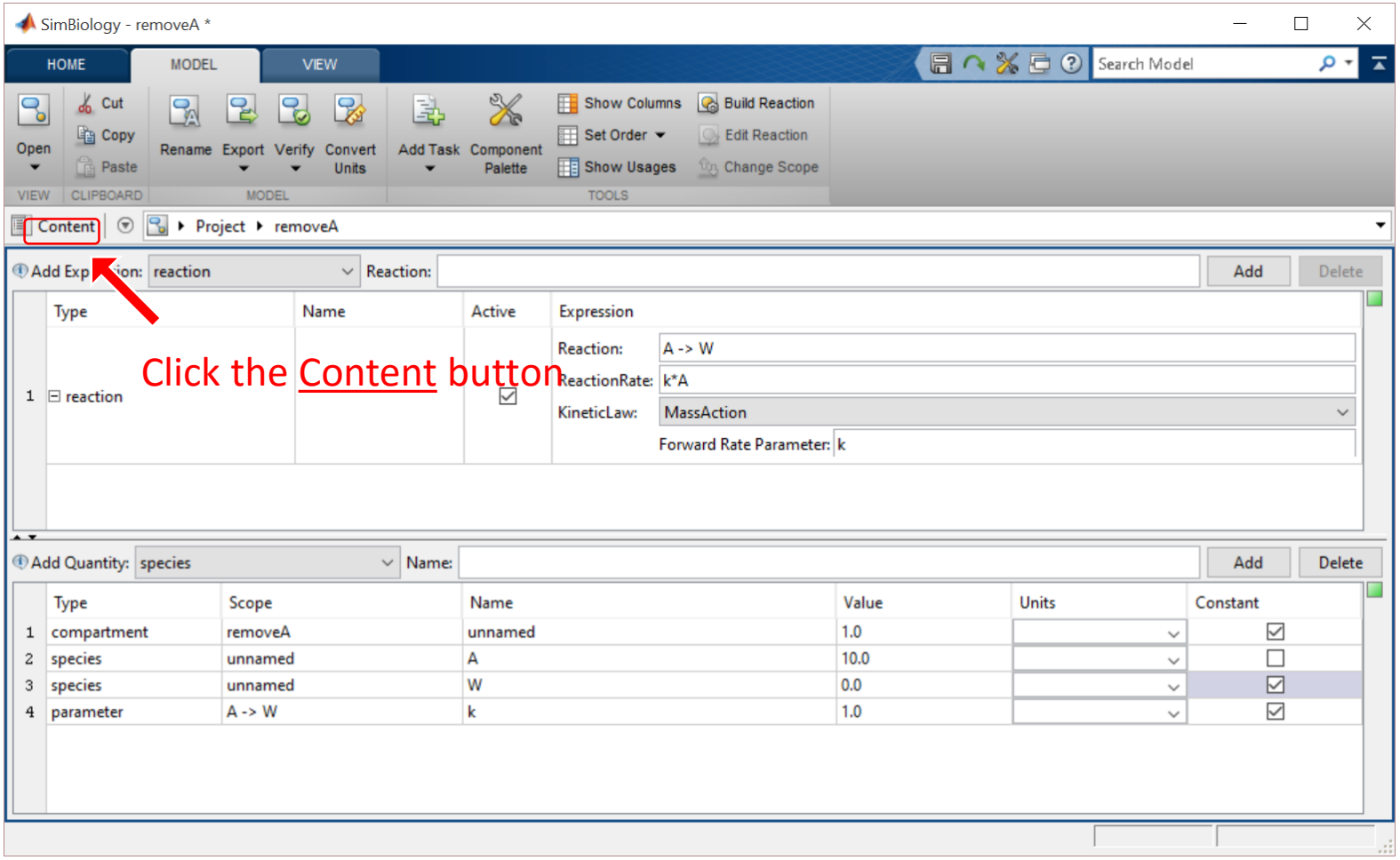
MATLAB Simbiology



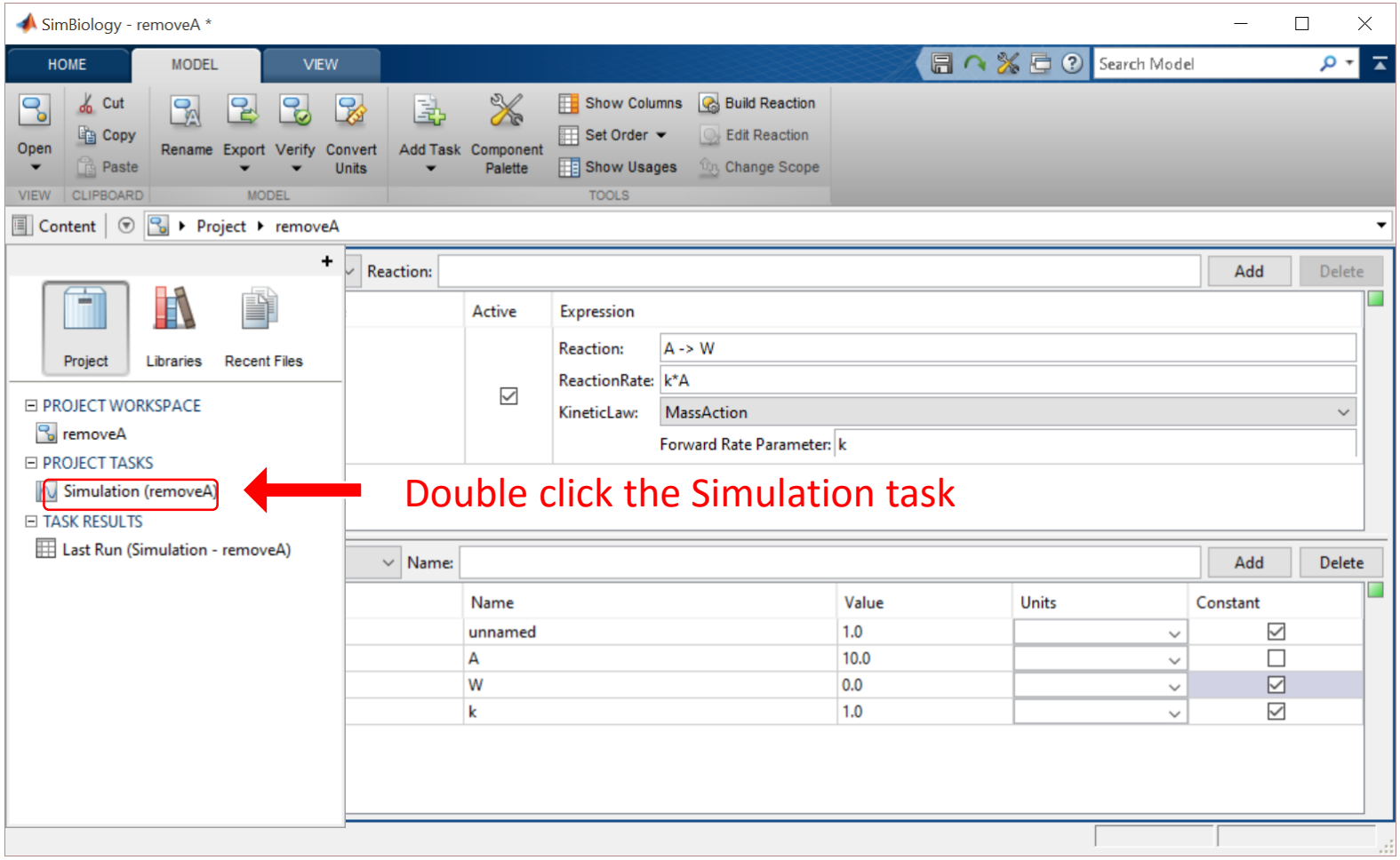
MATLAB Simbiology



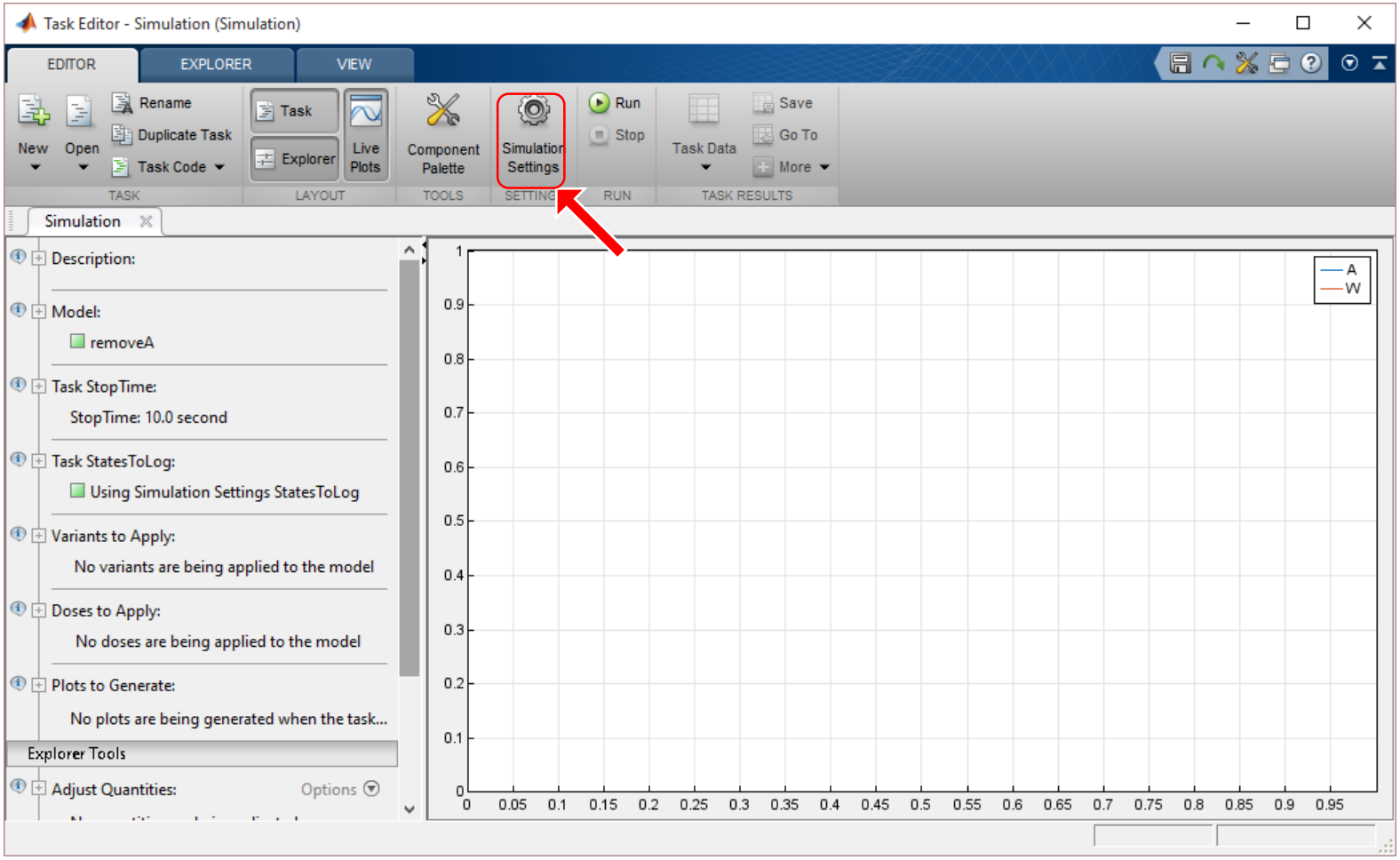
MATLAB Simbiology



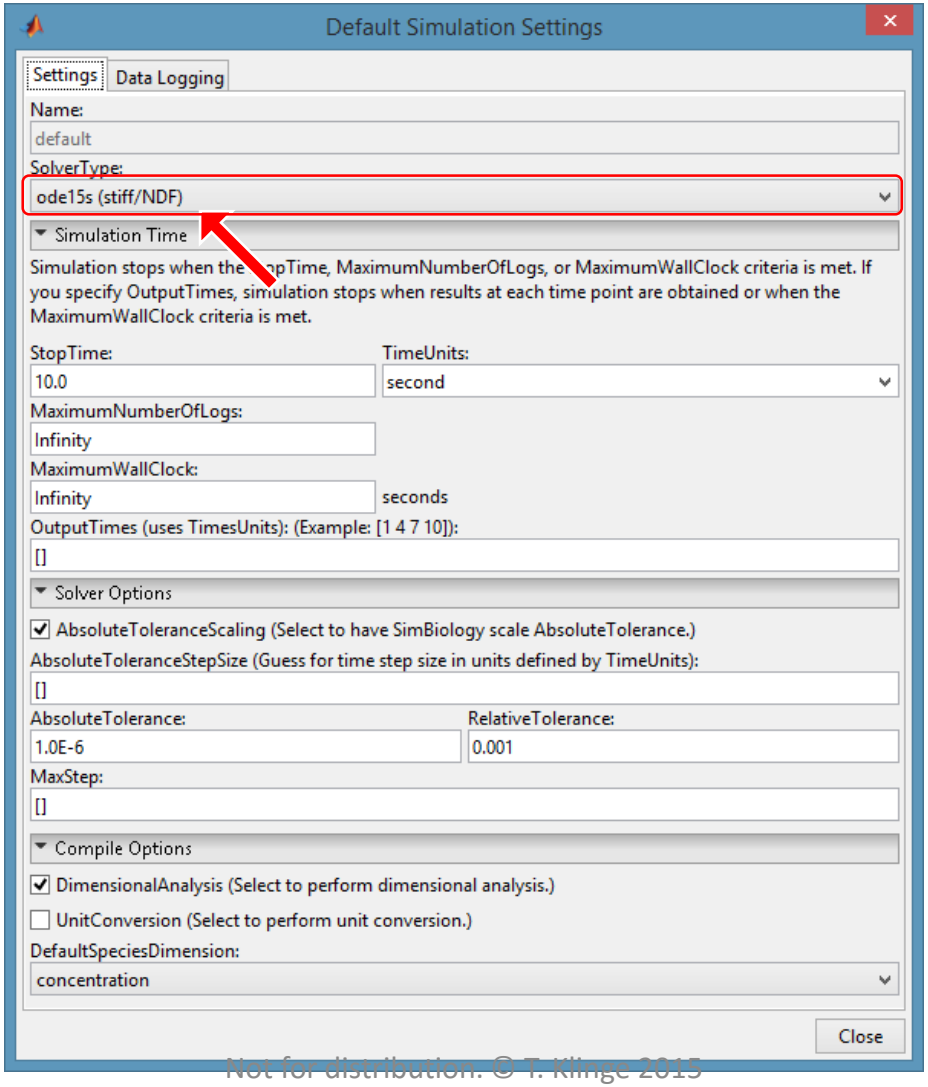
MATLAB Simbiology



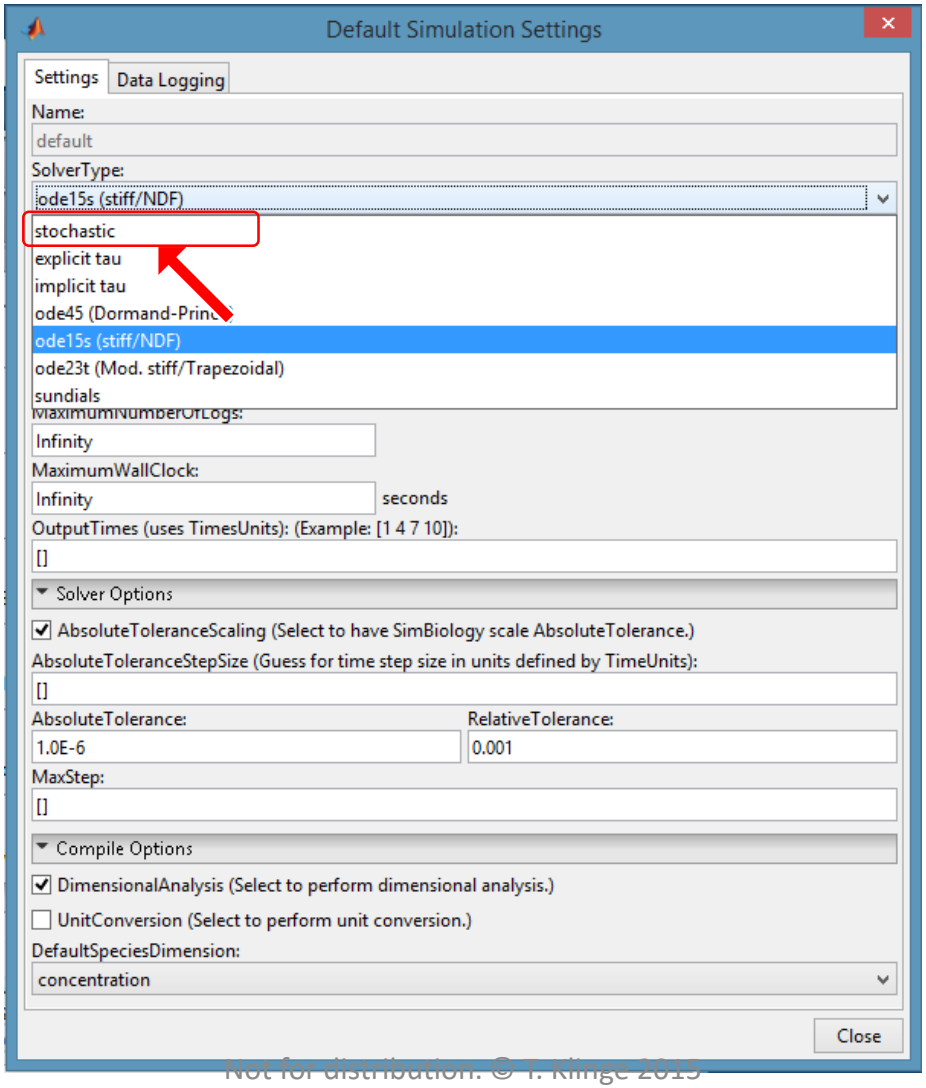
MATLAB Simbiology



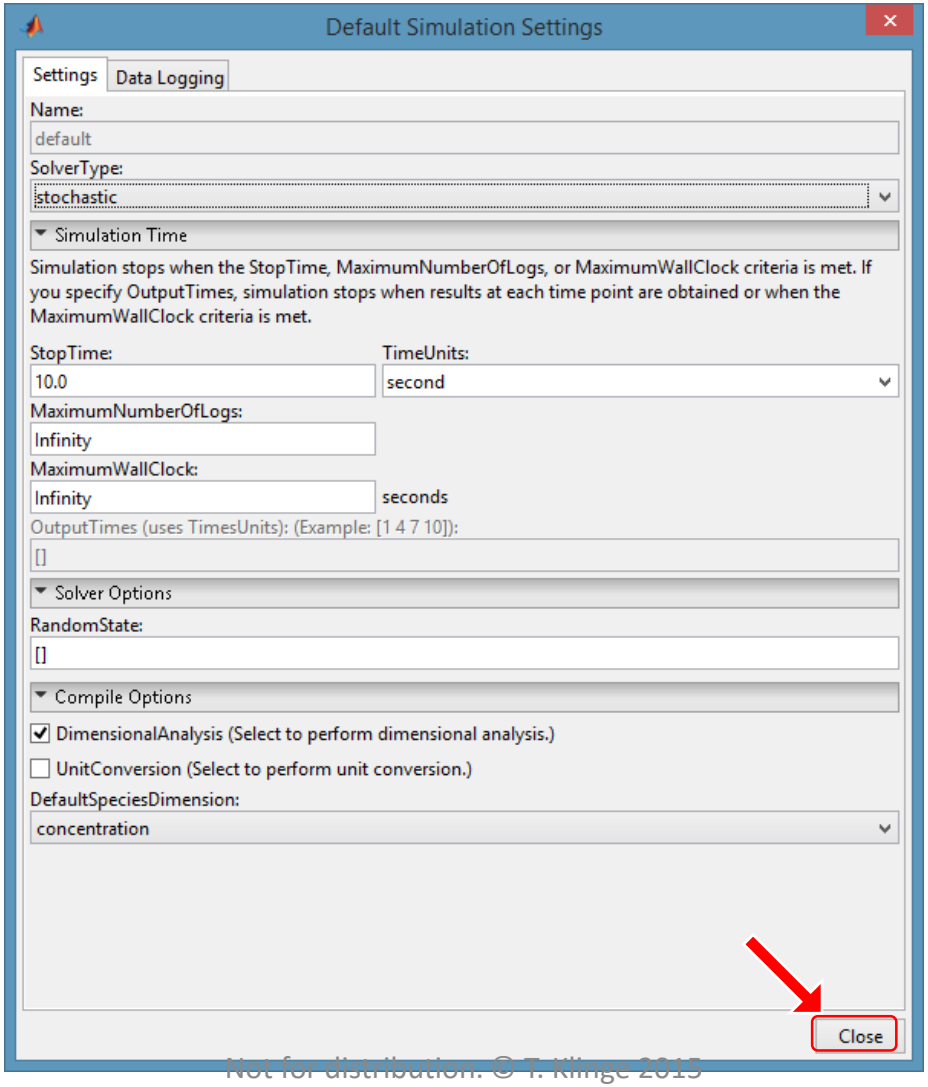
MATLAB Simbiology



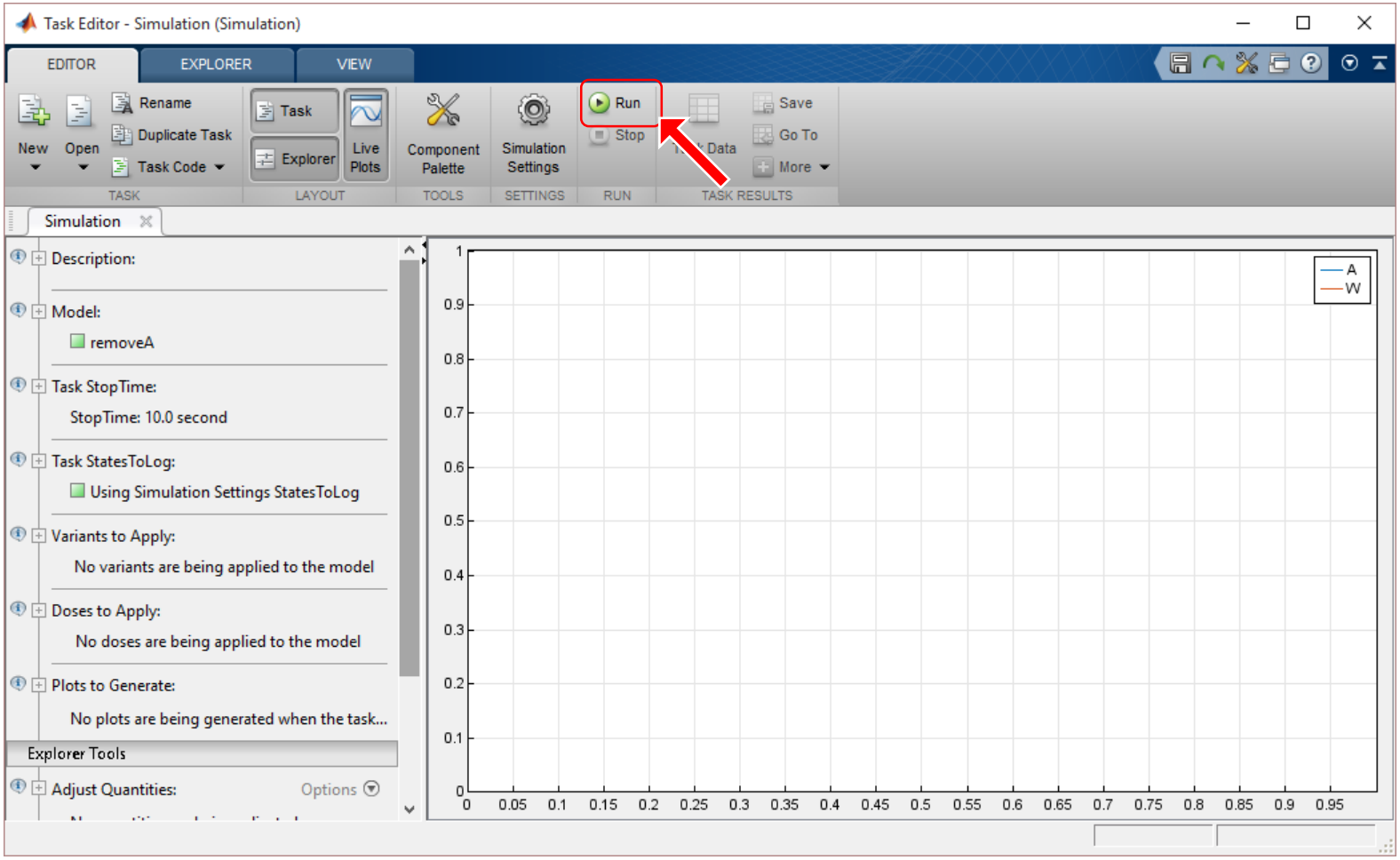
MATLAB Simbiology



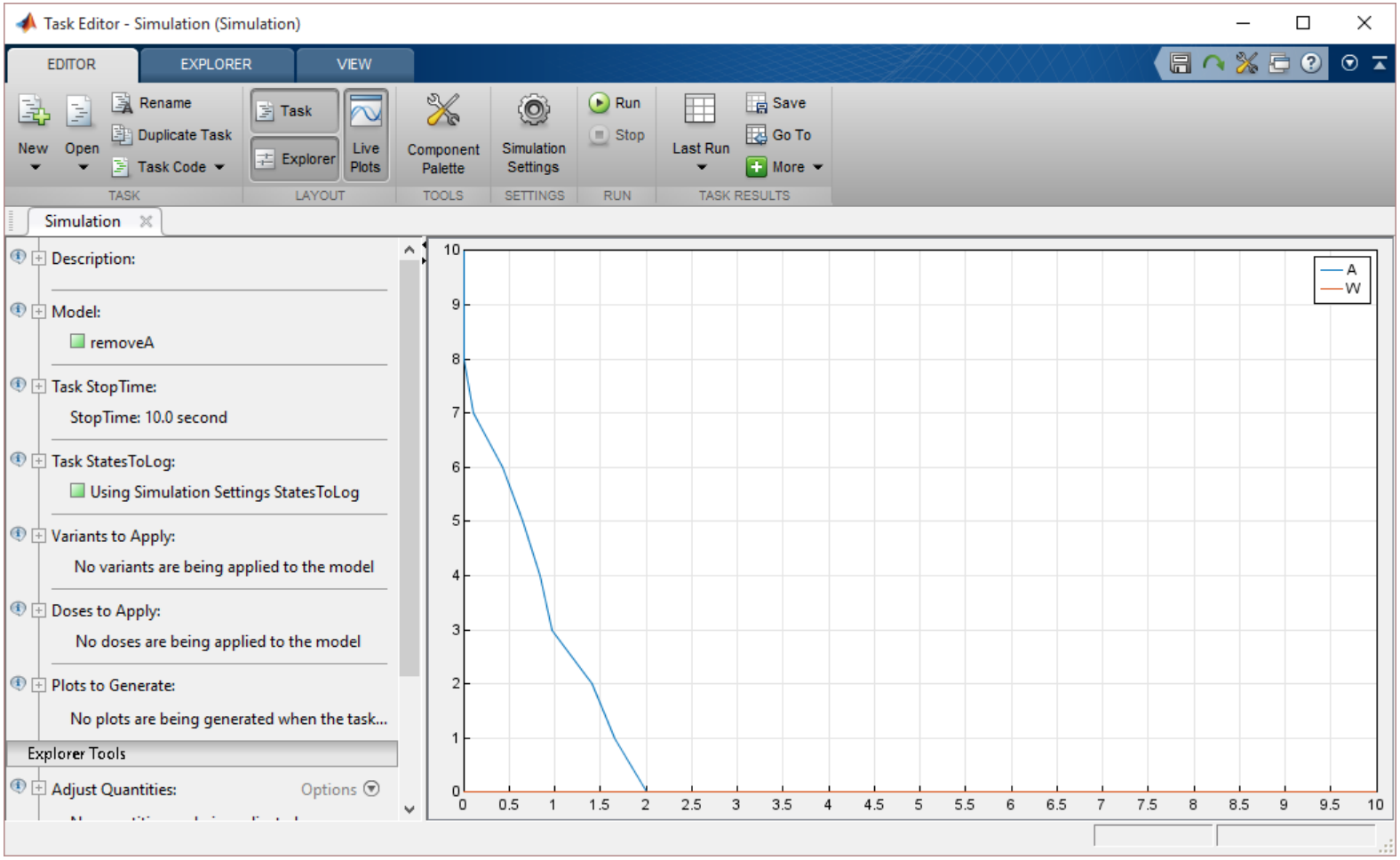
MATLAB Simbiology



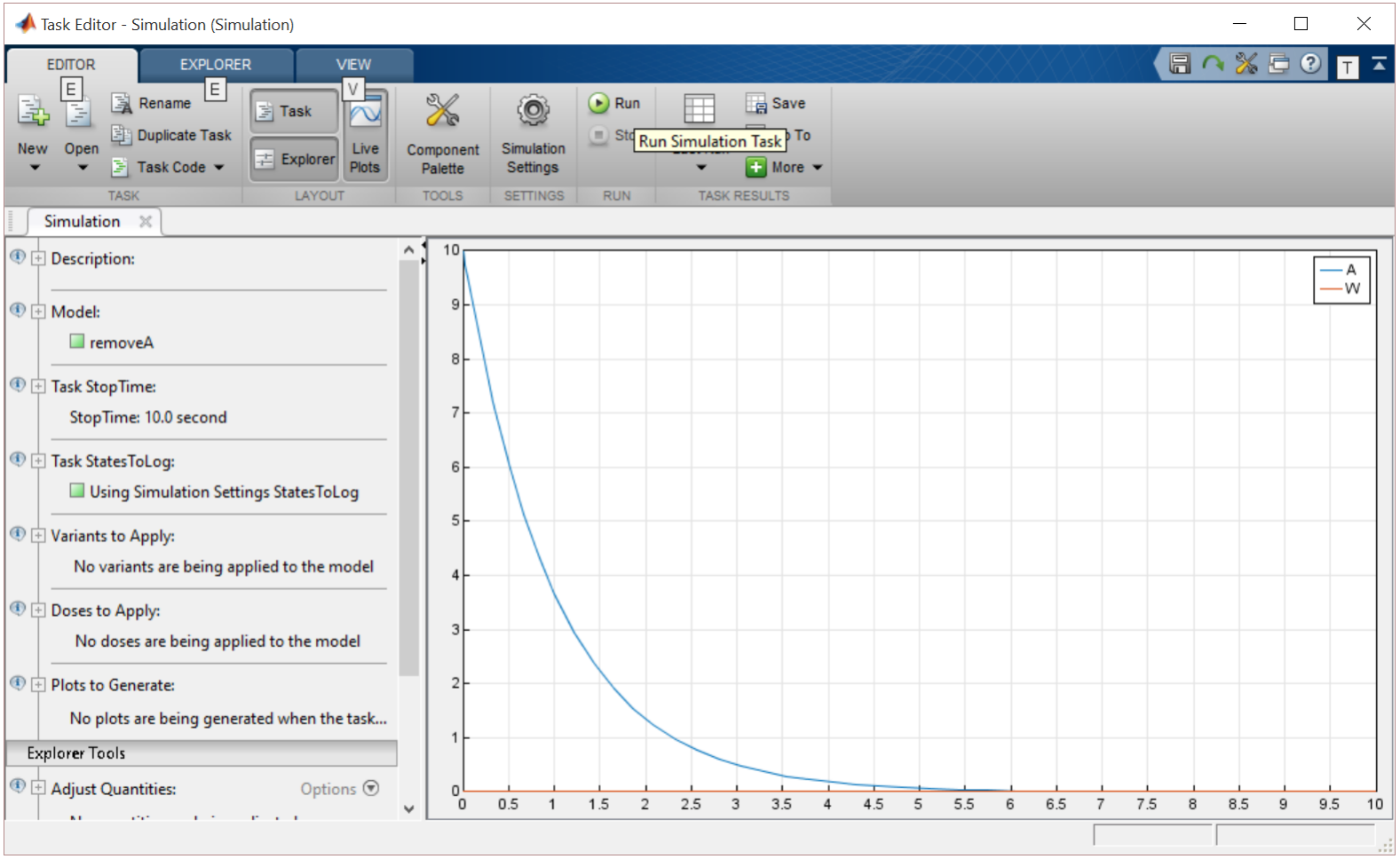
MATLAB Simbiology



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: $X = n, \quad S_1 = 1, \quad S_2 = S_3 = 0$

  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: $X = n, \quad S_1 = 1, \quad S_2 = S_3 = 0$

  With $n \in \{0,1\}$

- Output: Fix $\epsilon > 0$
  - If $n = 0,$ output $S_1 = S_2 = 0, \quad S_3 = 1$
  - If $n = 1,$ output $S_1 = S_3 = 0 \quad S_2 = 1$ 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

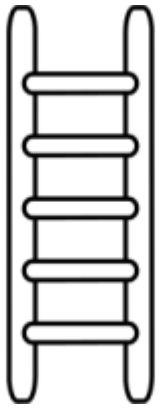
- Input: n molecules of X
- Output: $2n$ molecules of X with $p \geq 1 - \epsilon$
  - Want sequential behavior like:

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

$$X + L_i \rightarrow Xp + L_0 \quad \text{for all } i \in \{0 \ldots k\}$$
$$L_i + Y \rightarrow L_{i+1} + Y \quad \text{for all } i \in \{0 \ldots k-1\}$$
$$L_k + Y \rightarrow Z$$



