ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
TIK Institut für Technische Informatik und Kommunikationsnetze

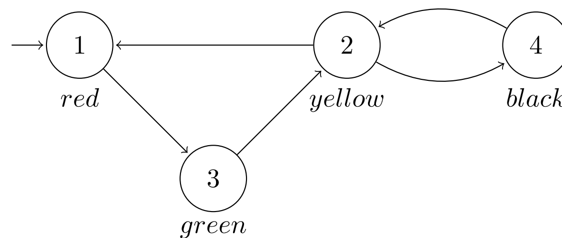
HS 2009

Dr. Kai Lampka
Jasmin Smula
Raphael Eidenbenz

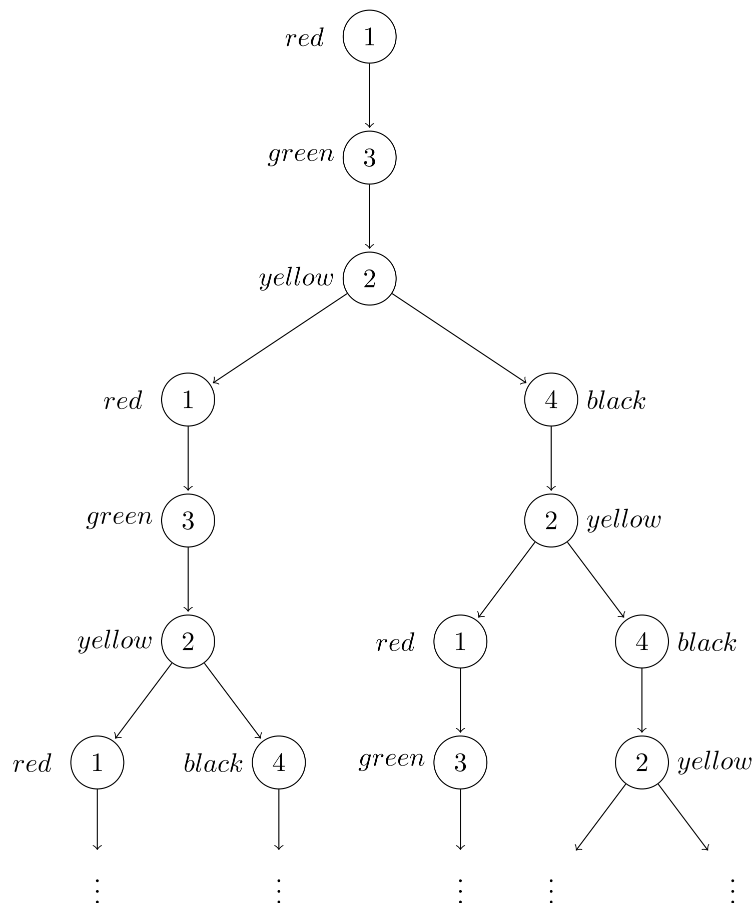
Discrete Event Systems
Solution to Exercise 10

1 Computation Tree Logic (CTL) Model Checking

a) Graph der Kripke-Struktur $\mathcal{K}$:



b) Der Computation-Tree bis zur Tiefe 7 für den Startzustand $\mathbb{S}_0$ sieht folgendermassen aus:



c) Ω_3 ist inkorrekt, weil $\exists$ nur in Kombination mit einem Temporaloperator ($\bigcirc, \Box, \bigcup, \Diamond$) definiert ist. Hier wird der Existenzquantor direkt mit einer Zustandsaussage benutzt. Ω_5 ist inkorrekt, weil der Operator $\exists\bigcirc$ eine Zustandsaussage erwartet, $(true \bigcup black)$ aber eine Pfadformel ist.

