Compositionality and Asynchrony

Johannes Åman Pohjola
CSIRO’s Data61 UNSW
Term 2 2021
Where we are at

Last lecture, we looked at at proof methods for termination (convergence and deadlock freedom) in sequential, shared-variable concurrent, and message-passing concurrent settings.
In this lecture, we will conclude our examination of proof methods for concurrency by examining compositional techniques. We will also discuss how to prove properties of asynchronous systems.
Analysis of AFR and L&G

Both are only applicable to *closed* systems.
Analysis of AFR and L&G

- Both are only applicable to *closed* systems.
- That means we always have to reason about the system as a whole, even including users modeled as processes.
Analysis of AFR and L&G

- Both are only applicable to *closed* systems.
- That means we always have to reason about the system as a whole, even including users modeled as processes.
- Using these methods, one cannot reason *compositionally*. Typically, non-compositional proof methods don’t scale and preclude re-use.
A *composition* proof method is a method by which the specification of a system can be inferred from the specifications of its constituents, without additional information about their internal structure.
Quotes on Compositionality

**de Roever et al.**

A *compositional* proof method is a method by which the specification of a system can be inferred from the specifications of its constituents, without additional information about their internal structure.

**F. B. Schneider, 1994**

Compositionality is a red herring.
Lamport (1997) – “Composition: a way to make proofs harder”

Systems are complicated. We master their complexity by building them from simpler components. This suggests that to master the complexity of reasoning about systems, we should prove properties of the separate components and then combine those properties to deduce properties of the entire system. In concurrent systems, the obvious choice of component is the process. So, compositional reasoning has come to mean deducing properties of a system from properties of its processes. I have long felt that this whole approach is rather silly. You don’t design a mutual exclusion algorithm by first designing the individual processes and then hoping that putting them together guarantees mutual exclusion.
Compositional Assertions

Key Idea

Handle communication with a special logical variable $h$, containing the history of all communication, i.e. a sequence of pairs of channels and messages $\langle C, x \rangle$. 
Compositional Interaction Theory

**Key Idea**

Handle communication with a special logical variable $h$, containing the *history* of all communication, i.e. a sequence of pairs of channels and messages $\langle C, x \rangle$.

A local assertion network $Q$ is *compositionally-inductive* for a sequential synchronous transition diagram $P = (L, T, s, t)$, written $P \vdash Q$, if

- $\models Q_\ell \land b \implies Q_{\ell'} \circ f$ for each $\ell \xrightarrow{b;f} \ell' \in T$.
- $\models Q_\ell \land b \implies Q_{\ell'} \circ (f \circ [h \leftarrow h \cdot \langle C, e \rangle])$, for each $\ell \xrightarrow{b;C \leftarrow e;f} \ell' \in T$.
- $\models Q_\ell \land b \implies \forall x (Q_{\ell'} \circ (f \circ [h \leftarrow h \cdot \langle C, x \rangle]))$, for each $\ell \xrightarrow{b;C \Rightarrow x;f} \ell' \in T$. 
Partial Correctness

Let $Q$ be an assertion network for a process $P$ and $Q_s$ and $Q_t$ be the assertions at the start and end states. We have the **Basic diagram rule**:

$$
\frac{P \vdash Q}{\{Q_s\} \ P \ {Q_t}}
$$
Partial Correctness

Let $Q$ be an assertion network for a process $P$ and $Q_s$ and $Q_t$ be the assertions at the start and end states. We have the Basic diagram rule:

$$P \vdash Q$$

$$\{Q_s\} P \{Q_t\}$$

We assume the history is empty initially with the Initialization rule:

$$\{\phi \land h = \varepsilon\} P \{\psi\}$$

$$\{\phi\} P \{\psi\}$$
Partial Correctness

Let $Q$ be an assertion network for a process $P$ and $Q_s$ and $Q_t$ be the assertions at the start and end states. We have the **Basic diagram rule**:

$$P \vdash Q$$

$$\{Q_s\} P \{Q_t\}$$

We assume the history is empty initially with the **Initialization rule**:

$$\{\phi \land h = \varepsilon\} P \{\psi\}$$

$$\{\phi\} P \{\psi\}$$

..the **Consequence rule** allows pre/post-conditions to be strengthened/weakened:

$$\phi \Rightarrow \phi'$$

$$\{\phi'\} P \{\psi'\}$$

$$\psi' \Rightarrow \psi$$

$$\{\phi\} P \{\psi\}$$
Parallel composition rule

Provided $\psi_i$ only makes assertions about local variables in $P_i$ and those parts of the history that involve channels read from/written to by $P_i$ we get this *compositional parallel composition rule*:

$$\left\{ \phi_1 \right\} P_1 \left\{ \psi_1 \right\} \left\{ \phi_2 \right\} P_2 \left\{ \psi_2 \right\} \implies \left\{ \phi_1 \land \phi_2 \right\} P_1 \parallel P_2 \left\{ \psi_1 \land \psi_2 \right\}$$