Afternoon Break

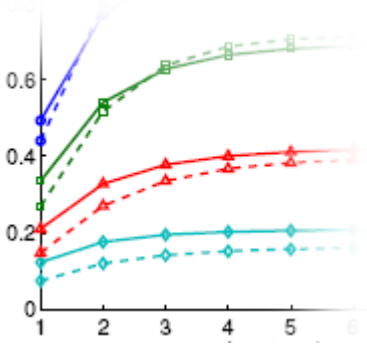
Software Tools

Molecular Programming tutorial
ASE 205

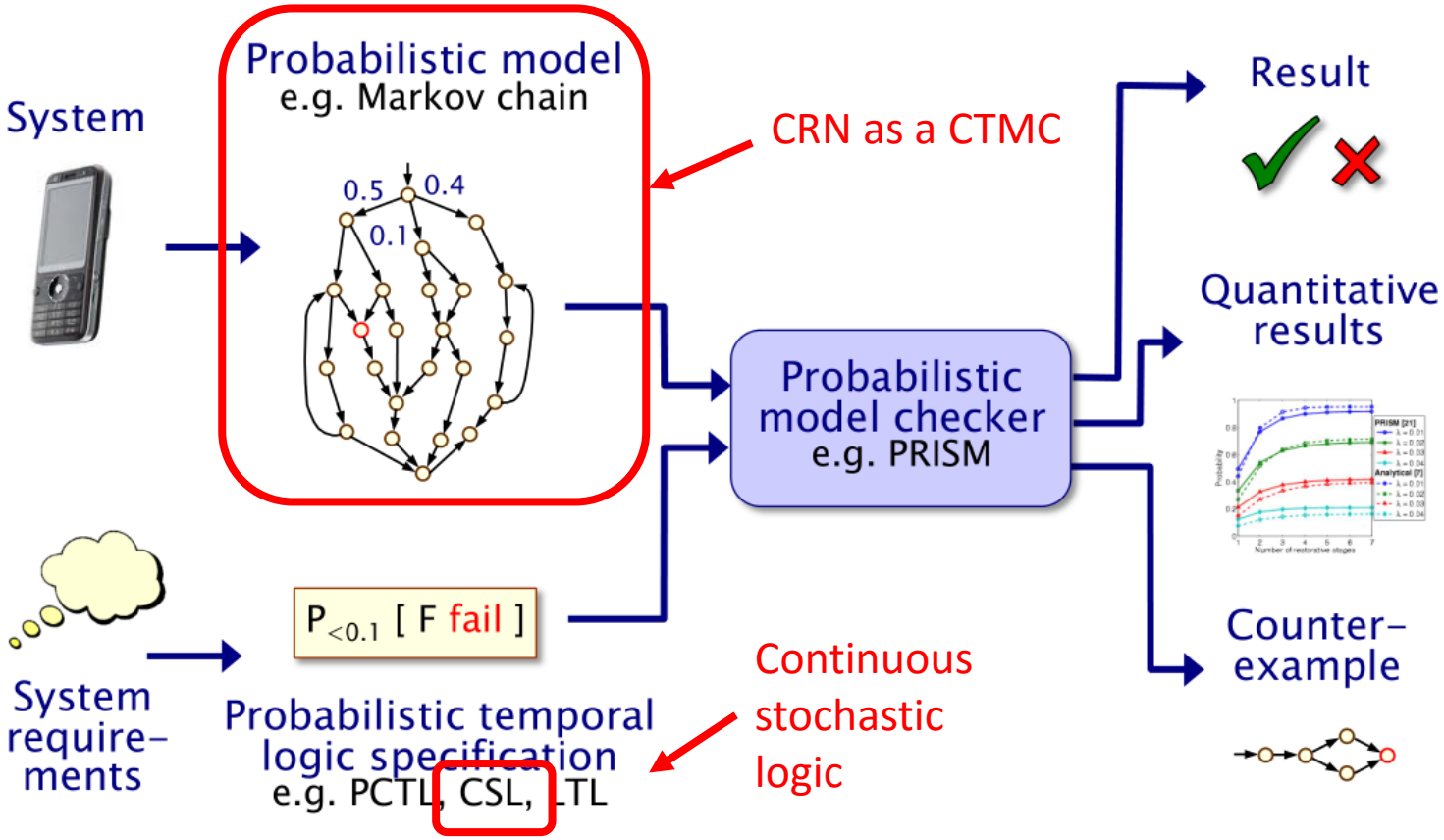
Jack Lutz, Robyn Lutz, James Lathrop, Titus Klinge

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)



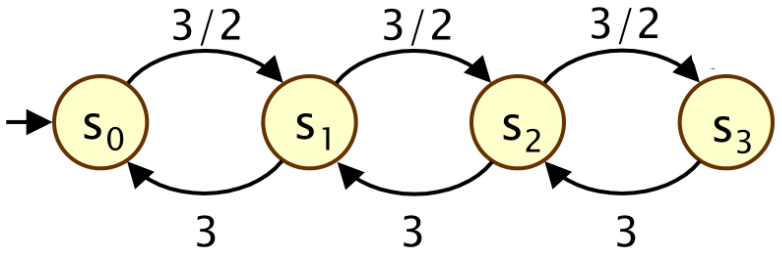
Probabilistic Model Checking



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

Probability of a path formula being satisfied

Steady state

- State formula

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

- 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 $\Box\phi$)
  - 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

$$\boldsymbol{\mathcal{P}}_{\geq 1}\Box\left[Th_H \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_4}\Diamond_{\leq w_{th}}\left(A_{trip} \vee \neg Th_H\right)\right]$$

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

PRISM - Probabilistic Syı

prismmodelchecker.org

www.prismmodelchecker.org

Search

Home • About • Downloads • Documentation • Manual • Tutorial • Publications • Case Studies • Support • Developers • 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.

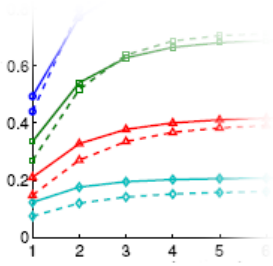
July 2015: A beta release of PRISM version 4.3 is now available. New features include external LTL-to-automata converters and various new model checking algorithms. Details here.

