Larissa Aspects and Design-By-Contract

David Staucl, Karine Altisen, Florence Maraninchi

Verimag, Grenoble, France
Introduction

- Reactive systems are systems which are in constant interaction with their environment
- Cross-cutting concerns exist in reactive systems
- Aspect-oriented programming modularizes cross-cutting concerns, but existing aspect languages cannot be used
- Larissa is an aspect language for the synchronous programming language Argos
- This talk:
  - present Argos and Larissa
  - combine design-by-contract with Larissa
Argos

- Hierarchical, synchronous automata language
- Basic element: complete and deterministic Mealy automata
- Interface: a set of inputs, a set of outputs
Argos

- Hierarchical, synchronous automata language
- Basic element: complete and deterministic Mealy automata
- Interface: a set of inputs, a set of outputs
- Operators: parallel product, encapsulation
Argos

- Hierarchical, synchronous automata language
- Basic element: complete and deterministic Mealy automata
- Interface: a set of inputs, a set of outputs
- Operators: parallel product, encapsulation
- Compiled into flat automata

![Diagram of Argos automata]

```
mod2
```

```
a/mod2
```

```
mod2
```

```
mod2/mod4
```

```
mod2
```

```
a
```

```
a/mod4
```

```
a
```

```
a
```
Larissa

- Aspect language for Argos
- Consists of pointcuts and advice:
  - a join point is a transition
  - pointcuts select transitions in automata
  - advice replaces these transitions
- This cannot be done with the existing operators
- We want to preserve semantic properties, e.g. preservation of equivalence
Pointcuts

- Observer automata which take as inputs the inputs and outputs of the program they observe
- Output $JP$ is emitted when the program is in a join point
- Independent of the implementation of the program
Pointcuts

– Observer automata which take as inputs the inputs and outputs of the program they observe
– Output $\text{JP}$ is emitted when the program is in a join point
– Independent of the implementation of the program

pointcut

join point program
toInit Advice

- When a join point is passed, program execution is changed:
  - emit some outputs \(O\)
  - go to some target state
  - target state defined by a finite input \(\text{trace}\)
- Example advice: trace \(a.a\), advice output \(d\)
tolnit Advice

– When a join point is passed, program execution is changed:
  – emit some outputs $O$
  – go to some target state
  – target state defined by a finite input trace
– Example advice: trace $a.a$, advice output $d$

\begin{center}
\begin{tikzpicture}

\node[state,fill=green] (a) at (0,0) {$a$};
\node[state,fill=green] (a/b) at (1,-1) {$a/b$};
\node[state,fill=green] (a/d) at (0,-2) {$a/d$};
\node[state,fill=green] (a/b) at (-1,-1) {$\bar{a}/b$};
\draw[->] (a) to (a/b);
\draw[->] (a/b) to (a/d);
\draw[->] (a/d) to (a);
\draw[->] (a) to (a/b);
\end{tikzpicture}
\end{center}

woven program
**toCurrent Advice**

- As toInit, but execute the trace from the source state of the join point
- Example advice: trace $\bar{a}$, advice output $d$
toCurrent Advice

– As toInit, but execute the trace from the source state of the join point
– Example advice: trace \( \overline{a} \), advice output \( d \)
Advice Program

– Add an automaton to the join point transition
– Example: tolnit advice, trace a.a, output d, inserted automaton

inserted automaton

join point program
Advice Program

- Add an automaton to the join point transition
- Example: toInit advice, trace a.a, output d, inserted automaton

inserted automaton  woven program
Results

– Case study: modelling the interface of a complex wrist-watch [Software Composition 06]
– Quite complex base program in Argos and several aspects
– Used aspects to build a product line
**Results**

- Case study: modelling the interface of a complex wrist-watch [Software Composition 06]
  - Quite complex base program in Argos and several aspects
  - Used aspects to build a product line

- Formal aspect interference analysis [FOAL 06]
  - Cheap proof of non-interference
  - Either for two aspects, or for two aspects and a program
Results

– Case study: modelling the interface of a complex wrist-watch [Software Composition 06]
  – Quite complex base program in Argos and several aspects
  – Used aspects to build a product line

– Formal aspect interference analysis [FOAL 06]
  – Cheap proof of non-interference
  – Either for two aspects, or for two aspects and a program

