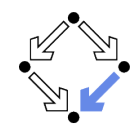
Nebenläufige Systeme

- Nebenläufigkeit (Concurrency)
  - Mehrere gleichzeitig aktive Komponenten.
  - Nicht-deterministisches Verhalten.
  - Oft: System läuft unaufhörlich (keine Termination).
  - Schwer/unmöglich zu testen.
- Sicherheitseigenschaft (Safety)
  - „Nichts Schlechtes wird je passieren.“
  - Keine mögliche Ausführung des Systems führt zu einem unerwünschten Zustand.
- Lebendigkeitseigenschaften (Liveness)
  - „Etwas Gutes wird letztendlich passieren.“
  - Jede mögliche Ausführung des Systems wird (unendlich oft) zu einem gewünschten Zustand führen.

Wir wollen die Sicherheit und Lebendigkeit von nebenläufigen Systemen sicherstellen.

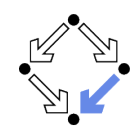


Transitionssysteme

Wir modellieren nebenläufige Systeme als (nichtdeterministische) „Transitionssysteme“.

- Zustandsraum: Menge S der Systemzustände.
  - Die möglichen Werte aller Variablen, sowohl der sichtbaren (Programmvariablen) als auch der versteckten (Programmzähler etc).
  - S kann endlich aber auch unendlich sein (wenn eine Variable unendlich viele Werte annehmen kann).
- Anfangszustände: Teilmenge $I \subseteq S$ von S.
  - Die Variablenwerte, mit denen eine Systemausführung beginnen kann.
  - Mehrere (auch unendlich viele) Anfangswerte möglich, wenn System (durch externe Eingaben) verschieden initialisiert werden kann.
- Zustandsüberführung: Relation $R \subseteq S \times S$ auf S.
  - Die Menge (s, s') der möglichen Zustandsübergänge von einem Vorgängerzustand s zu einem Nachfolgerzustand s'.
  - Nichtdeterminismus: zu einem Vorgängerzustand s kann es mehr als einen Nachfolgerzustand s' geben.

Ein Transitionssystem T ist ein solches Tripel $T = (S, I, R)$.

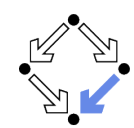


Die Beschreibung von Transitionssystemen

Ein Transitionssystem lässt sich durch logische Formeln beschreiben.

- Zustandsraum: $S := S_1 \times \ldots \times S_n$.
  - Ein Zustand besteht aus n Variablen mit Wertebereichen $S_1, \ldots, S_n$.
- Anfangsbedingung: $I(x_1, \ldots, x_n) :\Leftrightarrow \ldots$
  - Die Variablen $x_1, \ldots, x_n$ sind im Anfangszustand, wenn die Bedingung „. . . “ (mit Variablen $x_1, \ldots, x_n$) erfüllt ist.
- Zustandsüberführungsrelation: $R(\langle x_1, \ldots, x_n \rangle, \langle x'_1, \ldots, x'_n \rangle) :\Leftrightarrow \ldots$
  - Ein Vorgängerzustand $\langle x_1, \ldots, x_n \rangle$ kann in den Nachfolgerzustand $\langle x'_1, \ldots, x'_n \rangle$ übergehen, wenn die Bedingung „. . . “ (mit Variablen $x_1, \ldots, x_n, x'_1, \ldots, x'_n$) erfüllt ist.

Verwendung von Logik zur Beschreibung von Transitionssystemen.



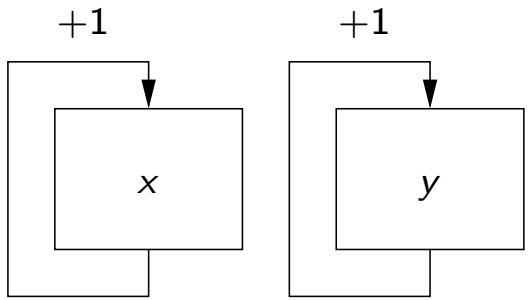
Beispiel

Ein System $C = (S, I, R)$ aus zwei Zählern x und y, die unabhängig voneinander (einzeln oder auch gemeinsam) erhöht werden können.

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

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

$$R(\langle x,y\rangle, \langle x',y'\rangle) :\Leftrightarrow$$
$$(x' = x+1 \wedge y' = y) \vee$$
$$(x' = x \wedge y' = y+1) \vee$$
$$(x' = x+1 \wedge y' = y+1)$$

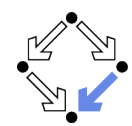


- Unendlich viele Anfangszustände:

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

- In jedem Zustand drei Möglichkeiten:
  - x wird erhöht und y bleibt gleich.
  - x bleibt gleich und y wird erhöht.
  - x und y werden beide erhöht.

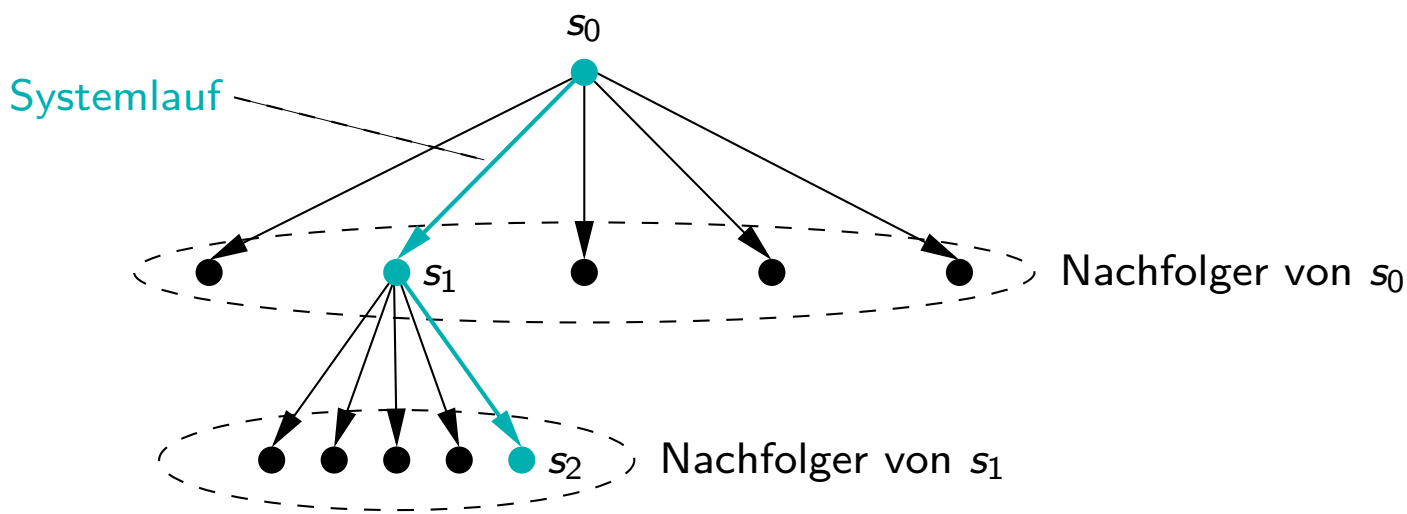
$$[x = 2, y = 3] \rightarrow [x = 3, y = 3]$$
$$\rightarrow [x = 2, y = 4]$$
$$\rightarrow [x = 3, y = 4]$$



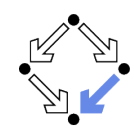
Systemläufe

Gegeben sei ein Transitionssystem $T = (S, I, R)$.

- Systemlauf: eine (endliche oder unendliche) Folge $s_0 \to s_1 \to s_2 \to \ldots$ von Zuständen in S.
  - s_0 ist ein Anfangszustand: $I(s_0)$.
  - $s_i \to s_{i+1}$ ist ein Zustandsübergang: $R(s_0, s_1)$.
  - Endet die Folge in einem Zustand s_n, dann hat dieser Zustand keinen Nachfolgerzustand: $\neg R(s_n, s')$, für alle $s' \in S$.



Ein Transitionssystem kann durch einen gerichteten Graphen mit den Zuständen als Knoten und den Übergängen als Kanten visualisiert werden.



Beispiel

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

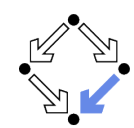
$$S := \mathbb{Z} \times \mathbb{Z}$$

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

$$\begin{aligned} R(\langle x, y\rangle, \langle x', y'\rangle) :\Leftrightarrow \\ (x' = x + 1 \wedge y' = y) \vee \\ (x' = x \wedge y' = y + 1) \vee \\ (x' = x + 1 \wedge y' = y + 1) \end{aligned}$$

$$\begin{aligned} [x = 2, y = 2] &\to [x = 3, y = 2] \to [x = 4, y = 2] \to [x = 4, y = 3] \to \ldots \\ &\to [x = 2, y = 3] \to [x = 3, y = 3] \to [x = 4, y = 3] \to \ldots \\ &\to [x = 2, y = 3] \to [x = 2, y = 4] \to [x = 2, y = 5] \to \ldots \\ &\to \ldots \end{aligned}$$

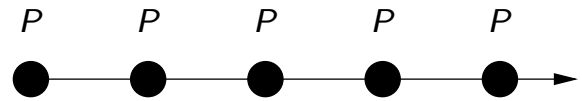
Unendlich viele Systemläufe vom gleichen Anfangszustand aus.



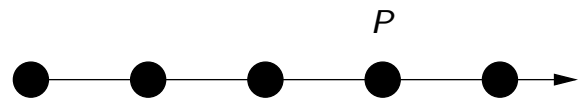
Systemeigenschaften

Die Eigenschaften eines Transitionssystems lassen sich als Formeln der linearen temporalen Logik (LTL) spezifizieren.

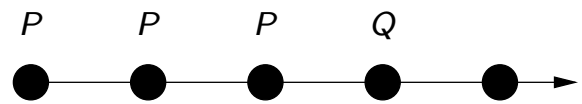
- Ein System S erfüllt eine LTL-Spezifikation P, wenn jeder mögliche Lauf von S die Eigenschaft P aufweist.
- Aktion: A
  - Klassische logische Formeln mit Variablen $x, y, \ldots$ und $x', y', \ldots$.
  - Das erste Zustandspaar (s_0, s_1) des Laufs erfüllt A mit $x, y, \ldots$ interpretiert in s_0 und $x', y', \ldots$ interpretiert in s_1.
- Always: $\Box P$



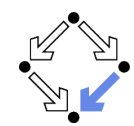
  - Der Lauf erfüllt ab jeder Position i die Eigenschaft P.
- Eventually: $\Diamond P$



  - Der Lauf erfüllt ab mindestens einer Position i die Eigenschaft P.
- Until: P U Q



  - Der Lauf erfüllt ab mindestens einer Position i die Eigenschaft Q, ab allen vorigen Positionen $j < i$ erfüllt er die Eigenschaft P.



