Modifying Contracts with Larissa Aspects

David Stauch
Verimag, Grenoble, France

Introduction

- Aspect-oriented programming modularizes cross-cutting concerns
- Cross-cutting concerns exist in reactive systems, but popular aspect languages as AspectJ can not be used
- Larissa is an aspect language for the synchronous programming language Argos
- Design-by-Contract abstracts program parts in contracts
- Adapted to synchronous languages by Lionel Morel
- This talk : combine design-by-contract with Larissa

Outline

- Motivation
- Existing Work
  - Argos and Contracts
  - Larissa
- Contributions
  - Weaving Aspects in Contracts
  - Case Study

Design by Contract

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

Design by Contract

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

```
class c {
   /* @assume i < 10 */
   /* @guarantee \result < 10 */
    int m(int i ){...}
}
```

Aspect-Oriented Programming

- **Basic concepts in aspect languages :**
  - **join point : a point where the aspect intervenes**
  - **pointcut : selects a set of join points**
  - **advice : inserts behavior at join points**

Aspect-Oriented Programming

- Basic concepts in aspect languages :
  - join point : a point where the aspect intervenes
  - pointcut : selects a set of join points
  - advice : inserts behavior at join points
- AspectJ example :

```
// pointcut selects executions of method m
pointcut pcm(int i) :
   execution(int c.m(int)) && args(i);

// advice adds 1 to argument and result
int around(int i) : pcm(i)
   {return 1 + proceed(i + 1);}
```

Aspects Modify Contracts

- **Adding an aspect may invalidate the contract of a method**
- **Consider contract of of method m with aspect**

```
/* @assume i < 10 */
/* @guarantee \result < 10 */
...
advice : return 1 + proceed(i + 1);
```

- **Contract invalid, aspect calls method with modified argument and modifies result**

Aspects Modify Contracts

- Adding an aspect may invalidate the contract of a method
- Consider contract of of method m with aspect

```
/* @assume i < 10 */
/* @guarantee \result < 10 */
...
advice : return 1 + proceed(i + 1);
```

- Contract invalid, aspect calls method with modified argument and modifies result
- Sometimes, a new contract can be derived

```
/* @assume i < 9 */
/* @guarantee \result < 11 */
```

Generating New Contracts

- **Idea : apply an aspect asp to a contract C, and obtain a new contract C' fulfilled by any $P \triangleleft$ asp, such that P fulfills C**

$$P \models C \Rightarrow P \triangleleft \textbf{asp} \models C'$$

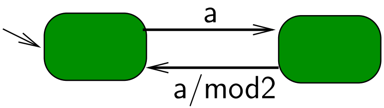
- **Goal : find a way to build C' automatically from C and asp, for Argos and Larissa aspects**

Outline

- Motivation
- Existing Work
  - Argos and Contracts
  - Larissa
- Contributions
  - Weaving Aspects in Contracts
  - Case Study

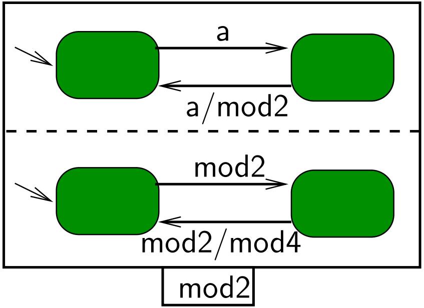
Argos

- **Hierarchical, synchronous automata language**
- **Basic element : complete and deterministic Mealy automata with Boolean signals**
- **Interface : a set of inputs, a set of outputs**



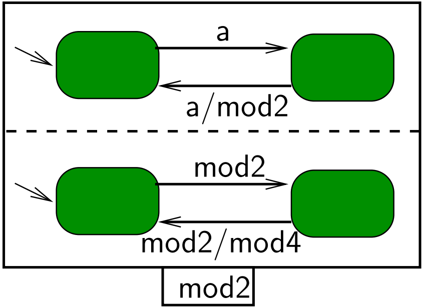
Argos

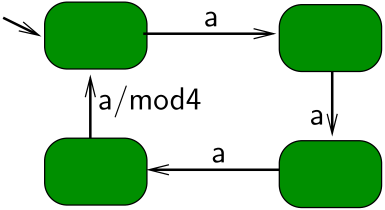
- **Hierarchical, synchronous automata language**
- **Basic element : complete and deterministic Mealy automata with Boolean signals**
- **Interface : a set of inputs, a set of outputs**
- **Operators : parallel product, encapsulation**