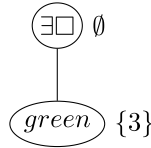
d) Ω_1, Ω_2 und Ω_7 sind bereits in ENF. Für die übrigen ergibt sich

$$\begin{aligned}
\Omega_4 &\equiv \neg\exists(true \bigcup \neg yellow) \\
\Omega_6 &\equiv \neg\exists(\neg black \bigcup \neg black) \wedge \neg\exists\Box\neg black \\
\Omega_8 &\equiv \neg\exists(\neg black \bigcup \neg true) \wedge \neg\exists\Box\neg black \\
\Omega_9 &\equiv \neg\exists\left(\neg(\exists\bigcirc black) \bigcup (yellow \wedge \neg(\exists\bigcirc black))\right) \wedge \neg\exists\Box\neg(\exists\bigcirc black) \\
\Omega_{10} &\equiv \exists(true \bigcup black) \\
\Omega_{11} &\equiv \forall(true \bigcup black) \equiv \neg\exists(\neg black \bigcup \neg true) \wedge \neg\exists\Box\neg black \\
\Omega_{12} &\equiv \neg\exists\left(\neg\psi \bigcup (\neg green \wedge \neg\psi)\right) \wedge \neg\exists\Box\neg\psi \\
\text{wobei } \psi &\equiv \neg\exists\left(\neg red \bigcup (\neg yellow \wedge \neg red)\right) \wedge \neg\exists\Box\neg red
\end{aligned}$$

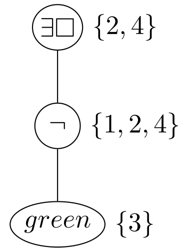
e) Für Ω_9 wird die Berechnung der $Satisfy(\phi)$-Mengen detailliert beschrieben (siehe unten). Für die restlichen Formeln kann die Berechnung analog dazu durchgeführt werden.

Die Syntaxbäume zu den syntaktisch korrekten CTL-Formeln Ω_i inklusive der Labels sehen dann wie folgt aus:

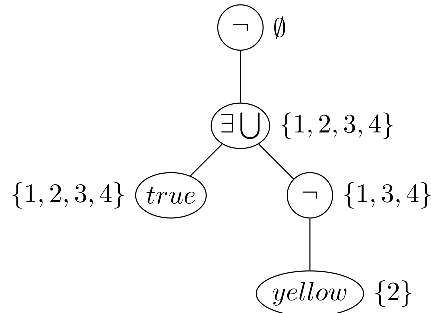
- $\Omega_1 = \exists\Box green$:



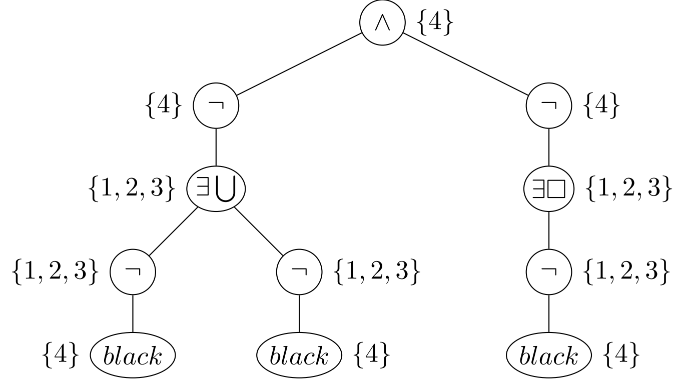
- $\Omega_2 = \exists\Box\neg green$:



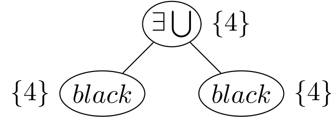
- $\Omega_4 = \forall\Box yellow = \neg\exists(true \bigcup \neg yellow)$:



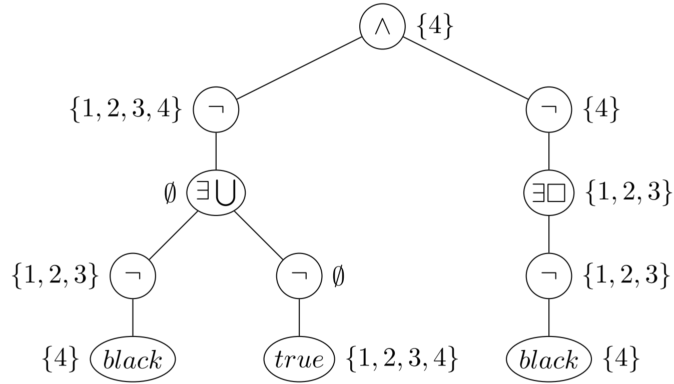
- $\Omega_6 = \forall(black \bigcup black) = \neg\exists(\neg black \bigcup \neg black) \wedge \neg\exists\Box\neg black$:



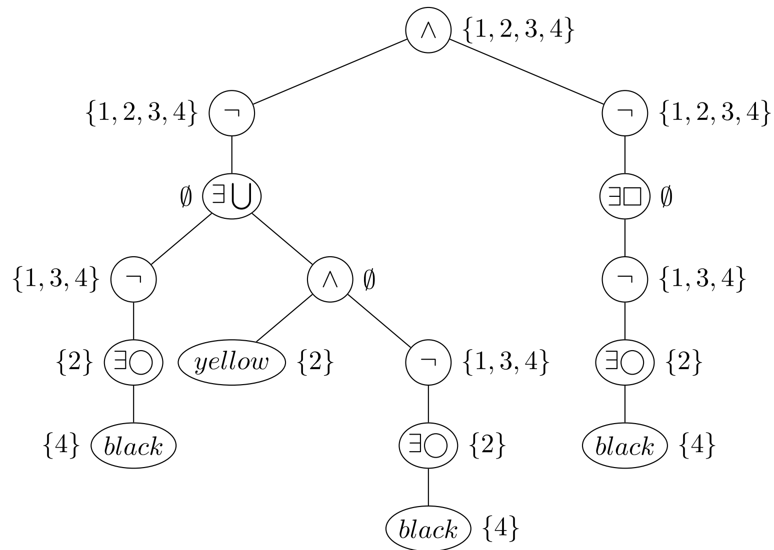
- $\Omega_7 = \exists(black \bigcup black)$:



- $\Omega_8 = \forall(\neg black \bigcup black) = \neg\exists(\neg black \bigcup \neg true) \wedge \neg\exists\Box\neg black$:



- $\Omega_9 = \neg\exists\Big(\neg(\exists\bigcirc black) \bigcup \big(yellow \wedge \neg(\exists\bigcirc black)\big)\Big) \wedge \neg\exists\Box\neg(\exists\bigcirc black)$:



An dieser Formel Ω_9 stellen wir beispielhaft die Berechnung der $Satisfy(\phi)$-Mengen vor, mit denen die Knoten der Syntaxbäume gelabelt werden. Die entsprechenden Formeln finden sich auf Folien 20ff. Die $Satisfy(\phi)$-Mengen werden zuerst für die Blätter des Syntaxbaums berechnet und dann von unten nach oben für die restlichen Knoten des Baums.

- $Satisfy(black) = \{\vec{s} \in \{1,2,3,4\} \mid \mathcal{L}(\vec{s}) = black\} = \{4\}$
- $Satisfy(yellow) = \{\vec{s} \in \{1,2,3,4\} \mid \mathcal{L}(\vec{s}) = yellow\} = \{2\}$
- $Satisfy(\exists \bigcirc black) = \{\vec{s} \in \{1,2,3,4\} \mid \mathcal{P}ost(\vec{s}) \cap Satisfy(black) \neq \emptyset\} = \{2\}$
- $Satisfy(\neg\exists \bigcirc black) = \mathbb{S} \setminus Satisfy(\exists \bigcirc black) = \{1,3,4\}$
- $Satisfy(yellow \wedge \neg\exists \bigcirc black) = Satisfy(yellow) \cap Satisfy(\neg\exists \bigcirc black) = \emptyset$
- $Satisfy(\exists\Box(\neg\exists \bigcirc black)) =: Satisfy(\phi)$:

  Hier müssen wir eine Fixpunktiteration durchführen. Wir berechnen die Mengen $Satisfy_i(\phi)$, bis wir einen Fixpunkt erreichen, also einen Punkt, ab dem gilt: $Satisfy_j(\phi) = Satisfy_i(\phi)\ \forall j > i$.
  (*Vorsicht:* Hier war ein Fehler auf den Folien! Auf Folie 22 muss es für $\phi = \exists\Box\varphi$ heissen: $Satisfy_{i+1}(\phi) := \{\vec{s} \in Satisfy_i(\varphi) \mid \mathcal{P}ost(\vec{s}) \cap Satisfy_i(\phi) \neq \emptyset\}$)

  * $Satisfy_0(\phi) = Satisfy(\neg\exists \bigcirc black) = \{1,3,4\}$
  * $Satisfy_1(\phi) = \{\vec{s} \in \{1,3,4\} \mid \mathcal{P}ost(\vec{s}) \cap \{1,3,4\} \neq \emptyset\} = \{1\}$
  * $Satisfy_2(\phi) = \{\vec{s} \in \{1\} \mid \mathcal{P}ost(\vec{s}) \cap \{1\} \neq \emptyset\} = \emptyset$

  Es ist klar, dass der Fixpunkt erreicht ist, da die Menge $Satisfy_3(\phi)$ (sowie alle weiteren Mengen $Satisfy_i(\phi)$) nicht mehr Elemente enthalten kann als $Satisfy_2(\phi)$. Damit erhält man $Satisfy(\exists\Box(\neg\exists \bigcirc black)) = \emptyset$.

- $Satisfy(\exists(\neg\exists \bigcirc black) \bigcup (yellow \wedge \neg\exists \bigcirc black)) =: Satisfy(\phi)$:

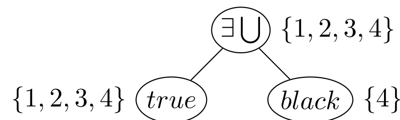
  Auch hier müssen wir eine Fixpunktiteration durchführen.

  * $Satisfy_0(\phi) = Satisfy(yellow \wedge \neg\exists \bigcirc black) = \emptyset$
  * $Satisfy_1(\phi) = \emptyset \cup \{\vec{s} \in \{1,3,4\} \mid \mathcal{P}ost(\vec{s}) \cap \emptyset \neq \emptyset\} = \emptyset$

  Der Fixpunkt ist erreicht, da auch alle weiteren Mengen $Satisfy_i(\phi)$ die leere Menge ergeben. Somit ist $Satisfy(\exists(\neg\exists \bigcirc black) \bigcup (yellow \wedge \neg\exists \bigcirc black)) = \emptyset$.

- $Satisfy(\neg\exists(\neg\exists \bigcirc black) \bigcup (yellow \wedge \neg\exists \bigcirc black)) = \mathbb{S} \setminus \emptyset = \{1,2,3,4\}$
- $Satisfy(\neg\exists\Box(\neg\exists \bigcirc black)) = \mathbb{S} \setminus \emptyset = \{1,2,3,4\}$
- $Satisfy(\neg\exists(\neg\exists \bigcirc black) \bigcup (yellow \wedge \neg\exists \bigcirc black) \wedge \neg\exists\Box(\neg\exists \bigcirc black)) = \{1,2,3,4\} \cap \{1,2,3,4\} = \{1,2,3,4\}$.

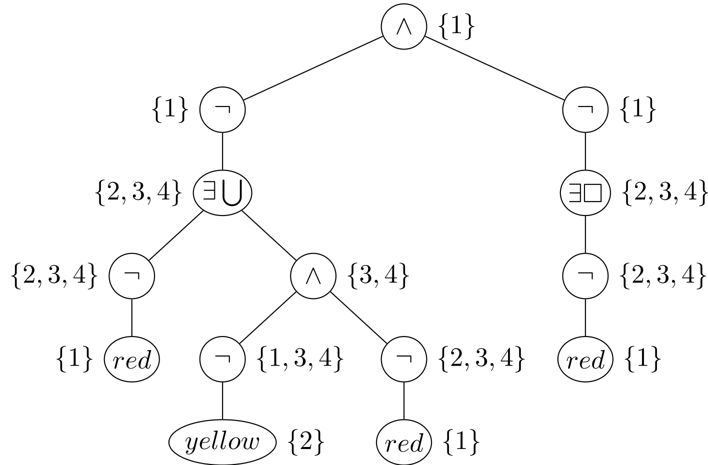
- $\Omega_{10} = \exists(true \bigcup black)$:



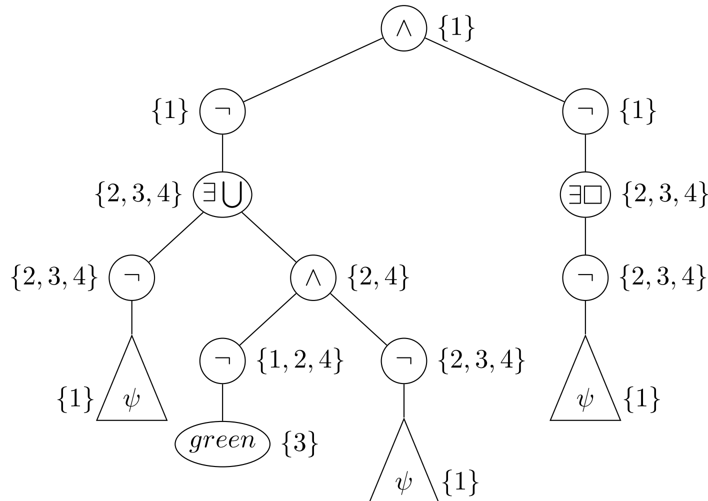
- $\Omega_{11} = \forall(true \bigcup black) = \neg\exists(\neg black \bigcup \neg true) \wedge \neg\exists\Box\neg black = \Omega_8$, siehe oben

- Ω_{12}:
  Der besseren Übersichtlichkeit halber stellen wir zuerst den Syntaxbaum für ψ auf und benutzen diesen später als Teilbaum im Syntaxbaum zu Ω_{12}.

  $\psi = \neg\exists\,(\neg red \bigcup (\neg yellow \wedge \neg red)) \wedge \neg\exists\Box\neg red$:



  $\Omega_{12} = \neg\exists\,(\neg\psi \bigcup (\neg green \wedge \neg\psi)) \wedge \neg\exists\Box\neg\psi$:



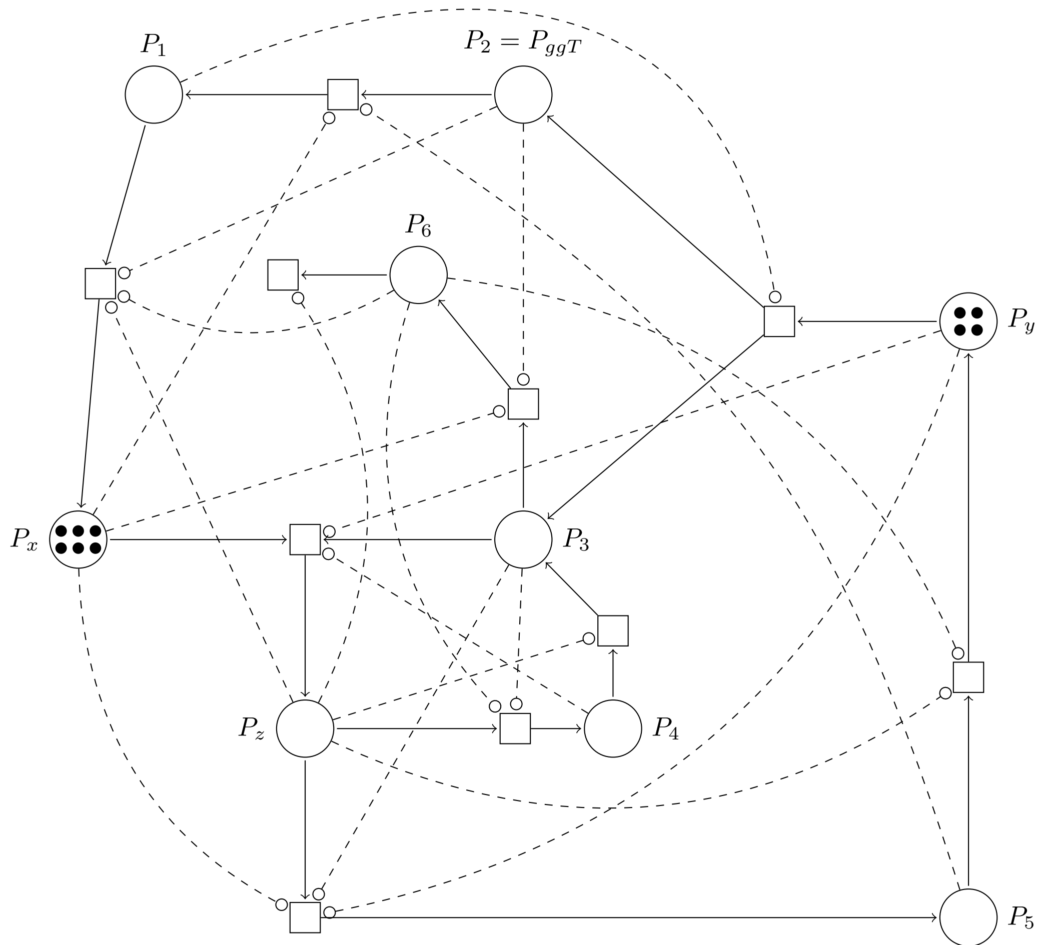
f) Eine Formel Ω wird von $\mathcal{K}$ erfüllt, wenn die Schnittmenge von $\mathbb{S}_0$ und $Satisfy_{\mathcal{K}}$ an der Wurzel des Syntaxbaumes nicht leer ist. Dies ist der Fall für Ω_9, Ω_{10} und Ω_{12}. Alle anderen sind nicht erfüllbar in $\mathcal{K}$.

