Synthesis from Scenarios

Christof Löding
RWTH Aachen
Germany

Yves Bontemps, University of Namur, Belgium
Pierre-Yves Schobbens, University of Namur, Belgium

based on the thesis of Yves Bontemps
OVERVIEW

Specification \rightarrow \text{Synthesis} \rightarrow \text{System}
OVERVIEW

Specification

Synthesis

System

Scenarios (live sequence charts)

games/strategies

open centralised
Introduction

**Specification**: live sequence charts
- concrete syntax
- abstract syntax
- semantics
- definable languages

**System**
- structure
- implementation

**Synthesis**
Why Scenarios?

Usually specifications are written in some logic

**Advantages:**
- clear semantics
- expressive
- well understood

**Drawbacks:** specifications for complete systems are
- tedious to write
- and difficult to read

\[ \text{AG}(\neg \text{req} \rightarrow \text{AX}(\text{req} \rightarrow \neg \text{E}((\neg \text{ack} \cup (\neg \text{req} \land \neg \text{ack} \land \text{E}(\neg \text{ack} \cup \text{req})))))) \]
Two simple scenarios (basic MSCs) for the specification of a coffee machine:

- **Scenario 1:**
  - Customer: insertCoin, askCoffee, prepCoffee, serveCoffee
  - Machine: askCoffee

- **Scenario 2:**
  - Customer: insertCoin
  - Machine: insertCoin, askMoney, giveMoney
PROBLEMS

- No distinction between triggering and responding events.
- Possible or required scenarios?
- Events that do not appear in a scenario: forbidden / don’t care?
- ...
PROBLEMS

- No distinction between triggering and responding events.
- Possible or required scenarios?
- Events that do not appear in a scenario: forbidden / don’t care?
  
  Problems addressed by Damm/Harel (1999):

  Live Sequence Charts (LSCs)
• Introduction

• **Specification**: live sequence charts
  – concrete syntax
  – abstract syntax
  – semantics
  – definable languages

• **System**
  – structure
  – implementation

• **Synthesis**
LSC – SYNTAX

Small subset of features introduced by Damm/Harel:

- Pre- and main charts for distinction of triggering and responding events
- Events considered by a scenario are made explicit
- Subcharts for modelling choice
LSC – Syntax by Example

pre-chart (triggering)
- askCoffee
- insertCoin
- prepCoffee

main chart (response)
- serveCoffee
giveMoney

co-regions (no order on events)
Customer
Machine

Dagstuhl, June 13, 2005
LSC – Syntax by Example

Customer

Machine

askCoffee

insertCoin

prepCoffee

serveCoffee

pre-chart (triggering)

main chart (response)
**LSC – Syntax by Example**

Diagram showing a scenario involving a Customer and a Machine.

- **Customer**:
  - askCoffee
  - insertCoin

- **Machine**:
  - prepCoffee
  - serveCoffee

- The scenario considers giving money.

The diagram indicates a pre-chart (triggering) and a main chart (response).
LSC – Syntax by Example

Customer

Machine

askCoffee

insertCoin

prepCoffee

serveCoffee

co-regions (unordered events)

pre-chart (triggering)

main chart (response)

considers giveMoney
Modelling `alternatives / choice` using subcharts:
Modelling **alternatives / choice** using subcharts:
**Abstract Syntax**

concrete: scenario

```
P_1   P_2   P_3   P_4
    \downarrow m_1
    \downarrow m_2
    \downarrow m_3
    \downarrow m_4
```

```
\text{ALT}
\downarrow m_5
\downarrow m_6
\downarrow m_7
\downarrow m_8
\downarrow m_9
\downarrow m_{10}
\downarrow m_{11}
```

Dagstuhl, June 13, 2005
**Abstract Syntax**

**Concrete:** scenario

**Abstract:** partial order

\[ P : \begin{array}{c} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \\ m_{10} \\ m_{11} \end{array} \]

\[ M : \begin{array}{c} m_4 \\ M_1 \\ m_5 \\ M_2 \\ m_8 \\ M_1 \\ m_9 \\ m_{10} \end{array} \]

\[ \text{ALT} \]

\[ P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \]
**Abstract Syntax**

**Concrete:** scenario

**Abstract:** partial order

\[
P : \quad \frac{m_1}{m_2} \frac{m_3}{m_4}
\]

\[
M : \quad \frac{m_4}{M_1 + M_2} \frac{m_5}{m_6} \frac{m_7}{m_8} \frac{m_9}{m_{10}} \frac{m_{11}}{}
\]
**Abstract Syntax**

