Process synthesis from multiple interaction specifications

Johannes Reich

2011-10-04
Table of Contents

1. Introduction

2. Nondeterministic Finite I/O Automata

3. Analysis of System Interactions

4. Synthesis of Interacting Systems

5. Discussion
Example of Complex Interacting Systems: A Ring of Resource Manager Processes $\mathcal{R}_i$
Example of Complex Interacting Systems: A Ring of Resource Manager Processes $\mathcal{R}_i$
A system $\mathcal{U}$ is interacting with a resource manager system $\mathcal{R}$ to get exclusive access to a resource. $\mathcal{U}$ can access the resource only while it is in its Crit(-ical) state.
A ring of resource managers $R_i$, $i = 1 \ldots n$ coordinates the exclusive access to a resource $U$ by exchange of a token.
Def.: A *nondeterministic finite I/O automaton* (NFIOA) is defined by a tuple $A = (Q, I, O, q_0, Acc, \Delta)$.

- $Q$ is the non-empty finite set of state values,
- $I$ and $O$ are the finite input and output alphabets where at least one of both is non empty,
- $q_0$ is the initial state value,
- $Acc$ is the acceptance component and
- $\Delta \subseteq Q \times Q \times I^c \times O^c$ is the transition relation.

For finite input sequences: $Acc \subseteq Q$.
For infinite input sequences: the $Acc$ depends on the computational model (Büchi, Rabin, Streett, Muller, etc.). The Muller acceptance condition is to accept all runs in which the set of infinitely often occurring state values is an element of $Acc$, that is $Acc \subseteq 2^Q$. 

Johannes Reich ()
Process synthesis from multiple interaction sp
2011-10-04 7 / 29
Interactions: Synchronizing NFIOAs

**Def.:** The *unsynchronized product* of a set of $n$ NFIOAs $A_k$ is defined by NFIOA $B = (Q, I, O, \bar{q}_0, \text{Acc}, \Delta)$, with $Q_B = \times Q_k$, $I_B = \times I_k$, $O_B = \times O_k$, $\bar{q}_{0B} = (q_{01}, \ldots, q_{0n})$, $\text{Acc}_B = \bigwedge \text{Acc}_k$ is the common acceptance component, $\Delta_B := \{ (\bar{p}, \bar{q}, \bar{i}, \bar{o}) | \bar{p} \text{ is a reachable state and } A_k \text{ provides a transition } (p_k, q_k, i_k, o_k) \text{ with } \bar{q} = p \left[ \begin{array}{c} q_k \\ p_k \end{array} \right] \text{ and } \bar{i} = \epsilon[i_k] \text{ and } \bar{o} = \epsilon[o_k] \}$. I also write $B = \bigotimes_{i=1}^{n} A_i$.

"Synchronization" then means to eliminate some transitions from the transition relation $\Delta_P$ and thereby to eliminate the symmetry of the unsynchronized product

**Def.:** Be $B = \bigotimes_{k=1}^{n} A_k$ an unsynchronized product automaton. An automaton $B^*$ with all components from $B$ except a reduced transition set $\Delta_{B^*} = \Delta_B \setminus \Delta^e$ with nonempty $\Delta^e \subset \Delta_B$ is a *synchronized* version of $B$. I also write $B^* = B \setminus \Delta^e$.
The Outer Synchronized Product: Channel Mediated System Interaction

Def.: An outer product of $k$ NFIOAs $\mathcal{P} = \bigotimes_{k=1}^{n} \mathcal{A}_k \setminus \Delta^C$ is called well formed if for every channel mediated transition attributable to a sender which sends a character different from $\epsilon$ there exists an induced transition of the receiver.

Def.: A well formed outer product is called consistent if for each reachable product state value the product acceptance conditions still holds.
The Outer Synchronized Product: The Protocol of Mutual Exclusion

Unsynchronized product

Channel mediated synchronized product
Def.: A protocol is the outer product of pairwise different NFIOAs, representing system projections, whose interactions happen completely through a set of channels.