g) Für diese Aufgabe können wir uns den Parse Tree anschauen. Ein Pfad, der mit Zustand 1 beginnt, $\Pi_4 = \Pi_6 = 1\ldots$, falsifiziert Ω_4 und Ω_6, weil bereits in Zustand 1 $\neg yellow$ sowie $\neg black$ gilt. Pfad $\Pi_8 = \Pi_{11} = 1, 3, 2, 1, 3, 2, \ldots$ falsifiziert Ω_8 und Ω_{11}, weil auf diesem unendlichen Pfad niemals Zustand 4 erreicht wird und somit nie *black* gilt.

2 Euklidscher Algorithmus

a) Der Algorithmus ermittelt den Rest z der Division ($z = x\ mod\ y$) und ruft sich selbst mit der kleineren Zahl, y, und dem Rest, z, wieder auf. Es werden vier Aufrufe getätigt, `ggt(225,60)`, `ggt(60,45)`, `ggt(45,30)` und `ggt(15,0)`. Das Resultat ist 15.

b) Folgendes Petrinetz modelliert Euklids Algorithmus:



Die Tokenverteilung modelliert den Startzustand für die Berechnung des ggT von 6 und 4. Das Petrinetz funktioniert wie folgt:

(1) Alle y Tokens aus P_y werden in P_2 sowie P_3 gefüllt.

(2) Es werden y Tokens aus P_x und P_3 nach P_z verschoben.

(3) Von P_z gelangen sie via P_4 wieder nach P_3.

(4) Schritt (2) und (3) wiederholen sich, bis P_x leer ist. P_z enthält jetzt den Rest $z = x \ mod \ y$.

(5) Alle y Tokens in P_2 werden nach P_1 überführt.

(6) Eventuelle restliche Tokens in P_3 werden nach P_6 verschoben.

(7) Die z Tokens in P_z gelangen nach P_5.

(8) Die überschüssigen Tokens in P_6 werden von der ausgangslosen Transition konsumiert.

(9) Die z Tokens in P_5 gelangen als Input der nächsten Rekursion nach P_y.

(10) Die y Tokens in P_1 werden als neuer Input nach P_x überführt.