**Concrete:** scenario

**Abstract:** partial order

\[ P : \begin{array}{c} m_1 \\ m_2 \\ m_3 \end{array} \]

\[ M : \begin{array}{c} m_4 \\ M_1 + M_2 \\ m_{11} \end{array} \]

\[ M_1 : \begin{array}{c} m_5 \\ m_6 \\ m_7 \end{array} \]

\[ M_2 : \begin{array}{c} m_8 \\ m_9 \\ m_{10} \end{array} \]
Labelled partial order (LPO) \( \mathcal{L} = (D, \leq, \Sigma_R, \lambda) \)

- finite set \( D \) of locations (domain)
- partial order relation \( \leq \subseteq D \times D \)
- label alphabet \( \Sigma_R \) (labels restricted by \( \mathcal{L} \))
- labelling function \( \lambda : D \rightarrow \Sigma_R \)
Labelled partial order (LPO) \( \mathcal{L} = (D, \leq, \Sigma_R, \lambda) \)

- finite set \( D \) of locations (domain)
- partial order relation \( \leq \subseteq D \times D \)
- label alphabet \( \Sigma_R \) (labels restricted by \( \mathcal{L} \))
- labelling function \( \lambda : D \to \Sigma_R \)

Example: \( \Sigma_R = \{a_1, a_2, a_3, a_4\} \)

\[ \begin{align*}
\mathcal{L} : & \quad a_1 \\
& \quad \Downarrow \quad a_2 \\
& \quad \quad \Downarrow \quad a_3 \\
& \quad \quad \Downarrow \quad a_1 \\
& \quad \quad \Downarrow \quad a_3 \\
\end{align*} \]

degenerate linearisations

\[ \begin{align*}
& \quad \quad a_1 a_2 a_1 a_1 a_3 a_3 \\
& \quad \quad a_1 a_2 a_1 a_3 a_1 a_3 \\
& \quad \quad a_1 a_2 a_3 a_1 a_1 a_3 \\
\end{align*} \]
Labelled partial order (LPO) \( \mathcal{L} = (D, \leq, \Sigma_R, \lambda) \)

- finite set \( D \) of locations (domain)
- partial order relation \( \leq \subseteq D \times D \)
- label alphabet \( \Sigma_R \) (labels restricted by \( \mathcal{L} \))
- labelling function \( \lambda : D \to \Sigma_R \)

Example: \( \Sigma_R = \{a_1, a_2, a_3, a_4\} \)

\[ \begin{array}{ccc}
\mathcal{L} : & a_1 & \text{linearisations} \\
| & a_2 & a_1a_2a_1a_1a_3a_3 & a_1a_5a_2a_1a_1a_3a_3 \models \mathcal{L} \\
| & a_3 & a_1a_2a_1a_3a_1a_3 \\
| & a_1 & a_1a_2a_3a_1a_1a_3 \\
& a_1 & a_1a_2a_1a_1a_4a_3a_3 \not\models \mathcal{L} \\
\end{array} \]

\( w \in \Sigma^* \) (with \( \Sigma_R \subseteq \Sigma \)) is a model of \( \mathcal{L} \) if \( w|_{\Sigma_R} \) linearises \( \mathcal{L} \)
A choice LPO:

\[ \mathcal{L} : \quad a_1 \quad a_2 \]
\[ \quad a_1 \quad \quad a_2 \quad \quad a_3 \]
\[ \quad a_1 \quad \quad \quad \quad \quad \quad \quad a_3 \quad \quad \quad \quad a_3 \quad \quad \quad \quad a_1 \]

\[ \mathcal{L}_1 : \quad a_2 \quad a_1 \quad a_4 \]
\[ \quad a_2 \quad \quad a_4 \quad \quad a_1 \]

\[ \mathcal{L}_1 + \mathcal{L}_2 \quad a_3 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad a_3 \quad a_1 \]

\[ \quad a_3 \quad \quad a_1 \]