– Implementation exists
Design by Contract

– Originally introduced by Bertrand Meyer for object-oriented programming
– A contract of a method consists of an assumption and a guarantee
– If the assumption holds when the method is called, the guarantee holds when the method returns
– Example:

```c
int m(int i) {
...
} /* @assume i ≤ 10 */
/* @guarantee \result ≤ 10 */
```
Aspects Modify Contracts

– Adding an aspect may invalidate the contract of a method

```java
int around(int i) : m(i) {
    return 1 + proceed(i + 1);
}
```
Aspects Modify Contracts

- Adding an aspect may invalidate the contract of a method

```java
int around(int i) : m(i){
    return 1 + proceed(i + 1);
}
```

- Sometimes, a new contract may be derived

```java
/*@assume i <= 9 */
/*@guarantee \result <= 11 */
```
Generating New Contracts

- Idea: apply an aspect \( \text{asp} \) to a contract \( C \), and obtain a new contract \( C' \) fulfilled by any \( P \triangleleft \text{asp} \), such that \( P \) fulfills \( C \)

\[ P \models C \Rightarrow P \triangleleft \text{asp} \models C' \]

- Goal: find a way to build \( C' \) automatically from \( C \) and \( \text{asp} \)
Ways of Modifying Contracts

– Aspects can modify contracts in multiple ways
– Refinement: If aspects weaken the assumption and reinforce the guarantee the new program can replace the old one and can also be used in other contexts
– If aspects reinforce the assumption or weaken guarantee, or modify them completely, $P \triangleleft \text{asp}$ must be considered a new program
Contracts for Reactive Systems

- **Assumptions** constrain the inputs from the environment
- **Guarantees** ensure properties on the outputs
- Example:
  - **Assumption**: input $a$ always occurs in pairs
  - **Guarantee**: input $a$ is immediately followed by output $b$
- Guarantee may not constrain inputs more than the assumption
Expressing Contracts with Observers

- Properties of reactive programs can be expressed with observers with a single output \text{err}
- A program fulfills a contract if, for any execution, the guarantee only emits \text{err} if the assumption emits \text{err} or has done so previously
Constructing a new Contract

– How can we obtain a new contract $C' = (\text{Ass}', \text{Gu}')$ such that

$$P \models (\text{Ass, Gu}) \Rightarrow P \triangleleft \text{asp} \models (\text{Ass}', \text{Gu}')$$

– Simulate the effect of the aspect on the program as far as possible on the assumption and the guarantee
– However, aspects cannot be applied to observers
– Transform observers into generator automata
– Apply aspect to generators
– Transform generators with aspects back to observers
– $\text{Ass}' = \text{obs}_{\text{Ass}}(\text{gen}(\text{Ass}) \triangleleft \text{asp})$
– $\text{Gu}' = \text{obs}_{\text{Gu}}(\text{gen}(\text{Gu}) \triangleleft \text{asp})$
Example

- Example aspect: advice output \( b \), trace \( a \)

Pointcut

Guarantee

Example aspect: advice output \( b \), trace \( a \)
Example

- Example aspect: advice output \( b \), trace \( a \)

Guarantee

Pointcut

\[ \overline{a}/b, \overline{a} \quad a, a/b \]

\[ \text{true}/b \]

\[ \text{gen(Guarantee)} \]

\[ \overline{b}/\text{err} \]

\[ a.b/\text{JP} \]
Example

- Example aspect: advice output b, trace a

\[
a \quad b / \text{err} \quad E
\]

Guarantee

\[
\overline{a} / b, \overline{a}
\]

\[
\text{gen}(\text{Guarantee}) \triangleleft \text{asp}
\]

Pointcut

\[
a.b / \text{JP}
\]

\[
\overline{a} / b, \overline{a}
\]

\[
a, a / b
\]

\[
\text{true} / b
\]

\[
\text{gen}(\text{Guarantee})
\]
Example

- Example aspect: advice output b, trace a

Pointcut

\[ a.b/JP \]

\[ a, a/b \]

\[ a, a/b, \overline{a} \]

\[ \text{gen}(	ext{Guarantee}) \]

\[ \text{asp} \]

\[ \text{E} \]

\[ \text{obs}_{\text{Gu}}(\text{gen}(	ext{Guarantee}) \triangleleft \text{asp}) \]
Some More Details

- Transformation into generators is standard
- Generators have the same interface as the program, plus output `err`, and are non-deterministic
Some More Details

