Compositional Reasoning about Concurrent Objects and Futures

Crystal Chang Din, Johan Dovland, Olaf Owe

Precise Modeling and Analysis group (PMA), University of Oslo, Norway

Outline

1. Introduction

2. Syntax and semantics

3. Reasoning system

4. Summary
We are here now...

1. Introduction

2. Syntax and semantics

3. Reasoning system

4. Summary
Introduction

- Concurrent and distributed systems
  - Difficulties of reasoning about concurrent and distributed systems
  - Necessity of compositional reasoning
  - Object orientation vs. distributed systems
  - Weakness of the existing paradigms for concurrent and distributed systems
Introduction

- Concurrent and distributed systems
- Difficulties of reasoning about concurrent and distributed systems
  - Necessity of compositional reasoning
  - Object orientation vs. distributed systems
  - Weakness of the existing paradigms for concurrent and distributed systems
Introduction

- Concurrent and distributed systems
- Difficulties of reasoning about concurrent and distributed systems
- Necessity of compositional reasoning
  - Object orientation vs. distributed systems
  - Weakness of the existing paradigms for concurrent and distributed systems
Introduction

- Concurrent and distributed systems
- Difficulties of reasoning about concurrent and distributed systems
- Necessity of compositional reasoning
- Object orientation vs. distributed systems
- Weakness of the existing paradigms for concurrent and distributed systems
Introduction

- Concurrent and distributed systems
- Difficulties of reasoning about concurrent and distributed systems
- Necessity of compositional reasoning
- Object orientation vs. distributed systems
- Weakness of the existing paradigms for concurrent and distributed systems
What do we consider?

- Concurrent objects
- Asynchronous method calls
- Shared futures
Abstract Behavioral Specification Language (ABS)

- We use the syntax of ABS
- What we provide in this work:
  - Compositionality
  - Compositional reasoning by means of class invariants over histories
What is a future?

An example:
Communication by asynchronous method calls and futures

Definition (Communication History/Trace)
A sequence of observable communication events between components.

\[ \langle o \rightarrow o', u, m, \bar{e} \rangle \]

\[ \langle o'' \leftrightarrow, u, e \rangle \]

\[ Ev_\rightarrow \] denotes \( \langle o \rightarrow o', u, m, \bar{e} \rangle \)

\[ Ev_\leftarrow \] denotes \( \langle o \leftarrow o', u, m, \bar{e} \rangle \)

\[ Ev_\leftrightarrow \] denotes \( \langle o \leftrightarrow o', u, e \rangle \)

\[ Ev_\leftarrow \] denotes \( \langle o \leftrightarrow o', u, e \rangle \)

\[ Ev_\leftrightarrow \] denotes \( \langle o \leftarrow u, e \rangle \)

\[ Ev_\rightarrow \] denotes \( \langle o \rightarrow u, m, \bar{e} \rangle \)

History well-formedness?
The order of events: \( Ev_\rightarrow \ Ev_\rightarrow \ Ev_\leftarrow \ [Ev_\leftarrow]^* \)
Communication by asynchronous method calls and futures

Definition (Communication History/Trace)
A sequence of observable communication events between components.

```
Ev→ denotes ⟨o → o', u, m, e⟩
Ev→ denotes ⟨o → o', u, m, e⟩
Ev← denotes ⟨← o', u, e⟩
Ev← denotes ⟨o ←, u, e⟩
```

History well-formedness?
The order of events: Ev→ Ev→ Ev← [Ev←]*
We are here now...

1. Introduction

2. Syntax and semantics

3. Reasoning system

4. Summary
ABS syntax and semantics for futures (kernel language)

- $fr := o'!m(e^*)$
  The caller generates a fresh future identity and continues without waiting.

- $v := fr?$
  The receiver object blocks while waiting for the future to be resolved.

- `await v := fr?`
  Since there is at most one process executing on an object at a time. The receiver object release the current process while waiting for the future to be resolved, and at the same time it is free to execute other suspended processes. (For simplicity this is omitted in the following presentation.)
ABS syntax and semantics for futures (kernel language)

- $fr := o!'m(e^*)$
  The caller generates a fresh future identity and continues without waiting.