PRISM 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-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 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)

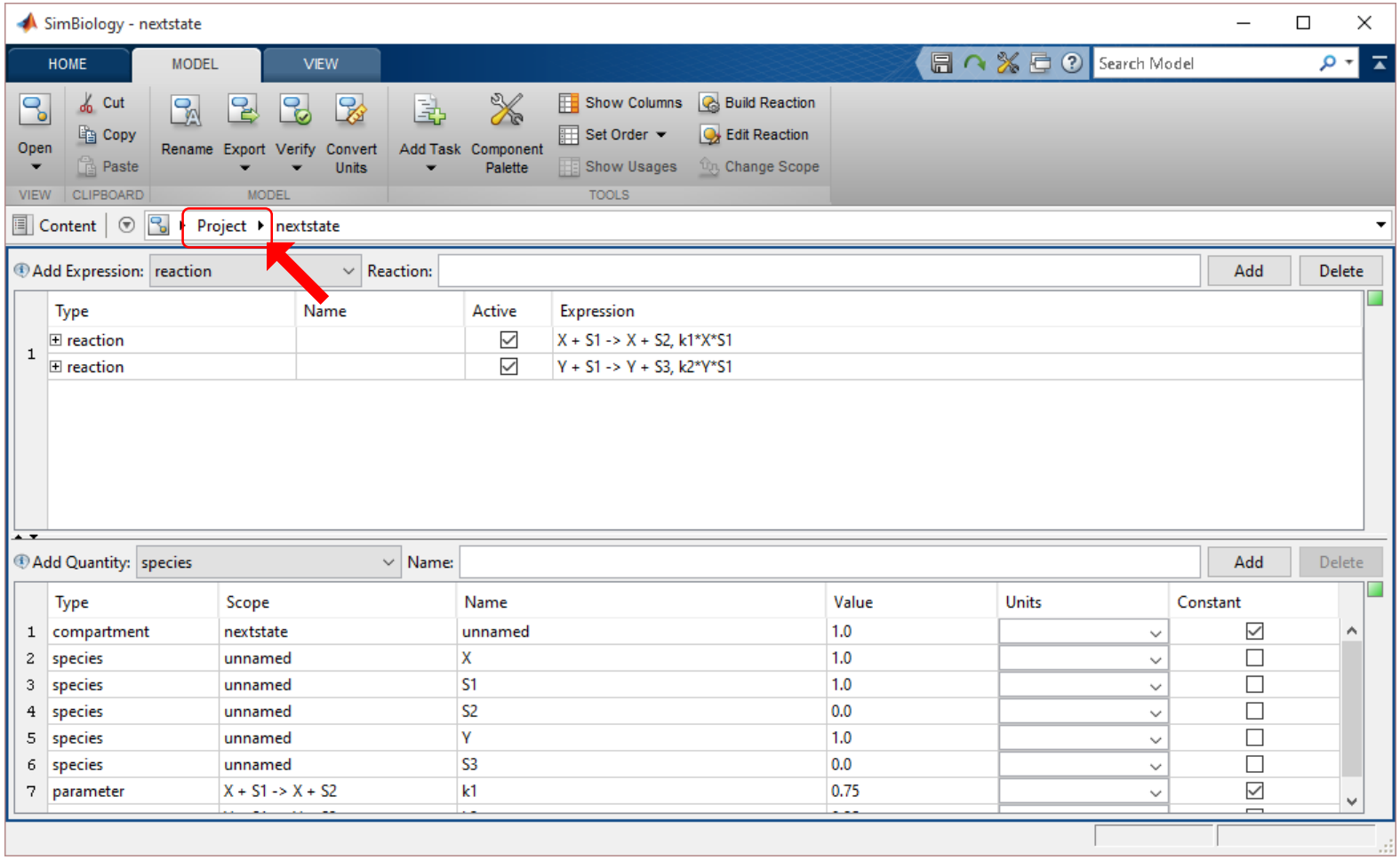
plus extensions of these models with costs and rewards.

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.

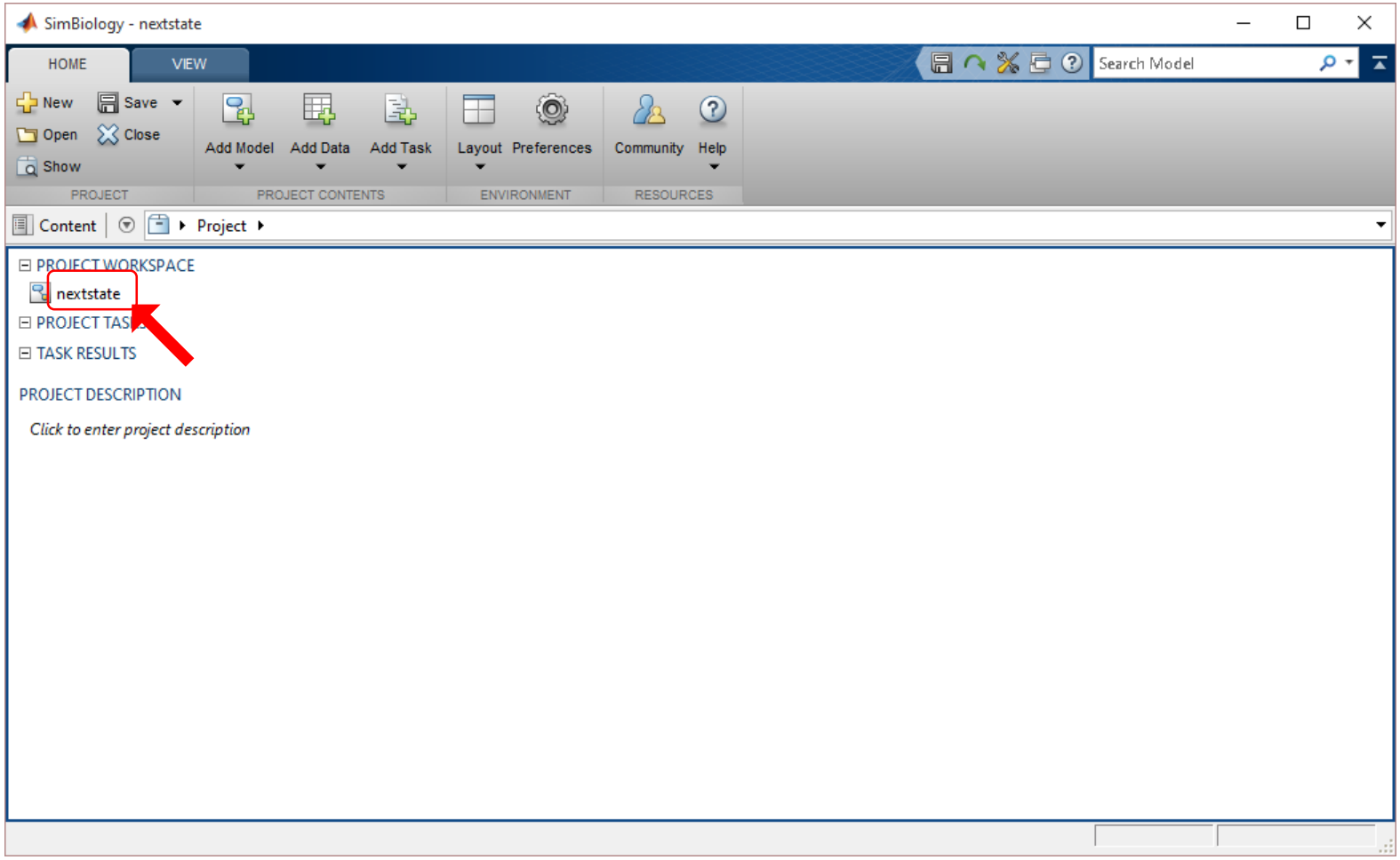
PRISM incorporates state-of-the art *symbolic* data structures and algorithms, based on BDDs (Binary Decision Diagrams) and MTBDDs (Multi-Terminal Binary Decision Diagrams) [KNP04b, Par02]. It also includes a *discrete-event simulation* engine, providing support for *approximate/statistical model checking*, 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).