Beispiel

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

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

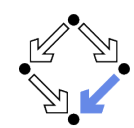
$$I(x, y) :\Leftrightarrow x = y \wedge y \geq 0$$

$$R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$$
$$(x' = x + 1 \wedge y' = y) \vee$$
$$(x' = x \wedge y' = y + 1) \vee$$
$$(x' = x + 1 \wedge y' = y + 1)$$

- Sicherheit: $\Box(x \geq 0 \wedge y \geq 0)$
  - Sowohl x als auch y werden nie negativ.
  - Das System erfüllt die Spezifikation, da jeder Lauf die Eigenschaft hat.
- Lebendigkeit: $\Diamond x \geq 1$.
  - Variable x wird irgendwann einmal einen Wert größer gleich 1 haben.
  - Das System erfüllt die Spezifikation nicht, da folgender Lauf die Eigenschaft nicht aufweist:

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

Lebendigkeitseigenschaften werden oft durch „unfaire“ Läufe verletzt; solche wollen wir ausschließen.



Fairness

- Unendlichkeit: Infinite P

  Infinite $P :\Leftrightarrow \Box\Diamond P$.

  - Ab jeder Position i gibt es eine Position $j \geq i$, ab der P gilt.
  - Die Eigenschaft P gilt unendlich oft.
- Stabilität: Stable P

  Stable $P :\Leftrightarrow \Diamond\Box P$.

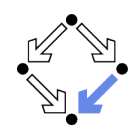
  - Ab einer Position i gilt ab allen Positionen $j \geq i$ die Eigenschaft P.
  - Die Eigenschaft P gilt irgendwann permanent.
- Ausführbarkeit: Enabled A
  - Aktion A beschreibt eine im aktuellen Zustand s ausführbare Transition: es gibt einen Zustand s' mit $R(s, s')$ sodass $A(s, s')$.
- Schwache (Weak) Fairness: WF A

  WF $A :\Leftrightarrow$ Stable (Enabled A) $\Rightarrow$ Infinite A.

  - Ist A ab einem gewissen Zeitpunkt permanent bereit, dann wird A auch (unendlich oft) ausgeführt.
- Starke (Strong) Fairness: SF A

  SF $A :\Leftrightarrow$ Infinite (Enabled A) $\Rightarrow$ Infinite A.

  - Ist A unendlich oft bereit, dann wird A auch (unendlich oft) ausgeführt.



Beispiel

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

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

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

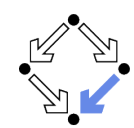
$$R(\langle x,y\rangle, \langle x',y'\rangle) :\Leftrightarrow$$
$$(x' = x+1 \wedge y' = y) \vee$$
$$(x' = x \wedge y' = y+1) \vee$$
$$(x' = x+1 \wedge y' = y+1)$$

- Lebendigkeit unter der Bedingung von Weak Fairness:

$$(\text{WF } x' = x+1) \Rightarrow \Diamond x \geq 1$$

  - Ist die Aktion $x' = x+1$ ab einem gewissen Zeitpunkt permanent bereit, wird sie unendlich oft ausgeführt.
  - Die Aktion ist immer bereit (Enabled $x' = x+1 \equiv \mathit{true}$) und wird daher unendlich oft ausgeführt.
  - Unter dieser Annahme gilt irgendwann $x \geq 1$ ($\Diamond x \geq 1$).

Es ist Aufgabe der Implementierung (z.B. des Prozess-Schedulers), die geforderten Fairness-Eigenschaften zu realisieren.



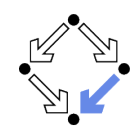
Die Temporale Logik der Aktionen (TLA)

- Leslie Lamport (Microsoft Research seit 2001).
  - ACM Turing Award 2013.
  - Theorie verteilter Systeme.
  - Verschiedene verteilte Algorithmen.
- TLA Spezifikation eines Systems:

  $$I \wedge \Box[R]_x \wedge \mathrm{WF}_x(A) \wedge \ldots$$

  - Anfangsbedingung I.
    - Formel mit Systemvariablen $x_1, \ldots, x_n$.
  - Zustandsüberführungsrelation R:
    - Formel mit Variablen $x_1, \ldots, x_n$ und $x'_1, \ldots, x'_n$.
    - Vektor $x = \langle x_1, x_2, \ldots, x_n \rangle$ aller Systemvariablen.
    - $[R]_x \equiv (R \vee x = x')$
    - $x = x'$: Stotterschritt (nichts ändert sich).
  - Fairnessbedingungen:
    - Konjunktion von Formeln $\mathrm{WF}_x(A)$ und/oder $\mathrm{SF}_x(A)$ für Aktionen A.

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

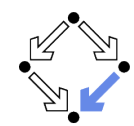


Beispiel

$$
\begin{aligned}
X \equiv & \wedge x' = x + 1 \\
& \wedge y' = y \\
Y \equiv & \wedge y' = y + 1 \\
& \wedge x' = x \\
S \equiv & \wedge (x = 0) \wedge (y = 0) \\
& \wedge \Box[X \vee Y]_{\langle x,y \rangle} \\
& \wedge \mathrm{WF}_{\langle x,y \rangle}(X) \wedge \mathrm{WF}_{\langle x,y \rangle}(Y)
\end{aligned}
$$

$$[x = 0, x = 0] \to [x = 1, y = 0] \to [x = 1, y = 0] \to [x = 1, y = 1] \to \ldots$$

Ein System wird in strukturierter Art und Weise durch logische Zusammensetzung von Aktionen beschrieben.



Verfeinerung von Spezifikationen

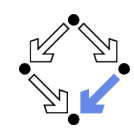
$$T \equiv (x = 0) \wedge \Box[x' = x + 1]_x \wedge \mathrm{WF}_x(x' = x + 1)$$

- Abstraktes System-Modell T.
  - Systemvariable x
  - Lauf $[x = 0] \rightarrow^* [x = 1] \rightarrow^* [x = 2] \rightarrow^* [x = 3] \rightarrow^* \ldots$
  - Stotterschritte $[x = a] \rightarrow^* [x = b]$:

$$[x = a] \rightarrow [x = a] \rightarrow [x = a] \rightarrow \ldots \rightarrow [x = a] \rightarrow [x = b]$$

- Konkretes System-Modell S (wie vorher):
  - Systemvariablen x, y.
  - Lauf $[x = 0, y = 0] \rightarrow^* [x = 0, y = 1] \rightarrow^* [x = 1, y = 1] \rightarrow^* \ldots$
- Logische Implikation $S \Rightarrow T$:
  - Jeder Lauf von S ist auch ein Lauf von T.
  - Änderungen von y sind Stotterschritte für T.
  - Aussagen, die für T bewiesen wurden, gelten auch für S.

TLA Formeln erlauben Verfeinerung von abstrakten zu konkreten Modellen.



TLA+

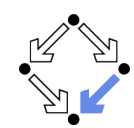
TLA ist nicht nur eine abstrakte Logik.

- TLA+: Eine formale Spezifikationssprache auf Basis von TLA.
  - Werte aus der klassischen Mengenlehre (kein statisches Typ-System).
  - Verwendet z.B. von Amazon Web Services.

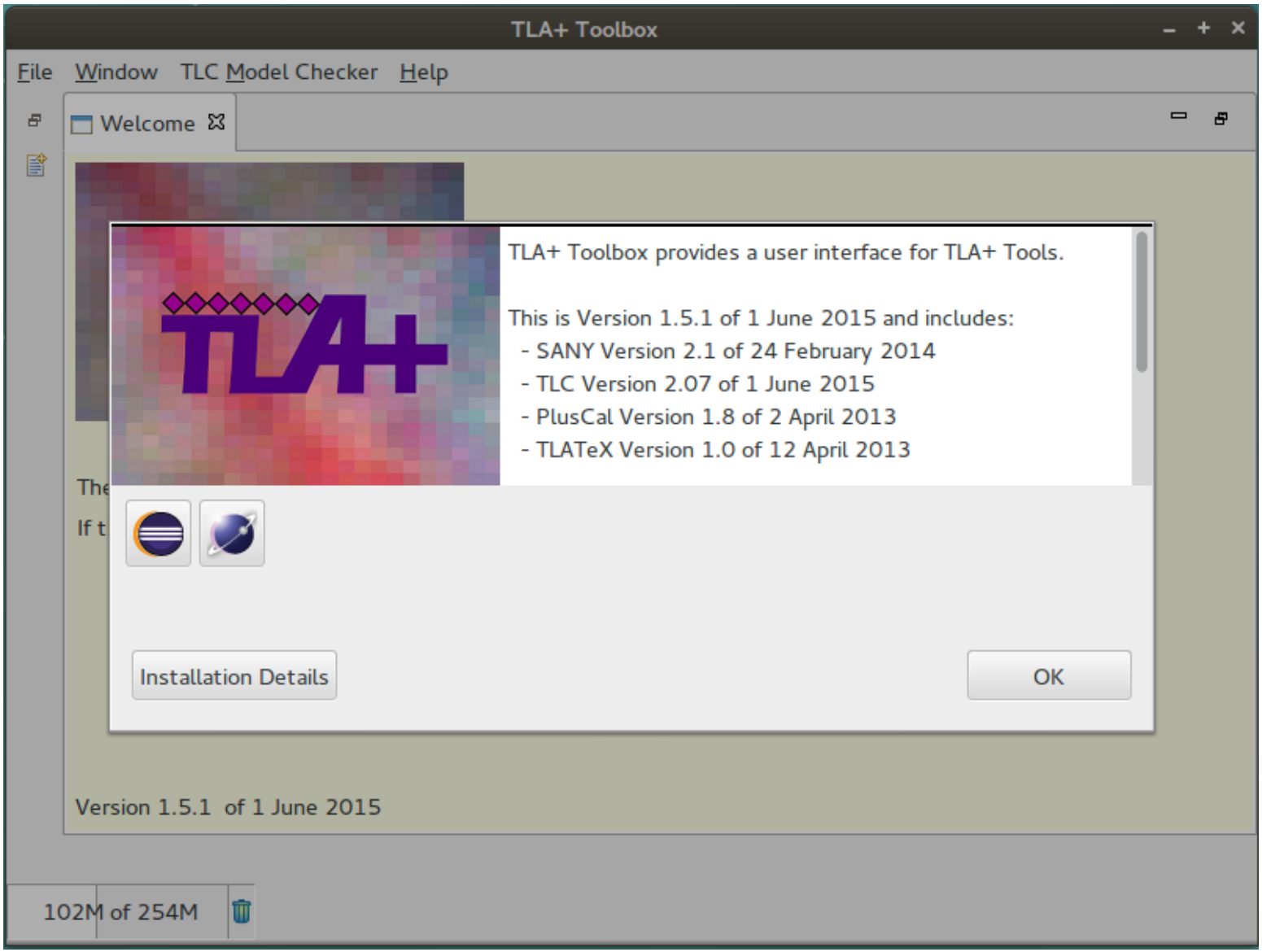
    Chris Newcombe et al. *How Amazon Web Services Uses Formal Methods.* Communications of the ACM, vol. 58 no. 4, pages 66-73, April 2015. `https://doi.org/10.1145/2699417`

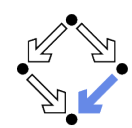
- TLA+ Toolbox: eine IDE für verschiedene TLA+-Werkzeuge.
  - Schreiben und Syntax-Überprüfung von TLA+ Spezifikationen.
  - Pretty-Printer zur Erzeugung von LaTeX-Dokumenten.
  - Übersetzer von der algorithmischen Sprache PlusCal nach TLA+.
  - Simulation und Model Checking von TLA+-Spezfikationen.
  - Konstruktion und Überprüfung von TLA+Beweisen.