Dagstuhl, June 13, 2005
A choice LPO:

\[ \mathcal{L} : \quad \begin{array}{c}
    a_1 \\
    a_2 \\
    a_1 \\
\end{array} \quad \mathcal{L}_1 : \quad \begin{array}{c}
    a_2 \\
    a_1 \\
\end{array} \quad \mathcal{L}_2 : \quad \begin{array}{c}
    a_3 \\
    a_1 \\
\end{array} \quad \mathcal{L}_1 + \mathcal{L}_2 : \quad \begin{array}{c}
    a_1 \\
    a_3 \\
\end{array} \]

\[ \text{expand}(\mathcal{L}) : \quad \begin{array}{c}
    a_1 \\
    a_2 \\
    a_1 \\
\end{array} \quad \begin{array}{c}
    a_2 \\
    a_1 \\
\end{array} \quad \begin{array}{c}
    a_4 \\
    a_3 \\
\end{array} \quad \begin{array}{c}
    a_3 \\
    a_2 \\
\end{array} \quad \begin{array}{c}
    a_3 \\
    a_1 \\
\end{array} \]
A choice LPO: 
\[ \mathcal{L} : \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_1 \]

\[ \mathcal{L}_1 : \quad a_2 \quad a_3 \quad a_1 \]

\[ \mathcal{L}_2 : \quad a_3 \quad a_1 \]

\[ \mathcal{L}_1 + \mathcal{L}_2 \]

\[ \text{expand}(\mathcal{L}) : \]

w satisfies \( \mathcal{L} \) if \( w \models \mathcal{L}' \) for some \( \mathcal{L}' \) in \( \text{expand}(\mathcal{L}) \)
An LSC is of the form $C = (P, M)$ where $P$ and $M$ are Choice LPOs over some alphabet $\Sigma_r$.

- $P$ is called pre-chart
- $M$ is called main chart
Abstract LSC

An LSC is of the form $C = (P, M)$ where $P$ and $M$ are Choice LPOs over some alphabet $\Sigma_r$.

- $P$ is called pre-chart
- $M$ is called main chart

Notes on translation concrete $\sim$ abstract:

- instantaneous messages: no distinction between send and receive (asynchronous messages can be modelled by explicitly adding a communication channel)
- $\leq$ induced by visual order on instance axes
- choice to capture the “ALT construct”
- label alphabet corresponds to “restricted messages”
LSC $C' = (P, M)$ over $\Sigma_r \subseteq \Sigma$, execution $\alpha \in \Sigma^\omega$

- $\alpha \models C$ iff “$P$ triggers $M$”:
  
  for all $u, v \in \Sigma^*, \alpha' \in \Sigma^\omega$ with $\alpha = uv\alpha'$:
  
  if $v \models P$

<table>
<thead>
<tr>
<th>$u$</th>
<th>$v$</th>
<th>$\alpha'$</th>
</tr>
</thead>
</table>

$\models P$
LSC $C = (P, M)$ over $\Sigma_r \subseteq \Sigma$, execution $\alpha \in \Sigma^\omega$

- $\alpha \models C$ iff “$P$ triggers $M$”:
  - For all $u, v \in \Sigma^*$, $\alpha' \in \Sigma^\omega$ with $\alpha = u v \alpha'$:
  - If $v \models P$ then $w \models M$ for some prefix $w$ of $\alpha'$
LSC $C = (P, M)$ over $\Sigma_r \subseteq \Sigma$, execution $\alpha \in \Sigma^\omega$

- $\alpha \models C$ iff “$P$ triggers $M$”: 
  for all $u, v \in \Sigma^*$, $\alpha' \in \Sigma^\omega$ with $\alpha = uv\alpha'$:
  if $v \models P$ then $w \models M$ for some prefix $w$ of $\alpha'$

- $L(C') = \{ \alpha \in \Sigma^\omega \mid \alpha \models C \}$
**SPECIFICATION**

**Specification** = set of LSCs

**Semantics**: conjunction
Introduction

Specification: live sequence charts
- concrete syntax
- abstract syntax
- semantics
- definable languages

System
- structure
- implementation

Synthesis
SAFETY AND LIVENESS

LSCs can do two things:

**Safety:** Forbid some events for the next step

**Liveness:** Require some events for the future
SAFETY AND LIVENESS

LSCs can do two things:

**Safety:** Forbid some events for the next step

**Liveness:** Require some events for the future

**Example:** \( C = (P, M), \Sigma_R = \{a_1, a_2, a_3, a_4\}, \Sigma = \Sigma_R \cup \{a_5\} \)

\[ P : \quad a_1 \quad a_3 \]  

\[ \quad a_4 \]  

\[ \text{expand}(M) : \quad a_1 \quad a_1 \]  

\[ \quad a_2 \]  

\[ a_2 \quad a_1 \]  

\[ a_4 \]  

\[ a_1 \]  

\[ a_1 \]  

\[ a_2 \]  

\[ a_1 \]  

\[ a_1 \]  

\[ a_2 \]  

\[ a_3 \]  

\[ a_3 \]  

\[ a_3 \]  

\[ a_3 \]
LSCs can do two things:

**Safety**: Forbid some events for the next step

**Liveness**: Require some events for the future

**Example**: \( C = (P, M), \Sigma_R = \{a_1, a_2, a_3, a_4\}, \Sigma = \Sigma_R \cup \{a_5\} \)

\[ P : \quad a_1 \quad a_3 \quad \text{expand}(M) : \quad a_1 \quad a_2 \quad a_3 \]

\[ u = a_1a_2a_3a_1a_4a_1a_2a_5a_1a_1 \quad \models P \]
SAFETY AND LIVENESS

LSCs can do two things:

**Safety:** Forbid some events for the next step

**Liveness:** Require some events for the future

Example: $C = (P, M), \Sigma_R = \{a_1, a_2, a_3, a_4\}, \Sigma = \Sigma_R \cup \{a_5\}$

\[P : \begin{array}{c}
| & a_1 & a_3 \\
\downarrow & & \\
& a_4 & \\
\end{array}
\]

\[u = a_1 a_2 a_3 a_1 a_4 a_1 a_2 a_5 a_1 a_1\]

\|\| = P

\[\text{expand}(M) : \begin{array}{c}
\begin{array}{c}
| & a_1 & a_2 & a_3 \\
\downarrow & & & \\
& a_4 & a_3 & \\
\end{array}
\end{array}\]

\[u \text{ forbids } \{a_1, a_4\} \text{ and requires } \{a_2, a_3\} \text{ (w.r.t. } C)\]
SAFETY AND LIVENESS

\( C = (P, M) \) over \( \Sigma_R, \alpha \in \Sigma_R \)

- \( \alpha \in \Sigma^\omega \) is called \( a \)-safe (w.r.t. \( C \)) if
  \[ \alpha = uaa' \Rightarrow a \text{ is not forbidden by } u \]

- \( \alpha \in \Sigma^\omega \) is called \( a \)-live (w.r.t. \( C \)) if
  \[ \alpha = u\alpha' \text{ and } u \text{ requires } a \Rightarrow \alpha' \text{ contains a letter from } \Sigma_R \]
\( C = (P, M) \) over \( \Sigma_R \), \( a \in \Sigma_R \)

- \( \alpha \in \Sigma^\omega \) is called \( a \)-safe (w.r.t. \( C \)) if
  \[ \alpha = u a a \alpha' \Rightarrow a \text{ is not forbidden by } u \]

- \( \alpha \in \Sigma^\omega \) is called \( a \)-live (w.r.t. \( C \)) if
  \[ \alpha = u a \alpha' \text{ and } u \text{ requires } a \Rightarrow \alpha' \text{ contains a letter from } \Sigma_R \]

**Theorem** \( \alpha \models C \) iff \( \alpha \) is \( a \)-safe and \( a \)-live for each \( a \in \Sigma_R \)
Theorem For every LSC $C = (P, M)$ one can build a deterministic Büchi-automaton $A_C$ such that $L(C) = L(A_C)$. 
**Theorem** For every LSC $C = (P, M)$ one can build a deterministic Büchi-automaton $A_C$ such that $L(C) = L(A_C)$.

**Construction:**
- recall last $|P| + |M|$ symbols of $\Sigma_R$ (suffices to determine the forbidden and required events)
- disallow transitions with currently forbidden events
- go to accepting state if
  - no event is required or
  - the last letter read was from $\Sigma_R$
**Theorem** For every LSC $C = (P, M)$ one can build a deterministic Büchi-automaton $A_C$ such that $L(C') = L(A_C)$.

