CTL Modelchecking

Model Checking, Seminar im Wintersemester 2004/05

Carsten Krapp

26.11.2004

Nach einem Buch von: E. M. Clarke, O. Grumberg, and D. Peled. Model Checking. MIT Press, 1999

Definition (Model Checker).

Ein Model Checker ist ein Algorithmus, der den Zustandsraum eines System nach unspezifizierten Abläufen durchsucht. [a]

[a] Prof. Dr. Armin Biere, Institut fur Computersysteme, ETH Zurich

Warum brauchen wir Modelchecker?

- Es ist nicht möglich, alle Systeme anhand ihres Input-Output-Verhaltens zu modellieren.
- Kritische Anwendungen dürfen nicht versagen und können eventuell nicht getestet werden unter Realbedingungen (Echtzeitsysteme, Sicherheitskritische Anwendungen)
- Müssen beweisen können, dass bestimmte Situationen eintreten bzw. nicht eintreten können.

Der Mars Rover „Spirit“

When the Mars rover Spirit went dark on Jan.21 (2003)...

„That appeared to fix the problems that had been identified with the initial load. But the new load also made possible a totally implausible sequence of events that would, many months later, silence Spirit...“

Dieses Missgeschick hat sicherlich einige Millionen Dollar gekostet und hätte verhindert werden können.

Eigenschaften eines Systems

Zustand, Transition und Valuation

- Ein Zustand ist die Beschreibung des System durch die Werte der Variablen zu der bestimmten Zeit.
- Eine Transition ist der Übergang von einem Zustand zu einem Folgestatus
- Sei V die Menge der Systemvariablen über einem Wertebereich D. Eine Valuation für V ist eine Funktion, die jeder Variable aus V einen Wert aus D zuordnet.

Die Kripke Struktur

Sei AP eine Menge von atomaren Propositionen. Eine Kripke Struktur M über AP ist ein Viertupel $M = (S, S_0, R, L)$ mit folgenden Eigenschaften:

- S ist eine endliche Menge von Zuständen, d.h. die Menge aller Valuationen für V.
- $S_0 \subseteq S$ ist die Menge der Anfangszustände, bzw. die Menge aller Valuationen s_0 für V, die S_0 erfüllen.
- $R \subseteq S \times S$ ist eine totale Relation. $R = \{(s, s') | \forall v \in V, v' \in V' : v \leftarrow s(v) \Rightarrow v' \leftarrow s'(v), s, s' \in S\}$.
- $L : S \rightarrow 2^{AP}$ ist eine Funktion, die jedem Zustand diejenigen atomaren Propositionen zuweist, die in ihm wahr sind.

Beispiel

Sei $V = \{x, y\}$, $D = \{0, 1\}$. Eine Valuation für V ist ein Paar $(d_1, d_2) \in D \times D$. Das System bestehe nur aus der Transition $x := (x + y) \mod 2$ mit dem Startzustand $x = 1,\ y = 1$. Die Menge der Anfangszustände wird dann durch $S_0(x, y) \equiv x = 1 \wedge y = 1$ dargestellt und es gilt $R(x, y, x', y') \equiv x' = (x + y) \mod 2 \wedge y' = y \Rightarrow$ Dann erhalten wir folgende Kripke Strukur $M = (S, S_0, R, L)$:

Beispiel

Sei $V = \{x, y\}$, $D = \{0, 1\}$. Eine Valuation für V ist ein Paar $(d_1, d_2) \in D \times D$. Das System bestehe nur aus der Transition $x := (x + y) \mod 2$ mit dem Startzustand $x = 1,\ y = 1$. Die Menge der Anfangszustände wird dann durch $S_0(x, y) \equiv x = 1 \wedge y = 1$ dargestellt und es gilt $R(x, y, x', y') \equiv x' = (x + y) \mod 2 \wedge y' = y \Rightarrow$ Dann erhalten wir folgende Kripke Strukur $M = (S, S_0, R, L)$:

- $S = D \times D$

Beispiel

Sei $V = \{x, y\}$, $D = \{0, 1\}$. Eine Valuation für V ist ein Paar $(d_1, d_2) \in D \times D$. Das System bestehe nur aus der Transition $x := (x + y) \mod 2$ mit dem Startzustand $x = 1,\ y = 1$. Die Menge der Anfangszustände wird dann durch $S_0(x, y) \equiv x = 1 \wedge y = 1$ dargestellt und es gilt $R(x, y, x', y') \equiv x' = (x + y) \mod 2 \wedge y' = y \Rightarrow$ Dann erhalten wir folgende Kripke Strukur $M = (S, S_0, R, L)$:

- $S = D \times D$
- $S_0 = \{(1, 1)\}$

Beispiel

Sei $V = \{x, y\}$, $D = \{0, 1\}$. Eine Valuation für V ist ein Paar $(d_1, d_2) \in D \times D$. Das System bestehe nur aus der Transition $x := (x + y) \mod 2$ mit dem Startzustand $x = 1,\ y = 1$. Die Menge der Anfangszustände wird dann durch $S_0(x, y) \equiv x = 1 \wedge y = 1$ dargestellt und es gilt $R(x, y, x', y') \equiv x' = (x + y) \mod 2 \wedge y' = y \Rightarrow$ Dann erhalten wir folgende Kripke Strukur $M = (S, S_0, R, L)$:

- $S = D \times D$
- $S_0 = \{(1, 1)\}$
- $R = \{((1, 1), (0, 1)), ((0, 1), (1, 1)), ((1, 0), (1, 0)), ((0, 0), (0, 0))\}$

Beispiel

Sei $V = \{x, y\}$, $D = \{0, 1\}$. Eine Valuation für V ist ein Paar $(d_1, d_2) \in D \times D$. Das System bestehe nur aus der Transition $x := (x + y) \mod 2$ mit dem Startzustand $x = 1,\ y = 1$. Die Menge der Anfangszustände wird dann durch $S_0(x, y) \equiv x = 1 \wedge y = 1$ dargestellt und es gilt $R(x, y, x', y') \equiv x' = (x + y) \mod 2 \wedge y' = y \Rightarrow$ Dann erhalten wir folgende Kripke Strukur $M = (S, S_0, R, L)$:

- $S = D \times D$
- $S_0 = \{(1, 1)\}$
- $R = \{((1, 1), (0, 1)), ((0, 1), (1, 1)), ((1, 0), (1, 0)), ((0, 0), (0, 0))\}$
- $L((1, 1)) = \{x = 1, y = 1\}, L((0, 1)) = \{x = 0, y = 1\}, L((1, 0)) = \{x = 1, y = 0\}, L((0, 0)) = \{x = 0, y = 0\}$

CTL* (Computation Tree Logic)

- Die CTL* Formeln beschreiben Eigenschaften eines Berechnungsbaumes
- Die Wurzel des Baumes ist ein Zustand einer Kripke Struktur
- Der Berechnungsbaum gibt alle möglichen Ausführungen an, die von der Wurzel aus erreichbar sind.
- CTL*-Formeln bestehen aus Pfadquantoren und Temporalen Operatoren
- Die Pfadquantoren beschreiben die Struktur des Berechnungsbaumes
- Die Temporalen Operatoren geben die Eigenschaften eines Pfades im Baum an

Pfadquantoren

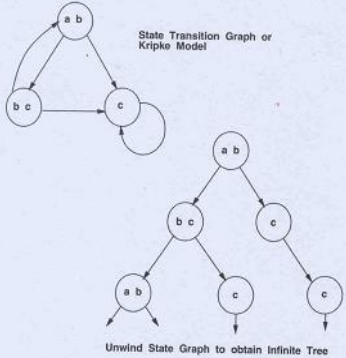

Figure 3.1
Computation trees.

Pfadquantoren

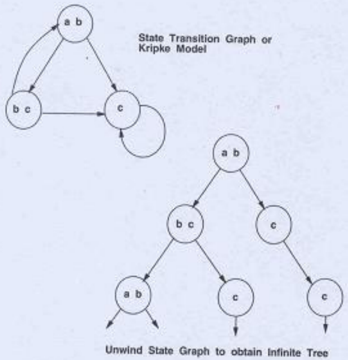

Figure 3.1
Computation trees.