Observe that we don’t need to prove anything like interference freedom or generate a proof obligation about each possible communication.
Parallel composition rule

Provided $\psi_i$ only makes assertions about local variables in $P_i$ and those parts of the history that involve channels read from/written to by $P_i$ we get this *compositional parallel composition rule*:

$$\begin{array}{c}
\{\phi_1\} P_1 \{\psi_1\} \quad \{\phi_2\} P_2 \{\psi_2\} \\
\{\phi_1 \land \phi_2\} P_1 \parallel P_2 \{\psi_1 \land \psi_2\}
\end{array}$$

Observe that we don’t need to prove anything like interference freedom or generate a proof obligation about each possible communication.

**Notation**

Define $h|_H$ as the history $h$ filtered to only contain those pairs $\langle C, x\rangle$ where $C \in H$. 
Example 2 once more

\[ s_1 \xrightarrow{C \leftarrow 1} l_1 \xrightarrow{C \leftarrow 2} t_1 \]

\[ s_2 \xrightarrow{C \Rightarrow x} l_2 \xrightarrow{C \Rightarrow x} t_2 \]
Example 2 once more

\[
\begin{align*}
\text{Compositionality} & \\
\text{Asynchrony} & \\
\text{Example 2 once more} & \\
\text{Diagram of Example 2} & \\
\end{align*}
\]
Example 2 once more cont’d

For the two output transitions we need to show

\[
\models h|_{\{C\}} = \varepsilon \implies h|_{\{C\}} = \langle C, 1 \rangle \circ [h \leftarrow h \cdot \langle C, 1 \rangle]
\]

\[
\models h|_{\{C\}} = \langle C, 1 \rangle \implies h|_{\{C\}} = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \circ [h \leftarrow h \cdot \langle C, 2 \rangle]
\]

which is obvious; and for the two input transitions

\[
\models h|_{\{C\}} = \varepsilon \implies \forall x \left( h|_{\{C\}} = \langle C, x \rangle \circ [h \leftarrow h \cdot \langle C, x \rangle] \right)
\]

\[
\models h|_{\{C\}} = \langle C, x \rangle \implies \forall x \exists y \left( h|_{\{C\}} = \langle C, y \rangle \cdot \langle C, x \rangle \circ [h \leftarrow h \cdot \langle C, x \rangle] \right)
\]

which also works out nicely.
Example 2 once more cont’d

Using the **Basic diagram rule** we may now deduce

\[
\begin{align*}
\{ h\mid \{ C \} = \varepsilon \} & \quad C \leftarrow 1; C \leftarrow 2 \quad \{ h\mid \{ C \} = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \} \\
\{ h\mid \{ C \} = \varepsilon \} & \quad C \Rightarrow x; C \Rightarrow x \quad \exists y. \quad h\mid \{ C \} = \langle C, y \rangle \cdot \langle C, x \rangle
\end{align*}
\]
Example 2 once more cont’d

Using the **Basic diagram rule** we may now deduce

\[
\{ h|_C = \varepsilon \} \quad C \leftarrow 1; \ C \leftarrow 2 \ \{ h|_C = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \}
\]

\[
\{ h|_C = \varepsilon \} \quad C \Rightarrow x; \ C \Rightarrow x \ \{ \exists y. \ h|_C = \langle C, y \rangle \cdot \langle C, x \rangle \}
\]

before applying the **parallel composition rule** to obtain

\[
\{ h|_C = \varepsilon \} \quad P \ \{ h|_C = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \land \exists y. \ h|_C = \langle C, y \rangle \cdot \langle C, x \rangle \}
\]
Example 2 once more cont’d

Using the Basic diagram rule we may now deduce

\[
\{ h|_C = \varepsilon \} \quad C \leftarrow 1; C \leftarrow 2 \quad \{ h|_C = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \} \\
\{ h|_C = \varepsilon \} \quad C \Rightarrow x; C \Rightarrow x \quad \exists y. \ h|_C = \langle C, y \rangle \cdot \langle C, x \rangle
\]

before applying the parallel composition rule to obtain

\[
\{ h|_C = \varepsilon \} \quad P \quad \{ h|_C = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \land \exists y. \ h|_C = \langle C, y \rangle \cdot \langle C, x \rangle \}
\]

which implies (via the rule of consequence):

\[
\{ h = \varepsilon \} \quad P \quad \{ x = 2 \}
\]
Example 2 once more cont’d

Using the **Basic diagram rule** we may now deduce

\[
\{ h|_C = \varepsilon \} \quad C \leftarrow 1; \quad C \leftarrow 2 \quad \{ h|_C = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \}
\]

\[
\{ h|_C = \varepsilon \} \quad C \Rightarrow x; \quad C \Rightarrow x \quad \{ \exists y. \quad h|_C = \langle C, y \rangle \cdot \langle C, x \rangle \}
\]

