# Synthese Reaktiver Systeme durch Live Sequence Charts

Frank Radmacher

RWTH Aachen

*Seminar: Message Sequence Charts (WS 2004/05)*

13. Januar 2005

# 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

# 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 Compter Science. Springer-Verlag, 2002.

# 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 Compter 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

# 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.

# Gliederung

# Erweiterung von MSCs zu Live Sequence Charts

- *Koregionen*: Ereignisse in beliebiger Reihenfolge.
- Drei Modi:
  - Initial LSCs
  - Existential LSCs (mögliches Verhalten)
  - Universal LSCs (notwendiges Verhalten)
- *Beschränkte Ereignisse*: dürfen nicht auftreten, wenn der Chart aktiv ist.
- LSCs bestehen aus Basic Charts.

# 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) ($S = \Box(C_1, C_2)$)
- *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).

# Nachrichtenalphabet

- *Ag*: Endliche Menge von *Instanzen* (*Ag* für engl. *Agents*).
- $\mathcal{M}$: Endliche Menge von *Nachrichtenbezeichnern*.

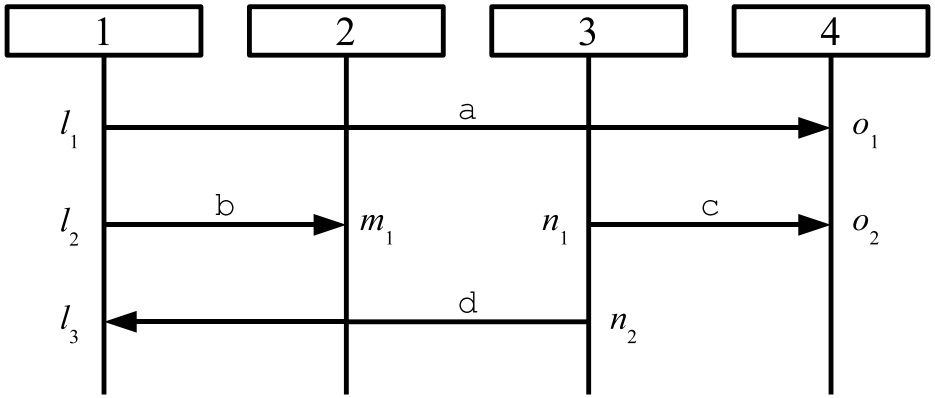
➥ Endliches Alphabet $\Sigma = Ag \times \mathcal{M} \times Ag$:

- 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

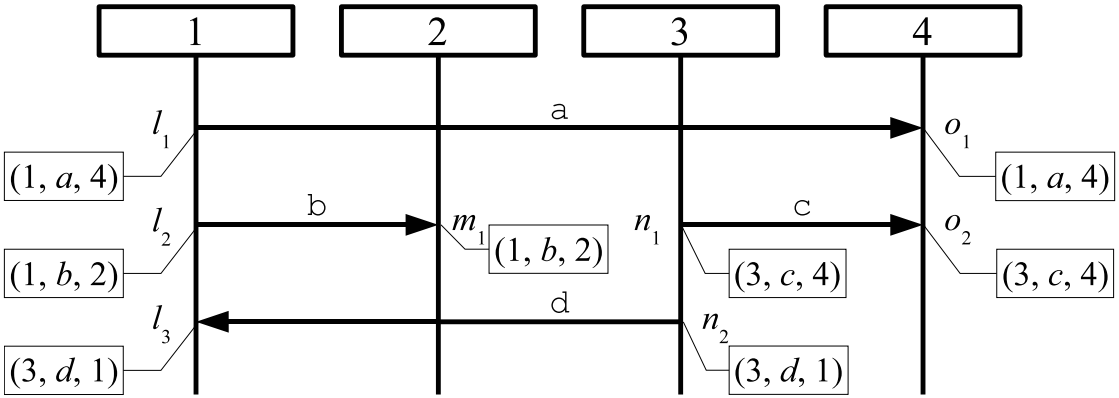
## Beschriftete Halbordnung - lpo




- 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 Äuivalenzklassen).