Argos

- **Hierarchical, synchronous automata language**
- **Basic element : complete and deterministic Mealy automata with Boolean signals**
- **Interface : a set of inputs, a set of outputs**
- **Operators : parallel product, encapsulation**
- **Semantics defined by compilation into flat automata**



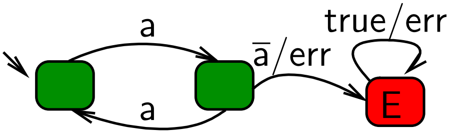


Contracts for Reactive Systems

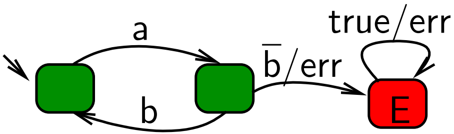
- **Assumptions constrain the inputs from the environment**
- **Guarantees ensure properties on the outputs**
- **Example with input a and output b :**
  - **Assumption : a always occurs in pairs**
  - **Guarantee : a is immidiatly followed by b**

Expressing Contracts with Observers

- **Properties of reactive programs can be expressed with observers with a single output err**
- **A program fulfills a contract if, for any execution, the guarantee only emits err if the assumption emits err**



Assumption



Guarantee

a always occurs in pairs **a is immidiatly followed by b**

Outline

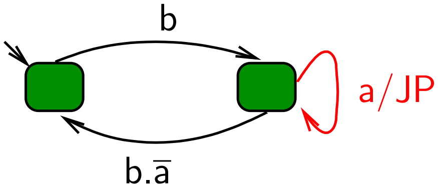
- **Motivation**
- **Existing Work**
  - **Argos and Contracts**
  - **Larissa**
- **Contributions**
  - **Weaving Aspects in Contracts**
  - **Case Study**

Larissa

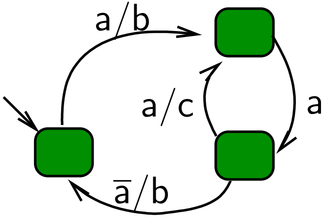
- Aspect language for Argos
- Contains same concepts as other aspect languages :
  - **join point** : a transition
  - **pointcut** : select transitions in automata
  - **advice** : replace join point transitions
- We want aspect weaving to be an operator of the language :
  - cannot be done with the existing operators
  - preserves determinism and completeness of programs
  - preserves equivalence between programs

Pointcuts

- **Pointcuts are observer automata with output JP**
- **JP is emitted when the program is in a join point**
- **Independent of the implementation of the program**



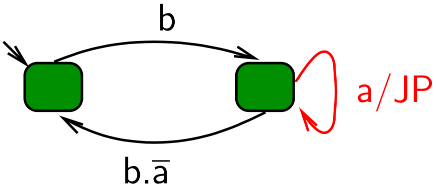
pointcut



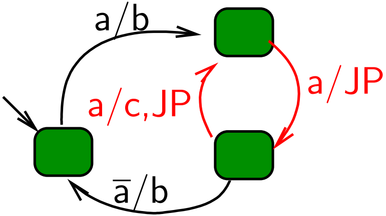
base program

Pointcuts

- **Pointcuts are observer automata with output JP**
- **JP is emitted when the program is in a join point**
- **Independent of the implementation of the program**



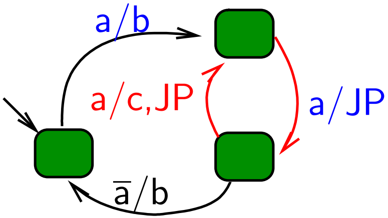
pointcut



join point program

toInit Advice

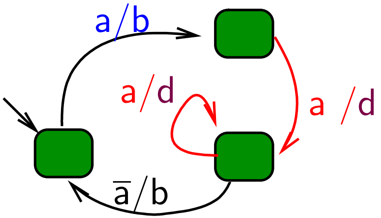
- When a join point is passed, execution is changed :
  - emit some outputs O
  - go to some target state
  - target state defined by a finite input trace
- Example advice : trace a.a, advice output d



join point program

toInit Advice

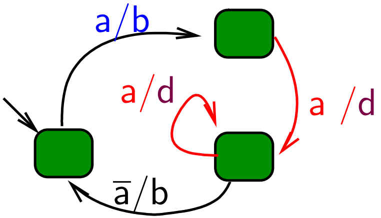
- **When a join point is passed, execution is changed :**
  - **emit some outputs O**
  - **go to some target state**
  - **target state defined by a finite input trace**
- **Example advice : trace a.a, advice output d**



woven program

toInit Advice

- **When a join point is passed, 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**



woven program

- **Other kinds of advice exist**

Outline

- Motivation
- Existing Work
  - Argos and Contracts
  - Larissa
- Contributions
  - Weaving Aspects in Contracts
  - Case Study