Es gibt zwei Pfadquantoren. Sie geben an, ob von einem Zustand aus alle oder einige Pfade eine gewisse Eigenschaft besitzen.

- **A** („Für alle Berechnungspfade“)

Pfadquantoren

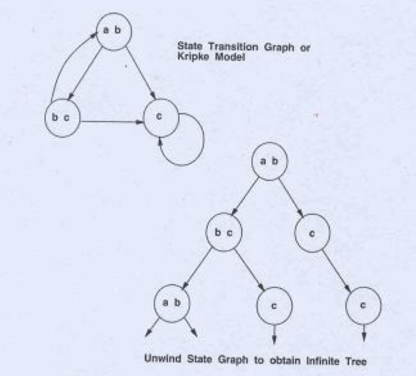

Figure 3.1
Computation trees.

Es gibt zwei Pfadquantoren. Sie geben an, ob von einem Zustand aus alle oder einige Pfade eine gewisse Eigenschaft besitzen.

- **A** („Für alle Berechnungspfade“)
- **E** („Für manche Berechnungspfade“)

Temporale Operatoren

Es gibt fünf Basisoperatoren:

Temporale Operatoren

Es gibt fünf Basisoperatoren:

- **X** (eine Eigenschaft soll im nächsten Zustand des Pfades gelten)

Temporale Operatoren

Es gibt fünf Basisoperatoren:

- **X** (eine Eigenschaft soll im nächsten Zustand des Pfades gelten)
- **F** (Es gibt einen Zustand im Pfad, der die geforderte Eigenschaft erfüllt)

Temporale Operatoren

Es gibt fünf Basisoperatoren:

- **X** (eine Eigenschaft soll im nächsten Zustand des Pfades gelten)
- **F** (Es gibt einen Zustand im Pfad, der die geforderte Eigenschaft erfüllt)
- **G** (eine Eigenschaft gilt in allen Zuständen des Pfades)

Temporale Operatoren

Es gibt fünf Basisoperatoren:

- **X** (eine Eigenschaft soll im nächsten Zustand des Pfades gelten)
- **F** (Es gibt einen Zustand im Pfad, der die geforderte Eigenschaft erfüllt)
- **G** (eine Eigenschaft gilt in allen Zuständen des Pfades)
- **U** (es gibt einen Zustand der eine gewisse Eigenschaft erfüllt und in allen anderen Zuständen auf dem Pfad gilt eine bestimmte andere Eigenschaft.)

Temporale Operatoren

Es gibt fünf Basisoperatoren:

- **X** (eine Eigenschaft soll im nächsten Zustand des Pfades gelten)
- **F** (Es gibt einen Zustand im Pfad, der die geforderte Eigenschaft erfüllt)
- **G** (eine Eigenschaft gilt in allen Zuständen des Pfades)
- **U** (es gibt einen Zustand der eine gewisse Eigenschaft erfüllt und in allen anderen Zuständen auf dem Pfad gilt eine bestimmte andere Eigenschaft.)
- **R** (eine Eigenschaft gilt bis zu einem Zustand, in dem auch eine weitere Eigenschaft gilt)

Syntax von CTL*-Formeln

Es gibt zwei Arten von Formeln in CTL*

1. Zustandsformeln, die in einem gegebenen Zustand wahr sind.
2. Pfadformeln, die auf einem gegebenen Pfad wahr sind

Syntax für die Zustands- und Pfadformeln:

- Ist $p \in AP$, dann ist p eine Zustandsformel.
- Sind f und g Zustandsformeln, dann auch $\neg f, f \vee g$ und $f \wedge g$.
- Ist f eine Zustandsformel, dann ist f auch eine Pfadformel.
- Ist f eine Pfadformel, dann sind $\mathbf{E}\ f$ und $\mathbf{A}\ f$ Zustandsformeln.
- Sind f und g Pfadformeln, so sind $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}\ f$, $\mathbf{F}\ f$, $\mathbf{G}\ f$, $f\ \mathbf{U}\ g$ und $f\ \mathbf{R}\ g$ Pfadformeln.