# Globale Ordnung der Ereignisse

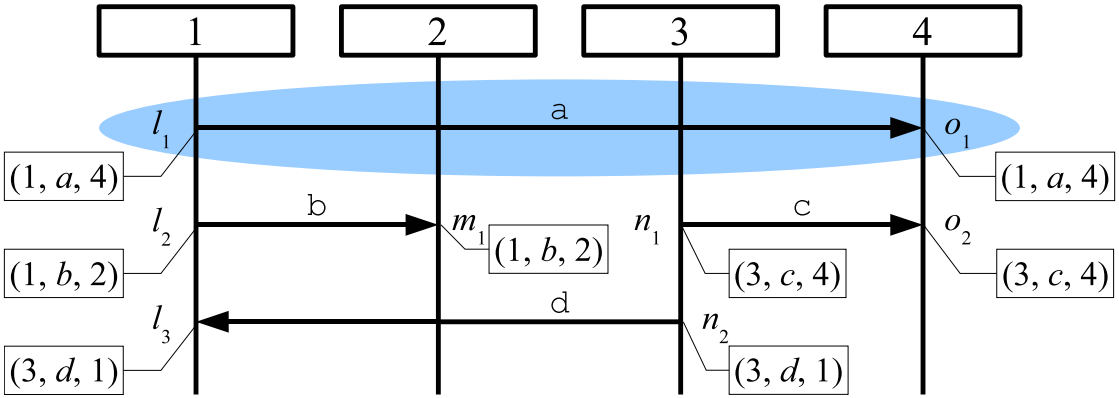
## Beschriftete Halbordnung - lpo




- 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 Äuivalenzklassen).

# Globale Ordnung der Ereignisse

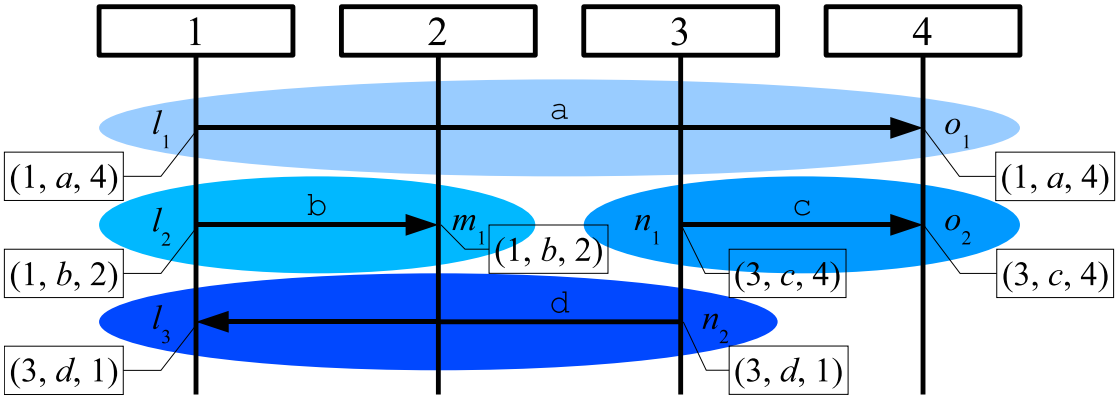
## Beschriftete Halbordnung - lpo

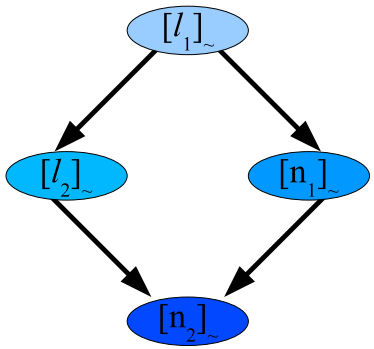



- Zusammenfassen zeitgleicher Ereignisse zu Äquivalenzklassen.
  - ➥ Strikte Halbordnung.
- Äquivalenzklassen werden eindeutig mit Ereignissen beschriftet.
- beschriftete strikte Halbordnung
  (engl. *labeled partial order*, kurz: lpo).
- Bezeichnung: $\mathcal{L}_C = \langle [L_C]_\sim, \prec_C, \phi_C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