The transition relation of the protocol has to be determined inductively.
The Inner Synchronized Product: Example "man in the middle"
Transitions can be eliminated, if

1. The projection of the synchronized product automaton must result in the original NFIOAs.
2. The common acceptance condition must still be fulfilled,
3. The elimination must provide the coordination semantics of all related interactions.
Unsynchronized Product Automaton
Synchronization, Step 1

The diagram illustrates the process of synchronization in interacting systems. It shows the flow of states and transitions labeled with events such as `req/ε`, `token/trigger`, `timeout/ε`, `ε/token`, `ε/cf-fin`, `fin/ε`, and `utl/ε`. The states are labeled with `remn`, `try`, `crit`, and `exit`, indicating different stages of the process. The transitions are marked with conditions and events that govern the progression through the states.
Synchronization, Step 1

The diagram illustrates the process synthesis from multiple interaction specifications. Nodes and edges are labeled with transitions and conditions.

- **remn**, abst
- **try**, abst
- **crit**, abst
- **exit**, abst
- **remn**, interim
- **try**, interim
- **crit**, interim
- **exit**, interim
- **remn**, avlb
- **try**, avlb
- **crit**, avlb
- **exit**, avlb

Transitions include:
- \( \epsilon / \text{token} \)
- \( \text{req} / \epsilon \)
- \( \text{timeout} / \epsilon \)
- \( \text{utl} / \epsilon \)
- \( \text{fin} / \epsilon \)
- \( \text{cf}_f \text{fin} / \epsilon \)

Conditions and actions are also depicted in the diagram.
Synchronization, Step 2

Diagram showing the transitions and states involving
- \( \text{remn}_c \), \( \text{abst} \)
- \( \text{remn}_c \), \( \text{interm} \)
- \( \text{remn}_c \), \( \text{avlb} \)
- \( \text{try}_c \), \( \text{abst} \)
- \( \text{try}_c \), \( \text{interm} \)
- \( \text{try}_c \), \( \text{avlb} \)
- \( \text{crit}_c \), \( \text{abst} \)
- \( \text{crit}_c \), \( \text{interm} \)
- \( \text{crit}_c \), \( \text{avlb} \)

States and transitions include:
- \( i \neq 1 \)
- \( \text{req/\( \epsilon \)} \), \( \text{token/trigger} \)
- \( \text{req/\( \epsilon \)} \)
- \( \text{timeout/\( \epsilon \)} \)
- \( \text{utl/\( \epsilon \)} \)
- \( \text{\( \epsilon/\text{cf-fin} \)} \)
- \( \text{\( \epsilon/\text{token} \)} \)
- \( \text{\( \text{fin/\( \epsilon \)} \)} \)
- \( \text{\( \text{timeout/\( \epsilon \)} \)} \)

The diagram illustrates the process synthesis from multiple interaction specifications.
Synchronization, Step 2
Synchronization and Determination

\begin{align*}
\text{Choose transition} & \quad \text{Eliminate state} \\
\end{align*}
Determination, Step 1

- \text{remn}_c \text{, abst}
- \text{try}_c \text{, abst}
- \text{exit}_c \text{, abst}
- \text{remn}_c \text{, intern}
- \text{try}_c \text{, intern}
- \text{crit}_c \text{, intern}
- \text{exit}_c \text{, intern}
- \text{remn}_c \text{, avlb}
- \text{try}_c \text{, avlb}
- \text{crit}_c \text{, avlb}
- \text{exit}_c \text{, avlb}

Transition labels:
- \epsilon/\text{token}
- \text{req}/\epsilon \text{, token/trigger}
- \epsilon/\text{cf}\_\text{req}
- \text{token/trigger}
- \text{utl}/\epsilon
- \text{fin}/\epsilon
- \epsilon/\text{cf}\_\text{fin}
- \text{timeout}/\epsilon
Determination, Step 1