Semantik von CTL*-Formeln

Sei M=(S,R,L) eine Kripke Struktur, f_1, f_2 Zustandsformeln, g_1, g_2 Pfadformeln und bezeichne π^i das Suffix eines Pfades π mit Startzustand s_i, dann definieren wir eine Relation $\models$ induktiv:

- $M, s \models p \Leftrightarrow p \in L(s)$
- $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1(s)$
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
- $M, s \models \mathbf{E}\ g_1 \Leftrightarrow$ es ex. ein Pfad π von s, so dass $M, \pi \models g_1$
- $M, s \models \mathbf{A}\ g_1 \Leftrightarrow$ jeder Pfad π von s, so erfüllt $M, \pi \models g_1$

Semantik von CTL*-Formeln

- $M, \pi \models f_1 \Leftrightarrow$ i ist der erste Zustand von π und $M, s \models f_1$
- $M, \pi \models \neg g_1 \Leftrightarrow M, \pi \not\models g_1(s)$
- $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$
- $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, \pi \models g_1$ and $M, \pi \models g_2$
- $M, \pi \models \mathbf{X}\ g_1 \Leftrightarrow M, \pi^1 \models g_1$
- $M, \pi \models \mathbf{F}\ g_1 \Leftrightarrow$ es gibt ein $k \geq 0$,so dass $M, \pi^k \models g_1$
- $M, \pi \models \mathbf{G}\ g_1 \Leftrightarrow$ für alle $i \geq 0$,gilt $M, \pi^i \models g_1$
- $M, \pi \models g_1\ \mathbf{U}\ g_2 \Leftrightarrow$ es gibt ein $k \geq 0$,so dass $M, \pi^k \models g_2$ und für alle $0 \leq j < k$, gilt $M, \pi^j \models g_1$
- $M, \pi \models g_1\ \mathbf{R}\ g_2 \Leftrightarrow$ für alle $j \geq 0$, $i < j$: $M, \pi^i \not\models g_1 \Rightarrow M, \pi^j \models g_2$

CTL (Computation Tree Logic)

CTL ist eine Teilmenge von CTL*. Auf jeden der temporalen Operatoren **X, F, G, U, R** muß sofort eine Pfadangabe folgen.

- Sind f und g Zustandsformeln, dann sind **X** f, **F** f, **G** f, f **U** g und f **R** g Pfadformeln.

CTL (Computation Tree Logic)

Es gibt zehn Basisoperatoren:

- **AX** und **EX**
- **AF** und **EF**
- **AG** und **EG**
- **AU** und **EU**
- **AR** und **ER**

CTL (Computation Tree Logic)

Es gibt zehn Basisoperatoren:

- **AX** und **EX**
- **AF** und **EF**
- **AG** und **EG**
- **AU** und **EU**
- **AR** und **ER**

Beispiel:
AG(**EF** Neustart)

CTL (Computation Tree Logic)

Es gibt zehn Basisoperatoren:

- **AX** und **EX**
- **AF** und **EF**
- **AG** und **EG**
- **AU** und **EU**
- **AR** und **ER**

Beispiel:
AG(**EF** Neustart)

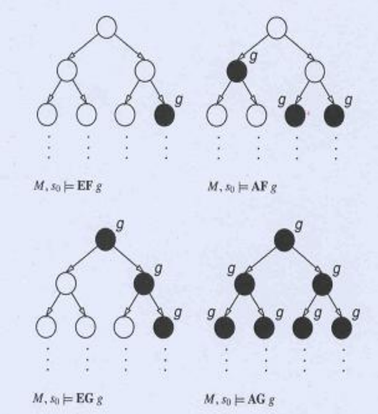

Figure 3.2
Basic CTL operators.

LTL (Linear Temporal Logic)

LTL ist eine Teilmenge von CTL*bei der die Operatoren die Ereignisse beschreiben, die auf einem einzelnen Berechnungspfad eintreten können.

- Ist $p \in AP$, dann ist p eine Pfadformel
- Sind f, g Pfadformeln, so auch $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}\ f$, $\mathbf{F}\ f$, $\mathbf{G}\ f$, $f\ \mathbf{U}\ g$ und $f\ \mathbf{R}\ g$.