# Globale Ordnung der Ereignisse

## Beschriftete Halbordnung - lpo

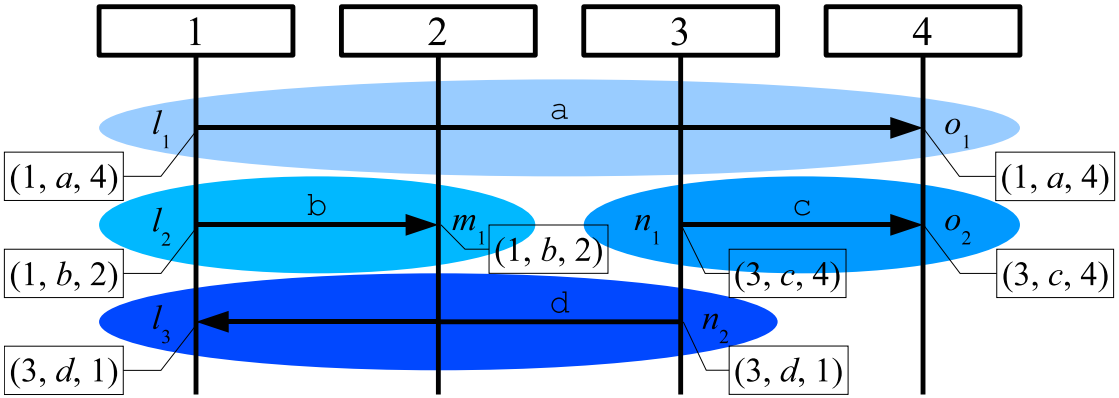


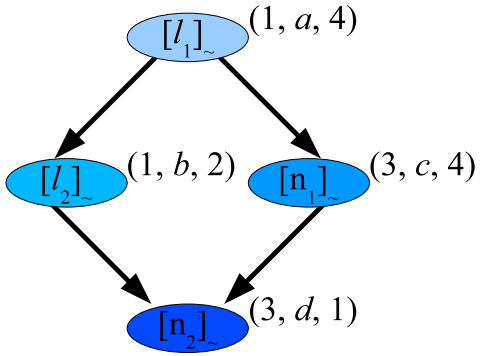




- Zusammenfassen zeitgleicher Ereignisse zu Äquivalenzklassen.
  - ➥ Strikte Halbordnung.
- Äquivalenzklassen werden eindeutig mit Ereignissen beschriftet.
- beschriftete strikte Halbordnung
  (engl. *labeled partial order*, kurz: lpo).
- Bezeichnung: $\mathcal{L}_C = \langle [L_C]_\sim, \prec_C, \phi_C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