- $v := fr?$
  The receiver object blocks while waiting for the future to be resolved.

- `await v := fr?`
  Since there is at most one process executing on an object at a time. The receiver object release the current process while waiting for the future to be resolved, and at the same time it is free to execute other suspended processes. (For simplicity this is omitted in the following presentation.)
ABS syntax and semantics for futures (kernel language)

- $fr := o'!m(e^*)$
  The caller generates a fresh future identity and continues without waiting.

- $v := fr?$
  The receiver object blocks while waiting for the future to be resolved.

- `await v := fr?`
  Since there is at most one process executing on an object at a time. The receiver object release the current process while waiting for the future to be resolved, and at the same time it is free to execute other suspended processes. (For simplicity this is omitted in the following presentation.)
Operational semantics

- **State transition:** \( s_1 \rightarrow s_2 \)
- Sending asynchronous method call from \( o \) to \( o' \):

\[
\begin{align*}
\mathcal{H} & \rightarrow \mathcal{H}_0 \\
o & \rightarrow \text{Prg} : (fr := o'!m(\bar{e}); \text{rest}), \\
\text{Cnt} & : (n), \\
\text{Att} & : (\ldots), \\
\text{Lvar} & : (\ldots)
\end{align*}
\]

\[
\begin{align*}
\text{Ev} & \rightarrow \\
\mathcal{H} & \rightarrow \mathcal{H}_0 \cdot \text{Ev} \\
o & \rightarrow \text{Prg} : (fr := (o, n); \text{rest}), \\
\text{Cnt} & : (\text{next}(n)), \\
\text{Att} & : (\ldots), \\
\text{Lvar} & : (\ldots)
\end{align*}
\]

\( \text{Ev} \rightarrow \) denotes \( \langle o \rightarrow o', (o, \text{cnt}), m, \bar{e} \rangle \)
where \( (o, n) \) is the fresh future identity.

- **ABS interpreter:** implementation in Maude by rewriting logic
Operational semantics

- State transition: \( s_1 \rightarrow s_2 \)
- Sending asynchronous method call from \( o \) to \( o' \):

\[
\begin{align*}
\mathcal{H} & \mapsto \mathcal{H}_0 \\
o & \mapsto Prg : (fr := o'!m(\bar{e}); rest), \\
Cnt & : (n), \\
Att & : (\ldots), \\
Lvar & : (\ldots)
\end{align*}
\]

\[
\begin{align*}
\mathcal{H} & \mapsto \mathcal{H}_0 \cdot Ev_{\rightarrow} \\
o & \mapsto Prg : (fr := (o, n); rest), \\
Cnt & : (\text{next}(n)), \\
Att & : (\ldots), \\
Lvar & : (\ldots)
\end{align*}
\]

\( Ev_{\rightarrow} \) denotes \( \langle o \rightarrow o', (o, cnt), m, \bar{e} \rangle \)
where \( (o, n) \) is the fresh future identity.

- \( ABS \) interpreter: implementation in Maude by rewriting logic.
Operational semantics

- State transition: $s_1 \rightarrow s_2$
- Sending asynchronous method call from $o$ to $o'$:

\[
\begin{align*}
\mathcal{H} & \mapsto \mathcal{H}_0 \\
o & \mapsto \text{Prg : (fr := o'!m(\bar{e}); rest)}, \\
Cnt & : (n), \\
Att & : (...), \\
Lvar & : (...)
\end{align*}
\]

\[
\begin{align*}
\mathcal{H} & \mapsto \mathcal{H}_0 \cdot Ev_{\rightarrow} \\
o & \mapsto \text{Prg : (fr := (o, n); rest)}, \\
Cnt & : (next(n)), \\
Att & : (...), \\
Lvar & : (...)
\end{align*}
\]

$Ev_{\rightarrow}$ denotes $\langle o \rightarrow o', (o, cnt), m, \bar{e} \rangle$
where $(o, n)$ is the fresh future identity.