Es kann gezeigt werden, dass die LTL Formel $\mathbf{A}(\mathbf{FG}p)$ nicht in CTL und dass die CTL Formel $\mathbf{AG}(\mathbf{EF}p)$ nicht in LTL ausgedrückt werden kann. Somit kann die CTL*Formel $\mathbf{A}(\mathbf{FG}p) \vee \mathbf{AG}(\mathbf{EF}p)$ weder in CTL noch in LTL ausgedrückt werden.

Fairness-Bedingungen

Manchmal ist es nicht nur wichtig, die Korrektheit auf einem Berechnungspfad zu zeigen, sondern auch dass Fairness-Bedingungen eingehalten werden.

Fairness-Bedingungen

Manchmal ist es nicht nur wichtig, die Korrektheit auf einem Berechnungspfad zu zeigen, sondern auch dass Fairness-Bedingungen eingehalten werden.

- Fairness kann nicht direkt in CTL abgebildet werden aber in CTL*

Fairness-Bedingungen

Manchmal ist es nicht nur wichtig, die Korrektheit auf einem Berechnungspfad zu zeigen, sondern auch dass Fairness-Bedingungen eingehalten werden.

- Fairness kann nicht direkt in CTL abgebildet werden aber in CTL*

$\Rightarrow$ Veränderung der Semantik von CTL wird notwendig. Wir nennen die neue Semantik von CTL faire Semantik.

Fairness-Bedingungen

Ein Pfad heißt fair, wenn jede Fairness-Bedingung unendlich oft auf einem Pfad gilt.

- eine faire Kripke Struktur ist ein Viertupel $M = (S, R, L, F)$, mit $F \subseteq 2^S$ als Menge der Fairness-Bedingungen.
- $\inf(\pi) = \{s | s = s_i \text{ für unendlich viele } i\}$
- π ist fair $\Leftrightarrow \forall P \in F : \inf(\pi) \cap P \neq \emptyset$
- Wir schreiben $M, s \models_F f$ wenn die Zustandsformel f wahr ist im Zustand s einer fairen Kripke Struktur.
- Wir schreiben $M, \pi \models_F g$ wenn die Pfadformel g wahr ist für einen Pfad π in einer fairen Kripke Struktur M.

CTL Model Checking

Um herauszufinden, welche Zustände S einer Kripke Struktur $M = (S, R, L)$ eine CTL Formel f erfüllen, benutzen wir folgenden Algorithmus:

- Markiere jeden Zustand s mit der Menge $label(s)$ von Teilformeln von f, die in s wahr sind.
- Wir beginnen mit $label(s) = L(s)$.
- Nun wird L(s) induktiv in Subformel aufgeteilt und diese werden $label(s)$ hinzugefügt.
- Terminiert der Algorithmus ,so gilt $M, S \models f$ iff $f \in label(s)$

CTL Model Checking

Es gilt:

- Jede CTL Formel kann durch die Terme $\neg$, $\vee$, **EX**,**EU**,**EG** dargestellt werden

THEOREM 1.

Es gibt einen Algorithmus, der entscheidet, ob ein CTL Formel f in einem Zustand s einer Kripke Struktur $M = (S, R, L)$ wahr ist, der Laufzeit $\mathcal{O}(|f| \cdot (|S| + |R|))$ hat.

Markierungsalgorithmen

- Die Markierung für Formeln der Art $\neg f$, $f \vee g$, $\mathbf{EX}\ f$ erfolgt kanonisch.
- Für Formeln der Art $g = \mathbf{E}[f_1 \mathbf{U}\ f_2]$ suche zunächst alle Zustände, die mit f_2 markiert sind. Gehe „zurück" mit R^{-1} und markiere alle Zustände, die man über einen Pfad, in dem alle Zustände mit f_1 markiert sind, mit g.
  Der Algorithmus *CheckEU* benötigt offensichtlich $\mathcal{O}(|S| + |R|)$ Zeit.

Der CheckEU-Algorithmus

```
procedure CheckEU(f1, f2)
    T := { s | f2 ∈ label(s) };
    for all s ∈ T do label(s) := label(s) ∪ { E[f1 U f2] };
    while T ≠ ∅ do
        choose s ∈ T;
        T := T \ {s};
        for all t such that R(t, s) do
            if E[f1 U f2] ∉ label(t) and f1 ∈ label(t) then
                label(t) := label(t) ∪ { E[f1 U f2] };
                T := T ∪ {t};
            end if;
        end for all;
    end while;
end procedure
```

Figure 4.1
Procedure for labeling the states satisfying $\mathbf{E}[f_1 \ \mathbf{U}\ f_2]$.

