

Synthese Reaktiver Systeme durch Live Sequence Charts

Frank Radmacher

RWTH Aachen

Seminar: Message Sequence Charts (WS 2004/05)

13. Januar 2005



Druckversion



Worum es geht ...

- Überlegenheit von Live Sequence Charts (LSCs) gegenüber MSCs.
- Synthese von Reaktiven Systemen möglich:
 - gegeben: LSC Spezifikation von Reaktivem System
 - gesucht: korrekte Implementierung oder Sabotageplan
- Vereinigung von Konstruktion (Codegenerierung) und Verifikation.

Gliederung

- 1 Einleitung
- 2 Grundlagen
- 3 Liveness und Safety
- 4 Synthese
- 5 Zusammenfassung

Literatur



Yves Bontemps, Pierre-Yves Schobbens, and Christof Löding.
Synthesizing open reactive systems from scenario-based specifications.
Fundamenta Informaticae, XX:1–31, 2004.



Yves Bontemps and Pierre-Yves Schobbens.
Synthesizing open reactive systems from scenario-based specifications.
In Felice Balarin and Johan Lilius, editors, *Proc. of the 3rd ACSD'03*,
pages 41–50, Guimarães, Portugal, June 2003. IEEE Computer Science
Press.



E. Grädel, W. Thomas, and T. Wilke, editors.
Automata, Logics, and Infinite Games.
Number 2500 in Lecture Notes in Computer Science. Springer-Verlag,
2002.

Reaktive Systeme

- Mehrere Komponenten (Prozesse),
in *fortwährender Wechselwirkung* miteinander.
- In der Regel *verteilte Systeme*.
- In der Regel *keine Terminierung* der Komponenten.
- Die Komponenten gehören zwei Gruppen an:
 - System
 - Umgebung

Probleme bisheriger Ansätze

- 1 Meist nur sequentielle Beschreibungen wie in HMSCs.
 - ➔ Lösungsansatz: Beschreibung von Interaktion und Reaktion durch LSCs.
- 2 Große Lücke zwischen Beispielszenarien (MSCs) und formaler Spezifikation (temporale Logiken).
 - ➔ Lösungsansatz: Anschauliche aber zugleich präzise Spezifikationssprache.
- 3 Mühsame *a posteriori* Verifikation.
 - ➔ Lösungsansatz: Korrektheit durch Konstruktion.

Erweiterung von MSCs zu Live Sequence Charts

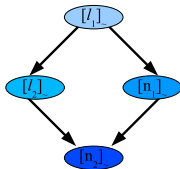
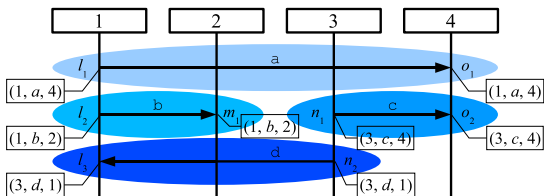
- *Koregionen*: Ereignisse in beliebiger Reihenfolge.
- Drei Modi:
 - Initial LSCs ($S = \triangleright C$)
 - Existential LSCs (mögliches Verhalten) ($S = \diamond C$)
 - Universal LSCs (notwendiges Verhalten) ($S = \square(C_1, C_2)$)
- *Beschränkte Ereignisse*: dürfen nicht auftreten, wenn der Chart aktiv ist.
- LSCs bestehen aus Basic Charts.

Nachrichtenalphabet

- Ag : Endliche Menge von *Instanzen* (Ag für engl. *Agents*).
- \mathcal{M} : Endliche Menge von *Nachrichtenbezeichnern*.
- ➔ Endliches Alphabet $\Sigma = Ag \times \mathcal{M} \times Ag$:
 - Menge aller möglicher Ereignisse.
 - $(a_1, m, a_2) \in \Sigma$: a_1 sendet eine Nachricht m an a_2
- Besonderheit: Alle Ereignisse sind unmittelbar (jedoch keine echte Einschränkung).