- $ABS$ interpreter: implementation in Maude by rewriting logic
Operational semantics

- State transition: $s_1 \rightarrow s_2$
- Sending asynchronous method call from $o$ to $o'$:

$$
\begin{align*}
\mathcal{H} &\rightarrow \mathcal{H}_0 \\
o &\rightarrow Prg: (fr := o'!m(\bar{e}); rest), \\
Cnt &:= (n), \\
Att &:= (\ldots), \\
Lvar &:= (\ldots)
\end{align*}
$$

$$
\begin{align*}
E_{v \rightarrow} \\
\mathcal{H} &\rightarrow \mathcal{H}_0 \cdot E_{v \rightarrow} \\
o &\rightarrow Prg: (fr := (o, n); rest), \\
Cnt &:= (\text{next}(n)), \\
Att &:= (\ldots), \\
Lvar &:= (\ldots)
\end{align*}
$$

$E_{v \rightarrow}$ denotes $\langle o \rightarrow o', (o, cnt), m, \bar{e} \rangle$
where $(o, n)$ is the fresh future identity.

- ABS interpreter: implementation in Maude by rewriting logic
Operational semantics

- State transition: \( s_1 \rightarrow s_2 \)
- Sending asynchronous method call from \( o \) to \( o' \):

\[
\begin{align*}
\mathcal{H} & \rightarrow \mathcal{H}_0 \\
o & \rightarrow Prg : (fr := o'!m(\bar{e}); rest), \\
Cnt & : (n), \\
Att & : (...), \\
Lvar & : (...) \\
\end{align*}
\]

\[
\begin{align*}
Ev & \rightarrow \\
\mathcal{H} & \rightarrow \mathcal{H}_0 \cdot Ev \\
o & \rightarrow Prg : (fr := (o, n); rest), \\
Cnt & : (\text{next}(n)), \\
Att & : (...), \\
Lvar & : (...) \\
\end{align*}
\]

\( Ev \rightarrow \) denotes \( \langle o \rightarrow o', (o, cnt), m, \bar{e} \rangle \)
where \( (o, n) \) is the fresh future identity.

- ABS interpreter: implementation in Maude by rewriting logic
An example history (a possible sequence of events)
A possible history of the CV example

class Adm(Prof p) {
    String req(){
        Fut<Fut<String>> fr; Fut<String> mb, String cv;
        fr := p!ask(); mb := fr?; cv := mb?; return cv}
}

class Prof(Stud s) {
    Fut<String> ask(){ Fut<String> mb; mb := s!getCV(); return mb}
}

class Stud(String cv) {
    String getCV(){return cv}
}

\[\langle _\to a, u, req, \varepsilon \rangle, \langle a \to p, fr, ask, \varepsilon \rangle, \langle a \to p, fr, ask, \varepsilon \rangle, \langle p \to s, mb, getCV, \varepsilon \rangle, \langle p \to s, mb, getCV, \varepsilon \rangle, \langle a \leftarrow fr, mb \rangle, \langle \leftarrow p, fr, mb \rangle, \langle p \to s, u_3, getCV, \varepsilon \rangle, \langle a \leftarrow fr, mb \rangle, \langle \leftarrow s, mb, cv \rangle, \langle a \leftarrow mb, cv \rangle, \langle \leftarrow a, u, cv \rangle\]

From which we can observe that:
The cv received by the Administrator is the one provided by the Student.
A possible history of the CV example

class Adm(Prof p) {
    String req()
    {
        Fut<Fut<String>> fr; Fut<String> mb, String cv;
        fr := p!ask(); mb := fr?; cv := mb?; return cv;
    }
}

class Prof(Stud s) {
    Fut<String> ask()
    {
        Fut<String> mb;
        mb := s!getCV(); return mb;
    }
}

class Stud(String cv) {
    String getCV()
    {
        return cv;
    }
}

⟨_ → a, u, req, ε⟩, ⟨a → p, fr, ask, ε⟩, ⟨a → p, fr, ask, ε⟩, ⟨p → s, mb, getCV, ε⟩,
⟨← p, fr, mb⟩, ⟨p → s, u₃, getCV, ε⟩,
⟨a ←, fr, mb⟩, ⟨← s, mb, cv⟩, ⟨a ←, mb, cv⟩, ⟨← a, u, cv⟩

From which we can observe that:
The cv received by the Administrator is the one provided by the Student.
We are here now...