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



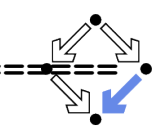
TLA+ Toolbox


TLA+ Toolbox
File Window TLC Model Checker Help
Welcome
TLA+ Toolbox provides a user interface for TLA+ Tools.
This is Version 1.5.1 of 1 June 2015 and includes:
- SANY Version 2.1 of 24 February 2014
- TLC Version 2.07 of 1 June 2015
- PlusCal Version 1.8 of 2 April 2013
- TLATeX Version 1.0 of 12 April 2013
Installation Details
OK
Version 1.5.1 of 1 June 2015
102M of 254M



Beispiel (Textdatei)

```
---------------------------- MODULE Counter ----------------------------
EXTENDS Naturals
VARIABLE x,y

(* the initial state condition *)
I == x = 0 /\ y = 0

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 *)

(* the whole specification *)
C == I /\ [][R]_var /\ WF_var(X) /\ WF_var(Y)

(* some properties *)
NotNegative == [](x >= 0 /\ y >= 0)
BecomeOne   == <>(x = 1 /\ y = 1)
=============================================================================
```



Spezifikation eines Systems C und einiger Eigenschaften.

Beispiel (LaTeX)

MODULE *Counter*

EXTENDS *Naturals*

VARIABLE $x,\ y$

the initial state condition

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

$$\begin{array}{lll} X \triangleq & \land\, x' = x + 1 & \text{increment } x \\ & \land\, y' = y & \\ Y \triangleq & \land\, x' = x & \text{increment } y \\ & \land\, y' = y + 1 & \\ R \triangleq & \lor\, X & \text{increment } x \text{ or } y \\ & \lor\, Y & \end{array}$$

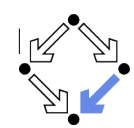
$$var \triangleq \langle x,\ y \rangle \quad \text{the system variables}$$

the whole specification

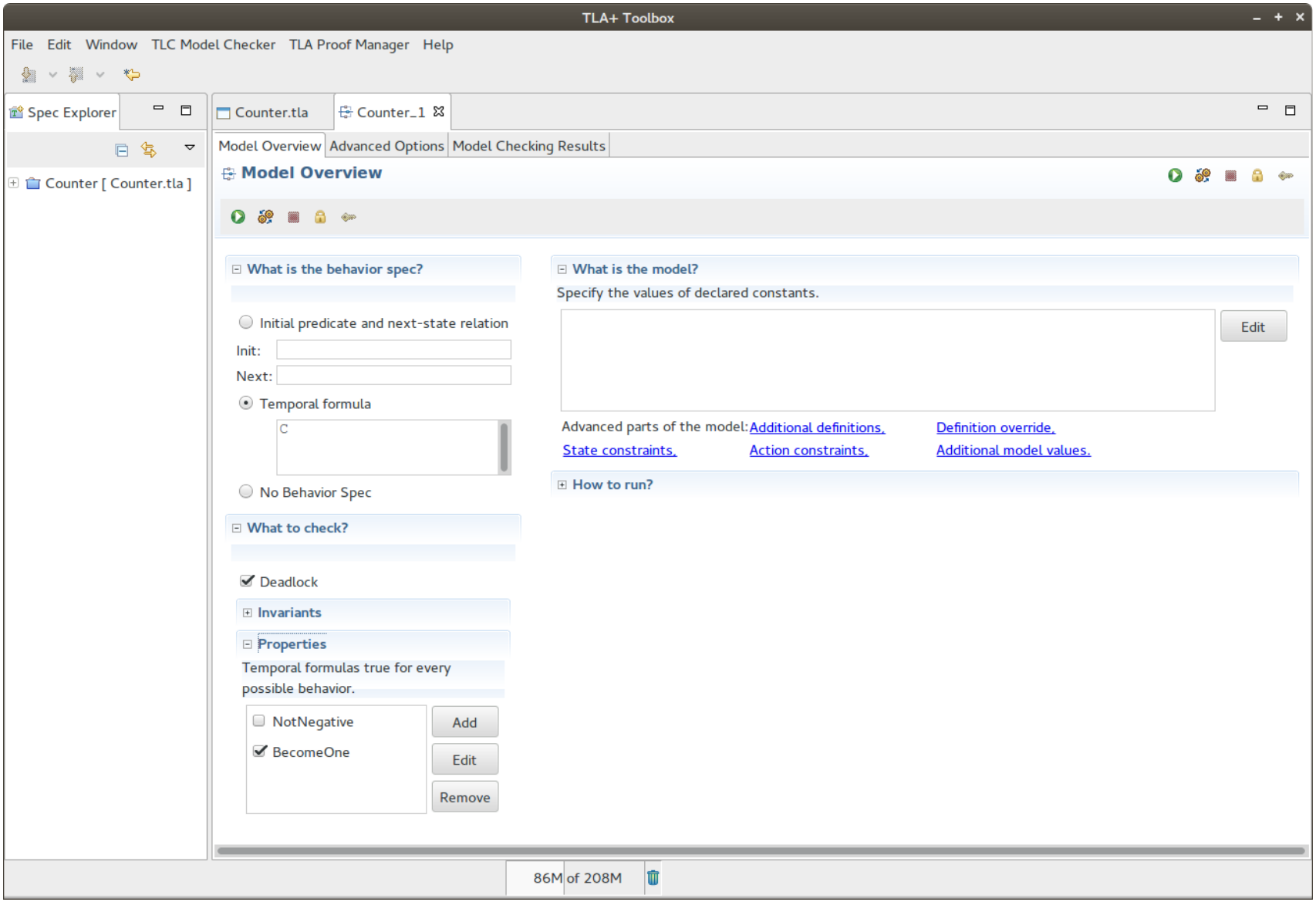
$$C \triangleq I \land \Box[R]_{var} \land \mathrm{WF}_{var}(X) \land \mathrm{WF}_{var}(Y)$$

some properties

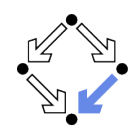
$$NotNegative \triangleq \Box(x \geq 0 \land y \geq 0)$$
$$BecomeOne \triangleq \Diamond(x = 1 \land y = 1)$$



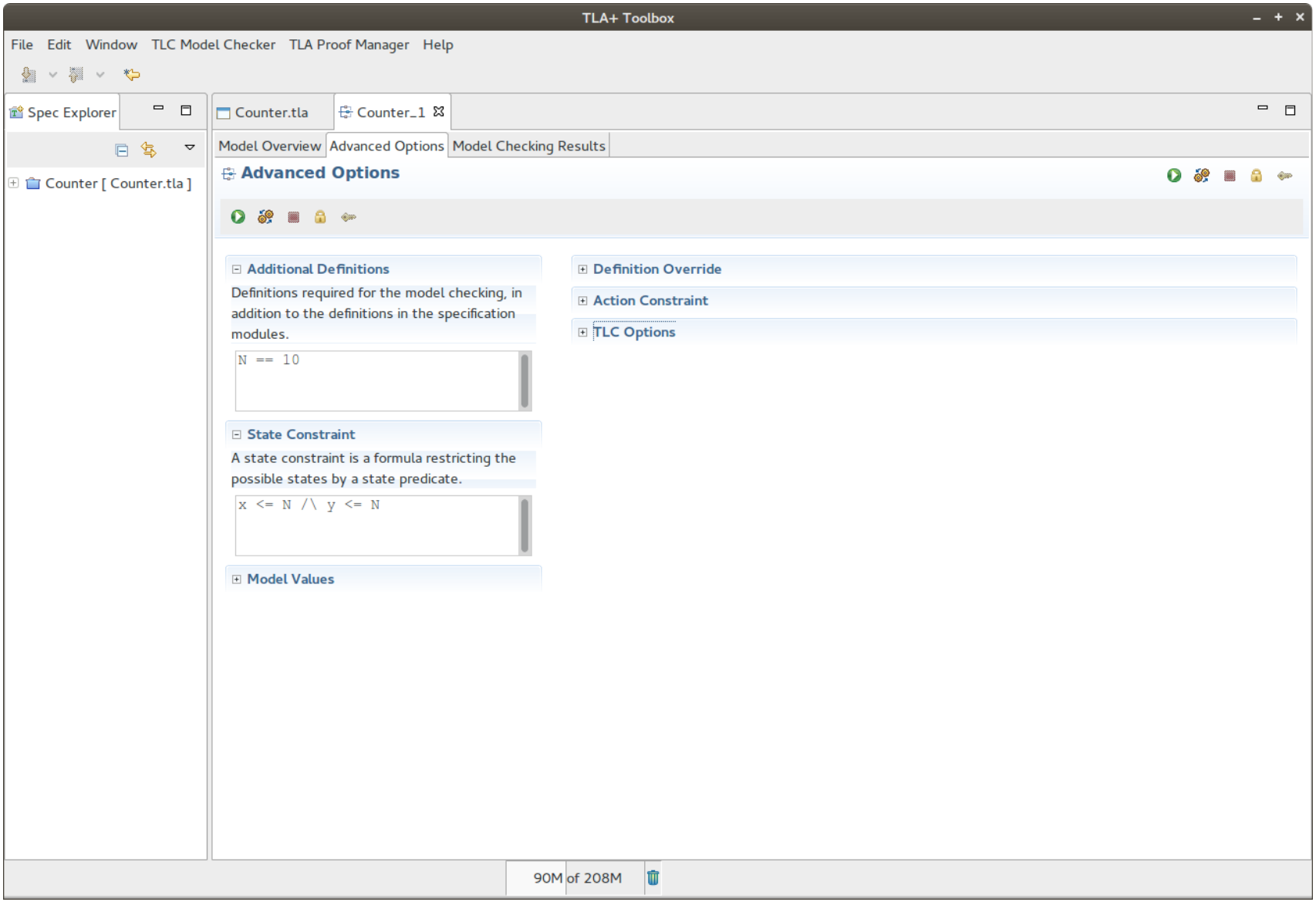
Der TLC Model Checker



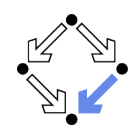
Wähle Spezifikation und zu überprüfende Eigenschaften.