- Transformation into generators is standard
- Generators have the same interface as the program, plus output `err`, and are non-deterministic
- Aspects can be woven into generators, but trace may lead to several target states
- Solution: add several advice transitions, introducing additional non-determinism
Some More Details

- Transformation into generators is standard
- Generators have the same interface as the program, plus output err, and are non-deterministic
- Aspects can be woven into generators, but trace may lead to several target states
- Solution: add several advice transitions, introducing additional non-determinism
- Transformation into observers different for assumption and guarantee
- Different handling of introduced non-determinism:
  - Assumption rejects all traces that can be rejected
  - Guarantee produces all traces that can be produced
Related Work

– Closest related work: Goldman and Katz, “Modular Generic Verification of LTL Properties for Aspects” (FOAL ’06)
– If a base program $P$ verifies a LTL property $\psi$, then $P \triangleleft \text{asp}$ verifies $\phi$
– $\phi$ build by weaving the aspect into $\psi$
– generic aspect model, restricted class of aspects
Related Work

- Closest related work: Goldman and Katz, “Modular Generic Verification of LTL Properties for Aspects” (FOAL ’06)
  - If a base program $P$ verifies a LTL property $\psi$, then $P \triangleleft \text{asp}$ verifies $\phi$
  - $\phi$ build by weaving the aspect into $\psi$
  - generic aspect model, restricted class of aspects
- Several approaches restrict the influence of aspects on the program, e.g.
  - Curtis and Leavens, “Behavioral Subtyping Analogy”
    Aspects must not violate the contract of a method
  - Dantas and Walker, “Harmless Advice”: Aspects must not influence the final result of the program
Conclusion

- Larissa: an aspect language with strong semantic properties
- Contracts: exploit semantic properties
- We have shown

\[ P \models (\text{Ass}, \text{Gu}) \]
\[ \Rightarrow P \triangleleft \text{asp} \models (\text{obs}_{\text{Ass}}(\text{gen(As}) \triangleleft \text{asp}), \text{obs}_{\text{Gu}}(\text{gen(Gu}) \triangleleft \text{asp})) \]

- Approach works on small examples
- Further work:
  - Validate approach on larger example
  - Extend approach to valued signals
Non-Deterministic Automata

- Aspects cannot be woven into observers
- Idea: use non-deterministic automata instead
- Same inputs as the program and same outputs plus err
- err signals that an input trace is not accepted
- NDAs are equivalent to observers
Weaving aspects into NDA

- Problem with weaving aspects: trace may lead to more than one state
- Solution: add an advice transition to every possible target state
- This introduces additional non-determinism
- Aspect can be woven into NDAs

\[ P \models (\text{Ass}, \text{Gu}) \Rightarrow P \triangleleft \text{asp} \models (\text{Ass} \triangleleft \text{asp}, \text{Gu} \triangleleft \text{asp}) \]
Weaving Aspects into Contracts

- Weaving aspects into NDAs introduces additional non-determinism
- For the modular verification theorem to hold, this non-determinism needs different treatment for pre- and guarantee
  - Assumption must reject all traces that can possibly be rejected
  - Guarantee must produce all traces that can possibly be produced
- This difference is taken into account during the transformation from NDA to observer
Converting NDAs into Observers – Assumptions

- Assumption must reject all traces that can possibly be rejected
- Remove all outputs, except err
- Determinize the automaton, but do not consider non-error transitions if an error transition is available
Converting NDAs into Observers – Assumptions

- Assumption must reject all traces that can possibly be rejected
- Remove all outputs, except err
- Determinize the automaton, but do not consider non-error transitions if an error transition is available

\[ \text{obs}_{\text{Ass}}(\text{gen}(\text{Assumption}) \triangleleft \text{asp}) \]
Converting NDAs into Observers – Guarantees

- Guarantee must produce all traces that can possibly be produced
- Convert the NDA into an observer which accepts exactly the traces the NDA produces
- Determinize the observer, but do not consider error transitions if a non-error transition is available

\[
\gen(Guarantee) \triangleleft \text{asp}
\]
Converting NDAs into Observers – Guarantees

- Guarantee must produce all traces that can possibly be produced
- Convert the NDA into an observer which accepts exactly the traces the NDA produces
- Determinize the observer, but do not consider error transitions if a non-error transition is available

\[ obs_{Gu}(gen(Guarantee) \bowtie asp) \]