(11) Dieser gesamte Prozess wird rekursiv aufgerufen, bis nach (4) P_x und P_3 leer sind. Das Resultat steht in P_{ggT}.

Bemerkung: Das Petrinetz ist streng genommen nicht ganz korrekt, denn am Ende des Algorithmus gibt es immer noch aktivierte Transitionen. Die Tokens in P_{ggT} könnten bspw. nach P_1 überführt werden, womit das Resultat dann nicht mehr ersichtlich wäre. Um dies zu verhindern, könnte man eine Kante von P_3 via Transition $t_{2,1}$ zu sich selbst und eine von P_x via $t_{z,4}$ zu sich selbst einführen, wobei $t_{i,j}$ die Transition zwischen P_i und P_j ist.

CTL-Zusammenfassung

Die Computation Tree Logic (CTL) ist eine Temporale Logik, die speziell zur Spezifikation und Verifikation von Computersystemen dient. Dabei geht es nicht um die Beschreibung von zeitlichen Abläufen, sondern um die Eigenschaften von Zuständen und deren Änderung in Systemabläufen. CTL wird aus

- atomaren Zustandsaussagen ($black, yellow, ..$),
- den booleschen Operatoren ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$) und
- Paaren von Pfadquantor ($\exists, \forall$) und Temporaloperator ($\bigcirc = next, \bigcup = until, \Box = always, \Diamond = eventually$) gebildet.

Wir können eine Eigenschaft eines Systems überprüfen, indem wir die entsprechende CTL-Formel im Computation Tree verifizieren, bzw. falsifizieren. Der *Computation Tree* eines Systems ist der Baum aller möglichen Ausführungspfade beginnend beim Initialzustand. Beachten Sie, dass die Zweige eines solchen Baumes unendlich tief sein können. Ein Baum erfüllt $\exists\varphi$ genau dann, wenn es in diesem beginnend bei der Wurzel einen Pfad gibt, der φ erfüllt. Ein Baum erfüllt $\forall\varphi$ genau dann, wenn jeder bei der Wurzel beginnende Pfad φ erfüllt.
Auf einem *Ausführungspfad* sind folgende Formelarten definiert:

$\bigcirc\phi$	Im nächsten Zustand gilt ϕ.
$\phi\bigcup\psi$	Bis zum ersten Auftreten von ψ gilt ϕ.
$\Box\phi$	In jedem Zustand gilt ϕ.
$\Diamond\phi$	In irgendeinem Zustand gilt ϕ.

Auf dem *Computation Tree* sind also folgende Formelarten definiert:

$\exists\bigcirc\phi$	In mind. einem der Kinderzustände der Wurzel gilt ϕ.
$\exists(\phi\bigcup\psi)$	Beginnend bei der Wurzel gibt es mind. einen Pfad für den gilt: bis zum ersten Auftreten von ψ gilt ϕ.
$\exists\Box\phi$	Es gibt einen Pfad, auf dem für jeden Zustand ϕ gilt.
$\exists\Diamond\phi$	In irgendeinem Zustand des Baumes gilt ϕ.
$\forall\bigcirc\phi$	In jedem Kind der Wurzel gilt ϕ.
$\forall(\phi\bigcup\psi)$	In jedem Pfad gilt ϕ bis zum ersten Auftreten von ψ.
$\forall\Box\phi$	In jedem Zustand des Baumes gilt ϕ.
$\forall\Diamond\phi$	Auf jedem Pfad gilt für mindestens einen Zustand ϕ.

Ferner gelten folgende Äquivalenzen:

$$\forall\bigcirc\phi \equiv \neg\exists\bigcirc\neg\phi \tag{1}$$
$$\forall\Box\phi \equiv \neg\exists\Diamond\neg\phi \equiv \neg\exists(true\bigcup\neg\phi) \tag{2}$$
$$\forall(\phi\bigcup\psi) \equiv \neg\exists(\neg\psi\bigcup(\neg\phi\wedge\neg\psi))\wedge\neg\exists\Box\neg\psi \tag{3}$$
$$\forall\Diamond\phi \equiv \neg\exists\Box\neg\phi \tag{4}$$