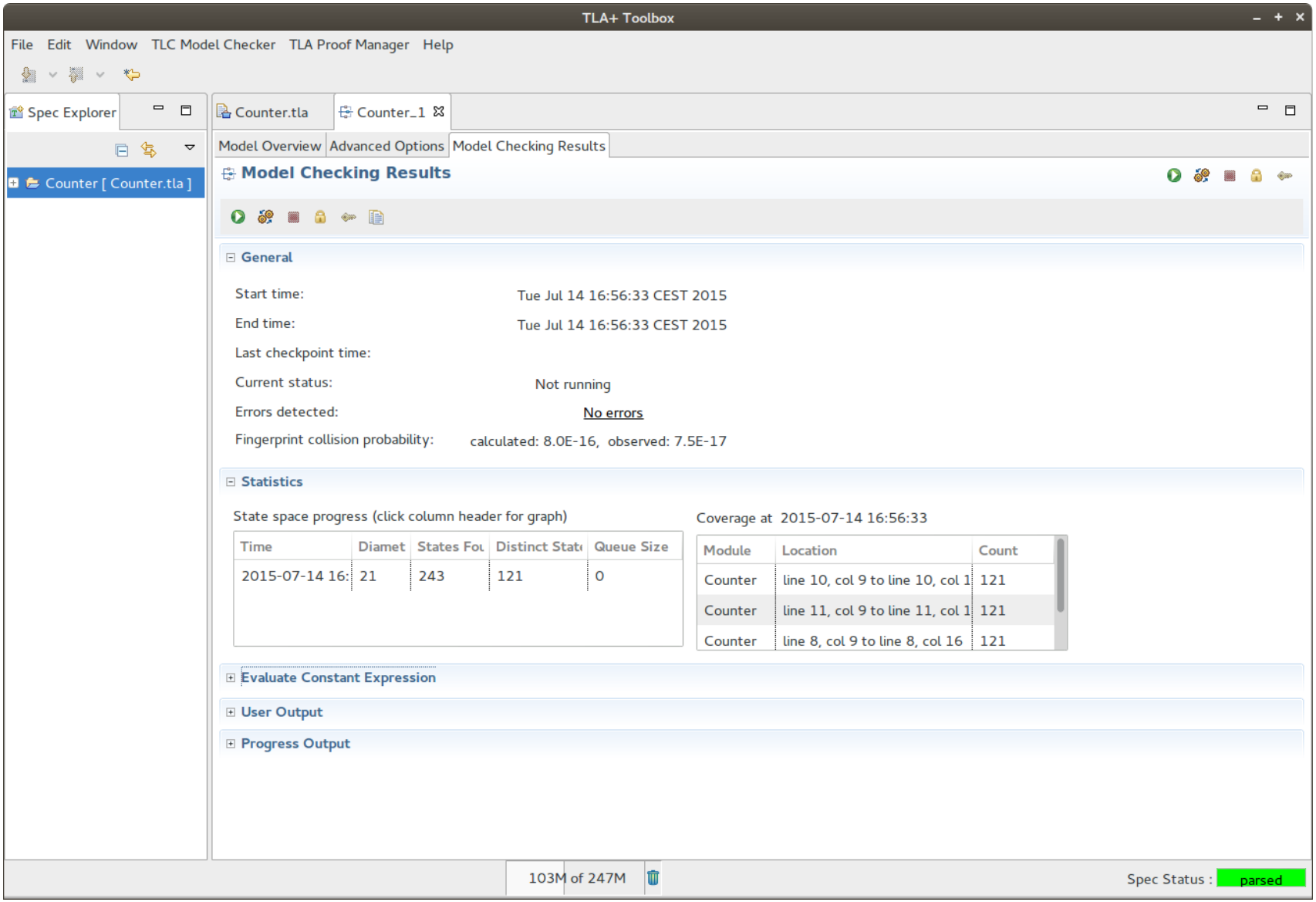
Der TLC Model Checker



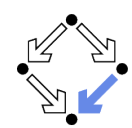
Wenn notwendig, beschränke Zustandsraum auf endliche Teilmenge.



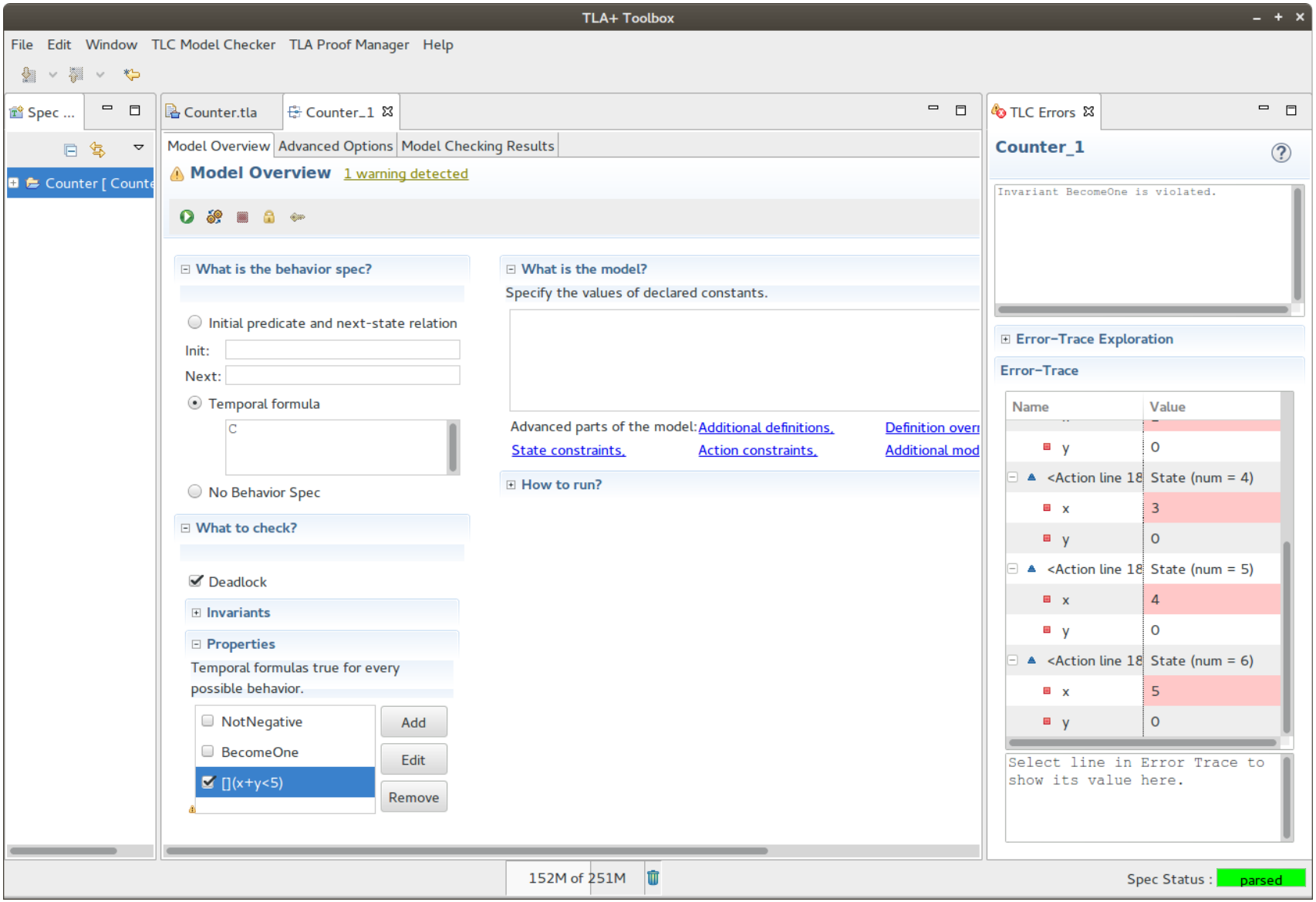
Der TLC Model Checker



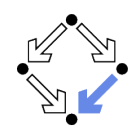
Überprüfe die Eigenschaften.



Der TLC Model Checker

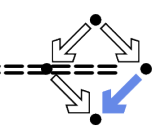


Im Fehlerfall wird ein verletzender Systemlauf dargestellt.



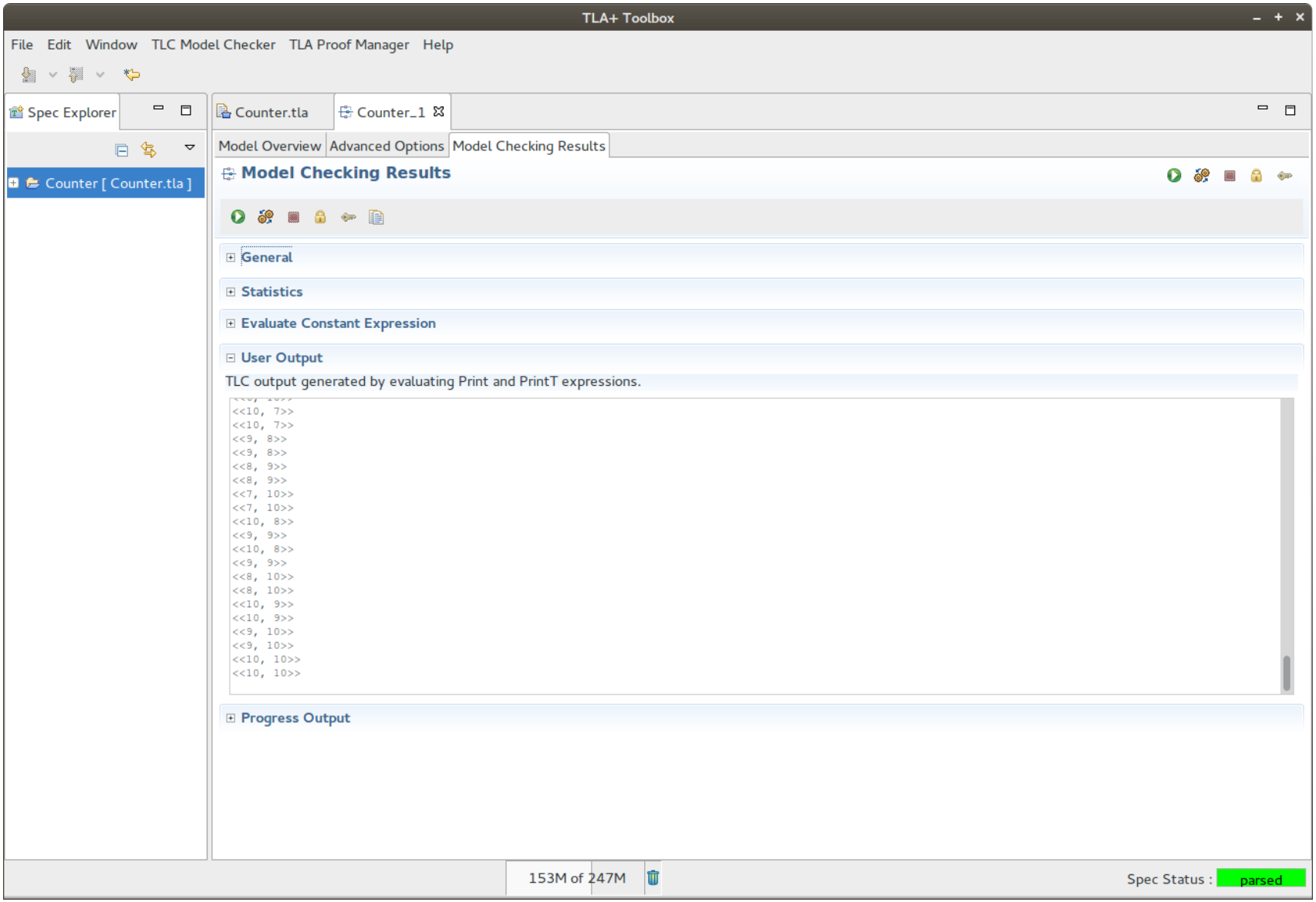
Beispiel

```
---------------------------- MODULE Counter ----------------------------
EXTENDS Naturals, TLC
VARIABLE x,y

(* the initial state condition *)
I == x = 0 /\ y = 0

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 *)

(* the whole specification *)
C == I /\ [][R /\ PrintT(<<x,y>>)]_var /\ WF_var(X) /\ WF_var(Y)

(* some properties *)
NotNegative == [](x >= 0 /\ y >= 0)
BecomeOne   == <>(x = 1 /\ y = 1)
=============================================================================
```

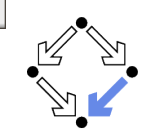


Mit Hilfe von Benutzer-Ausgaben kann das Modell validiert werden.