1. Introduction

2. Syntax and semantics

3. Reasoning system

4. Summary
Dynamic logic formula

\[ \psi \Rightarrow [s] \phi \]

Similar to Hoare triple: \( \{\psi\} s \{\phi\} \)
The order of events: $E_{v\rightarrow} \quad E_{v\rightarrow} \quad E_{v\leftarrow} \quad [E_{v\leftarrow}]^*$

$E_{v\rightarrow}$ denotes $\langle o \rightarrow o', u, m, \bar{e} \rangle$  $E_{v\rightarrow}$ denotes $\langle o \rightarrow o', u, m, \bar{e} \rangle$

$E_{v\leftarrow}$ denotes $\langle \leftarrow o', u, e \rangle$  $E_{v\leftarrow}$ denotes $\langle o \leftarrow, u, e \rangle$

(1)

$$
\vdash \forall fr'. \{ \mathcal{H} := \mathcal{H} \cdot E_{v\rightarrow} \parallel fr := fr' \} (wf(\mathcal{H}) \Rightarrow [s] \phi)
$$

$$
\vdash [fr = o'!m(\bar{e}); \ s] \phi
$$

(2)

$$
\vdash \forall v'. \{ \mathcal{H} := \mathcal{H} \cdot E_{v\leftarrow} \parallel v := v' \} (wf(\mathcal{H}) \Rightarrow [s] \phi)
$$

$$
\vdash [v := fr?; \ s] \phi
$$

(3)

$$
\Gamma_C \vdash (wf(\mathcal{H}) \land I_C(\bar{w}, \mathcal{H})) \Rightarrow

[\mathcal{H} := \mathcal{H} \cdot E_{v\rightarrow}; \ body_m; \mathcal{H} := \mathcal{H} \cdot E_{v\leftarrow}; ]

(wf(\mathcal{H}) \Rightarrow I_C(\bar{w}, \mathcal{H}))$$
The order of events: \( Ev \rightarrow Ev \rightarrow Ev \leftarrow [Ev\leftarrow]^* \)

\( Ev \rightarrow \) denotes \( \langle o \rightarrow o', u, m, \bar{e} \rangle \)

\( Ev \rightarrow \) denotes \( \langle o \rightarrow o', u, m, \bar{e} \rangle \)

\( Ev \leftarrow \) denotes \( \langle o', u, e \rangle \)

\( Ev \leftarrow \) denotes \( \langle o \leftarrow, u, e \rangle \)

(1)

\[
\vdash \forall fr'. \{ \mathcal{H} := \mathcal{H} \cdot Ev\rightarrow \mid fr := fr' \} \ (\text{wf}(\mathcal{H}) \Rightarrow [s]\phi) \\
\vdash [fr = o!'m(\bar{e}); \ s]\phi
\]

(2)

\[
\vdash \forall v'. \{ \mathcal{H} := \mathcal{H} \cdot Ev\leftarrow \mid v := v' \} \ (\text{wf}(\mathcal{H}) \Rightarrow [s]\phi) \\
\vdash [v := fr?; \ s]\phi
\]

(3)

\[
\Gamma_C \vdash (\text{wf}(\mathcal{H}) \land I_C(\overline{w}, \mathcal{H})) \Rightarrow \\
[\mathcal{H} := \mathcal{H} \cdot Ev\rightarrow; body_m; \mathcal{H} := \mathcal{H} \cdot Ev\leftarrow;] \\
(\text{wf}(\mathcal{H}) \Rightarrow I_C(\overline{w}, \mathcal{H}))
\]
Class invariants

$\mathcal{H}$ is local history.