# Globale Ordnung der Ereignisse

## Beschriftete Halbordnung - lpo

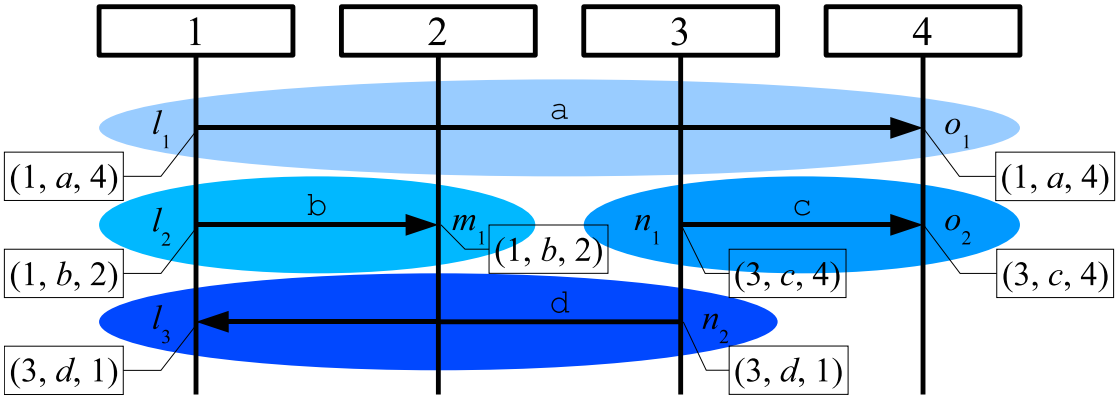


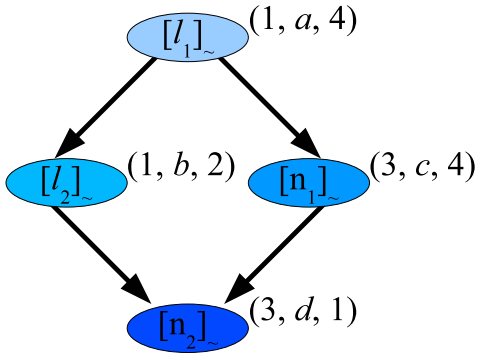




- Zusammenfassen zeitgleicher Ereignisse zu Äquivalenzklassen.
  - ➥ Strikte Halbordnung.
- Äquivalenzklassen werden eindeutig mit Ereignissen beschriftet.
- beschriftete strikte Halbordnung
  (engl. *labeled partial order*, kurz: lpo).
- Bezeichnung: $\mathcal{L}_C = \langle [L_C]_\sim, \prec_C, \phi_C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

# 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 Äuivalenzklassen).

# 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 ($r \models_L S$)**

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

1. $S = \triangleright C \ \wedge \ \exists i : i \geq 1 : \ e_1 \ldots e_i \models_B C$
2. $S = \Diamond C \ \wedge \ \exists i, j : 1 \leq i \leq j : \ e_i \ldots e_j \models_B C$
3. $S = \Box(C_1, C_2) \ \wedge \ \forall i, j : 1 \leq i \leq j :$

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

# Erfüllbarkeit
## von Live Sequence Charts

### Beispiel

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

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

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

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

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

# Erfüllbarkeit
## einer Spezifikation

**Definition ($R \models_S \mathcal{S}$)**

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

1. $\forall S \in \mathcal{S}_\triangleright \cup \mathcal{S}_\square : \quad \forall r \in R : \; r \models_L S$
2. $\forall S \in \mathcal{S}_\Diamond : \qquad \exists r \in R : \; r \models_L S.$

# 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**

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

# Cuts

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

# Cuts

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

Beispiel

# 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:

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

# 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:

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

# 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:

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

# Gliederung

# Notwendige und verbotene Ereignisse

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

**Beispiel**

$w = \texttt{insertCoin} \cdot \texttt{askCocoa} \cdot \texttt{prepCocoa}$

# Liveness und Safety

## Definition (Verbotene Ereignisse, Safety)

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

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

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

## Definition (Notwendige Ereignisse, Liveness)

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

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

$e_1 e_2 \ldots \in \Sigma^\omega$ heißt $e$-lebendig $\iff \forall i : 1 \leq i : require(e_1 \ldots e_i, e) \Longrightarrow \exists k : i < k : e = e_k$.

# Liveness und Safety

## Definition (Verbotene Ereignisse, Safety)

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

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

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

## Definition (Notwendige Ereignisse, Liveness)

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

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

$e_1 e_2 \ldots \in \Sigma^\omega$ heißt $e$-lebendig $\iff$ $\forall i : 1 \leq i : require(e_1 \ldots e_i, e) \Longrightarrow \exists k : i < k : e = e_k$.