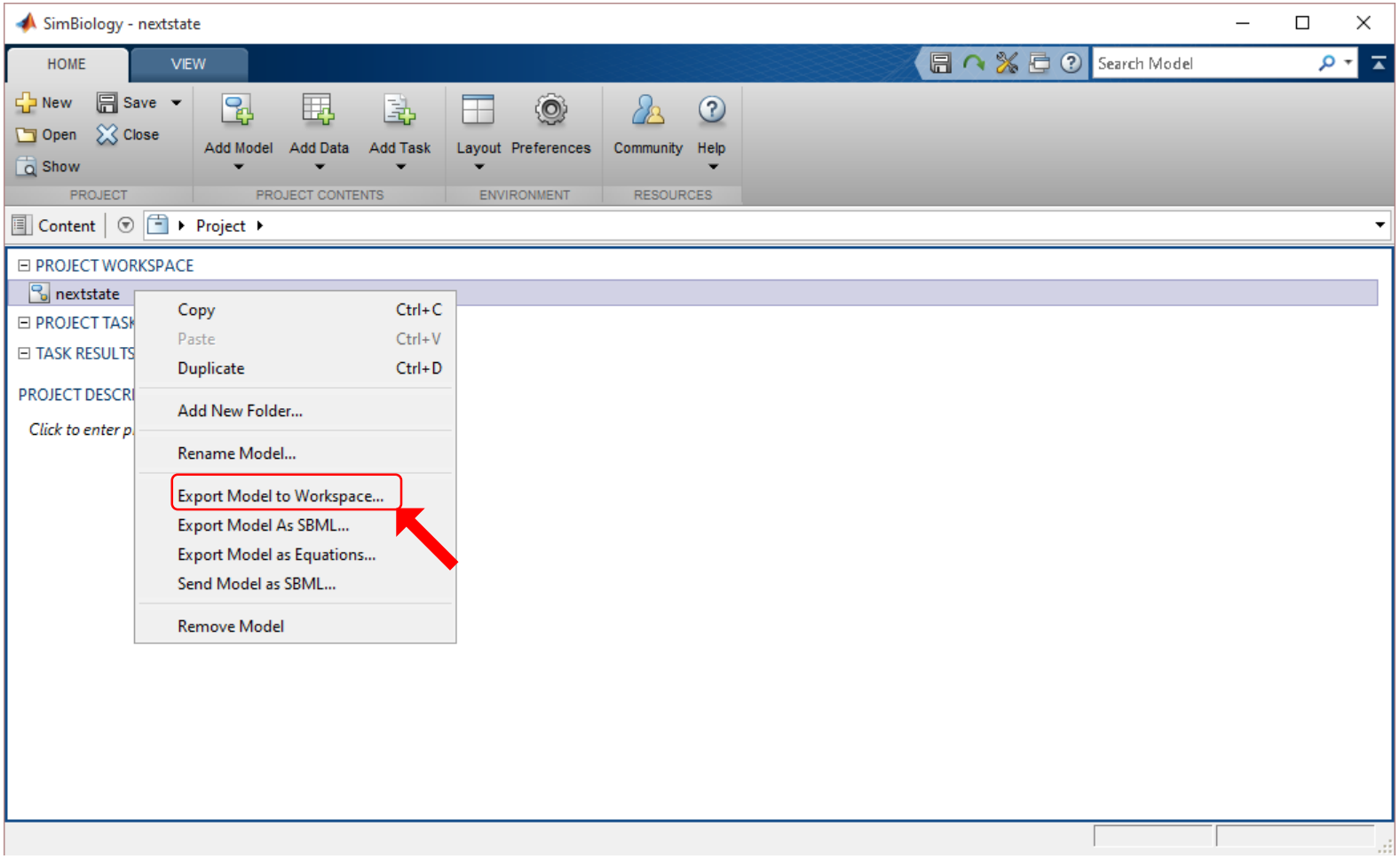
PRISM: Model Checker



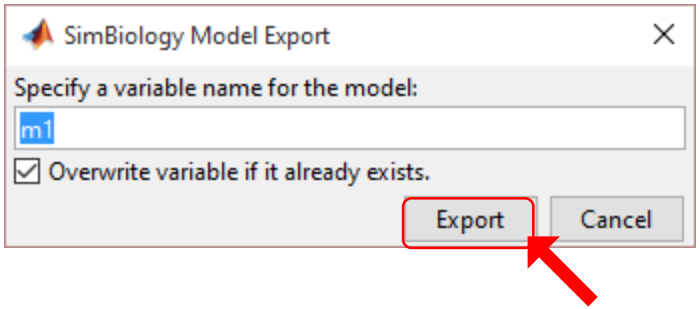
PRISM: Model Checker



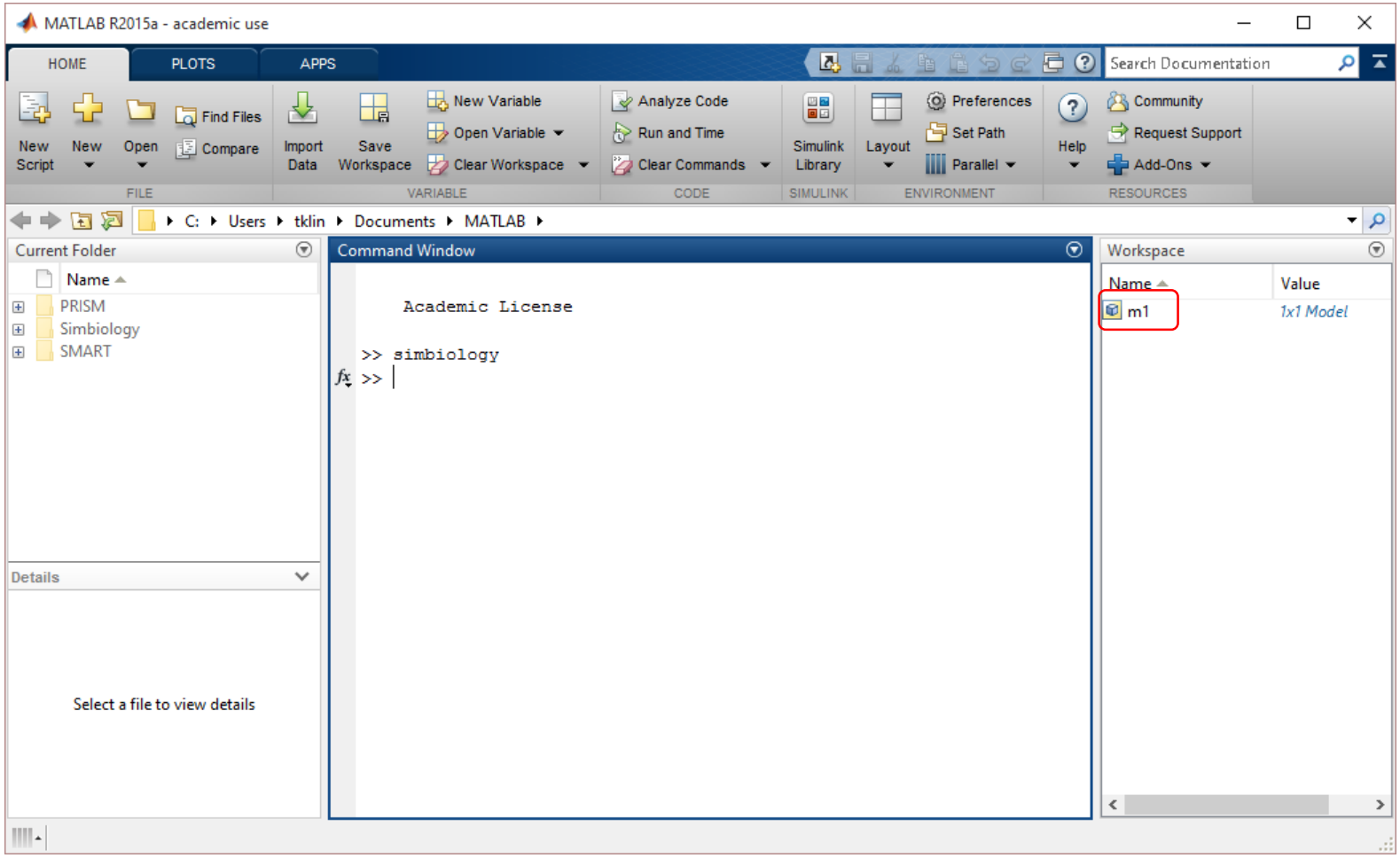
PRISM: Model Checker



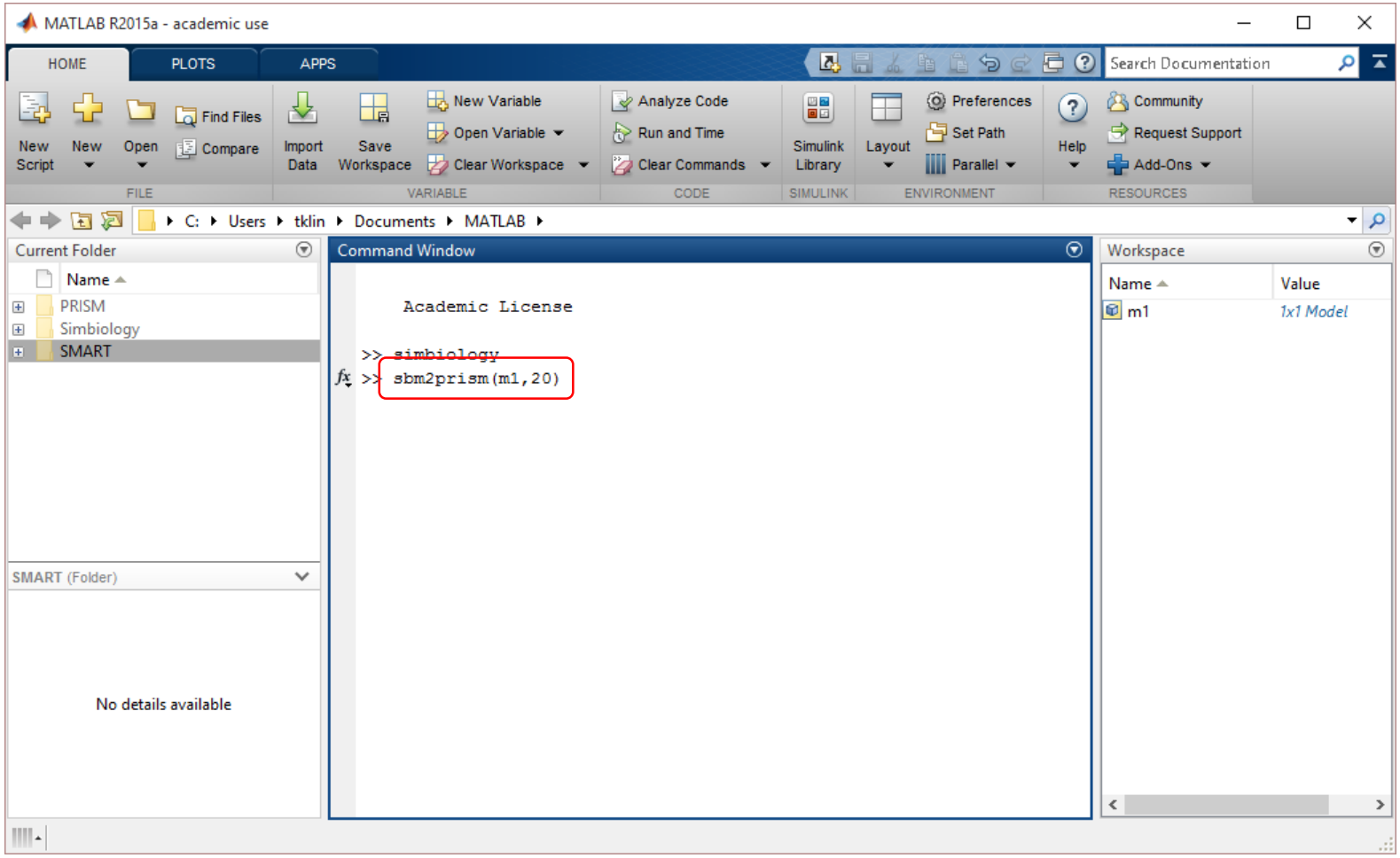
PRISM: Model Checker



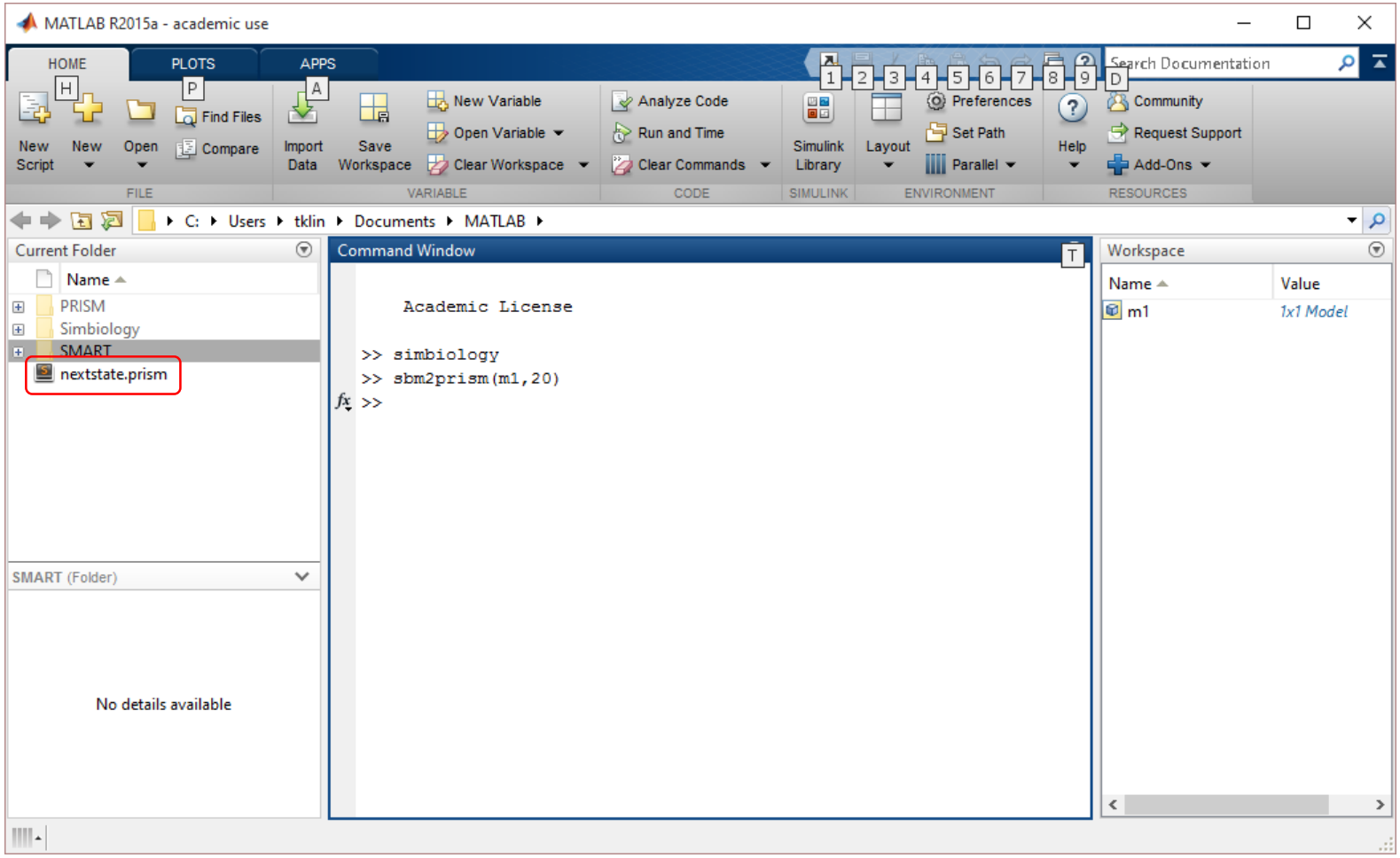
PRISM: Model Checker



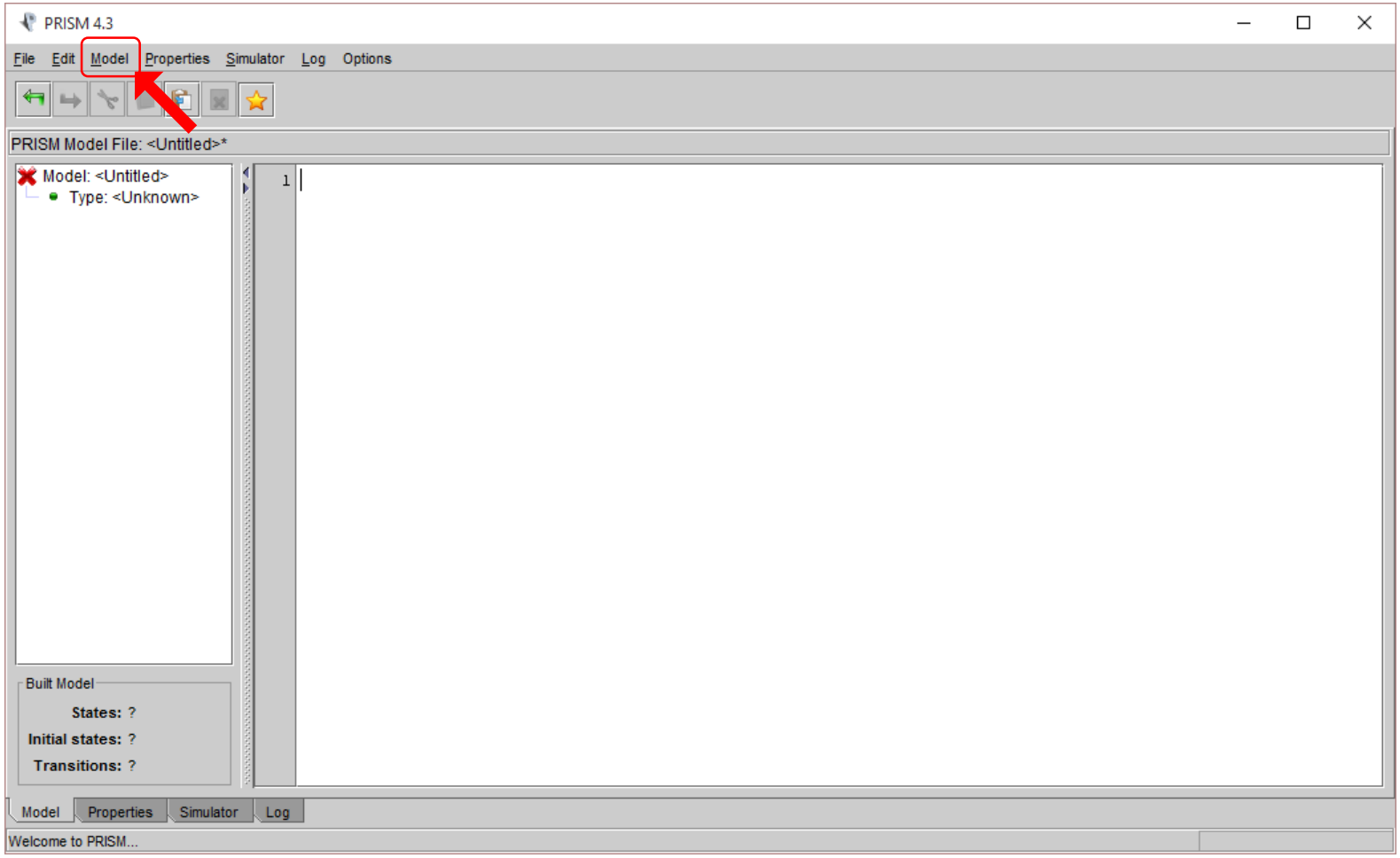
PRISM: Model Checker



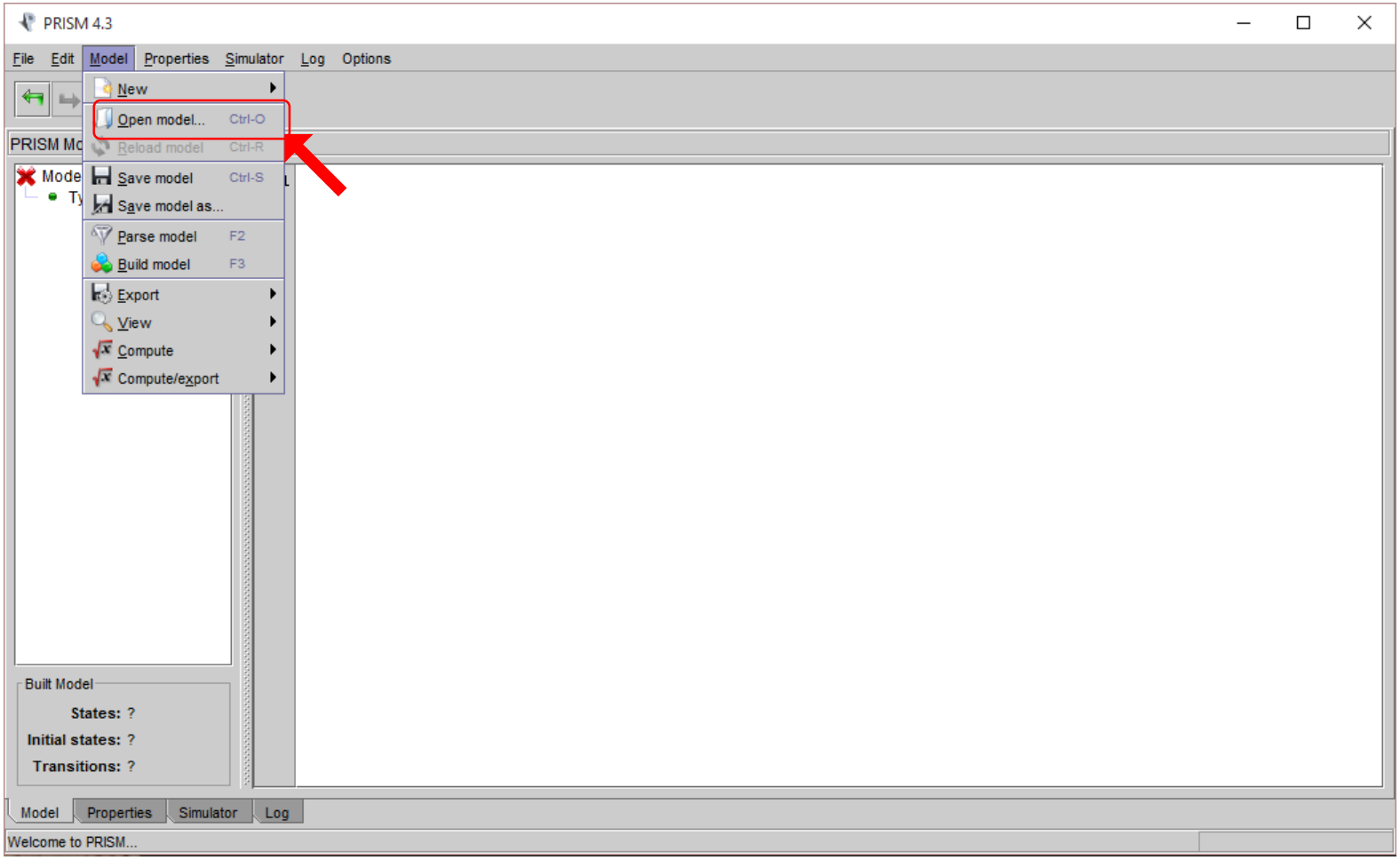
PRISM: Model Checker



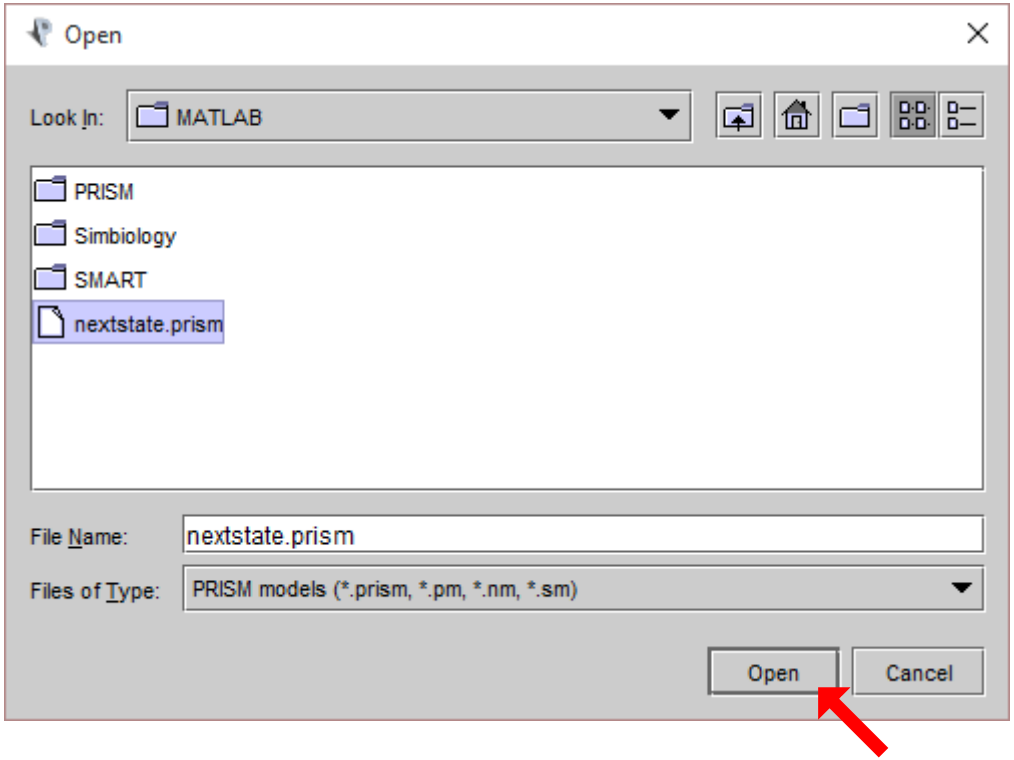
PRISM: Model Checker



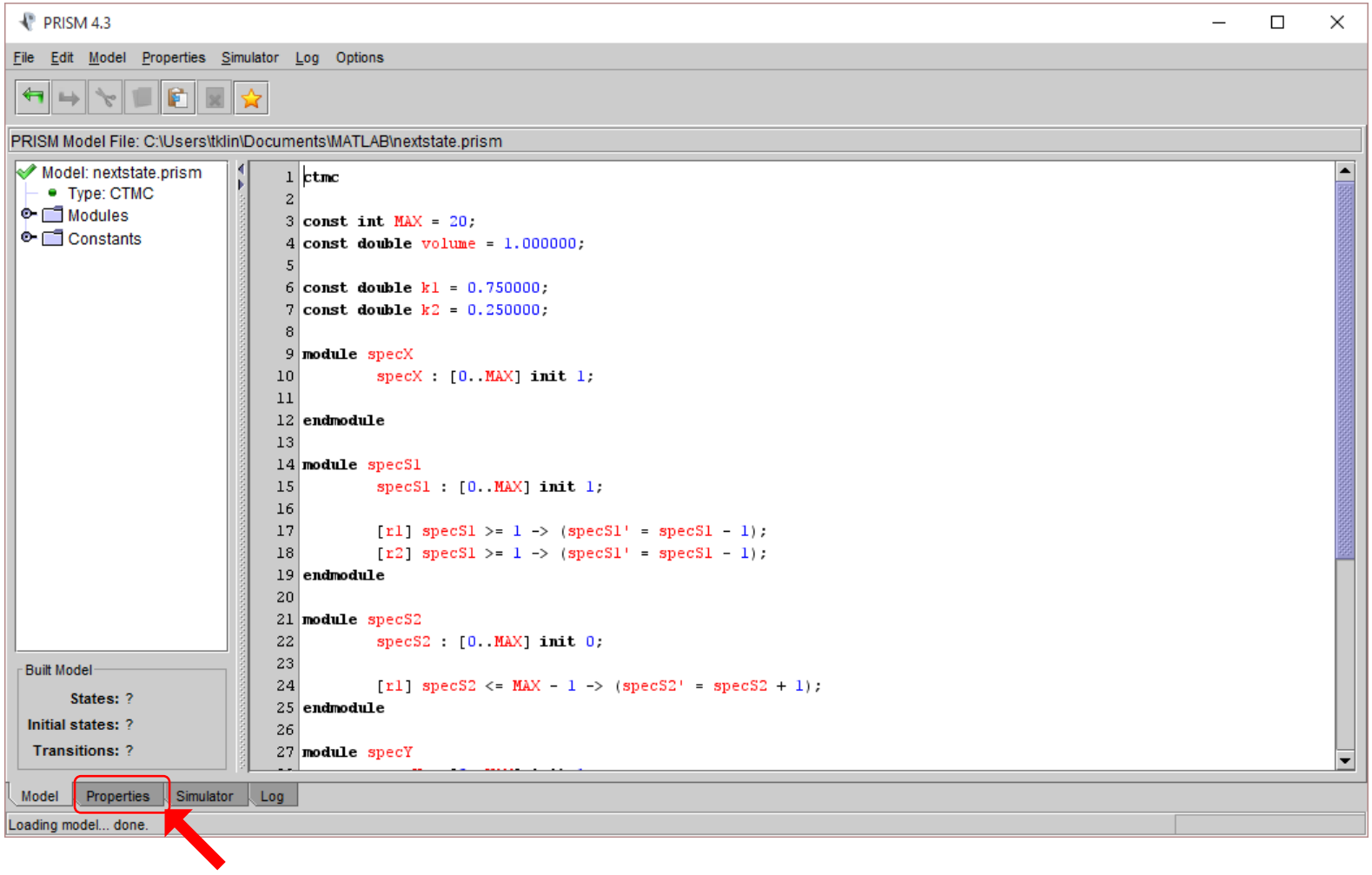
PRISM: Model Checker



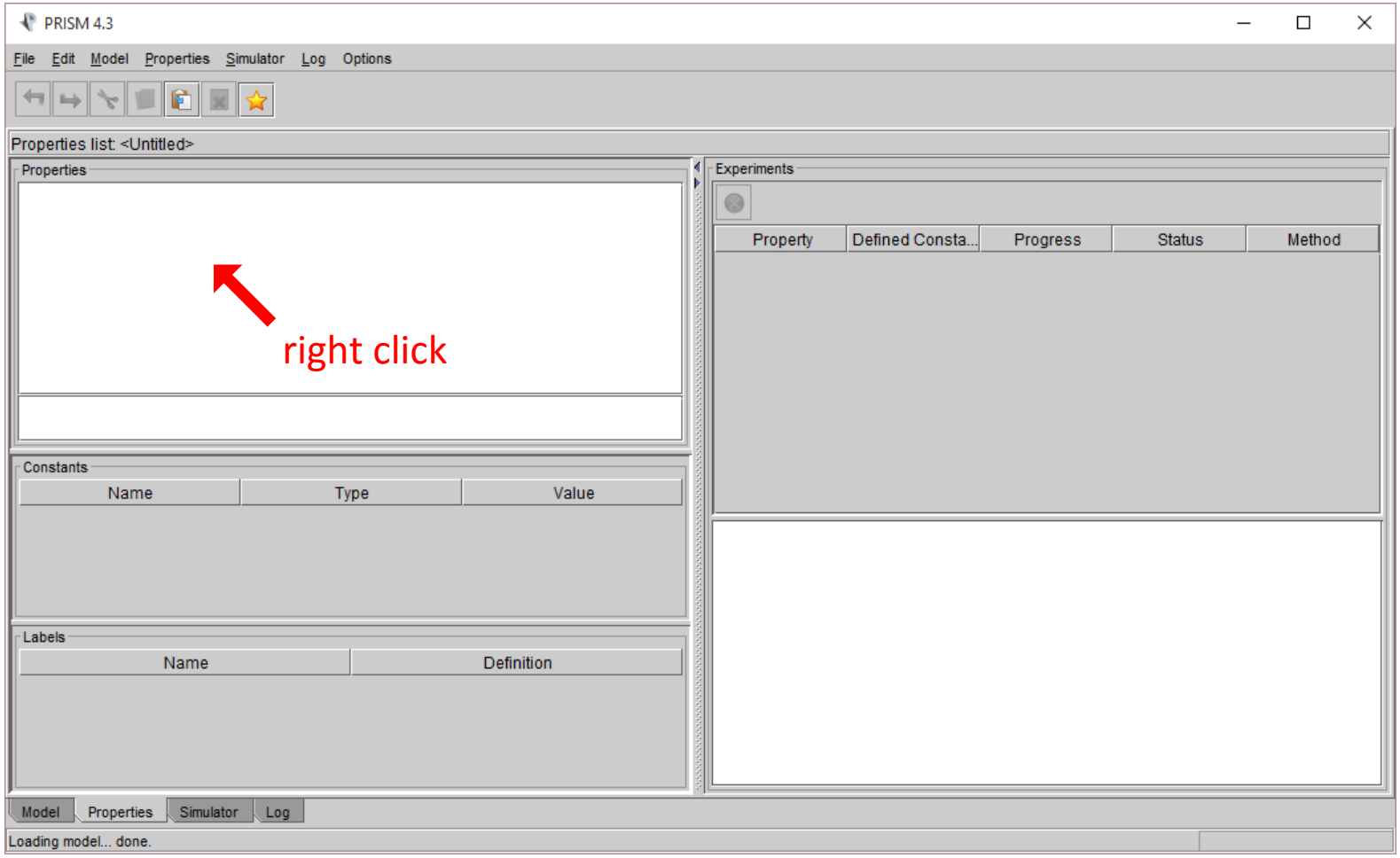