Der *CheckEG*-Algorithmus

Sei $M' = (S', R', L')$, wobei $S' = \{s \in S | M, s \models f_1\}$, $R' = R|_{S' \times S'}$ und $L' = L|_{S'}$ (R' muss nach dieser Konstruktion nicht mehr total sein). Es kann gezeigt werden, dass $M, s \models \mathbf{EG}\ f_1$ genau dann, wenn gilt:

1. $s \in S'$
2. Es existiert ein Pfad in M' der von s zu einem Knoten t geht in einem nichttrivialen stark zusammenhängenden Graphen (S', R').

Der *CheckEG*-Algorithmus

- Partitioniere also den Graphen (S', R') in stark zusammenhängende Komponenten. Dies hat Komplexität $\mathcal{O}(|S'| + |R'|)$.
- Nun finde alle Zustände in nichtrivialen Komponenten.
- benutze R' und finde alle Zustände, die über einen Pfad erreicht werden, in dem jeder Zustand mit f_1 markiert ist. Dies hat Laufzeit $\mathcal{O}(|S| + |R|)$.
  Der Algorithmus *CheckEG* benötigt offensichtlich $\mathcal{O}(|S| + |R|)$ Zeit.

Der CheckEG-Algorithmus

procedure $CheckEG(f_1)$
    $S' := \{ s \mid f_1 \in label(s) \}$;
    $SCC := \{ C \mid C$ is a nontrivial SCC of $S' \}$;
    $T := \bigcup_{C \in SCC} \{ s \mid s \in C \}$;
    for all $s \in T$ **do** $label(s) := label(s) \cup \{ \mathbf{EG}\ f_1 \}$;
    while $T \neq \emptyset$ **do**
        choose $s \in T$;
        $T := T \setminus \{s\}$;
        for all t **such that** $t \in S'$ **and** $R(t, s)$ **do**
            if $\mathbf{EG}\ f_1 \notin label(t)$ **then**
                $label(t) := label(t) \cup \{ \mathbf{EG}\ f_1 \}$;
                $T := T \cup \{t\}$;
            end if;
        end for all;
    end while;
end procedure

Figure 4.2
Procedure for labeling the states satisfying $\mathbf{EG}\ f_1$.

Zu Theorem 1.

- Um eine zusammengesetzte CTL Formel f überprüfen zu können, wende die Markierungsalgorithmen auf die Subformeln von f an, indem wir mit der kürzesten, tiefsten verschachtelten Formel beginnen.
- Jede Überprüfung einer Subformel benötigt $\mathcal{O}(|S| + |R|)$ Zeit.
- f hat höchstens $|f|$ Subformeln. Es folgt also:
- Der gesamte Algorithmus hat Komplexität $\mathcal{O}(|f| \cdot (|S| + |R|))$.

Beispiel. (Anhand eines Mikrowellenofens)

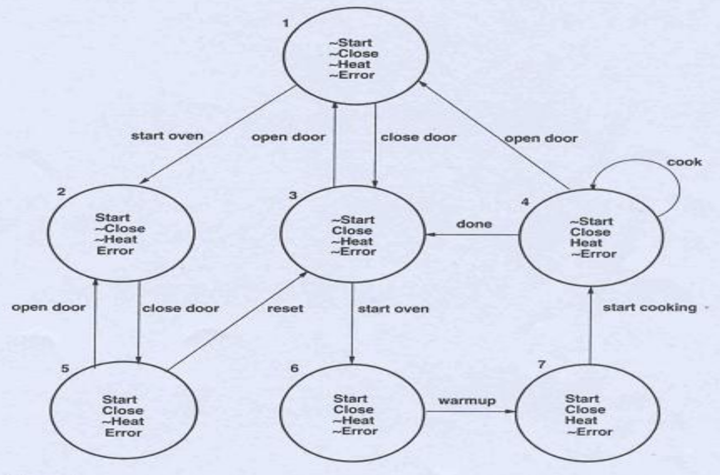

Figure 4.3
Microwave oven example.