# Erfüllbarkeit entspricht Liveness und Safety

**Satz (Liveness + Safety = LSCs)**

*$\mathcal{S}$: Spezifikation, nur bestehend aus Universal LSCs,*
*$R$: Menge aus unendlichen Läufen.*

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

- 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.

# Gliederung

# 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 (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 ($\mathbf{WB}(\Sigma_E)$)**

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

Menge aller Läufe mit wohlverhaltender Umgebung: $\mathbf{WB}(\Sigma_E)$.

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

# 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 ($\mathbf{WB}(\Sigma_E)$)**

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

Menge aller Läufe mit wohlverhaltender Umgebung: $\mathbf{WB}(\Sigma_E)$.

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

# 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 ($\mathbf{WB}(\Sigma_E)$)**

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

Menge aller Läufe mit wohlverhaltender Umgebung: $\mathbf{WB}(\Sigma_E)$.

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

# 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 ($\mathbf{WB}(\Sigma_E)$)**

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

Menge aller Läufe mit wohlverhaltender Umgebung: $\mathbf{WB}(\Sigma_E)$.

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

# 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 ($\mathbf{WB}(\Sigma_E)$)**

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

Menge aller Läufe mit wohlverhaltender Umgebung: $\mathbf{WB}(\Sigma_E)$.

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

# 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 ($\mathbf{WB}(\Sigma_E)$)**

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

Menge aller Läufe mit wohlverhaltender Umgebung: $\mathbf{WB}(\Sigma_E)$.

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

# 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 ($\mathbf{WB}(\Sigma_E)$)**

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

Menge aller Läufe mit wohlverhaltender Umgebung: $\mathbf{WB}(\Sigma_E)$.

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

# Das Konsistenzproblem

- Universal LSC Spezifikation $\mathcal{S}$ ist konsistent $\Longleftrightarrow$ Konsistenzbedingung erfüllt.
- Eine Konsistenzbedingung:
  $\exists f : \forall w \in \mathit{Out}(f) :$
  $w$ ist $\Sigma_E$-sicher $\wedge$ $w$ ist $\Sigma_E$-lebendig
  $\Longrightarrow$ $w$ ist $\Sigma_S$-sicher $\wedge$ $w$ ist $\Sigma_S$-lebendig
- $\mathit{Out}(f) \cap \mathbf{WB}(\Sigma_E) \models_S \mathcal{S}$

# Das Konsistenzproblem

- Universal LSC Spezifikation $\mathcal{S}$ ist konsistent $\Longleftrightarrow$ Konsistenzbedingung erfüllt.
- Eine Konsistenzbedingung:
  $\exists f : \forall w \in \mathit{Out}(f) :$
  $w$ ist $\Sigma_E$-sicher $\wedge$ $w$ ist $\Sigma_E$-lebendig
  $\Longrightarrow$ $w$ ist $\Sigma_S$-sicher $\wedge$ $w$ ist $\Sigma_S$-lebendig
- $\mathit{Out}(f) \cap \mathbf{WB}(\Sigma_E) \models_S \mathcal{S}$

# Algorithmus (Modellierung als Spiel)
## Transitionssystem

- Transitionen zwischen sicheren Zuständen:

**Definition ($\gamma \overset{e}{\Longmapsto} \gamma'$)**

Sei $S = \Box(C_1, C_2)$ und $Gen(S) = Gen(C_1 \cdot C_2)$.
$\forall \gamma, \gamma' \in Gen(S), e \in \Sigma : \gamma \overset{e}{\Longmapsto} \gamma' \iff$