Globale Ordnung der Ereignisse

Beschriftete Halbordnung - lpo



- Zusammenfassen zeitgleicher Ereignisse zu Äquivalenzklassen.
 ➔ Strikte Halbordnung.
- Äquivalenzklassen werden eindeutig mit Ereignissen beschriftet.
- beschriftete strikte Halbordnung
 (engl. *labeled partial order*, kurz: **lpo**).
- Bezeichnung: $\mathcal{L}_C = \langle [L_C]_{\sim}, \prec_C, \phi_C \rangle$
- \mathcal{L}_C : Menge aller erreichbaren Positionen
 (gemeint sind eigentlich Äquivalenzklassen).

Erfüllbarkeit

von Basic Charts

- **Linearisierung**: Folge von Ereignissen.
formal: *lineare Ordnung* (totale Ordnung), die die lpo erweitert.

Definition ($r \models_B C$)

Ereignisfolge $r = e_1 \dots e_n \in \Sigma^*$ erfüllt Basic Chart $C \iff$
 $r|_{\Sigma(C)}$ ist Linearisierung der lpo $\langle [L_C]_{\sim}, \prec_C, \phi_C \rangle$.

Erfüllbarkeit

von Live Sequence Charts

Definition ($r \models_L S$)

$r = e_1 e_2 \dots \in \Sigma^\omega$ erfüllt einen LSC $S \iff$

- ① $S = \triangleright C \wedge \exists i : i \geq 1 : e_1 \dots e_i \models_B C$
- ② $S = \diamond C \wedge \exists i, j : 1 \leq i \leq j : e_i \dots e_j \models_B C$
- ③ $S = \square(C_1, C_2) \wedge \forall i, j : 1 \leq i \leq j :$

$$(e_i \dots e_j \models_B C_1) \implies (\exists k : j \leq k : e_{j+1}, \dots, e_k \models_B C_2).$$

Erfüllbarkeit

von Live Sequence Charts

Beispiel

$(\text{askCocoa} \cdot \text{insertCoin} \cdot \text{prepCocoa} \cdot \text{serveCocoa})^\omega$ (1)

$(\text{askCocoa} \cdot \text{moneyBack})^\omega$ (2)

$(\text{askCocoa} \cdot \text{insertCoin} \cdot \text{prepCocoa} \cdot \text{moneyBack} \cdot \text{serveCocoa})^\omega$ (3)

$\text{askCocoa} \cdot \text{insertCoin} \cdot \text{prepCocoa} \cdot (\text{iAmEmpty})^\omega$ (4)

- (1) und (2) sind jeweils Modell von LSC (a).
- (3) und (4) sind *keine* Modelle von LSC (a).

Erfüllbarkeit einer Spezifikation

Definition ($R \models_S S$)

Eine Menge unendlicher Läufe R ist Modell von $S \iff$

- 1 $\forall S \in \mathcal{S}_{\triangleright} \cup \mathcal{S}_{\square} : \forall r \in R : r \models_L S$
- 2 $\forall S \in \mathcal{S}_{\diamond} : \exists r \in R : r \models_L S.$

Probleme mit der Definition von Erfüllbarkeit

- Semantik eines Charts hängt maßgeblich von Linearisierung ab.
- Mit einem Wort können zugleich mehrere Positionen im Chart erreicht werden.

Beispiel

$w = \text{getWater} \cdot \text{pourWater} \cdot \text{getWater} \cdot \text{pourWater} \cdot \text{getWater} \cdot \text{pourWater}$

Cuts

- Cut $c \subseteq \mathcal{L}_C$: „Schon erreichte“ Positionen (bzw. Äquivalenzklassen).
- Cuts sind vorgängerabgeschlossen.
- Erreichbarkeit von Cuts: $c \xrightarrow{e} c'$

Generierung von Cuts

Definition (Generierung von Cuts)

$w \in \Sigma^*$ generiert einen Cut c in einer lpo $\mathcal{L}_C \iff$
ein Suffix von $w|_{\Sigma(C)}$ ist eine Linearisierung von c .

- Menge aller von w generierten Cuts: $gen(w, \mathcal{L}_C)$
- ➔ Zustandsbeschreibung eines LSCs.