\[ I_{Adm(p)}(\mathcal{H}) \triangleq \mathcal{H} \leq \]
\[ \langle c \rightarrow \text{this}, d, \text{req}, \varepsilon \rangle, \langle \text{this} \rightarrow p, u, \text{ask}, \varepsilon \rangle, \langle \text{this} \leftarrow, u, u' \rangle, \]
\[ \langle \text{this} \leftarrow, u', \text{cv} \rangle, \langle \leftarrow \text{this}, d, \text{cv} \rangle. \]
\[ \text{some } c, d, u, u', \text{cv} \]?

\[ I_{Prof(s)}(\mathcal{H}) \triangleq \mathcal{H} \leq \]
\[ \langle c \rightarrow \text{this}, u, \text{ask}, \varepsilon \rangle, \langle \text{this} \rightarrow s, u', \text{getCV}, \varepsilon \rangle, \langle \leftarrow \text{this}, u, u' \rangle. \]
\[ \text{some } c, u, u' \]?

\[ I_{Stud(cv)}(\mathcal{H}) \triangleq \mathcal{H} \leq \]
\[ \langle c \rightarrow \text{this}, u, \text{getCV}, \varepsilon \rangle, \langle \leftarrow \text{this}, u, \text{cv} \rangle. \]
\[ \text{some } c, u \]?

The order of the history events must be obeyed locally but can be interleaved in the global setting according to the scheduler.
Global invariant of the objects $a$, $p$ and $s$

$h$ is global history.

From the class invariants above, we can derive the following global invariant for the subsystem of $s : Stud(cv), p : Prof(s), a : Adm(p)$:

$$I(h) \triangleq wf(h) \land l_{Adm(p)}(h/a)_a^{this} \land l_{Prof(s)}(h/p)_p^{this} \land l_{Stud(cv)}(h/s)_s^{this}$$

where wellformedness allows us to relate the different object histories.
Reasoning about the example

Because the Administrator and the Student communicate with each other through the same future, we derive that the CV received by the Administrator is the one provided by the Student.
We are here now...

1. Introduction

2. Syntax and semantics

3. Reasoning system

4. Summary
Conclusion

- A compositional reasoning system for concurrent objects and futures
- A reasoning system for the ABS language in Dynamic Logic
- Futures are handled by events appearing in the histories
- Classes can be specified independently from the surroundings
- Modularity is achieved
- Global specification is realized from composing local specifications
Future Work

- Proof of soundness with respect to the operational semantics including a global communication history.
- The reasoning system is currently being implemented within the KeY framework at Technical University Darmstadt.
- Semi-automatic verification of large case studies using KeY
  - An elevator system has been implemented, simulated and tested according to the generated histories.
Comparison to Related Work

Papers:

- **Observable behavior of distributed systems: Component reasoning for concurrent objects.** (by Din et al.)
  - We revise the four-event semantics to deal with futures, allowing several readings of the same future, possibly by different objects.

- **A system for compositional verification of asynchronous objects.** (by Ahrendt et al.)
  - They present a compositional verification system in dynamic logic for the Creol language, but without futures.
  - The denotational semantics features the similar four communication events.
  - However, the reasoning system is based on the two-event semantics, which requires more complex rules.
Comparison to Related Work

Papers:

- **Observable behavior of distributed systems: Component reasoning for concurrent objects.** (by Din et al.)
  - We revise the four-event semantics to deal with futures, allowing several readings of the same future, possibly by different objects.

- **A system for compositional verification of asynchronous objects.** (by Ahrendt et al.)
  - They present a compositional verification system in dynamic logic for the Creol language, but without futures.
  - The denotational semantics features the similar four communication events.
  - However, the reasoning system is based on the two-event semantics, which requires more complex rules.
Comparison to Related Work

Papers:

- A complete guide to the future. (by de Boer et al.)
  - Futures are treated as visible objects, rather than our approach where futures are reflected by events in histories.
  - Global reasoning about futures is obtained by means of global invariants, rather than by compositional rules.
  - Therefore, the environment of a class need to be known at the verification time which is not required in our system.
Thank You