PRISM: Model Checker



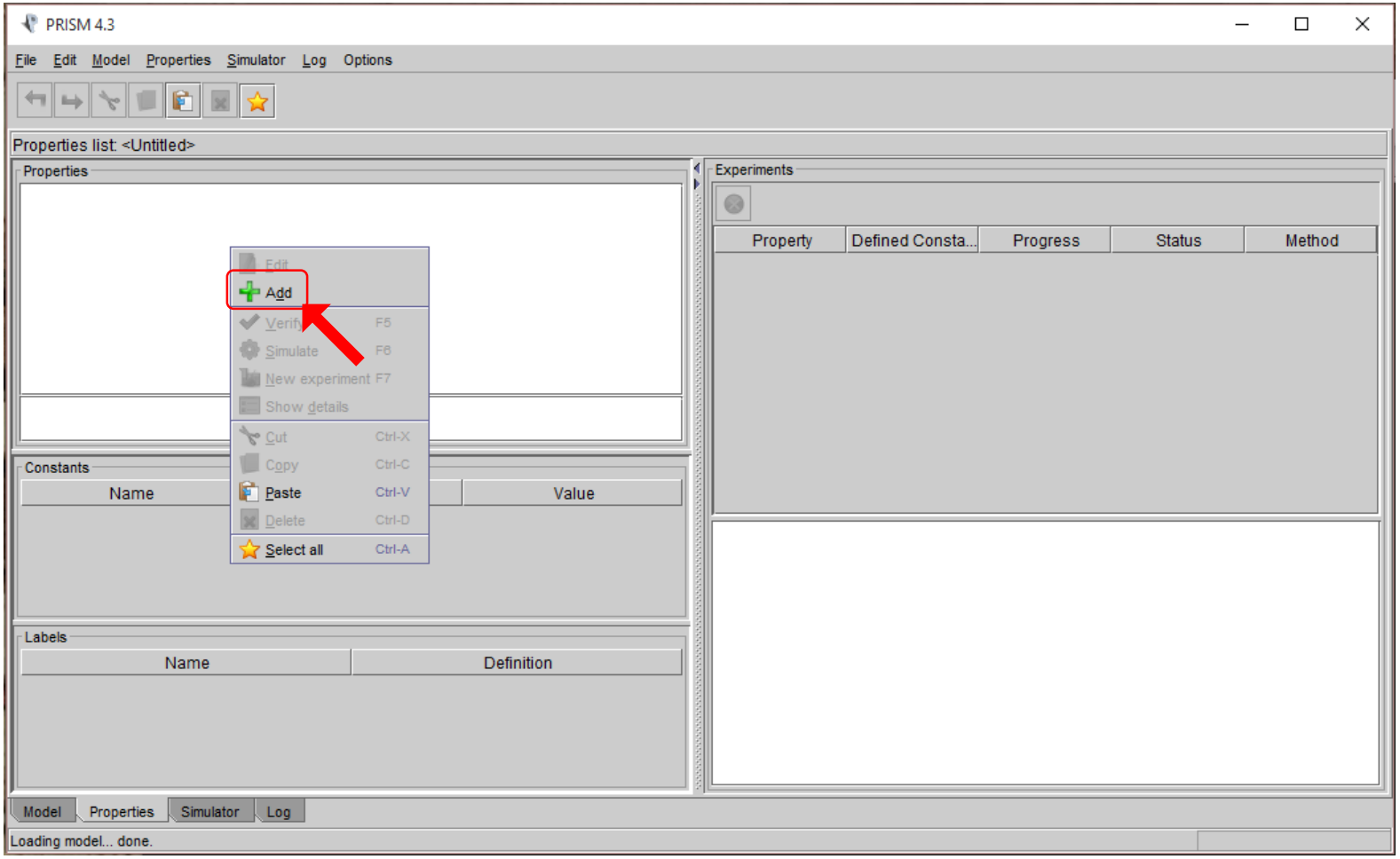
PRISM: Model Checker



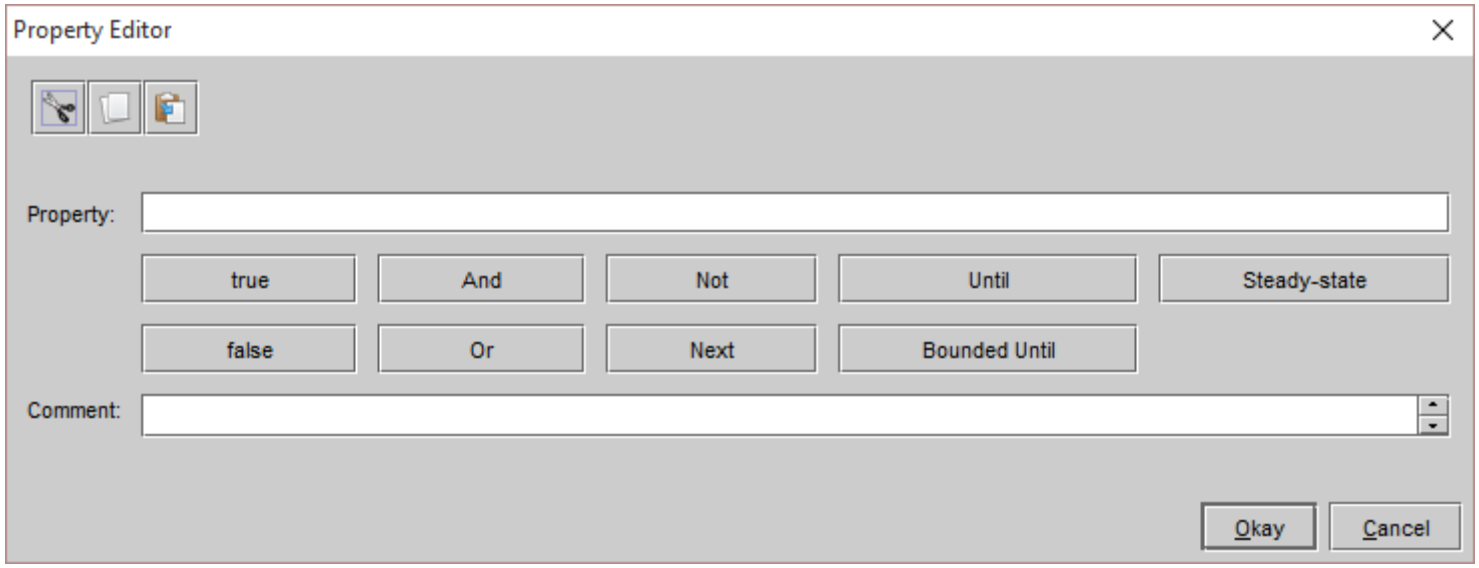
PRISM: Model Checker



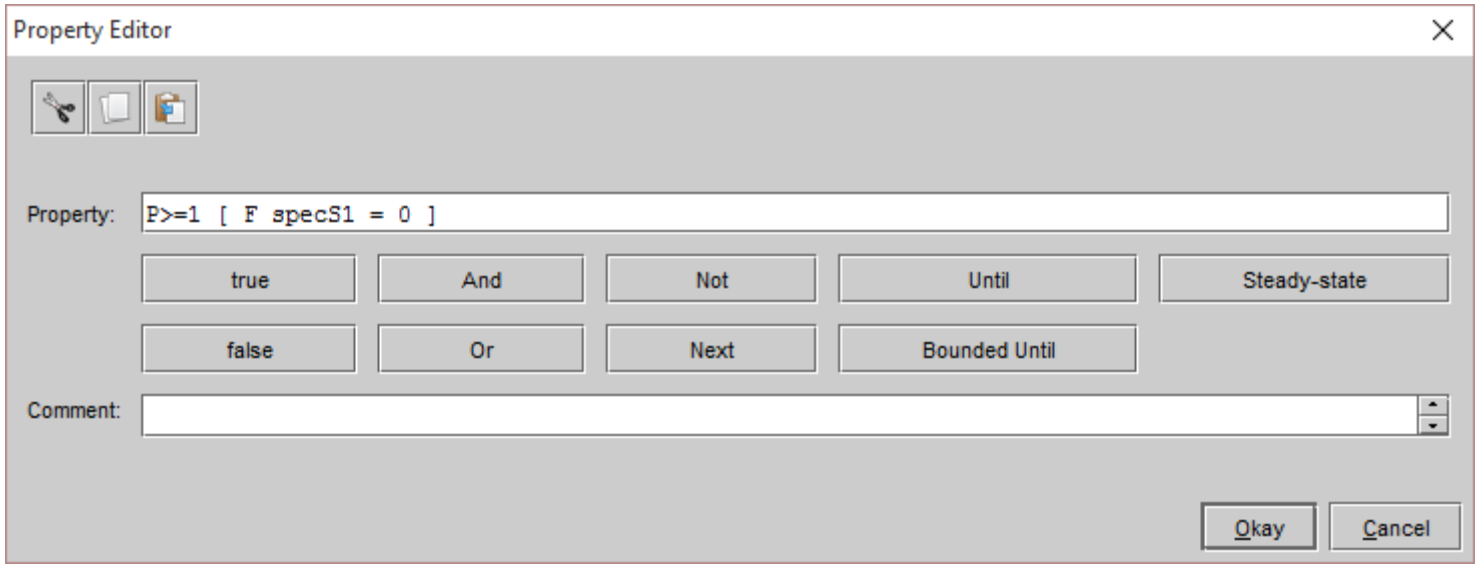
PRISM: Model Checker



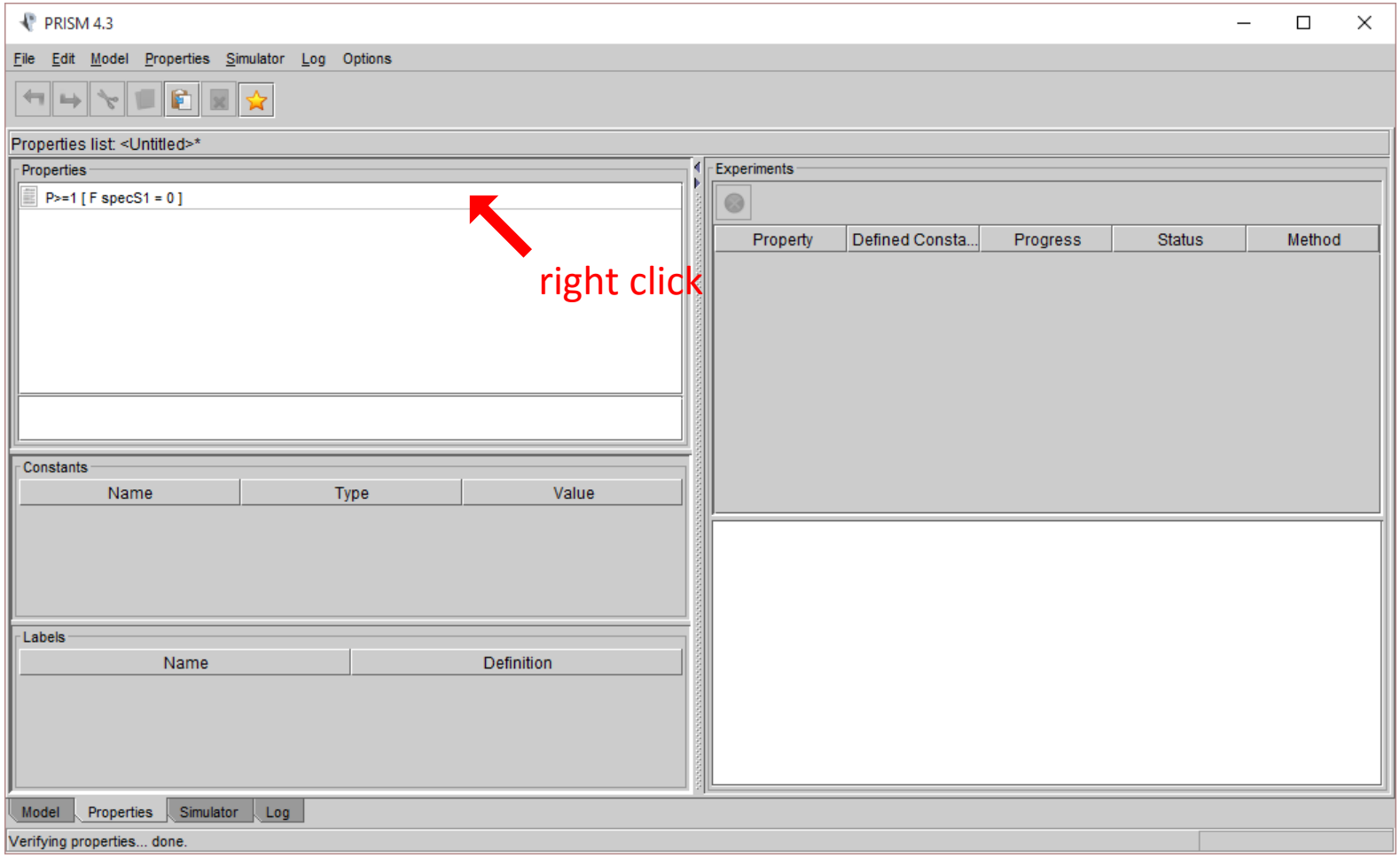
PRISM: Model Checker



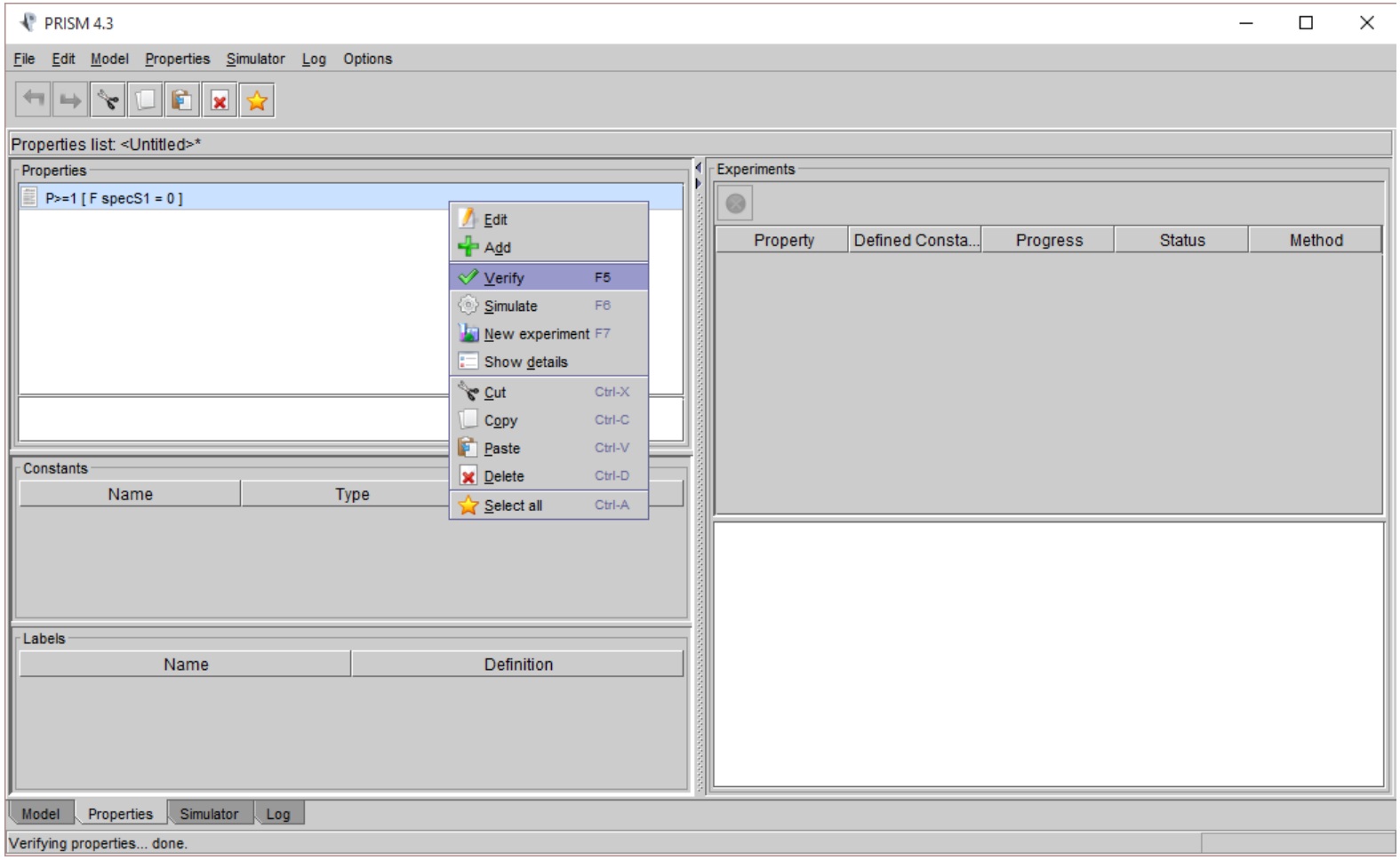
PRISM: Model Checker



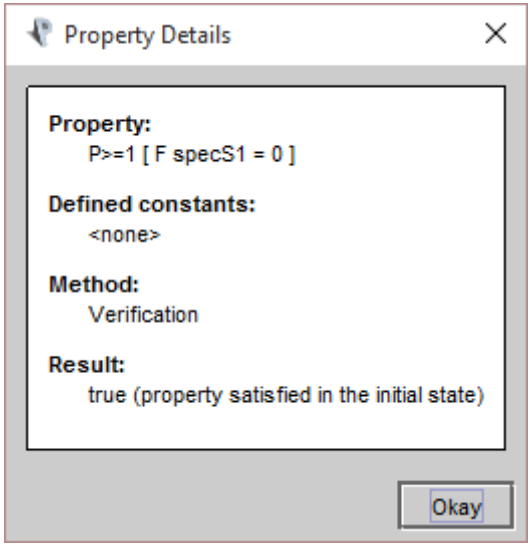
PRISM: Model Checker



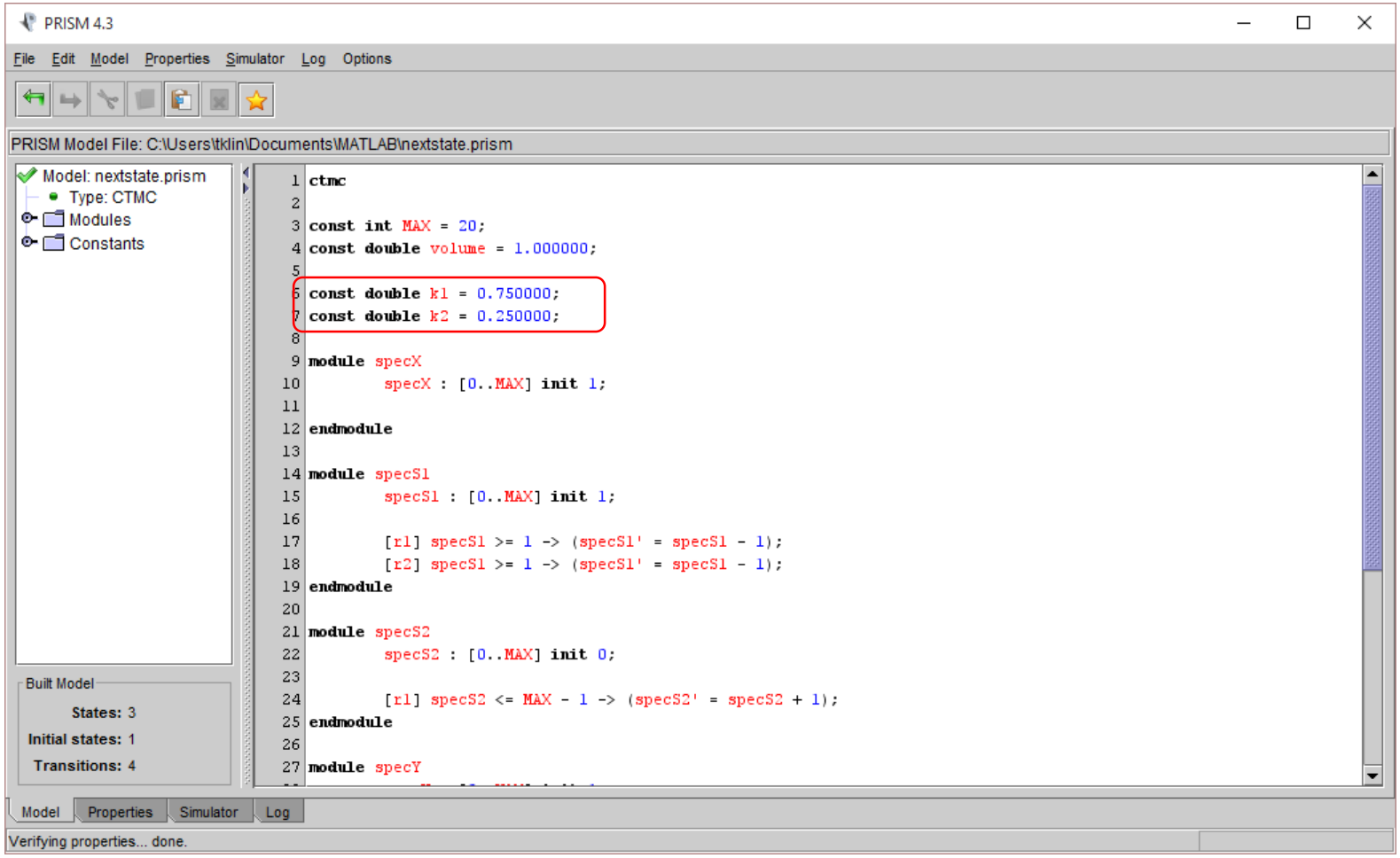
PRISM: Model Checker



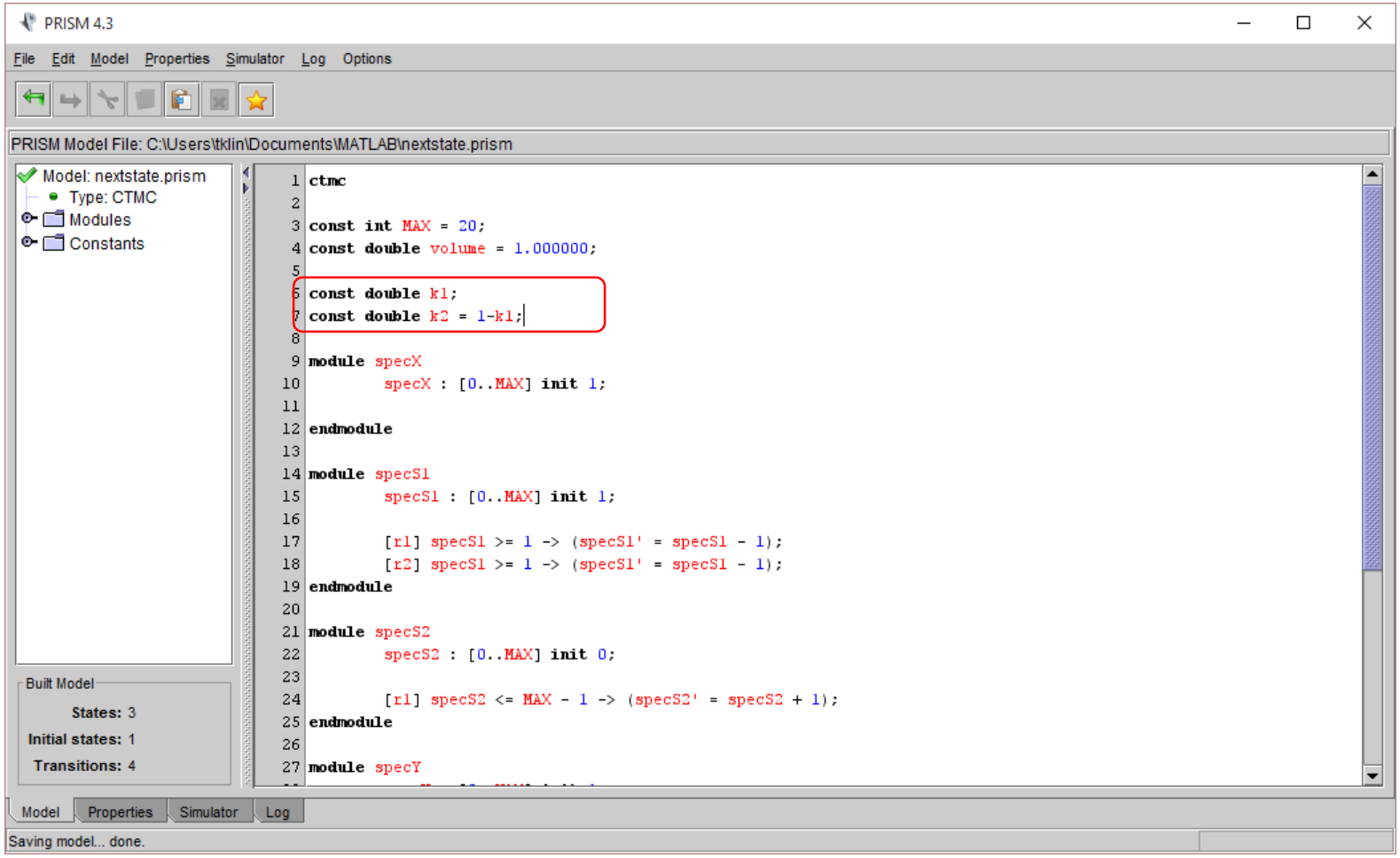
PRISM: Model Checker



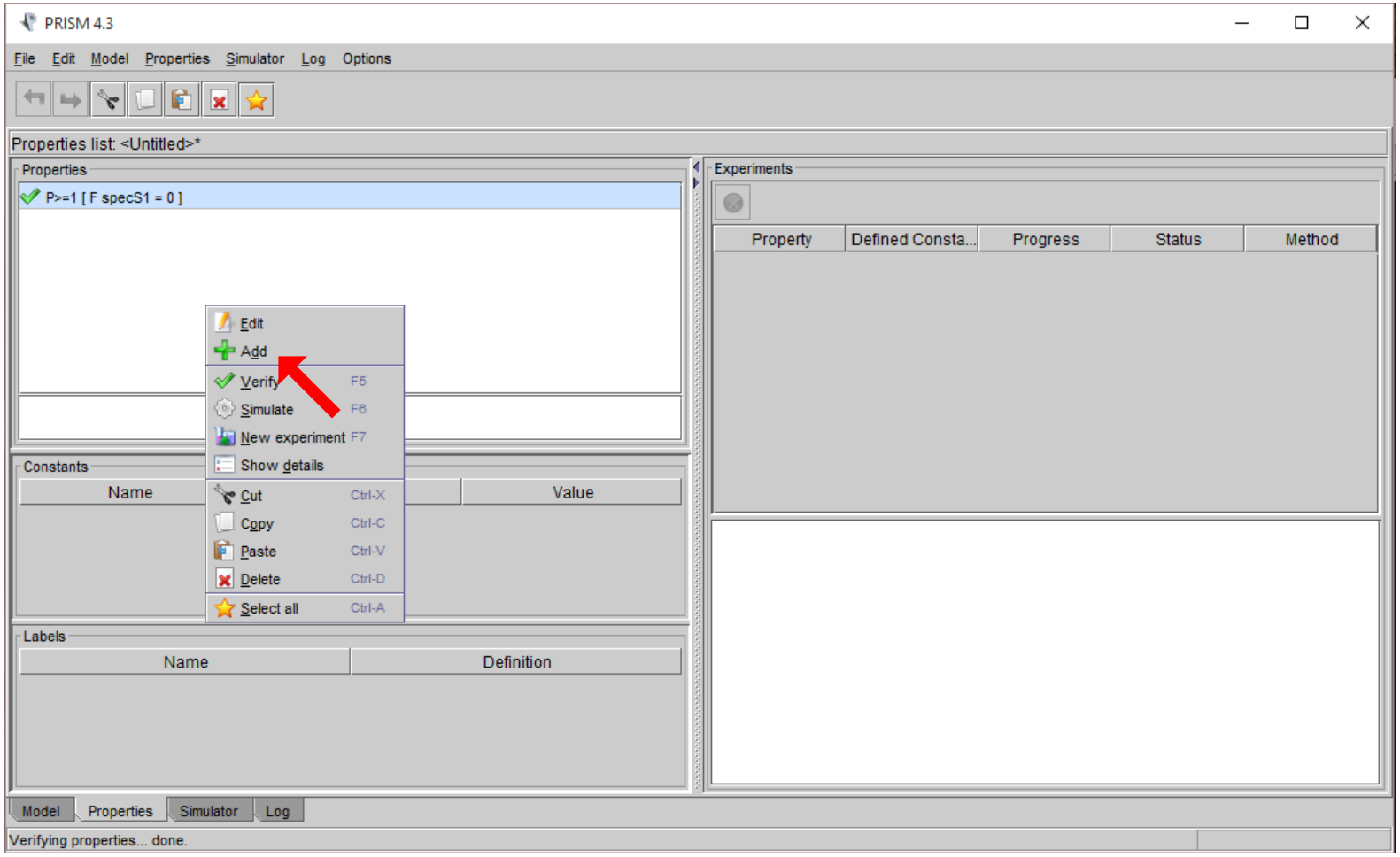
PRISM: Model Checker