- Das ergibt für einen Chart C für sämtliche Wörter:
 $Gen(C) = \{gen(w, \mathcal{L}_C) \mid w \in \Sigma^*\}$.
- Beschreibung von **endlichen Zustandsraum**:

$$|Gen(\mathcal{L}_C)| = 2^{O(n \log n)}$$

Notwendige und verbotene Ereignisse

- In jedem Zustand (Menge von Cuts) eines LSCs gibt es
 - verbotene Ereignisse
 - notwendige Ereignisse
- Verbotene Ereignisse treten nie auf \implies **Safety**.
- Notwendige Ereignisse treten irgendwann auf \implies **Liveness**.

Beispiel

```
w = insertCoin · askCocoa · prepCocoa
```

Liveness und Safety

Definition (Verbotene Ereignisse, Safety)

S **verbietet** $e \in \Sigma$ hinter w ($\text{forbid}(w, e)$) $\iff \exists \square(C_1, C_2) \in S \exists c \in \text{gen}(w, C_1 \cdot C_2)$, so daß:

- $\mathcal{L}_{C_1} \subseteq c \subset \mathcal{L}_{C_1 \cdot C_2}$, d. h. der Chart ist aktiv;
- $e \in \Sigma(C_1 \cdot C_2)$;
- $\nexists c' : c \xrightarrow{e} c'$.

$e_1 e_2 \dots \in \Sigma^\omega$ heißt **e-sicher** $\iff \forall i : 1 \leq i : \text{forbid}(e_1 \dots e_i, e) \implies e \neq e_{i+1}$.

Definition (Notwendige Ereignisse, Liveness)

S **benötigt** $e \in \Sigma$ hinter $w \in \Sigma^*$ ($\text{require}(w, e)$) $\iff \exists \square(C_1, C_2) \in S \exists c \in \text{gen}(w, C_1 \cdot C_2)$, so daß:

- $\mathcal{L}_{C_1} \subseteq c \subset \mathcal{L}_{C_1 \cdot C_2}$, d. h. der Chart ist aktiv;
- $e \in \Sigma(C_1 \cdot C_2)$;
- $\exists c' : c \xrightarrow{e} c'$.

$e_1 e_2 \dots \in \Sigma^\omega$ heißt **e-lebendig** $\iff \forall i : 1 \leq i : \text{require}(e_1 \dots e_i, e) \implies \exists k : i < k : e = e_k$.

Erfüllbarkeit entspricht Liveness und Safety

Satz (Liveness + Safety = LSCs)

S : Spezifikation, nur bestehend aus Universal LSCs,

R : Menge aus unendlichen Läufen.

$$R \models_S S \iff \forall r \in R : \forall e \in \Sigma : r \text{ ist } e\text{-sicher und } e\text{-lebendig.}$$

- Errungenschaft: Eine Sicherheitsbedingung und Lebendigkeitsbedingung pro Ereignis.

Was haben wir bis jetzt erreicht?

- Cuts
 - Endlicher Zustandsraum von LSCs für beliebige Läufe.
 - Zustand: $gen(w, \mathcal{L}_C) \in Gen(C)$.
- Liveness und Safety
 - Bedingungen für Zustandsübergänge, die weiterhin Erfüllbarkeit ermöglichen.

Problemstellung (1)

- Spezifikation beschreibt sowohl **System** als auch **Umgebung** (engl. **Environment**).
- Partitionierung der Instanzen: $Ag = E \uplus S$
- Partitionierung der Ereignisse: $\Sigma = \Sigma_E \uplus \Sigma_S$
- LSC Spezifikation:
 - $e \in \Sigma_E$: gegebene Annahmen.
 - $e \in \Sigma_S$: zu erfüllende Garantien.
- Ziel: Implementierung für das **System** finden, falls möglich.

Problemstellung (2)

- gesucht: Strategie f für das System $f(f(r) \cdot r') \in \Sigma^*$
- Resultat einer Strategie f : $Out(f) \in \mathcal{P}(\Sigma^\omega)$
- Annahmen über die **Umgebung**:

Definition (**WB**(Σ_E))

Eine Umgebung E heißt **wohlverhaltend** (engl. well-behaving) in einem Lauf $r \in \Sigma^\omega \iff r$ ist Σ_E -sicher \wedge Σ_E -lebendig.

