A Model-Checking Approach to Safe SFCs

Ralf Huuck
School of Computer Science & Engineering, University of New South Wales, Sydney, Australia

Ben Lukoschus
Department of Computer Science, University of Kiel, Germany

Nanette Bauer
BASF AG, Ludwigshafen, Germany

CESA 2003 · Lille, France · July 9–11, 2003
Overview

• Sequential Function Charts (SFCs)
• “Unsafe” and “unreachable” SFCs
• Definition of “safe” SFCs
• Algorithmic checking for “safe” SFCs:
  o Execution model for SFCs
  o Formal specification of “safe”
  o Model checking
• Summary, future work
Sequential Function Charts (SFCs)

- Graphical programming language for PLCs
- Based on Petri nets and Grafcet
- Syntax and informal semantics defined in IEC 61131-3
- Concepts:
  - Actions (embedding of other PLC languages)
  - Parallelism
  - Hierarchy
Sequential Function Charts: Components

- Step
- Transition
- Guard

Action Block
- Action Name
- Action Qualifier

Actions:
- S: action1
- N: action2
- R: action1
Sequential Function Charts: Transition Types

\[(\{s_1\}, g, \{s_2\})\]  \[(\{s_1\}, g, \{s_2, s_3, s_4\})\]  \[(\{s_1, s_2, s_3\}, g, \{s_4\})\]

\[(\{s_1\}, g_1, \{s_2\})\]  \[(\{s_1\}, g_2, \{s_3\})\]  \[(\{s_1\}, g_3, \{s_4\})\]

\[(\{s_1, s_2, s_3\}, g, \{s_4, s_5\})\]
“Unsafe” and “unreachable” SFCs
“Unsafe” and “unreachable” SFCs
“Unsafe” and “unreachable” SFCs
“Unsafe” and “unreachable” SFCs
“Unsafe” and “unreachable” SFCs
“Unsafe” and “unreachable” SFCs

IEC 61131-3 calls these SFCs “unsafe”/“unreachable” . . .
“Unsafe” and “unreachable” SFCs

IEC 61131-3 calls these SFCs “unsafe”/“unreachable” . . .

But construction still possible in many programming environments!
“Safe” SFCs

“Safe” = absence of “unsafe” and “unreachable”

Informal (graphical):

- no jumps between parallel branches
- no jumps out of parallel branches
- every opening parallel branch is closed correctly
“Safe” = absence of “unsafe” and “unreachable”

**Informal (graphical):**
- no jumps between parallel branches
- no jumps out of parallel branches
- every opening parallel branch is closed correctly

**Formal (Petri net execution model):**
- In each execution there is at most one token in each step.
- For every closing parallel transition there is an execution that uses this transition.
Check for “Safe” SFCs = Reachability Problem

“Safe” as reachability:

1. No state can be reached in which more than one token can enter a step.

2. For every closing parallel transition a state is reachable in which this transition can be used.
Check for “Safe” SFCs $\equiv$ Reachability Problem

“Safe” as reachability:

1. No state can be reached in which more than one token can enter a step.
2. For every closing parallel transition a state is reachable in which this transition can be used.

$\Rightarrow$ Checking by model checking (Cadence SMV):

- Abstraction of SFCs
- Modelling of SFC executions in CaSMV
- Definition of “safe” in CaSMV
CaSMV model for SFCs: Variables

Abstraction of the token flow:

- no program variables
- no actions
- guards are replaced by unconstrained Boolean variables
- one Boolean variable $s_i$ for each step
  \[(s_i = true: \text{step } s_i \text{ has a token})\]

State changes of the variables:

- discrete transition system
- relation “next” between old and new values
CaSMV model for SFCs: Transitions

Activity of step $s_i$ in the next cycle:

$$\text{next}(s_i) \equiv s_i_{-\text{will\_be\_entered}} \lor (s_i \land s_i_{-\text{will\_not\_be\_left}})$$
CaSMV model for SFCs: Transitions

Activity of step $s_i$ in the next cycle:

$$\text{next}(s_i) \equiv s_i\text{-will\_be\_entered} \lor (s_i \land s_i\text{-will\_not\_be\_left})$$

Step $s_i$ will be entered in the next cycle:

$$s_i\text{-will\_be\_entered} \equiv$$

$$(\exists t = (S, g, T) \in Tr : s_i \in T \land \text{next}(g) \land \bigwedge_{s_j \in S} s_j$$

$\land \forall t' = (S, g', T') \in Tr \setminus \{t\} : \text{next}(g') \Rightarrow \bigwedge_{s_k \in T' \setminus T} \neg \text{next}(s_k))$$
CaSMV model for SFCs: Transitions

Activity of step $s_i$ in the next cycle:

$$\text{next}(s_i) \equiv s_i\_will\_be\_entered \lor (s_i \land s_i\_will\_not\_be\_left)$$

**Step $s_i$ will be entered in the next cycle:**

$$s_i\_will\_be\_entered \equiv (\exists t = (S, g, T) \in Tr : s_i \in T \land \text{next}(g) \land \bigwedge_{s_j \in S} s_j \land \forall t' = (S, g', T') \in Tr \setminus \{t\} : \text{next}(g') \Rightarrow \bigwedge_{s_k \in T' \setminus T} \neg \text{next}(s_k))$$

**Step $s_i$ will not be left in the next cycle:**

$$s_i\_will\_not\_be\_left \equiv \neg \exists (S, g, T) \in Tr : s_i \in S \land \text{next}(g) \land \bigwedge_{s_j \in S} s_j$$
Requirement 1: At most one token in a step

More than one token in a step:

\[
\text{next}(\text{token\_overflow}) \equiv \bigvee_{s_i \in St} (s_i \wedge \bigvee_{(S,g,T)\in Tr} \left( s_i \in T \wedge \text{next}(g) \wedge \bigwedge_{s_j \in S} s_j \right))
\]

\[
\bigvee (\bigvee_{(S_1,g_1,T_1)\in Tr} (s_i \in T_1 \cap T_2 \wedge \text{next}(g1) \wedge \text{next}(g2) \wedge \bigwedge_{s_j \in S_1 \cup S_2} s_j))
\]

CaSMV specification: SPEC AG !token\_overflow
Requirement 2: Closing parallel transitions

We show for each transition $(S, g, T) \in Tr$ with $|S| > 1$:
There exists an execution in which all $s_i \in S$ are active.

**CaSMV specification:** SPEC EF $\&_{s_i \in S} s_i$
Implemented as a tool:

- Input: SFC in IEC 61131-3 or Siemens syntax
- Output: CaSMV code and CTL specification

Output of CaSMV:

- OK – SFC is “safe”
- Error trace (helpful to locate the problem)

⇒ requires only minimal interaction by the user
Summary and Future Work

Summary

- The problem of “unsafe” and “unreachable” SFCs
- Algorithmic approach to check for “safe” SFCs:
  - abstract CaSMV model
  - tool-supported automatic verification

Future work

- Embed tool into PLC programming environments
- Combine with other automated verification approaches, e.g., static analysis