Beispiel

Überprüfe die CTL Formel $\mathbf{AG}(Start \rightarrow \mathbf{AF}\ Heat)$. (bzw. $\neg\mathbf{EF}(Start \wedge\ \mathbf{EG}\ \neg Heat)$

- Suche zuerst die Zustände, die die atomaren Formeln erfüllen
  - $S(Start) = \{2, 5, 6, 7\}$
  - $S(\neg Heat) = \{1, 2, 3, 5, 6\}$
- Um $S(\mathbf{EG}\ \neg Heat)$ finden zu können, bestimme zunächst die Menge der nichttrivialen stark zusammenhängenden Komponenten (SCC) in $S' = S(\neg Heat)$
  - SCC$=\{\{1, 2, 3, 5\}\}$
  - $S(\mathbf{EG}\ \neg Heat) = \{1, 2, 3, 5\}$
- $S(Start \wedge \mathbf{EG}\ \neg Heat) = \{2, 5\}$

Beispiel

- Um $S(\mathbf{EF}(Start \wedge \mathbf{EG}\ \neg Heat)$ zu berechnen, beginnen wir mit $S(Start \wedge \mathbf{EG}\ \neg Heat)$
  - benutze R^{-1} und finde alle Knoten, von denen aus man $S(Start \wedge \mathbf{EG}\ \neg Heat)$ erreichen kann.
  - $S(\mathbf{EF}(Start \wedge \mathbf{EG}\ \neg Heat)) = \{1, 2, 3, 4, 5, 6, 7\}$
- Wir erhalten $S(\neg\mathbf{EF}(Start \wedge \mathbf{EG}\ \neg Heat)) = \emptyset$
  Da der Anfangszustand nicht enthalten ist, können wir schließen, dass die Kripke Struktur, der gegebenen Spezifikation nicht genügt.

Woran scheitert das Beispiel?

Woran scheitert das Beispiel?

Es ist möglich unendlich häufig mal hintereinander den Kreis $\pi = 1, 2, 5, 3, 1$ zu benutzen. Wir müssen also wieder die Fairness-Bedingungen beachten.

CTL Model Checking und Fairness

Sei $M = (S, R, L, F)$ eine faire Kripke Struktur, wobei $F = \{P_1, \ldots, P_k\}$ die Menge der Fairness-Bedingungen ist. Eine SCC K in M ist fair in Bezug auf F genau dann, wenn für alle $P_i \in F$ ein Zustand $t_i \in (K \cap P_i)$ existiert.

Theorem 2.

Ähnlich wie vorhin können wir zeigen:

- Es existiert ein Algorithmus, der bestimmt, ob eine CTL Formel f bezüglich der fairen Semantik in einem Zustand s der Struktur $M = (S, R, L, F)$ wahr ist, mit Laufzeit $\mathcal{O}(|f| \cdot (|S| + |R|) \cdot |F|)$

Zurück zum Mikrowellenofen-Beispiel

Wir betrachten nun nur die Pfade, bei denen der Benutzer die Mikrowelle unendlich oft korrekt bedient.

- Es soll also unendlich oft gelten: $F = \{P\}$ mit $P = \{s | s \models Start \wedge Close \wedge \neg Error\}$
- Berechnen wir nun die Menge der SCC über $S' = S(\neg Heat,$ dann ist $\{1, 2, 3, 5\}$ nicht fair.
- Also $S(\mathbf{EG}\neg Heat) = \emptyset$
- $S(\mathbf{EF}(Start \wedge \mathbf{EG}\neg Heat)) = \emptyset$
- $S(\neg(\mathbf{EF}(Start \wedge \mathbf{EG}\neg Heat))) = \{1, 2, 3, 4, 5, 6, 7\}$

Das Programm erfüllt also die Formel unter den gegeben Fairness-Bedingungen.

Was liefert und CTL

Was liefert und CTL

- Einfache Syntax und Semantik

Was liefert und CTL

- Einfache Syntax und Semantik
- Geringe Laufzeit

Was liefert und CTL

- Einfache Syntax und Semantik
- Geringe Laufzeit
- Problem Fairness abzubilden

Fragen?