Menge aller Läufe mit wohlverhaltender Umgebung: **WB**(Σ_E).

- $Out(f) \cap \mathbf{WB}(\Sigma_E) \models_S \mathcal{S}$

Das Konsistenzproblem

- Universal LSC Spezifikation \mathcal{S} ist **konsistent** \iff Konsistenzbedingung erfüllt.
- Eine Konsistenzbedingung:
 $\exists f : \forall w \in \text{Out}(f) :$
 w ist Σ_E -sicher $\wedge w$ ist Σ_E -lebendig
 $\implies w$ ist Σ_S -sicher $\wedge w$ ist Σ_S -lebendig
- $\text{Out}(f) \cap \mathbf{WB}(\Sigma_E) \models_S \mathcal{S}$

Algorithmus (Modellierung als Spiel)

Transitionssystem

- Transitionen zwischen sicheren Zuständen:

Definition ($\gamma \stackrel{e}{\Longrightarrow} \gamma'$)

Sei $S = \square(C_1, C_2)$ und $Gen(S) = Gen(C_1 \cdot C_2)$.

$\forall \gamma, \gamma' \in Gen(S), e \in \Sigma : \gamma \stackrel{e}{\Longrightarrow} \gamma' \iff$

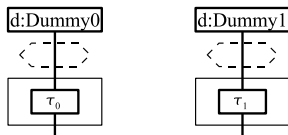
- $e \notin \Sigma(S)$ und es gilt $\gamma = \gamma'$,
- oder $e \in \Sigma(S)$ und es gilt
 - $\gamma' = \{\emptyset\} \cup \{c' \mid \exists c \in \gamma : c \xrightarrow{e} c'\}$
 - und $\forall c \in \gamma : \mathcal{L}_{C_1} \subseteq c \subset \mathcal{L}_{C_1 \cdot C_2} \implies \exists c' : c \xrightarrow{e} c'$

- $\{\emptyset\} \stackrel{w}{\Longrightarrow} \gamma \iff w$ ist Σ -sicher und $\gamma = gen(w, C_1 \cdot C_2)$

Algorithmus (Modellierung als Spiel)

„Dummy“-Ereignisse zum Spielerwechsel

- τ_0 : Ende einer Ereignisfolge des **Systems** (Spieler 0)
- τ_1 : Ende einer Ereignisfolge der **Umgebung** (Spieler 1)
- Zugehörige Fairness Szenarien:



- Ereignisse in Σ_E und Σ_S tragen Indexnummern.
 τ_0 bzw. τ_1 sind das $[|\Sigma_S| + 1]$ -te bzw. $[|\Sigma_E| + 1]$ -te Ereignis.

Algorithmus (Modellierung als Spiel)

Spielgraph (1)

Spielgraph: $G_S = \langle V, V_0, \Delta, \Omega \rangle$

- V - Menge der Knoten:

$$V = (\{0, 1\} \times \Sigma \times \text{Gen}(S_1) \times \dots \times \text{Gen}(S_n) \times [|\Sigma_S| + 1] \times [|\Sigma_E| + 1]) \cup \{\text{sink}_0, \text{sink}_1\}$$

Ein Knoten hat die Form $(i, e, \gamma_1, \dots, \gamma_n, c_0, c_1)$

- $V_0 = \{\text{sink}_1\} \cup \{(i, e, \gamma_1 \dots \gamma_n, c_0, c_1) \in V \mid i = 0\}$

Algorithmus (Modellierung als Spiel)

Spielgraph (2)

Spielgraph: $G_S = \langle V, V_0, \Delta, \Omega \rangle$

- Δ - Transitionsrelation:

- 1 $\Delta((i, e, \gamma_1 \dots \gamma_n, c_0, c_1), (i', e', \gamma'_1 \dots \gamma'_n, c'_0, c'_1)),$
falls „im Wesentlichen“ $\gamma_j \xrightarrow{e'} \gamma'_j, (1 \leq j \leq n)$
d. h. Lauf bleibt Σ -sicher.
- 2 $\Delta((i, e, \gamma_1 \dots \gamma_n, c_0, c_1), \text{sink}_i),$
sonst.
- 3 $\Delta(\text{sink}_i, \text{sink}_i).$