**Construction:**

- recall last $|P| + |M|$ symbols of $\Sigma_R$
  (suffices to determine the forbidden and required events)
- disallow transitions with currently forbidden events
- go to accepting state if
  - no event is required or
  - the last letter read was from $\Sigma_R$

**Theorem** For every LSC $C = (P, M)$ one can construct an equivalent LTL-formula.
• Introduction

• **Specification**: live sequence charts
  – concrete syntax
  – abstract syntax
  – semantics
  – definable languages

• **System**
  – structure
  – implementation

• Synthesis
**System Structure**

System structure: \( \langle \text{Proc}, \text{Sys}, (\Sigma^s_p)_{p \in \text{Proc}}, (\Sigma^r_p)_{p \in \text{Proc}} \rangle \)

- \( \text{Proc} \): set of processes
- \( \text{Sys} \subseteq \text{Proc} \): processes controlled by the system
- \( \Sigma^s_p \): set of messages process \( p \) can send (pairwise disjoint)
- \( \Sigma^r_p \): set messages process \( p \) can receive (pairwise disjoint)

**Notations**

- \( \text{Env} := \text{Proc} \setminus \text{Sys} \)
- \( \Sigma_p = \Sigma^s_p \cup \Sigma^r_p \)
- \( P \subseteq \text{Proc} \): \( \Sigma_P = \bigcup_{p \in P} \Sigma_p \) (same for \( \Sigma^r_P \) and \( \Sigma^s_P \))
- \( \Sigma = \Sigma_{\text{Proc}} \)
An implementation for a set $P \subseteq S_{sys}$ is a function

$$f_P : (\Sigma_P)^* \rightarrow 2^{\Sigma_P}$$

providing possible next messages based on the knowledge of previously observed messages.
An implementation for a set $P \subseteq Sys$ is a function

$$f_P : (\Sigma_P)^* \rightarrow 2^{\Sigma_P}$$

providing possible next messages based on the knowledge of previously observed messages.

An execution $\alpha \in \Sigma^\omega$ is an outcome of $f_P$ if each message sent by some process from $P$ complies with $f_P$:

$$\alpha = uaa' \text{ with } a \in \Sigma_P \Rightarrow a \in f_P(u|_{\Sigma_P})$$

The set of all possible outcomes is denoted by $\text{out}(f_P)$.
**System Implementation**

- An **implementation** for a set $P \subseteq Sys$ is a function

  $$f_P : (\Sigma_P)^* \rightarrow 2^{\Sigma_P}$$

  providing possible next messages based on the knowledge of previously observed messages.

- An execution $\alpha \in \Sigma^\omega$ is an **outcome** of $f_P$ if each message sent by some process from $P$ complies with $f_P$:

  $$\alpha = ua\alpha' \text{ with } a \in \Sigma_P \Rightarrow a \in f_P(u|\Sigma_P)$$

  The set of all possible outcomes is denoted by $\text{out}(f_P)$

- **Distributed implementation**: implementations $f_p$ for each $p \in Sys$

- **Centralised implementation**: implementation $f_{Sys}$ for $Sys$. 
• Introduction

• **Specification**: live sequence charts
  – concrete syntax
  – abstract syntax
  – semantics
  – definable languages

• **System**
  – structure
  – implementation

• **Synthesis**
**Synthesis Problems**

Given: system structure \( \langle \text{Proc}, \text{Sys}, (\Sigma^s_p)_{p \in \text{Proc}}, (\Sigma^r_p)_{p \in \text{Proc}} \rangle \)

specification \( S = \{C_1, \ldots, C_n\} \)

**Distributed Synthesis:** Construct a distributed implementation \((f_p)_{p \in \text{Sys}}\)
that is “correct w.r.t. the specification”.

Centralised Synthesis: Construct a centralised implementation \((f_p)_{p \in \text{Sys}}\)
that is “correct w.r.t. the specification”.

Dagstuhl, June 13, 2005
SYNTHESIS PROBLEMS

Given: system structure \( \langle Proc, Sys, (\Sigma^s_p)_{p \in Proc}, (\Sigma^r_p)_{p \in Proc} \rangle \)

specification \( S = \{C_1, \ldots, C_n\} \)

Distributed Synthesis: Construct a distributed implementation \( (f_p)_{p \in Sys} \)

that is “correct w.r.t. the specification”.