Der TLC Model Checker



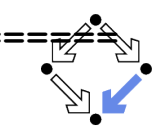
Die besuchten Zustände werden ausgedruckt.



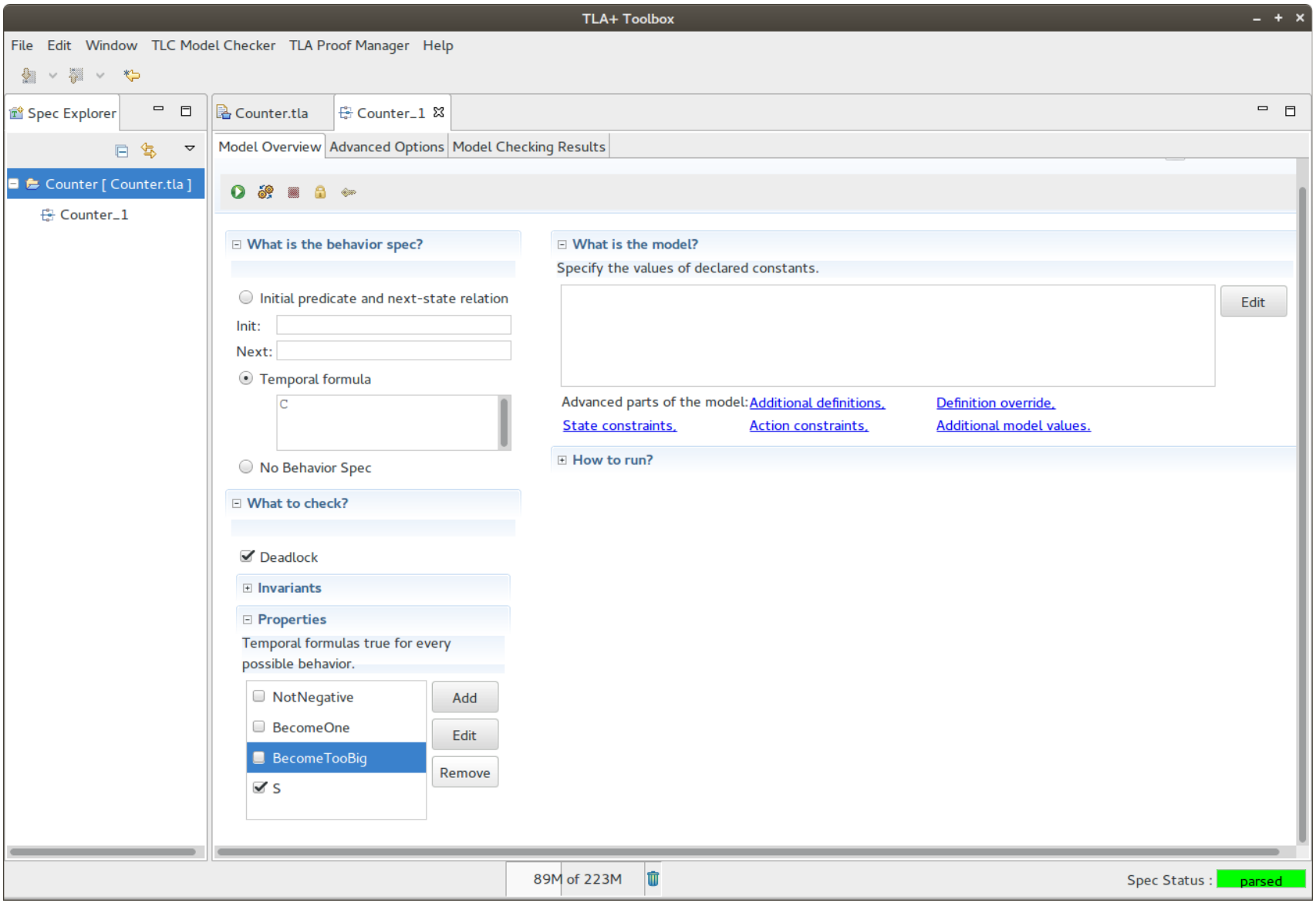
Der TLC Model Checker

```
---------------------------- MODULE Counter ----------------------------
EXTENDS Naturals
VARIABLE x,y

(* the initial state condition *)
I == x = 0 /\ y = 0

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 *)

(* the whole specification *)
C == I /\ [][R]_var /\ WF_var(X) /\ WF_var(Y)

(* another system *)
S == (x = 0) /\ [][x' = x+1]_x /\ WF_x(x' = x+1)
=============================================================================
```

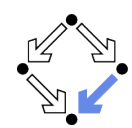
Spezifikation eines zweiten (abstrakteren) Systems S.



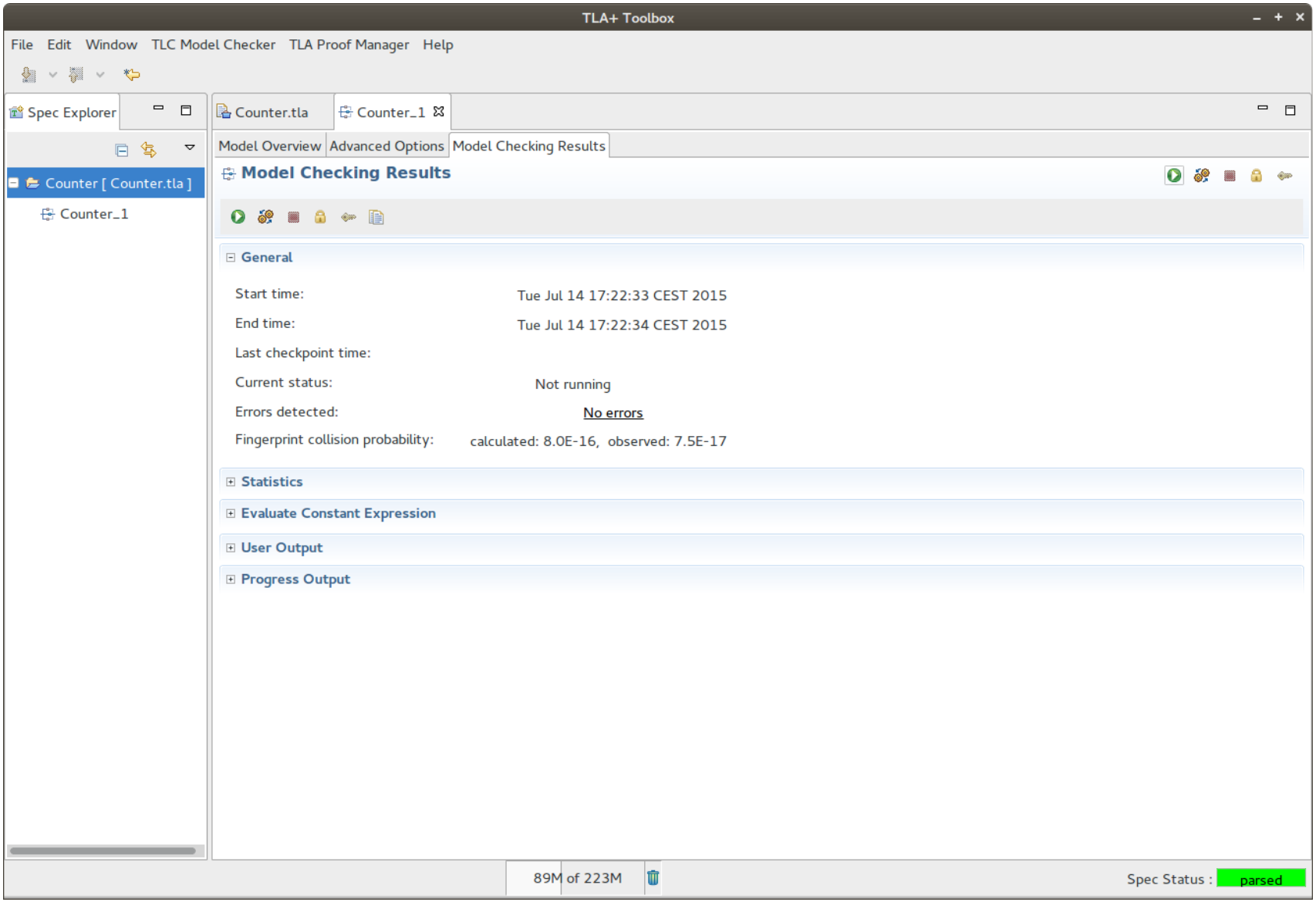
Der TLC Model Checker



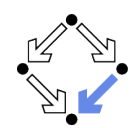
Überprüfung ob C die Spezifikation S verfeinert ($C \Rightarrow S$).



Der TLC Model Checker

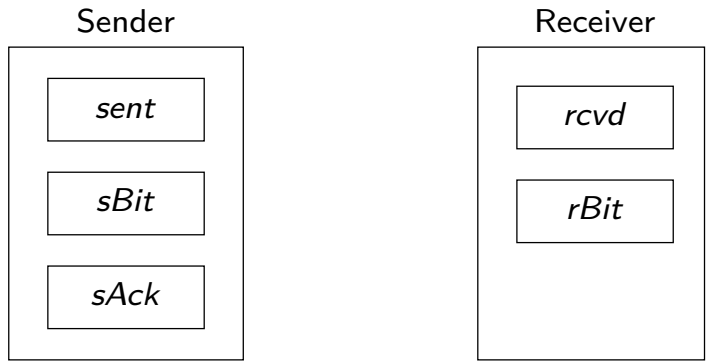


System C ist eine gültige Verfeinerung von S.



Das Alternierende Bit Protokoll (Gemeinsamer Speicher)

Übertragung einer Folge von Bits zwischen zwei asynchronen Komponenten mittels gemeinsamer Register.



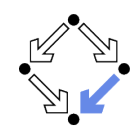
var $sBit \in \{0,1\}, sAck \in \{0,1\}, rBit \in \{0,1\}, sent \in Data, rcvd \in Data$
init $sBit = sAck = rBit$

Sender :	||	Receiver :
loop		**loop**
wait $sAck = sBit$		**wait** $rBit \neq sBit$
$sent = \ldots; sBit = 1 - sBit$		$rcvd = sent; rBit = sBit$
		$sAck = rBit$

- Korrektheitseigenschaft:

  $$\forall d \in Data.\ sent = d \wedge sBit \neq sAck \rightsquigarrow rcvd = d$$

  - Response: $P \rightsquigarrow Q \equiv \Box(P \Rightarrow \Diamond Q)$
  - Auf Anforderung P folgt immer Antwort Q.