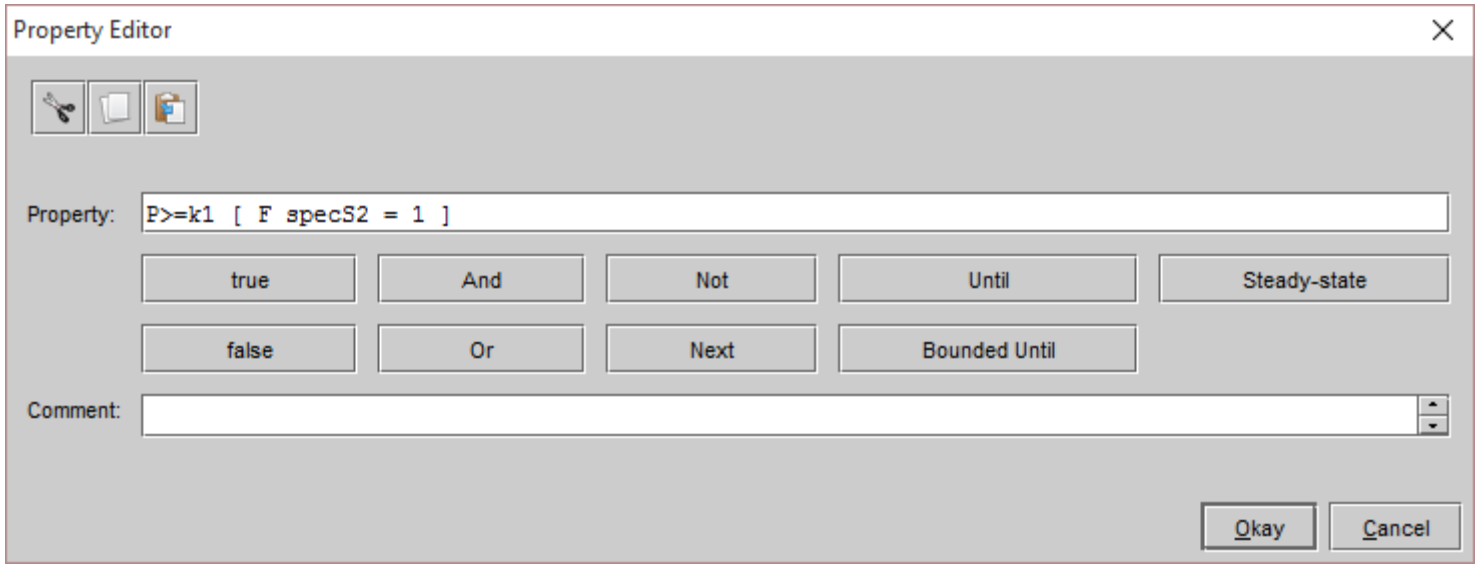
PRISM: Model Checker



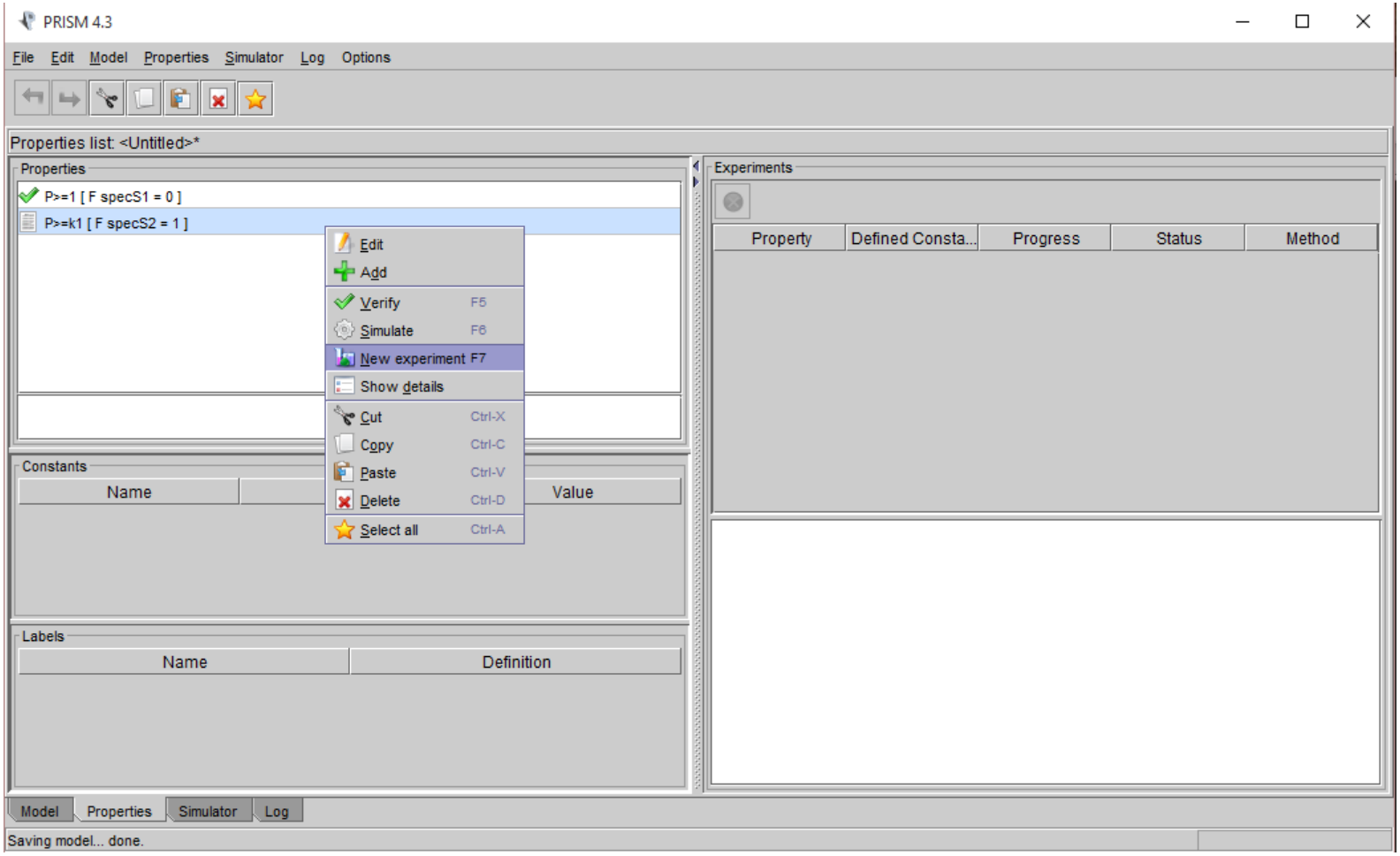
PRISM: Model Checker



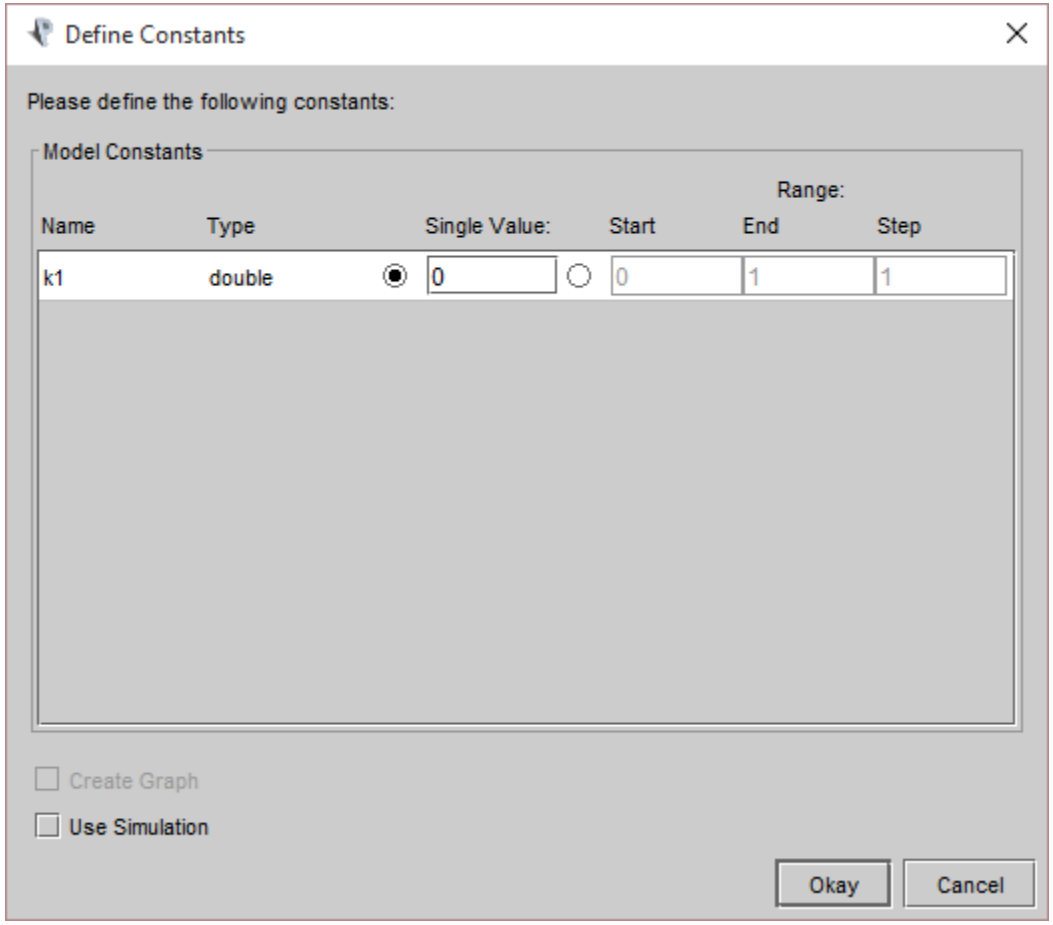
PRISM: Model Checker



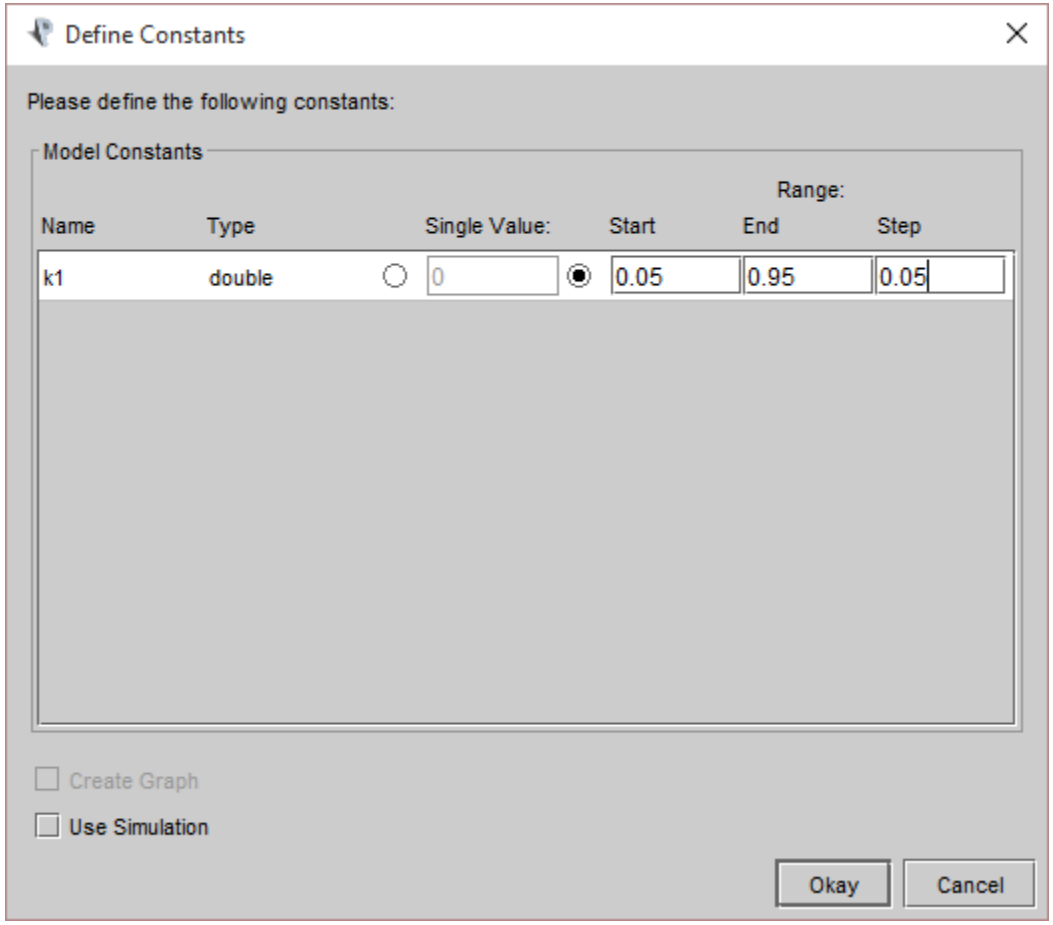
PRISM: Model Checker



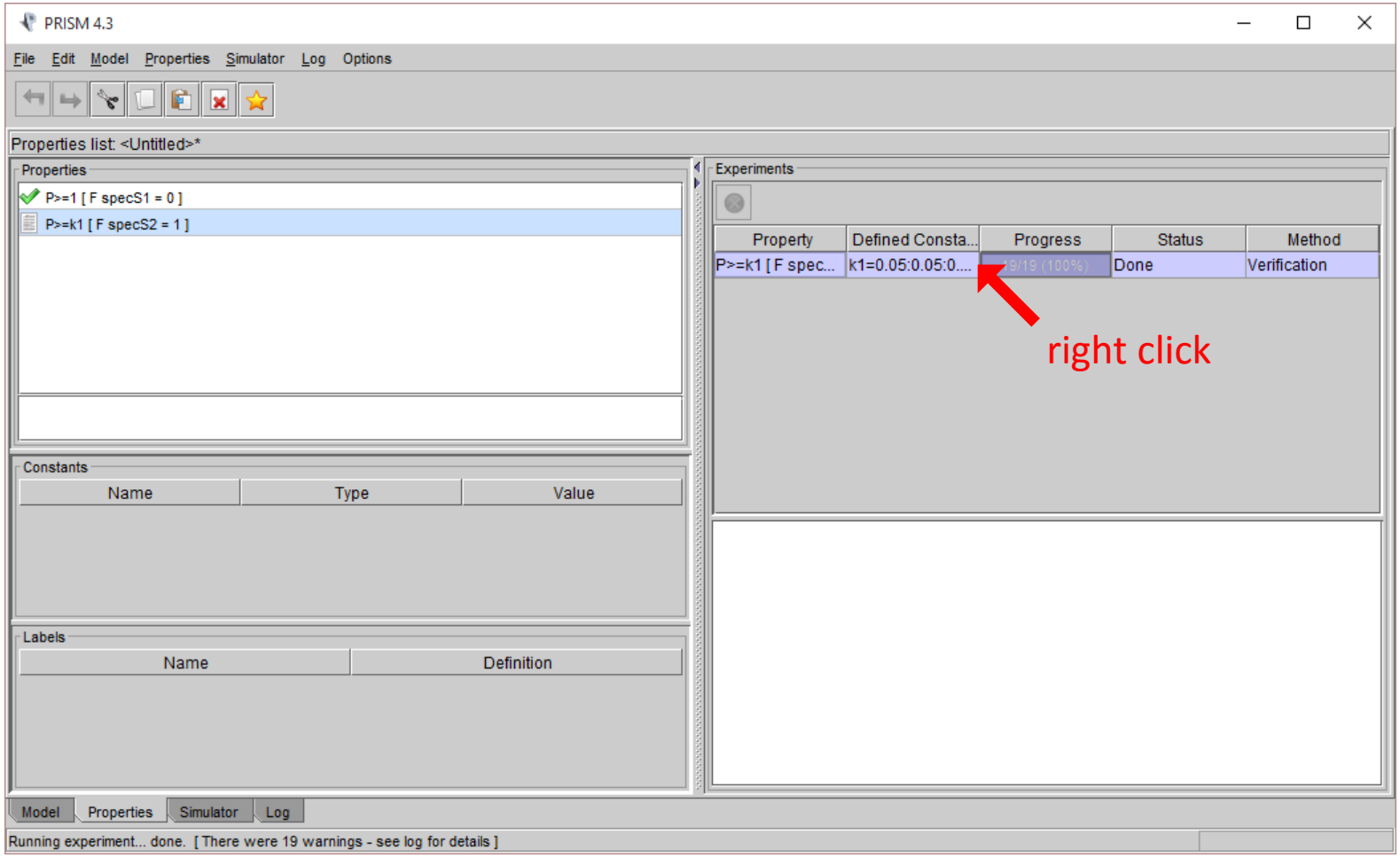
PRISM: Model Checker



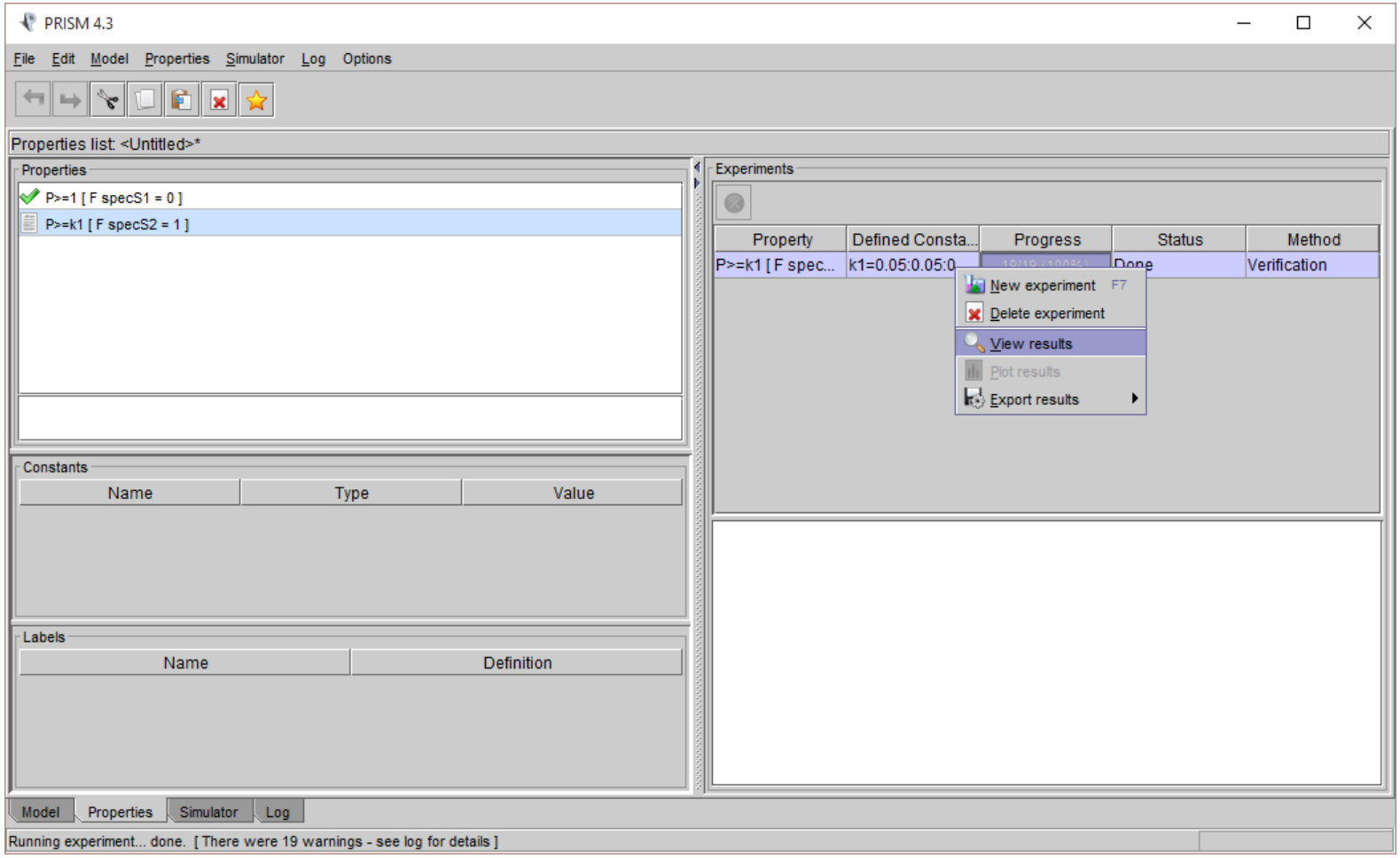
PRISM: Model Checker



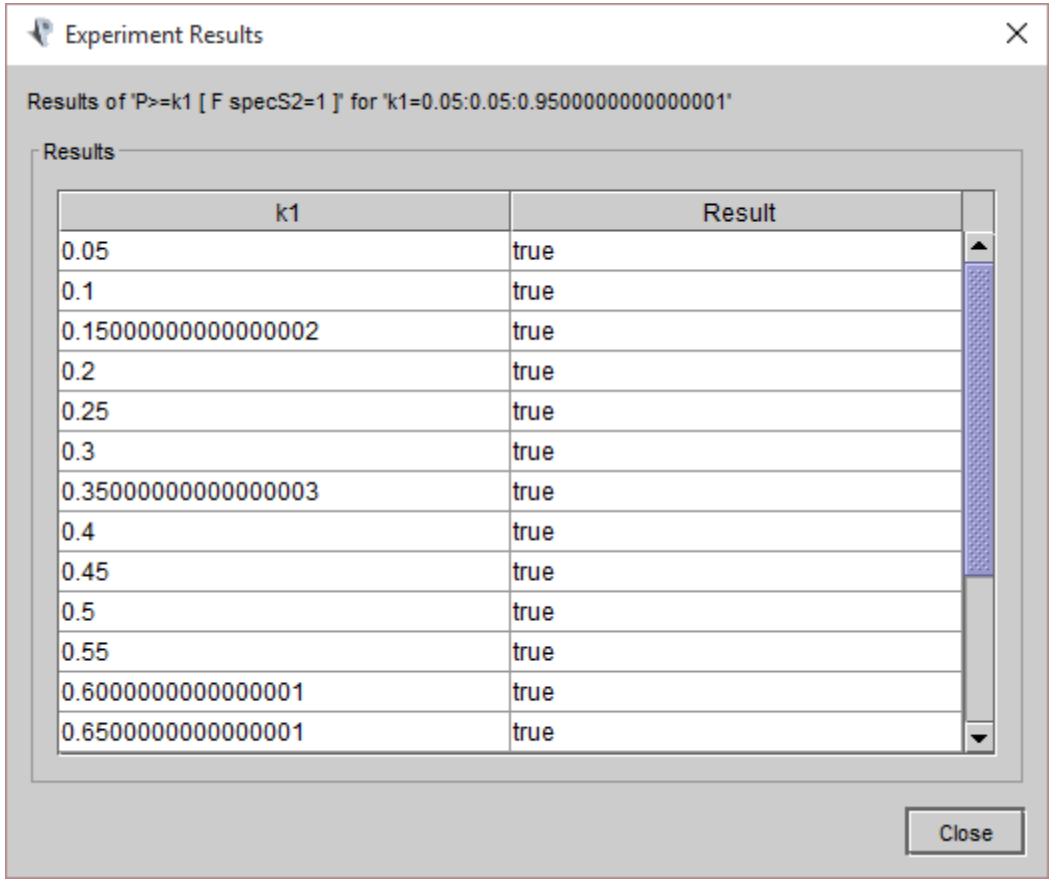
PRISM: Model Checker



PRISM: Model Checker



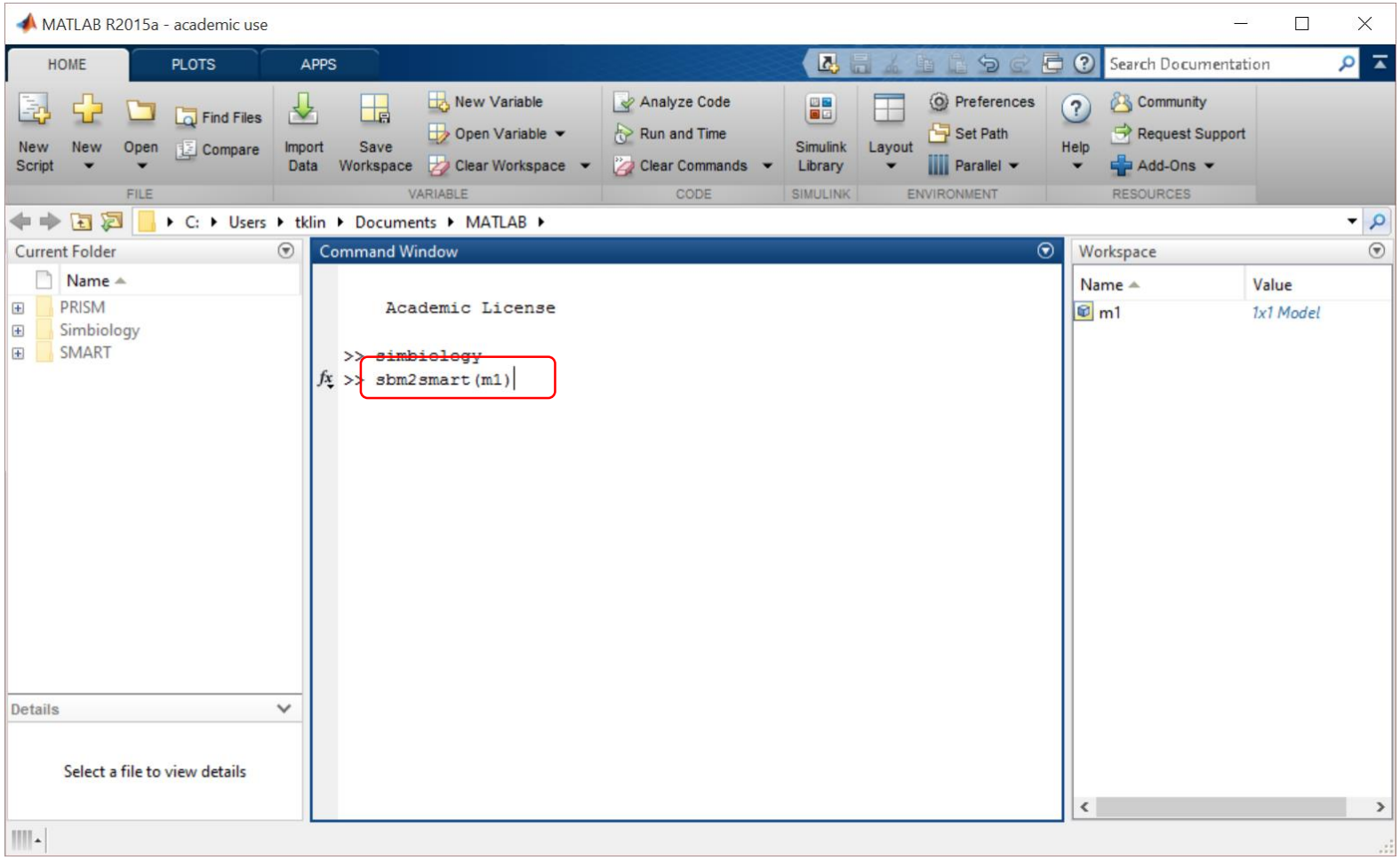
PRISM: Model Checker



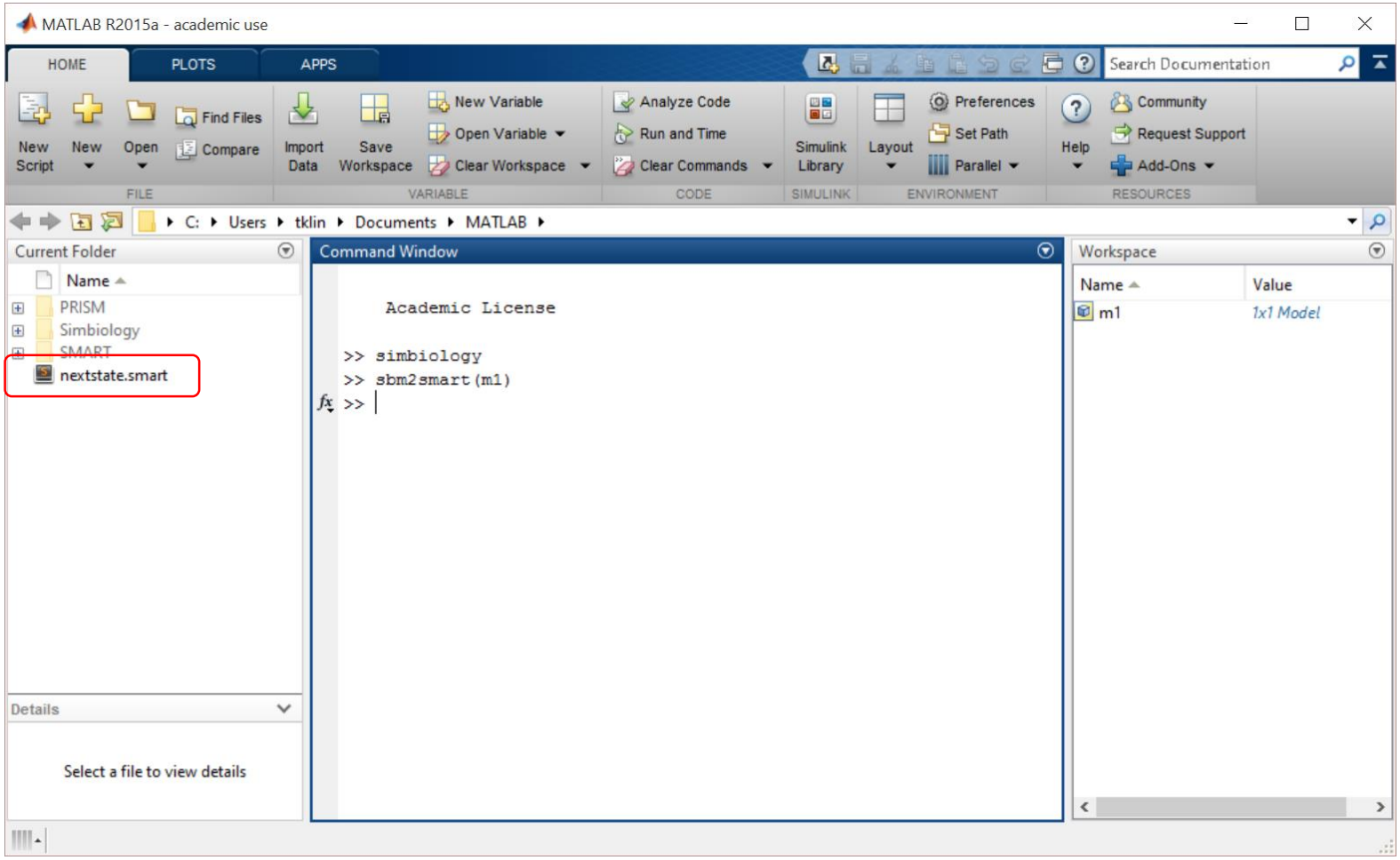
SMART: Model Checker