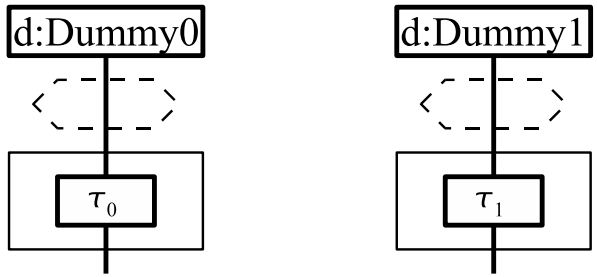
- $e \notin \Sigma(S)$ und es gilt $\gamma = \gamma'$,
- oder $e \in \Sigma(S)$ und es gilt
  - $\gamma' = \{\emptyset\} \cup \{c' \mid \exists c \in \gamma : c \xrightarrow{e} c'\}$
  - und $\forall c \in \gamma : \mathcal{L}_{C_1} \subseteq c \subset \mathcal{L}_{C_1 \cdot C_2} \Longrightarrow \exists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

- $\{\emptyset\} \overset{w}{\Longmapsto} \gamma \iff w$ ist $\Sigma$-sicher und $\gamma = gen(w, C_1 \cdot C_2)$

# Algorithmus (Modellierung als Spiel)

## „Dummy“-Ereignisse zum Spielerwechsel

- $\tau_0$: Ende einer Ereignisfolge des Systems (Spieler 0)
- $\tau_1$: Ende einer Ereignisfolge der Umgebung (Spieler 1)
- Zugehörige Fairness Szenarien:




- Ereignisse in $\Sigma_E$ und $\Sigma_S$ tragen Indexnummern.
  $\tau_0$ bzw. $\tau_1$ 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:

  $V = (\{0,1\} \times \Sigma \times Gen(S_1) \times \ldots Gen(S_n) \times [|\Sigma_S| + 1] \times [|\Sigma_E| + 1])$
  $\cup \ \{sink_0, sink_1\}$

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

- $V_0 = \{sink_1\} \cup \{(i, e, \gamma_1 \ldots \gamma_n, c_0, c_1) \in V \mid i = 0\}$

# Algorithmus (Modellierung als Spiel)
## Spielgraph (1)

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

- $V$ - Menge der Knoten:

  $V = (\{0,1\} \times \Sigma \times Gen(S_1) \times \ldots Gen(S_n) \times [|\Sigma_S| + 1] \times [|\Sigma_E| + 1])$
  $\cup \ \{sink_0, sink_1\}$

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

- $V_0 = \{sink_1\} \cup \{(i, e, \gamma_1 \ldots \gamma_n, c_0, c_1) \in V \mid i = 0\}$

# Algorithmus (Modellierung als Spiel)
## Spielgraph (1)

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

- $V$ - Menge der Knoten:

  $V = (\{0,1\} \times \Sigma \times Gen(S_1) \times \ldots Gen(S_n) \times [|\Sigma_S| + 1] \times [|\Sigma_E| + 1])$
  $\cup \ \{sink_0, sink_1\}$

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

- $V_0 = \{sink_1\} \cup \{(i, e, \gamma_1 \ldots \gamma_n, c_0, c_1) \in V \mid i = 0\}$

# Algorithmus (Modellierung als Spiel)
## Spielgraph (2)

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

- $\Delta$ - Transitionsrelation:

1. $\Delta((i, e, \gamma_1 \ldots \gamma_n, c_0, c_1), (i', e', \gamma'_1 \ldots \gamma'_n, c'_0, c'_1))$,
   falls „im Wesentlichen “ $\gamma_j \overset{e'}{\Longmapsto} \gamma'_j, (1 \leq j \leq n)$
   d. h. Lauf bleibt $\Sigma$-sicher.
2. $\Delta((i, e, \gamma_1 \ldots \gamma_n, c_0, c_1), sink_i)$,
   sonst.
3. $\Delta(sink_i, sink_i)$.