Das Alternierende Bit Protokoll (Gemeinsamer Speicher)

MODULE $ABCorrectness$

EXTENDS $Naturals$
CONSTANTS $Data$
VARIABLES $sBit$, $sAck$, $rBit$, $sent$, $rcvd$

$$ABCInit \triangleq sBit \in \{0, 1\} \land sAck = sBit \land rBit = sBit \land sent \in Data \land rcvd \in Data$$

$$CSndNewValue(d) \triangleq \land sAck = sBit \land sent' = d \land sBit' = 1 - sBit \\ \land \text{UNCHANGED } \langle sAck, rBit, rcvd \rangle$$

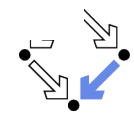
$$CRcvMsg \triangleq \land rBit \neq sBit \land rBit' = sBit \land rcvd' = sent \\ \land \text{UNCHANGED } \langle sBit, sAck, sent \rangle$$

$$CRcvAck \triangleq \land rBit \neq sAck \land sAck' = rBit \\ \land \text{UNCHANGED } \langle sBit, rBit, sent, rcvd \rangle$$

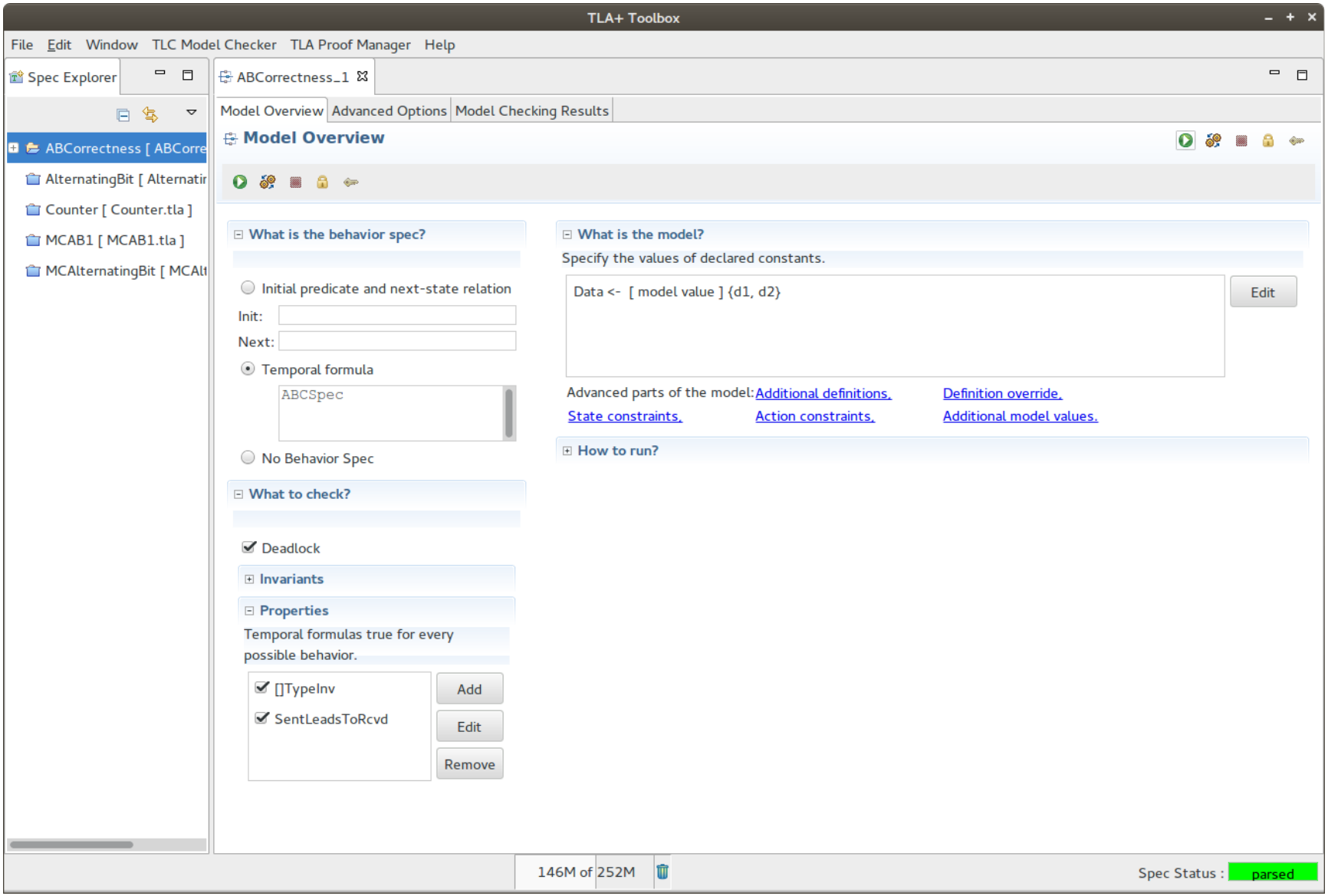
$$ABCNext \triangleq (\exists\, d \in Data : CSndNewValue(d)) \lor CRcvMsg \lor CRcvAck$$

$$cvars \triangleq \langle sBit, sAck, rBit, sent, rcvd \rangle$$
$$ABCSpec \triangleq ABCInit \land \Box[ABCNext]_{cvars} \land \text{WF}_{cvars}(CRcvMsg) \land \text{WF}_{cvars}(CRcvAck)$$

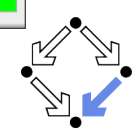
$$TypeInv \triangleq sBit \in \{0, 1\} \land sAck \in \{0, 1\} \quad \land rBit \in \{0, 1\} \land sent \in Data \land rcvd \in Data$$
$$SentLeadsToRcvd \triangleq \forall\, d \in Data : (sent = d) \land (sBit \neq sAck) \rightsquigarrow (rcvd = d)$$



Model Checking des Protokolls (Gemeinsamer Speicher)

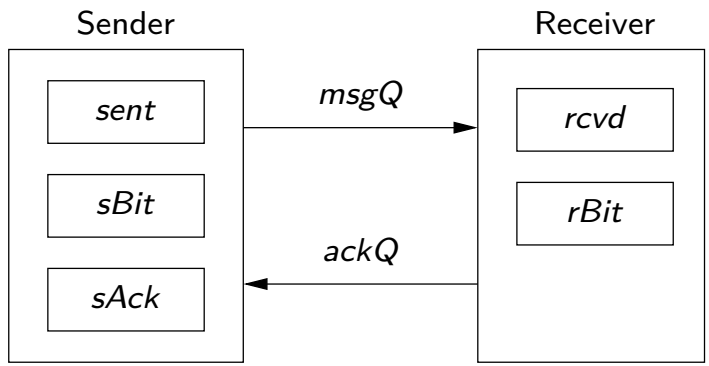


Kein Fehler: das Protokoll erfüllt die Spezifikation.



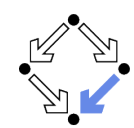
Das Alternierende Bit Protokoll (Verteilt)

Übertragung einer Folge von Bits zwischen zwei asynchronen Komponenten mittels *verlustbehafteter* Kommunikationskanäle.



- *msgQ* : überträgt Nachrichten $\langle sBit, sent \rangle$.
  - Neue Werte nach Aktualisierung durch Sender.
- *ackQ* : überträgt Nachrichten *rBit*.
  - Neue Werte nach Aktualisierung durch Receiver.

Das Protokoll soll die gleiche Korrektheitseigenschaft wie das Protokoll für gemeinsamen Speicher aufweisen.



Das Alternierende Bit Protokoll (Verteilt)

MODULE *AlternatingBit*

EXTENDS *Naturals*, *Sequences*

CONSTANTS *Data*

VARIABLES $msgQ$, $ackQ$, $sBit$, $sAck$, $rBit$, $sent$, $rcvd$

$$ABInit \triangleq \land msgQ = \langle\rangle \land ackQ = \langle\rangle \land sBit \in \{0, 1\} \land sAck = sBit \land rBit = sBit \land sent \in Data \land rcvd \in Data$$

.....

$$ABNext \triangleq \lor (\exists\, d \in Data : SndNewValue(d)) \lor ReSndMsg \lor RcvMsg \lor SndAck \lor RcvAck \lor LoseMsg \lor LoseAck$$

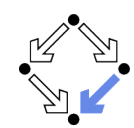
$$abvars \triangleq \langle msgQ, ackQ, sBit, sAck, rBit, sent, rcvd\rangle$$

$$ABSpec \triangleq \land ABInit \land \Box[ABNext]_{abvars} \land \mathrm{WF}_{abvars}(ReSndMsg) \land \mathrm{WF}_{abvars}(SndAck) \land \mathrm{SF}_{abvars}(RcvMsg) \land \mathrm{SF}_{abvars}(RcvAck)$$

$$ABTypeInv \triangleq \land msgQ \in Seq(\{0, 1\} \times Data) \land ackQ \in Seq(\{0, 1\}) \land sBit \in \{0, 1\} \land sAck \in \{0, 1\} \land rBit \in \{0, 1\} \land sent \in Data \land rcvd \in Data$$

INSTANCE *ABCorrectness*

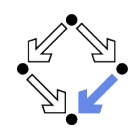
Der Kern der Spezifikation.



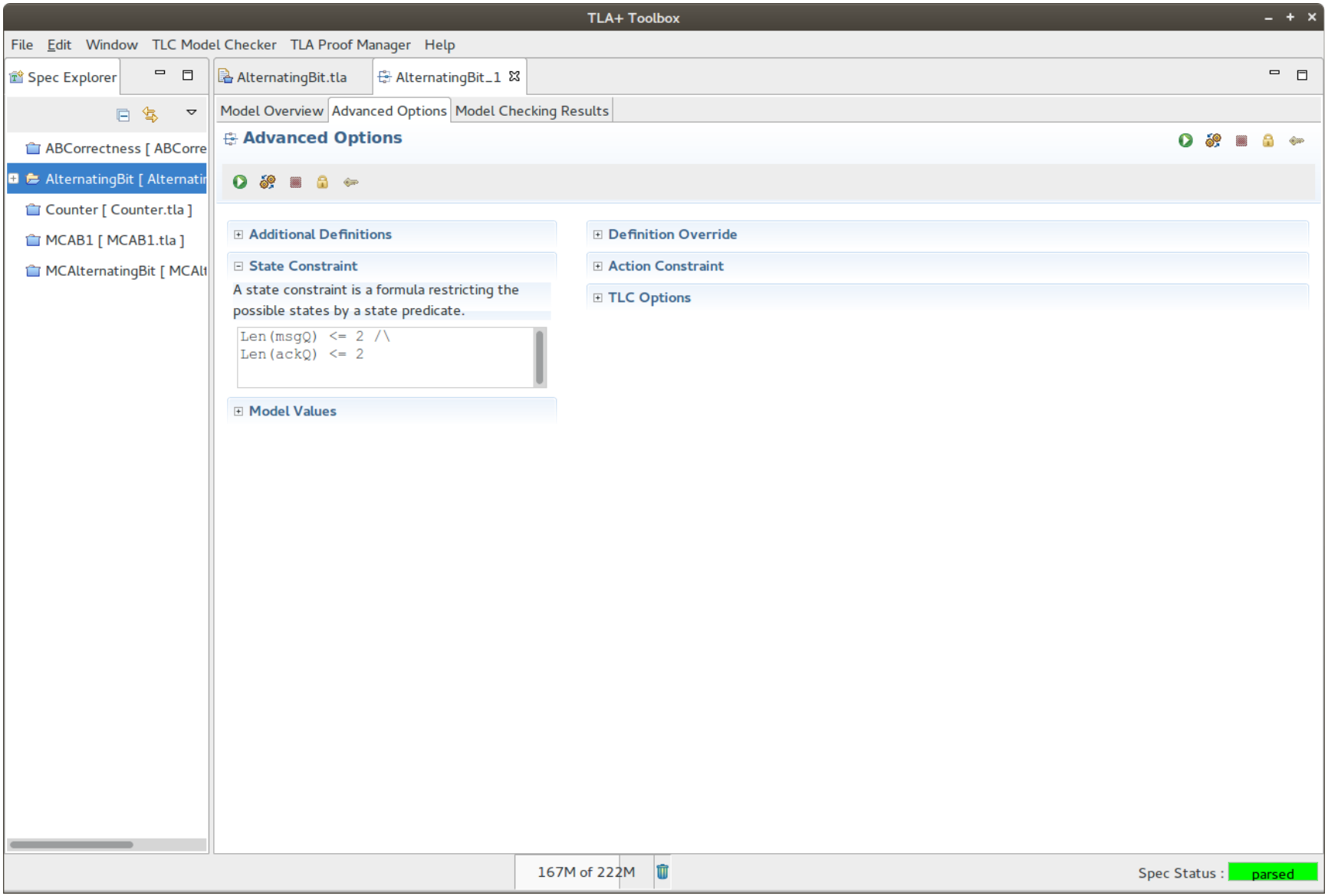
Das Alternierende Bit Protokoll (Verteilt)