[Diagram of interacting system states and transitions]

- $\text{remn}_c, \text{abst}$
  - $i \neq 1$
  - $\text{try}_c, \text{abst}$
  - $\text{crit}_c, \text{interm}$
  - $\text{exit}_c, \text{abst}$

- $\text{remn}_c, \text{avlb}$
  - $i = 1$
  - $\text{try}_c, \text{interm}$
  - $\text{crit}_c, \text{avlb}$

Transitions:
- $\text{timeout/\epsilon}$
- $\text{token/trigger}$
- $\text{req/\epsilon}$
- $\text{fin/\epsilon}$
- $\text{util/\epsilon}$

Note: States and transitions represent abstract states and interactions within a system.
Determination, Step 2

- \( \text{remn}_{c}, \text{abst} \)
  - \( \text{try}_{c}, \text{abst} \)
    - \( \text{crit}_{c}, \text{interm} \)
      - \( \text{exit}_{c}, \text{abst} \)
    - \( \text{timeout/} \epsilon \)
      - \( \text{token/trigger, cf}\_\text{req} \)
      - \( \epsilon/\text{cf}\_\text{fin} \)
    - \( \text{req/} \epsilon \)
      - \( \text{timeout/} \epsilon \)
      - \( \text{token/trigger, cf}\_\text{fin} \)
  - \( \text{timeout/} \text{token} \)
    - \( \text{token/trigger} \)
  - \( \text{req/} \text{cf}\_\text{req} \)
  - \( \text{fin/} \text{cf}\_\text{fin} \)
- \( \text{remn}_{c}, \text{avlb} \)
  - \( \text{crit}_{c}, \text{avlb} \)
  - \( \text{req/} \epsilon \)
  - \( \text{util/} \epsilon \)
Determination, Step 3

Diagram of interacting systems with nodes and transitions labeled with conditions and events.
Determination, Step 3
Processes

**Def.:** Be $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ a set of $n \geq 2$ consistent protocols with a role $A_{k,i}$ in each protocol $\mathcal{P}_i$ which specifies the behavior of a projection of a single system (symbolized by the common index $k$) such that for all $i \neq j$, $A_{k,i} \not\subseteq A_{k,j}$ . A fully synchronized product of these roles $\mathcal{R} = \bigotimes_{i=1}^{n} A_{k,i} \setminus \Delta^e$ is called a **process**.

Thereby a process is again an NFIOA, but - based on its construction - with the important property, that it coordinates at least two different roles. In the special case that a DFIOA results, possible after additional $\epsilon$-merge, a process specifies a finite system.
Summary

- Computer science is a subspecialty of system theory.
- Process theory is more complex than I have thought.
- Network-like interactions are not deterministic, but are nondeterministic. The actions are not determined by the interactions!
- Nondeterminism is not introduced to abstract from details of implementation (C. Hoare, 1985) or only for convenience (N. Lynch, 1996), but nondeterminism seems to be the precondition for an efficient description of interactions between many different systems, where all interactions happen on the same level of abstraction.
- ”Outer Synchronization” represents causal relation between the input and output of different systems: a sender and a receiver
- ”Inner Synchronization” represents causal relation between the input and output of the same system.
Possible Consequences

- What is “loose coupling”? Abstraction from algorithm (Doug Kaye, 2003) versus sensible interaction of complex systems with few common states (Robert B. Glassmann, 1973)

- Imperative programming languages are formal calculi to describe systems and their interactions. “Soundness”: every correct formal description describes an interacting system. “Completeness”: every system and interaction can be described.

- Protocols are genuine entities of their own. Their implementation should be much better supported by programming languages (“implements protocol”-annotations, language support for document declarations like data type declarations, ...)

- We can start to think about a “process oriented architecture”
Thank You!

Any questions?

Johannes.Reich@sophoscape.de