Undecidable
SYNTHESIS PROBLEMS

Given: system structure $\langle Proc, Sys, (\Sigma^s_p)_{p \in Proc}, (\Sigma^r_p)_{p \in Proc} \rangle$
specification $S = \{ C_1, \ldots, C_n \}$

Distributed Synthesis: Construct a distributed implementation $(f_p)_{p \in Sys}$
that is “correct w.r.t. the specification”.
Undecidable

Centralised Synthesis: Construct a centralised implementation $f_{Sys}$ that is
“correct w.r.t. the specification”.
Centralised setting:

- distinction of processes not necessary
- only required information: set of messages controlled by the system
  \[ \sim \Sigma_{Sys}^S \text{ and } \Sigma_{Env}^S \]
Correct Centralised Implementation

Centralised setting:
- distinction of processes not necessary
- only required information: set of messages controlled by the system 
  \( \sim \Sigma^s_{Sys} \text{ and } \Sigma^s_{Env} \)

Correct implementation (for specification \( S = \{C_1, \ldots, C_n\} \))
- standard definition: \( \forall \alpha \in \text{out}(f_{Sys}) : \alpha \models S \)
Centralised setting:

- distinction of processes not necessary
- only required information: set of messages controlled by the system

\[ \sim \Sigma_{Sys}^s \text{ and } \Sigma_{Env}^s \]

**Correct implementation** (for specification \( S = \{C_1, \ldots, C_n\} \))

- standard definition: \( \forall \alpha \in \text{out}(f_{Sys}) : \alpha \models S \)

**Problem**: environment could make unsafe moves or refuse to send required messages
**CORRECT CENTRALISED IMPLEMENTATION**

Centralised setting:

- distinction of processes not necessary
- only required information: set of messages controlled by the system
  \[ \sim \Sigma^s_{Sys} \text{ and } \Sigma^s_{Env} \]

**Correct implementation** (for specification \( S = \{C_1, \ldots, C_n\} \))

- standard definition: \( \forall \alpha \in \text{out}(f_{Sys}) : \alpha \models S \)

  **Problem**: environment could make unsafe moves or refuse to send required messages

  **Solution**: restrict to well behaved environments.

  An implementation \( f_{Sys} \) is correct w.r.t. \( S \) iff

  \[ \forall \alpha \in \text{out}(f_{Sys}) \left\{ \begin{array}{c}
  \alpha \text{ is } \Sigma^s_{Env} \text{-safe } \Rightarrow \alpha \text{ is } \Sigma^s_{Sys} \text{-safe} \\
  \alpha \text{ is } \Sigma^s_{Env} \text{-live } \Rightarrow \alpha \text{ is } \Sigma^s_{Sys} \text{-live}
\end{array} \right. \]
The problem of centralised synthesis of LSC specifications can be solved in exponential time.
Centralised Synthesis

Theorem The problem of centralised synthesis of LSC specifications can be solved in exponential time.

Construction: \( S = \{C_1, \ldots, C_n\} \), \( C_i = (P_i, M_i) \) over \( \Sigma_R^i \)

Game \( G_S \): winning strategy for player \( S_{sys} \) \( \leadsto \) correct implementation
**Centralised Synthesis**

**Theorem** The problem of centralised synthesis of LSC specifications can be solved in exponential time.

**Construction:** \( S = \{C_1, \ldots, C_n\}, C_i = (P_i, M_i) \) over \( \Sigma^i_R \)

Game \( G_S \): winning strategy for player \( Sys \) \( \rightsimeq \) correct implementation

- **Vertices** of \( G_S \): \( \{Sys, Env\} \times (\prod_{i=1}^{n} (\Sigma^i_R)^{|P_i|+|M_i|}) \times \{0, \ldots, n\}^2 \)
  - a flag indicates whose turn it is
  - for each \( i \) the last \( |P_i| + |M_i| \) messages from \( \Sigma^i_R \) are stored
  - two counters for verifying the liveness conditions for each \( C_i \)
Centralised Synthesis

**Theorem** The problem of centralised synthesis of LSC specifications can be solved in exponential time.

**Construction:** \( S = \{C_1, \ldots, C_n\}, C_i = (P_i, M_i) \) over \( \Sigma_R^i \)

Game \( G_S \): winning strategy for player \( Sys \) \( \leadsto \) correct implementation

- **Vertices** of \( G_S \): \( \{Sys, Env\} \times (\prod_{i=1}^{n}(\Sigma_R^i)^{|P_i|+|M_i|}) \times \{0, \ldots, n\}^2 \)
  - a flag indicates whose turn it is
  - for each \( i \) the last \( |P_i| + |M_i| \) messages from \( \Sigma_R^i \) are stored
  - two counters for verifying the liveness conditions for each \( C_i \)

- **Transitions**
  - arbitrary finite sequence of moves for \( Env \), single moves for \( Sys \)
  - unsafe moves lead to special state loosing for respective player
  - liveness counter is updated from \( i \) to \( i + 1 \) if the liveness condition for \( C_i \) is satisfied for the respective player (\( n + 1 \equiv 0 \))
The problem of centralised synthesis of LSC specifications can be solved in exponential time.