$$
\begin{array}{lcl}
SndNewValue(d) & \triangleq & \land\ sAck = sBit \land sent' = d \land sBit' = 1 - sBit \\
 & & \land\ msgQ' = Append(msgQ, \langle sBit', d\rangle) \\
 & & \land\ \text{UNCHANGED}\ \langle ackQ, sAck, rBit, rcvd\rangle \\
ReSndMsg & \triangleq & \land\ sAck \neq sBit \\
 & & \land\ msgQ' = Append(msgQ, \langle sBit, sent\rangle) \\
 & & \land\ \text{UNCHANGED}\ \langle ackQ, sBit, sAck, rBit, sent, rcvd\rangle \\
RcvMsg & \triangleq & \land\ msgQ \neq \langle\rangle \land msgQ' = Tail(msgQ) \land rBit' = Head(msgQ)[1] \land rcvd' = Head(msgQ)[2] \\
 & & \land\ \text{UNCHANGED}\ \langle ackQ, sBit, sAck, sent\rangle \\
SndAck & \triangleq & \land\ ackQ' = Append(ackQ, rBit) \\
 & & \land\ \text{UNCHANGED}\ \langle msgQ, sBit, sAck, rBit, sent, rcvd\rangle \\
RcvAck & \triangleq & \land\ ackQ \neq \langle\rangle \land ackQ' = Tail(ackQ) \land sAck' = Head(ackQ) \\
 & & \land\ \text{UNCHANGED}\ \langle msgQ, sBit, rBit, sent, rcvd\rangle \\
Lose(q) & \triangleq & \land\ q \neq \langle\rangle \\
 & & \land\ \exists\, i \in 1\,..\,Len(q) : q' = [j \in 1\,..\,(Len(q)-1) \mapsto \text{IF}\ j < i\ \text{THEN}\ q[j]\ \text{ELSE}\ q[j+1]] \\
 & & \land\ \text{UNCHANGED}\ \langle sBit, sAck, rBit, sent, rcvd\rangle \\
LoseMsg & \triangleq & Lose(msgQ) \land \text{UNCHANGED}\ ackQ \\
LoseAck & \triangleq & Lose(ackQ)\ \land \text{UNCHANGED}\ msgQ
\end{array}
$$

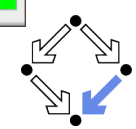
Die Aktionen der Spezifikation.



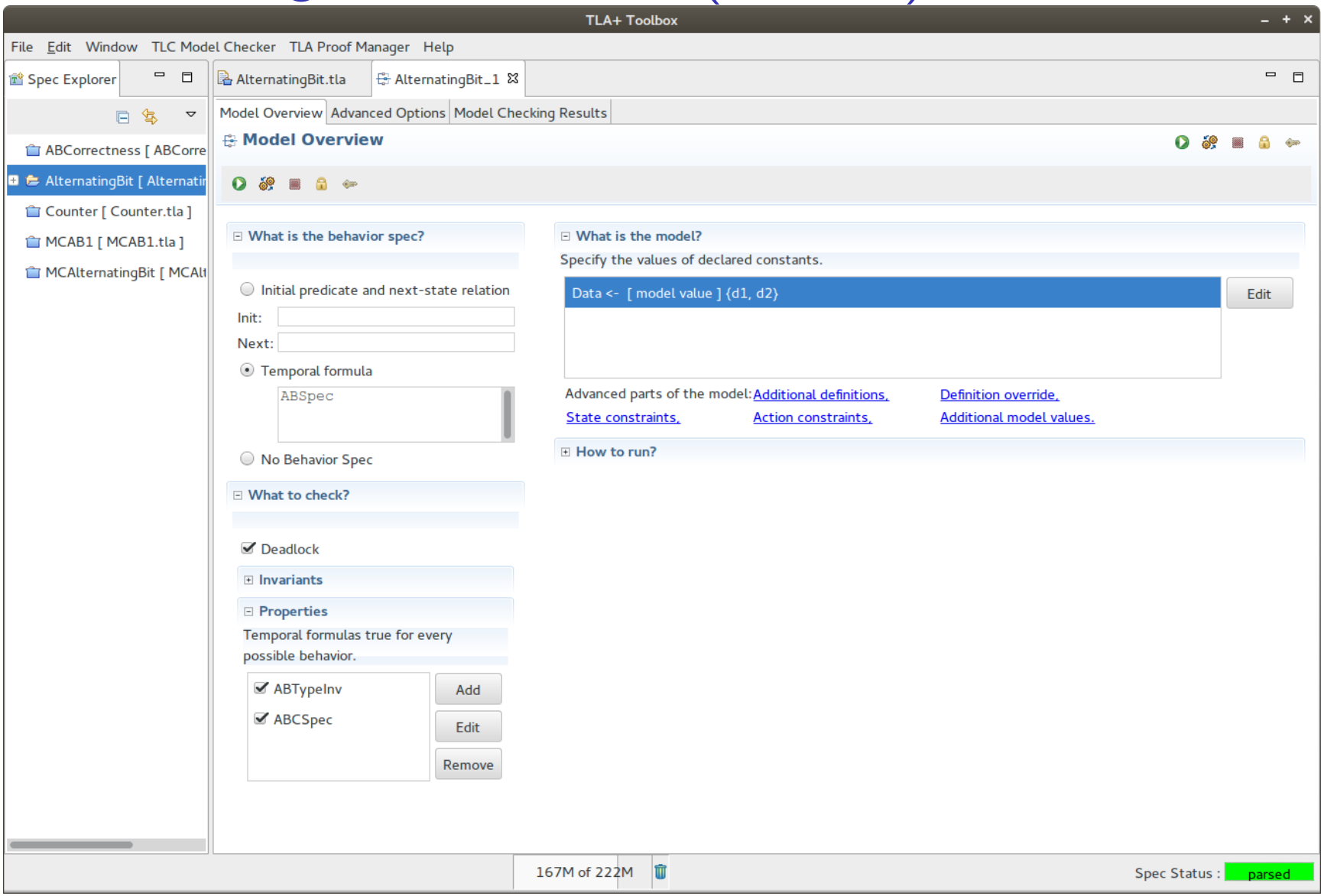
Zustandsraum des Protokolls (Verteilt)



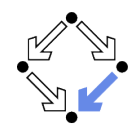
Einschränkung des Zustandsraums auf eine endliche Teilmenge.



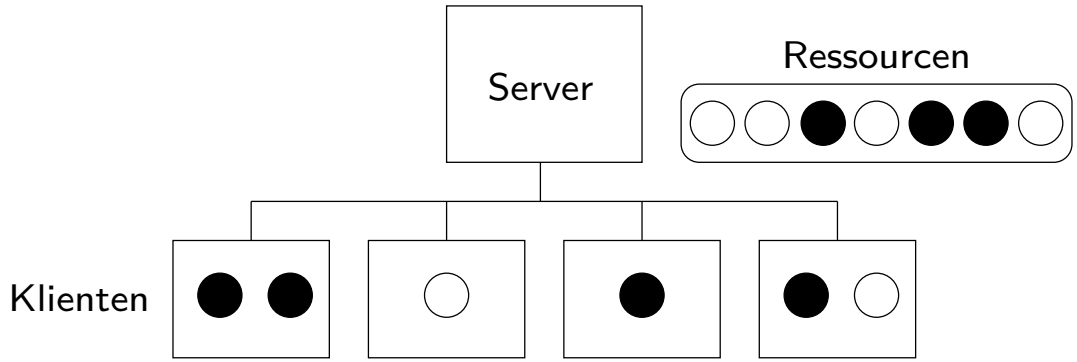
Model Checking des Protokolls (Verteilt)



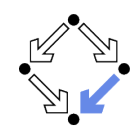
Kein Fehler: das Protokoll verfeinert die ursprüngliche Spezifikation und erbt damit automatisch seine Korrektheitseigenschaft.



Ein Verteilter Ressourcen-Zuteiler



- Ein Server teilt einer Menge von Klienten verschiedene Ressourcen zu.
- Ein Klient, der keine Ressourcen hält und keine ausstehenden Anfragen hat, kann eine bestimmte Menge der Ressourcen anfordern.
- Der Server kann dem Klienten einige oder alle der gewünschten Ressourcen zuteilen.
  - Ressourcenanfragen können in mehreren Teilen erledigt werden; der Klient kann möglicherweise bereits mit einem Teil weiterarbeiten.
- Der Klient kann eine Teilmenge der gehaltenen Ressourcen freigeben; letztendlich muss er alle gehaltenen Ressourcen wieder freigeben.
- Sicherheit: keine Ressource wird zwei Klienten gleichzeitig zugeteilt.
- Lebendigkeit: jede Ressourcenanfrage wird letztendlich erfüllt.

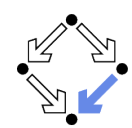


Ein Verteilter Ressourcen-Zuteiler

Das Verfahren operiert mit den folgenden Variablen.

- Server:
  - *unsat*[*c*]: die vom Klienten *c* angeforderten aber vom Server noch nicht zugeteilten Ressourcen.
  - *alloc*[*c*]: die vom Klienten *c* angeforderten Ressourcen, für die der Server dem Klienten eine Zuteilung gesendet hat.
  - *sched*: eine Liste der Klienten mit noch offenen Anfragen.
    - Ältere Anfragen scheinen in der Liste weiter vorne auf und werden bei der Zuteilung im Falle von Konflikten bevorzugt.
- Klient *c*:
  - *requests*[*c*]: die vom Klienten angeforderten Ressourcen, für die er vom Server noch keine Zuteilung erhalten hat.
  - *holding*[*c*]: die vom Klienten gehaltenen Ressourcen.
- Netzwerk:
  - *network* : die sich im Netzwerk befindlichen Nachrichten.

Da Nachrichten sich noch im Netzwerk befinden können, unterscheidet sich die Server-Sicht im allgemeinen von der Klienten-Sicht.