before applying the **parallel composition rule** to obtain

\[
\{ h|_C = \varepsilon \} \quad P \quad \{ h|_C = \langle C, 1 \rangle \cdot \langle C, 2 \rangle \wedge \exists y. \quad h|_C = \langle C, y \rangle \cdot \langle C, x \rangle \}
\]

which implies (via the rule of consequence):

\[
\{ h = \varepsilon \} \quad P \quad \{ x = 2 \}
\]

and finally the **initialisation rule** takes us to

\[
\{ \text{TRUE} \} \quad P \quad \{ x = 2 \}
\]
Asynchrony

Consider a process $P$ that sends a file $a$ on the channel $C$ to the process $Q$, which saves it to $b$.

- $a[i] \neq \text{EOF}; C \leftarrow a[i]; i \leftarrow i + 1$
- $a[i] = \text{EOF}; C \leftarrow \text{EOF}; i \leftarrow i + 1$
- $C \Rightarrow b[j]; j \leftarrow j + 1$
- $j > 0 \land b[j - 1] = \text{EOF}$

How do we verify this if $C$ is asynchronous?

\[ a[i] \neq \text{EOF}; C \leftarrow a[i]; i \leftarrow i + 1 \]
\[ a[i] = \text{EOF}; C \leftarrow \text{EOF}; i \leftarrow i + 1 \]
\[ C \Rightarrow b[j]; j \leftarrow j + 1 \]
\[ j > 0 \land b[j - 1] = \text{EOF} \]
Asynchrony

Consider a process \( P \) that sends a file \( a \) on the channel \( C \) to the process \( Q \), which saves it to \( b \).

\[
\begin{align*}
  a[i] &\neq \text{EOF}; C \leftarrow a[i]; i \leftarrow i + 1 \\
  C &\Rightarrow b[j]; j \leftarrow j + 1
\end{align*}
\]

How do we verify this if \( C \) is asynchronous?

---

Compositionality
Convert to Synchronous

\[ a[i] \neq \text{EOF}; A \leftarrow a[i]; i \leftarrow i + 1 \]

\[ B \Rightarrow b[j]; j \leftarrow j + 1 \]

\[ a[i] = \text{EOF}; A \leftarrow \text{EOF}; i \leftarrow i + 1 \]

\[ j > 0 \land b[j - 1] = \text{EOF} \]
Convert to Synchronous

\( a[i] \neq \text{EOF}; A \leftarrow a[i]; i \leftarrow i + 1 \)

\( B \Rightarrow b[j]; j \leftarrow j + 1 \)

\( a[i] = \text{EOF}; A \leftarrow \text{EOF}; i \leftarrow i + 1 \)

\( j > 0 \land b[j - 1] = \text{EOF} \)

\( A \Rightarrow x; q \leftarrow q \cdot x \)

\( q \neq \varepsilon; B \leftarrow \text{head}(q); q \leftarrow \text{tail}(q) \)
Compositionally

By adding an extra process with two synchronous channels to explicitly manage the queue, we convert this asynchronous system to a synchronous one. We can now use AFR, Levin and Gries or the compositional method.
By adding an extra process with two synchronous channels to explicitly manage the queue, we convert this asynchronous system to a synchronous one. We can now use AFR, Levin and Gries or the compositional method. Using the compositional method, we have the desired postcondition:

$$\exists i. \ a[i] = \text{EOF} \land a[0 \ldots i] = b[0 \ldots i]$$
Compositionally

By adding an extra process with two synchronous channels to explicitly manage the queue, we convert this asynchronous system to a synchronous one. We can now use AFR, Levin and Gries or the compositional method. Using the compositional method, we have the desired postcondition:

$$\exists i. \; a[i] = \text{EOF} \land a[0 \ldots i] = b[0 \ldots i]$$

And the following assertion network:

- $$Q(p_s) \equiv \hat{h}_{\{A\}} = a[0 \ldots i] \land \text{EOF} \notin a[0 \ldots i]$$
- $$Q(p_t) \equiv \hat{h}_{\{A\}} = a[0 \ldots i] \land \text{EOF} \notin a[0 \ldots i - 1] \land a[i - 1] = \text{EOF}$$
- $$Q(q_s) \equiv \hat{h}_{\{B\}} = b[0 \ldots j]$$
- $$Q(q_t) \equiv \hat{h}_{\{B\}} = b[0 \ldots j] \land b[j - 1] = \text{EOF}$$
- $$Q(C) \equiv \hat{h}_{\{A\}} = \hat{h}_{\{B\}} \cdot q$$

Proof obligations will be informally described.
What Now?

If time allows, we’ll take a brief detour into the world of *process algebra*, a high level formalism for describing concurrent systems. Either way, we’ll then discuss distributed algorithms.