# Algorithmus (Modellierung als Spiel)
## Spielgraph (3)

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

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

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

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

  - $F_E = \{(i, e, \gamma_1 \ldots \gamma_n, c_0, c_1) \mid c_1 = |\Sigma_E| + 1\} \cup \{sink_0\}$
  - $F_S = \{(i, e, \gamma_1 \ldots \gamma_n, c_0, c_1) \mid c_0 = |\Sigma_S| + 1\} \cup \{sink_1\}$

- Entspricht der Konsistenzbedingung

  $\Sigma_E$-sicher $\Longrightarrow$ ($\Sigma_S$-sicher $\wedge$ ($\Sigma_E$-lebendig $\Longrightarrow$ $\Sigma_S$-lebendig)).

# Algorithmus (Modellierung als Spiel)
## Spielgraph (3)

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

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

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

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

  - $F_E = \{(i, e, \gamma_1 \ldots \gamma_n, c_0, c_1) \mid c_1 = |\Sigma_E| + 1\} \cup \{sink_0\}$
  - $F_S = \{(i, e, \gamma_1 \ldots \gamma_n, c_0, c_1) \mid c_0 = |\Sigma_S| + 1\} \cup \{sink_1\}$

- Entspricht der Konsistenzbedingung

  $\Sigma_E$-sicher $\Longrightarrow$ ($\Sigma_S$-sicher $\wedge$ ($\Sigma_E$-lebendig $\Longrightarrow$ $\Sigma_S$-lebendig)).

# Umformen in Paritätsspiel (1)

- Universal LSC Spezifikation $\mathcal{S}$ konsistent $\Longleftrightarrow$
  $\exists f$ : $f$ 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 $\Omega_c : V \to \{1, \ldots, k\}$ für ein $k \in \mathbb{N}$
- Streett Bedingung: $\Omega = \text{Streett}(\{(F_E, F_S)\})$
  $:= \{r \in V^\omega \mid \inf(r) \cap F_E \neq \emptyset \Rightarrow \inf(r) \cap F_S \neq \emptyset\}$
- Äquiv. Paritätsbedingung: Färbung $\Omega$ mit
  - $\Omega_c(v) = 2$, falls $v \in F_S$;
  - $\Omega_c(v) = 1$, falls $v \in F_E \backslash F_S$;
  - $\Omega_c(v) = 0$, sonst.

# Umformen in Paritätsspiel (2)

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

# Umformen in Paritätsspiel (2)

- Paritätsspiel: Spielgraph $G_S = \langle V, V_0, \Delta, \Omega_c \rangle$
  mit Färbung $\Omega_c : V \to \{1, \ldots, k\}$ für ein $k \in \mathbb{N}$
- Streett Bedingung: $\Omega = \text{Streett}(\{(F_E, F_S)\})$
  $:= \{r \in V^\omega \mid \inf(r) \cap F_E \neq \emptyset \Rightarrow \inf(r) \cap F_S \neq \emptyset\}$
- Äquiv. Paritätsbedingung: Färbung $\Omega$ 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:

$$(\{0,1\} \times \Sigma \times Gen(S_1) \times \ldots Gen(S_s) \times [|\Sigma_S| + 1] \times [|\Sigma_E| + 1])$$
$$+$$
$$|Gen(\mathcal{L}_C)| = 2^{O(n \log n)}$$
$$=$$
$$O(|\Sigma|\ |\Sigma_S|\ |\Sigma_E|\ 2^{s\ n \log n})$$

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

# 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:

$$(\{0,1\} \times \Sigma \times \mathit{Gen}(S_1) \times \ldots \mathit{Gen}(S_s) \times [|\Sigma_S| + 1] \times [|\Sigma_E| + 1])$$
$$+$$
$$|\mathit{Gen}(\mathcal{L}_C)| = 2^{O(n \log n)}$$
$$=$$
$$O(|\Sigma|\ |\Sigma_S|\ |\Sigma_E|\ 2^{s\ n \log n})$$

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

# Gliederung

# 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.

# 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.