Ein Verteilter Ressourcen-Zuteiler

MODULE *DistributedAllocator*

EXTENDS *Naturals*, *Sequences*

CONSTANTS *Clients*, *Resources*

VARIABLES *unsat*, *alloc*, *sched*, *requests*, *holding*, *network*

$Messages \triangleq [type : \{\text{“request”}, \text{“allocate”}, \text{“return”}\}, clt : Clients, rsrc : \text{SUBSET } Resources]$

$Drop(seq, i) \triangleq SubSeq(seq, 1, i-1) \circ SubSeq(seq, i+1, Len(seq))$

$available \triangleq Resources \setminus (\text{UNION } \{alloc[c] : c \in Clients\})$

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$

$Init \triangleq$

$\land\ unsat = [c \in Clients \mapsto \{\}] \land alloc = [c \in Clients \mapsto \{\}]$

$\land\ requests = [c \in Clients \mapsto \{\}] \land holding = [c \in Clients \mapsto \{\}]$

$\land\ sched = \langle\rangle \land network = \{\}$

$Next \triangleq$

$\lor\ \exists\, m \in network : RReq(m) \lor RAlloc(m) \lor RRet(m)$

$\lor\ \exists\, c \in Clients, S \in \text{SUBSET } Resources : Request(c, S) \lor Allocate(c, S) \lor Return(c, S)$

$vars \triangleq \langle unsat, alloc, sched, requests, holding, network\rangle$

$Liveness \triangleq$

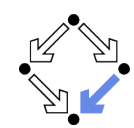
$\land\ \forall\, c \in Clients : \text{WF}_{vars}(requests[c] = \{\} \land Return(c, holding[c]))$

$\land\ \forall\, c \in Clients : \text{WF}_{vars}(\exists\, S \in \text{SUBSET } Resources : Allocate(c, S))$

$\land\ \forall\, m \in Messages : \text{WF}_{vars}(RReq(m)) \land \text{WF}_{vars}(RAlloc(m)) \land \text{WF}_{vars}(RRet(m))$

$Specification \triangleq Init \land \Box[Next]_{vars} \land Liveness$

Der Kern der Spezifikation.



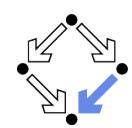
Ein Verteilter Ressourcen-Zuteiler

$RReq(m) \triangleq$
$\land\ m \in network \land m.type = \text{“request”}$
$\land\ alloc[m.clt] = \{\}$ * don't handle request messages *prematurely*(!)
$\land\ unsat' = [unsat \text{ EXCEPT } ![m.clt] = m.rsrc]$
$\land\ network' = network \setminus \{m\}$
$\land\ sched' = \text{IF } m.clt \in Range(sched) \text{ THEN } sched \text{ ELSE } Append(sched, m.clt)$
$\land\ \text{UNCHANGED } \langle alloc, requests, holding \rangle$

$RAlloc(m) \triangleq$
$\land\ m \in network \land m.type = \text{“allocate”}$
$\land\ holding' = [holding \text{ EXCEPT } ![m.clt] = @ \cup m.rsrc]$
$\land\ requests' = [requests \text{ EXCEPT } ![m.clt] = @ \setminus m.rsrc]$
$\land\ network' = network \setminus \{m\}$
$\land\ \text{UNCHANGED } \langle unsat, alloc, sched \rangle$

$RRet(m) \triangleq$
$\land\ m \in network \land m.type = \text{“return”}$
$\land\ alloc' = [alloc \text{ EXCEPT } ![m.clt] = @ \setminus m.rsrc]$
$\land\ network' = network \setminus \{m\}$
$\land\ \text{UNCHANGED } \langle unsat, sched, requests, holding \rangle$

Der Empfang von Nachrichten.



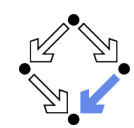
Ein Verteilter Ressourcen-Zuteiler

$$
\begin{array}{l}
Request(c,\ S) \triangleq \\
\quad \land\ requests[c] = \{\} \land holding[c] = \{\} \\
\quad \land\ S \neq \{\} \land requests' = [requests \text{ EXCEPT } ![c] = S] \\
\quad \land\ network' = network \cup \{[type \mapsto \text{“request”},\ clt \mapsto c,\ rsrc \mapsto S]\} \\
\quad \land\ \text{UNCHANGED } \langle unsat,\ alloc,\ sched,\ holding \rangle
\end{array}
$$

$$
\begin{array}{l}
Allocate(c,\ S) \triangleq \\
\quad \land\ S \neq \{\} \land S \subseteq available \cap unsat[c] \\
\quad \land\ \exists\, i \in \text{DOMAIN } sched : \\
\qquad \land\ sched[i] = c \\
\qquad \land\ \forall\, j \in 1 \,..\, i-1 : unsat[sched[j]] \cap S = \{\} \\
\qquad \land\ sched' = \text{IF } S = unsat[c] \text{ THEN } Drop(sched,\ i) \text{ ELSE } sched \\
\quad \land\ alloc' = [alloc \text{ EXCEPT } ![c] = @ \cup S] \\
\quad \land\ unsat' = [unsat \text{ EXCEPT } ![c] = @ \setminus S] \\
\quad \land\ network' = network \cup \{[type \mapsto \text{“allocate”},\ clt \mapsto c,\ rsrc \mapsto S]\} \\
\quad \land\ \text{UNCHANGED } \langle requests,\ holding \rangle
\end{array}
$$

$$
\begin{array}{l}
Return(c,\ S) \triangleq \\
\quad \land\ S \neq \{\} \land S \subseteq holding[c] \\
\quad \land\ holding' = [holding \text{ EXCEPT } ![c] = @ \setminus S] \\
\quad \land\ network' = network \cup \{[type \mapsto \text{“return”},\ clt \mapsto c,\ rsrc \mapsto S]\} \\
\quad \land\ \text{UNCHANGED } \langle unsat,\ alloc,\ sched,\ requests \rangle
\end{array}
$$

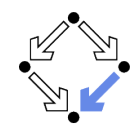
Das Verschicken von Nachrichten.



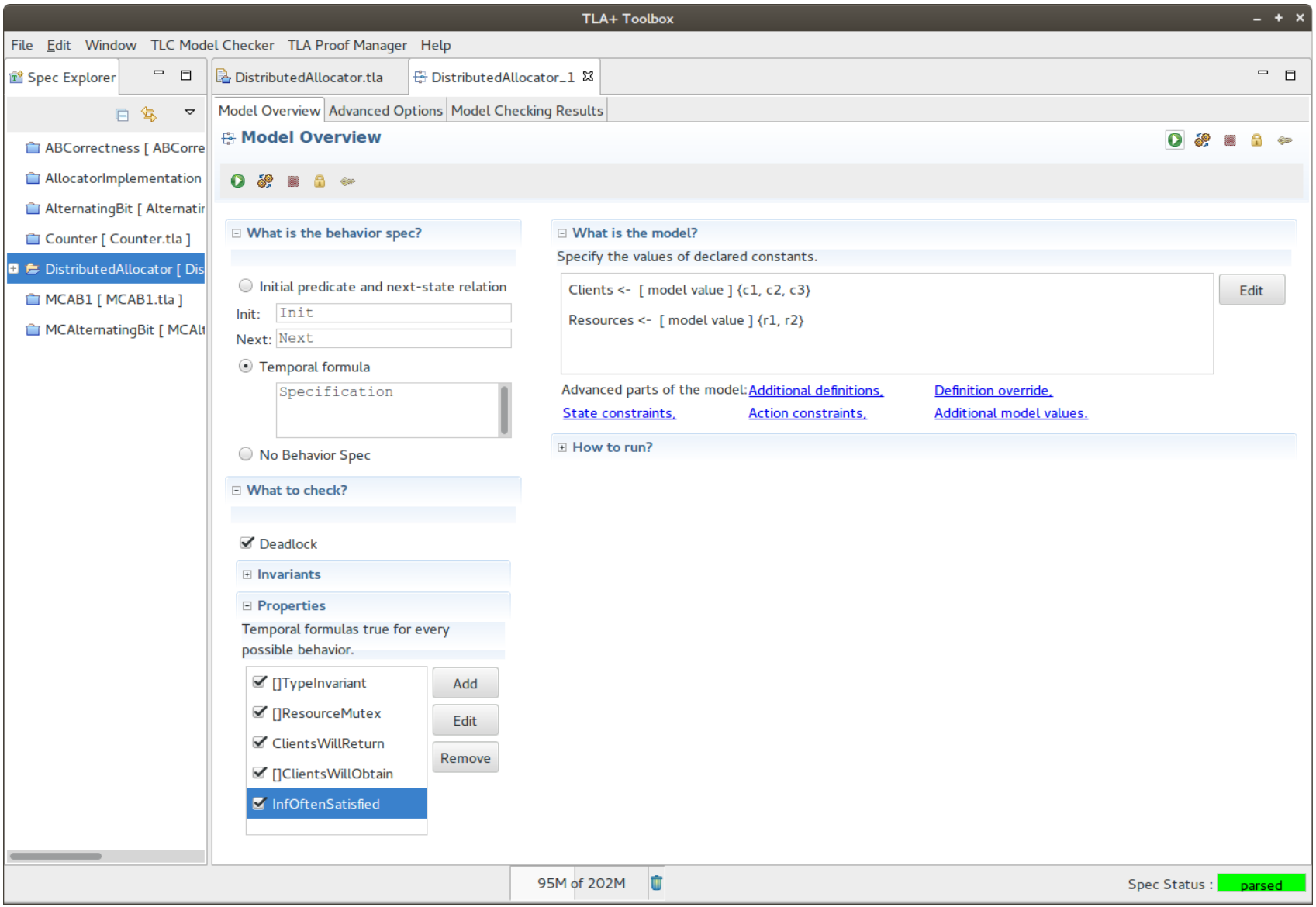
Ein Verteilter Ressourcen-Zuteiler

$$
\begin{array}{l}
\textit{TypeInvariant} \triangleq \\
\quad \land \textit{unsat} \in [\textit{Clients} \to \text{SUBSET } \textit{Resources}] \land \textit{alloc} \in [\textit{Clients} \to \text{SUBSET } \textit{Resources}] \\
\quad \land \textit{requests} \in [\textit{Clients} \to \text{SUBSET } \textit{Resources}] \land \textit{holding} \in [\textit{Clients} \to \text{SUBSET } \textit{Resources}] \\
\quad \land \textit{sched} \in \textit{Seq}(\textit{Clients}) \land \textit{network} \in \text{SUBSET } \textit{Messages} \\
\textit{ResourceMutex} \triangleq \\
\quad \forall\, c1,\, c2 \in \textit{Clients} : \textit{holding}[c1] \cap \textit{holding}[c2] \neq \{\} \Rightarrow c1 = c2 \\
\textit{ClientsWillReturn} \triangleq \\
\quad \forall\, c \in \textit{Clients} : (\textit{requests}[c] = \{\} \leadsto \textit{holding}[c] = \{\}) \\
\textit{ClientsWillObtain} \triangleq \\
\quad \forall\, c \in \textit{Clients},\, r \in \textit{Resources} : r \in \textit{requests}[c] \leadsto r \in \textit{holding}[c] \\
\textit{InfOftenSatisfied} \triangleq \\
\quad \forall\, c \in \textit{Clients} : \Box\Diamond(\textit{requests}[c] = \{\})
\end{array}
$$

Die Korrektheitseigenschaften.



Model Checking des Verteilter Ressourcen-Zuteilers



Der Zuteiler erfüllt die Eigenschaften (für 3 Klienten und 2 Ressourcen).