Algorithmus (Modellierung als Spiel)

Spielgraph (3)

Spielgraph: $G_S = \langle V, V_0, \Delta, \Omega \rangle$

- $\Omega \subseteq V^\omega$ - Gewinnbedingung für das **System**:

$$\Omega = \text{Streett}(\{(F_E, F_S)\})$$

$$:= \{r \in V^\omega \mid \text{inf}(r) \cap F_E \neq \emptyset \Rightarrow \text{inf}(r) \cap F_S \neq \emptyset\}$$

- $F_E = \{(i, e, \gamma_1 \dots \gamma_n, c_0, c_1) \mid c_1 = |\Sigma_E| + 1\} \cup \{\text{sink}_0\}$
 - $F_S = \{(i, e, \gamma_1 \dots \gamma_n, c_0, c_1) \mid c_0 = |\Sigma_S| + 1\} \cup \{\text{sink}_1\}$
- Entspricht der Konsistenzbedingung
 Σ_E -sicher \implies (Σ_S -sicher \wedge (Σ_E -lebendig \implies Σ_S -lebendig)).

Umformen in Paritätsspiel (1)

- Universal LSC Spezifikation \mathcal{S} **konsistent** \iff
 $\exists f : f$ ist Gewinnstrategie für Spieler 0 auf $G_{\mathcal{S}}$.
 - Ziel: Finden einer Gewinnstrategie f .
 - Für Paritätsspiele existieren dazu bereits effiziente Algorithmen.
- ➔ Transformation von $G_{\mathcal{S}}$ in Paritätsspiel. ▶ Skip

Umformen in Paritätsspiel (2)

- Paritätsspiel: Spielgraph $G_S = \langle V, V_0, \Delta, \Omega_c \rangle$
mit **Färbung** $\Omega_c : V \rightarrow \{1, \dots, k\}$ für ein $k \in \mathbb{N}$
- Streett Bedingung: $\Omega = \text{Streett}(\{(F_E, F_S)\})$
 $:= \{r \in V^\omega \mid \text{inf}(r) \cap F_E \neq \emptyset \Rightarrow \text{inf}(r) \cap F_S \neq \emptyset\}$
- Äquiv. Paritätsbedingung: Färbung Ω mit
 - $\Omega_c(v) = 2$, falls $v \in F_S$;
 - $\Omega_c(v) = 1$, falls $v \in F_E \setminus F_S$;
 - $\Omega_c(v) = 0$, sonst.

Komplexität

- Finden von Gewinnstrategien für Paritätsspiele mit nur 3 Farben in polynomineller Zeit bezüglich der Anzahl der Knoten möglich.
- Algorithmus liefert Implementierung f , falls \mathcal{S} konsistent, oder Sabotageplan, falls \mathcal{S} inkonsistent.

- Komplexität:

$$\begin{aligned}
 & (\{0, 1\} \times \Sigma \times \text{Gen}(\mathcal{S}_1) \times \dots \times \text{Gen}(\mathcal{S}_s) \times [|\Sigma_S| + 1] \times [|\Sigma_E| + 1]) \\
 & \quad + \\
 & |\text{Gen}(\mathcal{L}_C)| = 2^{O(n \log n)} \\
 & \quad = \\
 & O(|\Sigma| |\Sigma_S| |\Sigma_E| 2^{s n \log n})
 \end{aligned}$$

- Konsistenzproblem ist sowohl co-NP-schwer als auch NP-schwer.

Zusammenfassung

- Zustandsbeschreibung von LSCs durch Cuts.
- Liveness und Safety
 - Bedingungen pro Ereignis.
 - Bedingungen kann Instanzen zugeordnet werden.
- Konsistenzproblem.
- Lösung durch spieltheoretischen Algorithmus.

- Ausblick:
 - Benachteiligung unerwünschter Strategien („Mercifulness“).
 - Erweiterungen für Initial und Existential LSCs.

Vielen Dank
für die Aufmerksamkeit.