- 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

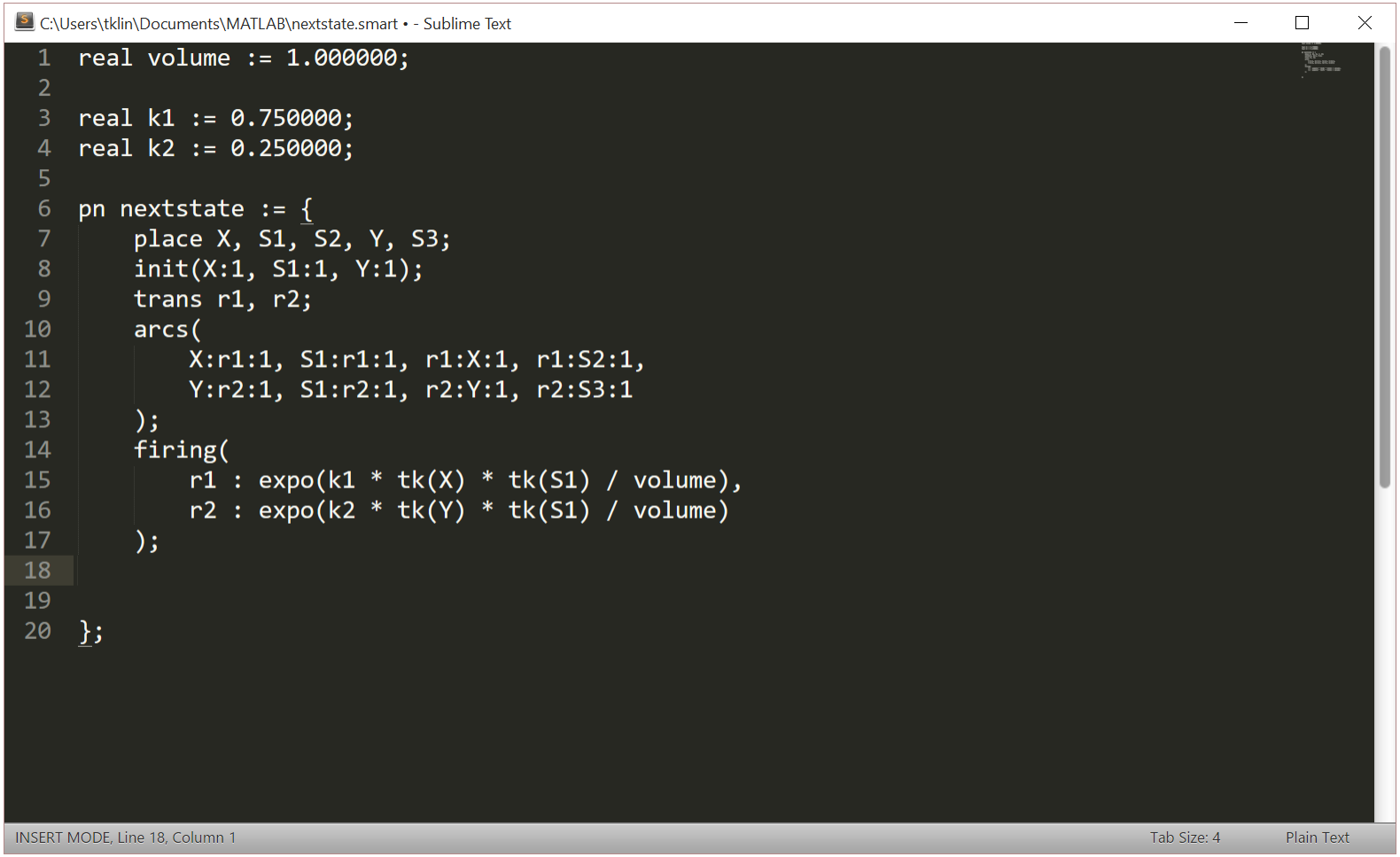
SMART: Model Checker



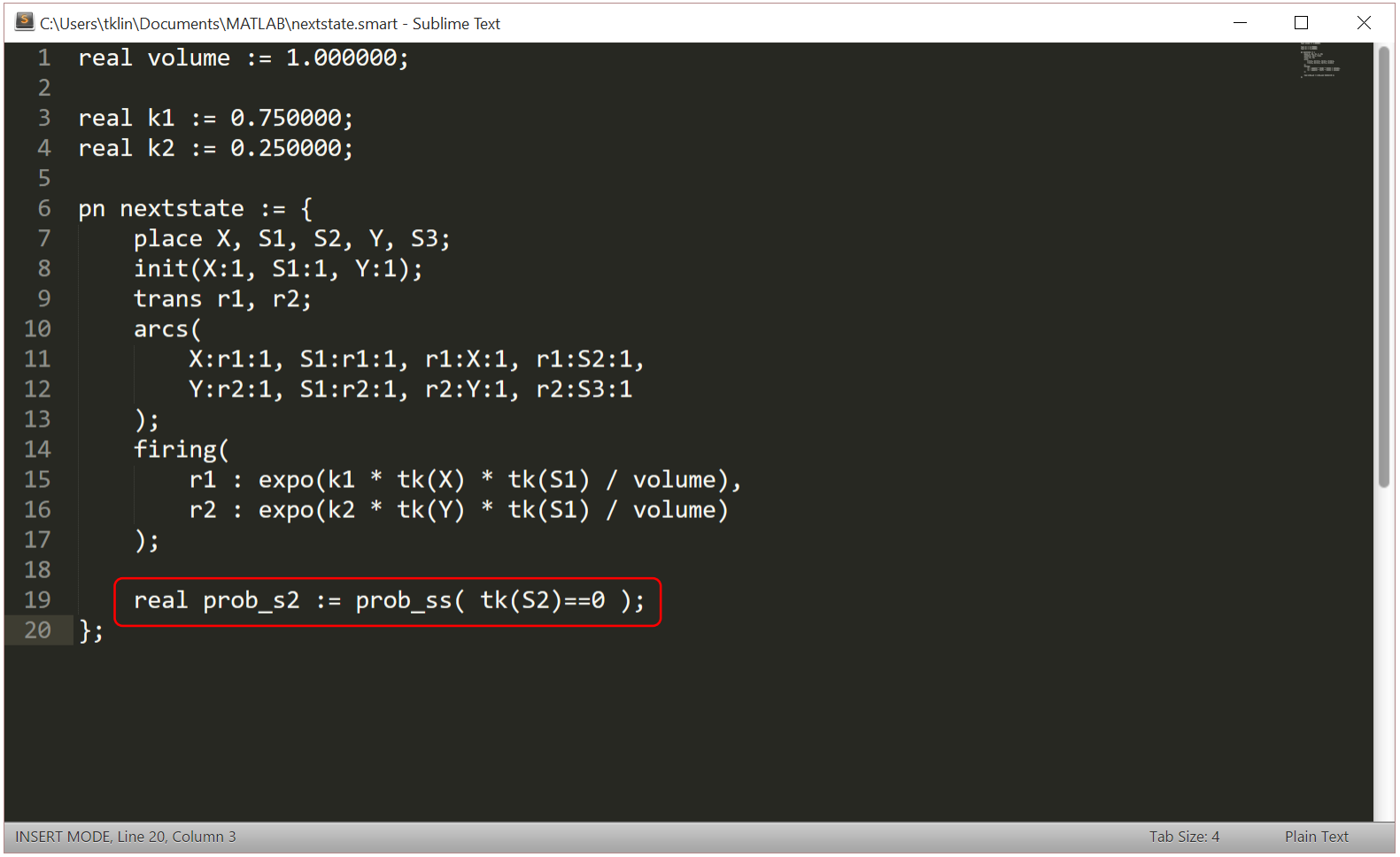
SMART: Model Checker



SMART: Model Checker



SMART: Model Checker



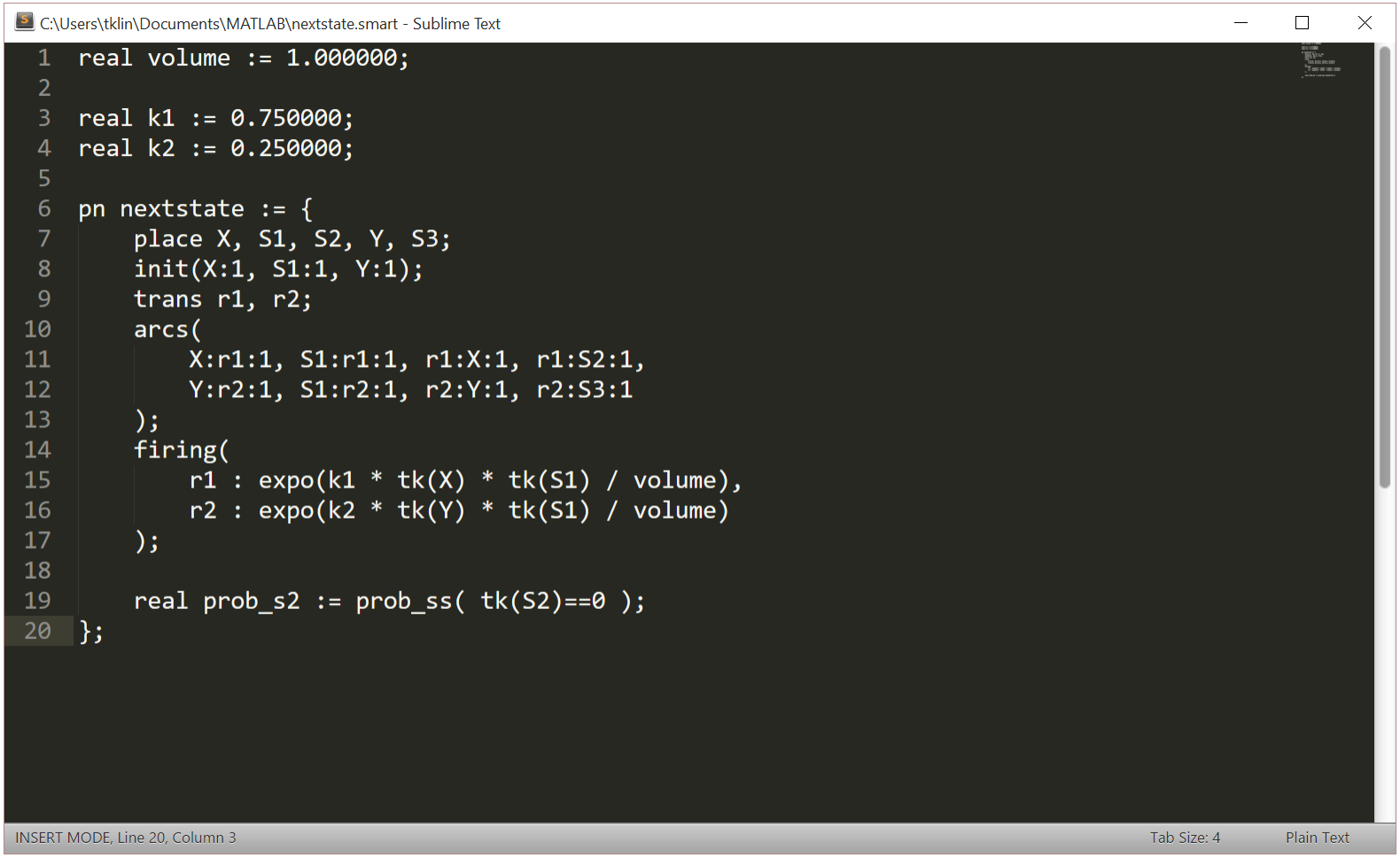
SMART: Model Checker

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

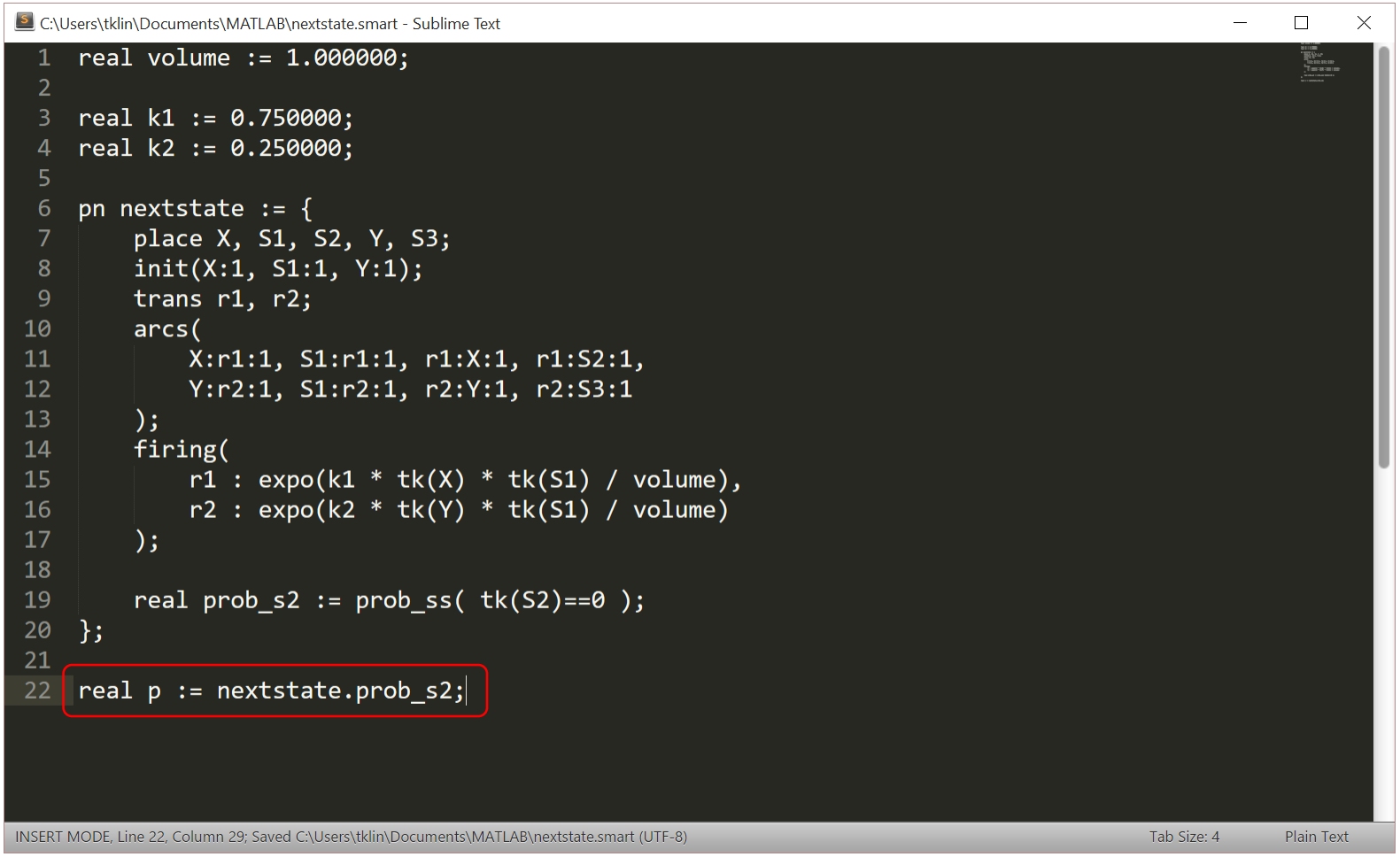
Just like the S operator

Number of tokens of S2

SMART: Model Checker



SMART: Model Checker



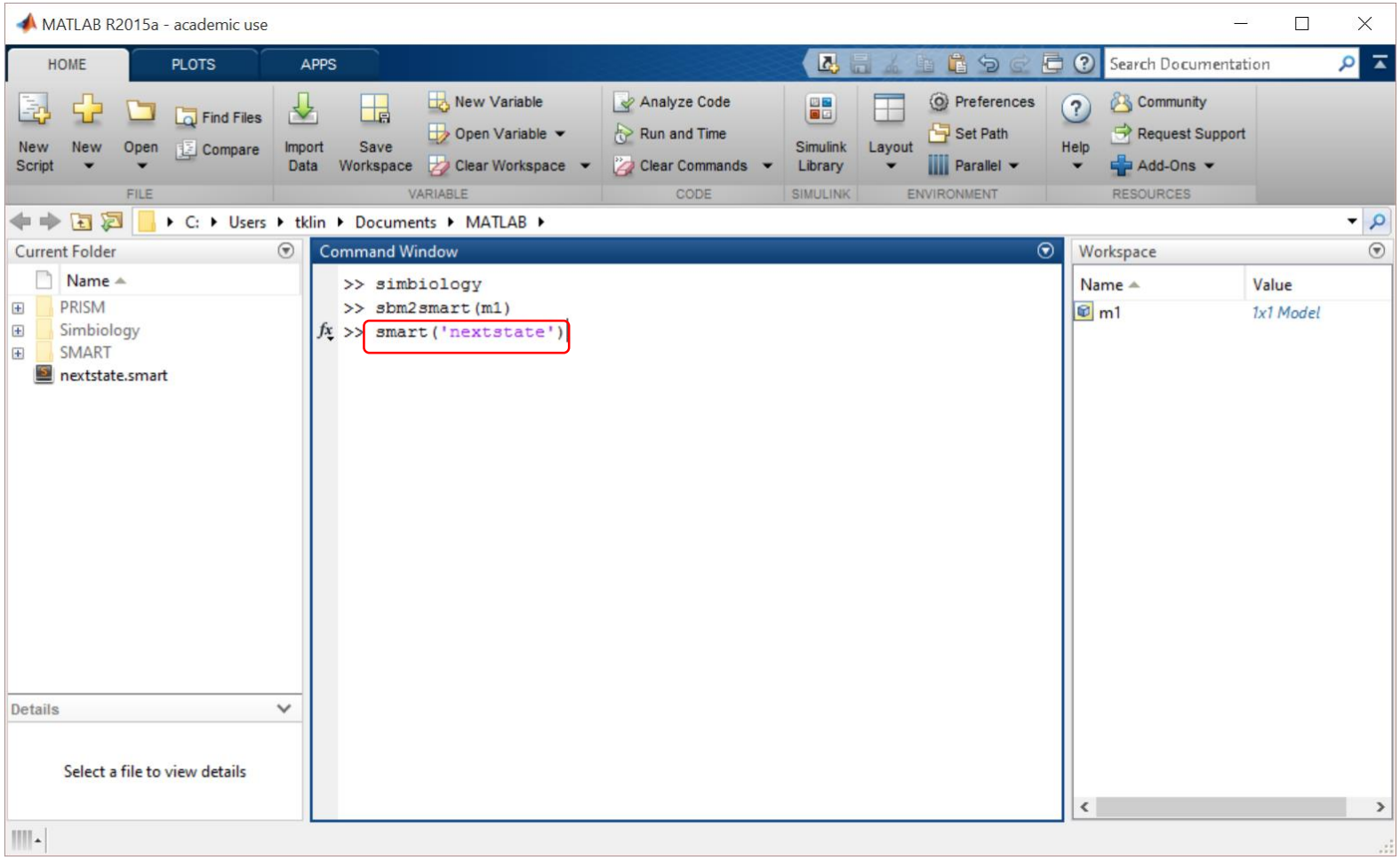
SMART: Model Checker

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

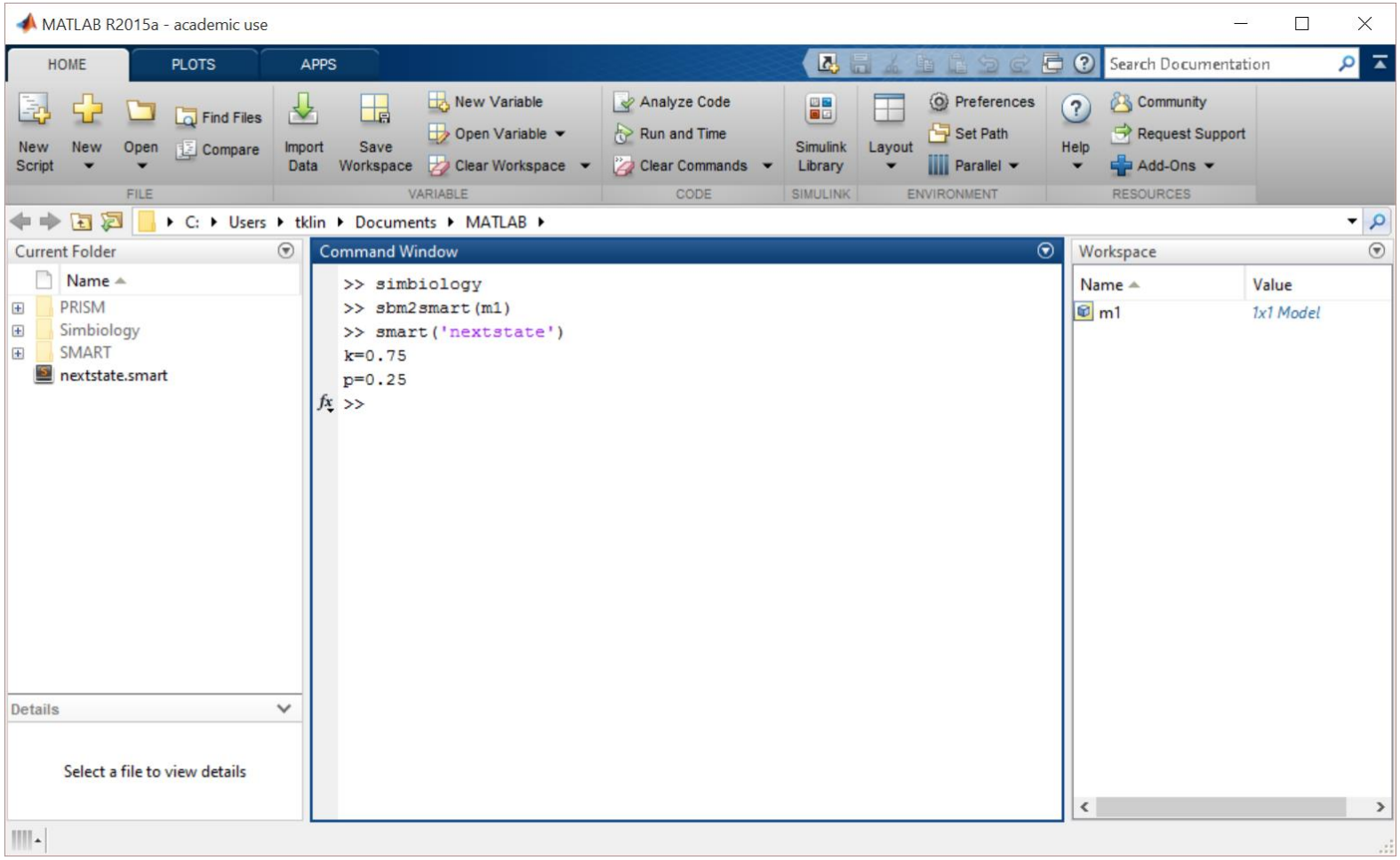
```
real volume := 1.000000;

real k1 := 0.750000;
real k2 := 0.250000;

pn nextstate := {
    place X, S1, S2, Y, S3;
    init(X:1, S1:1, Y:1);
    trans r1, r2;
    arcs(
        X:r1:1, S1:r1:1, r1:X:1, r1:S2:1,
        Y:r2:1, S1:r2:1, r2:Y:1, r2:S3:1
    );
    firing(
        r1 : expo(k1 * tk(X) * tk(S1) / volume),
        r2 : expo(k2 * tk(Y) * tk(S1) / volume)
    );

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

real p := nextstate.prob_s2;

print("k=0.75\n");
print("p=",p,"\n");
```

INSERT MODE, Line 25, Column 20 Tab Size: 4 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)