Construction: \( S = \{C_1, \ldots, C_n\} \), \( C_i = (P_i, M_i) \) over \( \Sigma_R^i \)

Game \( G_S \): winning strategy for player \( Sys \) \( \rightsquigarrow \) correct implementation

- **Vertices** of \( G_S \): \( \{Sys, Env\} \times (\prod_{i=1}^{n} (\Sigma_R^i)^{|P_i|+|M_i|}) \times \{0, \ldots, n\}^2 \)
  - a flag indicates whose turn it is
  - for each \( i \) the last \( |P_i| + |M_i| \) messages from \( \Sigma_R^i \) are stored
  - two counters for verifying the liveness conditions for each \( C_i \)

- **Transitions**
  - arbitrary finite sequence of moves for \( Env \), single moves for \( Sys \)
  - unsafe moves lead to special state losing for respective player
  - liveness counter is updated from \( i \) to \( i + 1 \) if the liveness condition for \( C_i \) is satisfied for the respective player \((n + 1 \equiv 0)\)

- **Winning condition**: infinitely often \( E \rightarrow \) infinitely often \( F \)
  - \( E/F = \) vertices with liveness counter for \( Env/Sys \) is 0
Merciful Strategies

In $G_S$: infinitely many visits of $E \equiv \Sigma_{Env}^s$-liveness
ininitely many visits of $F \equiv \Sigma_{Sys}^s$-liveness

Problem: What happens if $Sys$ wins by preventing $Env$ from visiting $E$?
(possible reason: faulty specification)
Merciful Strategies

In $G_S$:  infinitely many visits of $E \doteq \Sigma_{Env}^s$-liveness
          infinitely many visits of $F \doteq \Sigma_{Sys}^s$-liveness

Problem: What happens if $Sys$ wins by preventing $Env$ from visiting $E$?
          (possible reason: faulty specification)

Solution: “merciful strategy” for $Sys$

- If the play is in a vertex from where $E$ is reachable, then $Env$ must be able to reach $E$ against the strategy of $Sys$. 
Merciful Strategies

In $G_S$: infinitely many visits of $E \triangleq \sum^s_{Env}$-liveness
infinitely many visits of $F \triangleq \sum^s_{Sys}$-liveness

**Problem:** What happens if $Sys$ wins by preventing $Env$ from visiting $E$?  
(possible reason: faulty specification)

**Solution:** “merciful strategy” for $Sys$

- If the play is in a vertex from where $E$ is reachable, then $Env$ must be able to reach $E$ against the strategy of $Sys$.

winning / not merciful

---

Theorem: A merciful winning strategy can be computed by solving a parity game with $4$ colours on a slightly bigger game graph.

Dagstuhl, June 13, 2005 Synthesis from Scenarios – p.28
In $G_S$: infinitely many visits of $E \triangleq \sum_{Env}^s$-liveness
infinitely many visits of $F \triangleq \sum_{Sys}^s$-liveness

**Problem:** What happens if $Sys$ wins by preventing $Env$ from visiting $E$?
(possible reason: faulty specification)

**Solution:** “merciful strategy” for $Sys$

- If the play is in a vertex from where $E$ is reachable, then $Env$ must be able to reach $E$ against the strategy of $Sys$.

winning and merciful
**Merciful Strategies**

In $G_S$: infinitely many visits of $E \bowtie \sum_{Env}^s$-liveness
infinitely many visits of $F \bowtie \sum_{Sys}^s$-liveness

**Problem:** What happens if $Sys$ wins by preventing $Env$ from visiting $E$?
(possible reason: faulty specification)

**Solution:** “merciful strategy” for $Sys$

- If the play is in a vertex from where $E$ is reachable, then $Env$ must be able to reach $E$ against the strategy of $Sys$.

*winning and merciful*

---

**Theorem** A merciful winning strategy can be computed by solving a parity game with 4 colours on a slightly bigger game graph.

---

Dagstuhl, June 13, 2005
**CONCLUSION**

**Live sequence charts**
- intuitive / easy to write
- assumptions on environment modelled in a natural way
- not very expressive (LTL ∩ DBA)

**Synthesis**
- Distributed synthesis is undecidable
- Centralised synthesis in EXPTIME
  - game based approach
  - assumptions on environment have to be treated carefully