Constructing a new Contract

- How can we obtain a new contract $C' = (\mathsf{A}', \mathsf{G}')$, such that for any program P we have

$$P \models (\mathsf{A}, \mathsf{G}) \Rightarrow P \triangleleft \mathsf{asp} \models (\mathsf{A}', \mathsf{G}')$$

- **Idea :** Simulate the effect of the aspect on the program as far as possible on the assumption and the guarantee

Technical Overview

- **Problem : aspects cannot be applied directly to observer automata**

Technical Overview

- **Problem : aspects cannot be applied directly to observer automata**
- **Solution :**
  - **Transform observers into generator automata :** gen

Technical Overview

- **Problem : aspects cannot be applied directly to observer automata**
- **Solution :**
  - **Transform observers into generator automata :** gen
  - **Apply aspect to generators**

Technical Overview

- **Problem : aspects cannot be applied directly to observer automata**
- **Solution :**
  - **Transform observers into generator automata :** gen
  - **Apply aspect to generators**
  - **Transform woven generators back to observers :** obs

Technical Overview

- **Problem : aspects cannot be applied directly to observer automata**
- **Solution :**
  - **Transform observers into generator automata :** gen
  - **Apply aspect to generators**
  - **Transform woven generators back to observers :** obs
  - **Different for assumption and guarantee :**
    - $\mathbf{A}' = obs_{\mathbf{A}}(gen_{\mathbf{A}}(\mathbf{A}) \triangleleft \mathbf{asp})$
    - $\mathbf{G}' = obs_{\mathbf{G}}(gen_{\mathbf{G}}(\mathbf{G}) \triangleleft \mathbf{asp})$

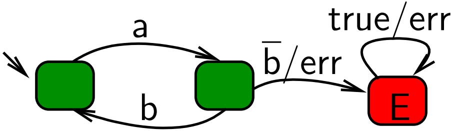
Technical Overview

- **Problem : aspects cannot be applied directly to observer automata**
- **Solution :**
  - **Transform observers into generator automata :** gen
  - **Apply aspect to generators**
  - **Transform woven generators back to observers :** obs
  - **Different for assumption and guarantee :**
    - $\mathsf{A}' = obs_{\mathsf{A}}(gen_{\mathsf{A}}(\mathsf{A}) \triangleleft \mathsf{asp})$
    - $\mathsf{G}' = obs_{\mathsf{G}}(gen_{\mathsf{G}}(\mathsf{G}) \triangleleft \mathsf{asp})$
- **Then,**

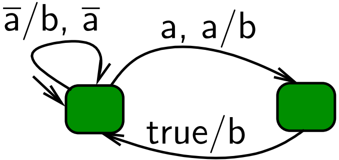
$$P \models (\mathsf{A}, \mathsf{G}) \Rightarrow P \triangleleft \mathsf{asp} \models (\mathsf{A}', \mathsf{G}')$$

Observers to Generators : gen_{G}

- Transform observer into a generator that generates exactly the traces the observer accepts
- Generators are nondeterministic
- Guarantee generators have no error transitions
  - error transitions correspond to input/output combinations that are never produced
  - must not be considered by the aspect



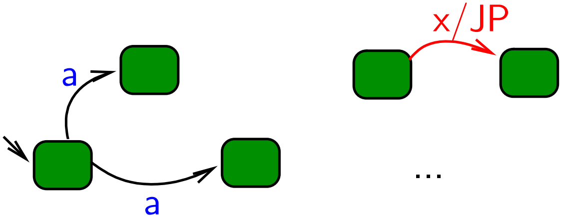
Guarantee



gen_{G}(**Guarantee**)

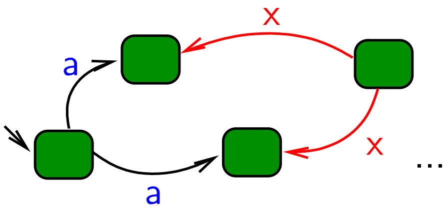
Weaving Aspects in Generators

- Aspects can be woven into generators, but trace may lead to several target states
- Solution : add several advice transitions, introducing additional non-determinism



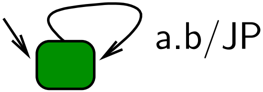
Weaving Aspects in Generators

- Aspects can be woven into generators, but trace may lead to several target states
- Solution : add several advice transitions, introducing additional non-determinism



Example – Weaving Aspects

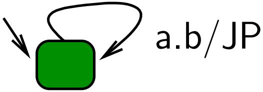
- **Example aspect : advice output b, trace a**
- **Trace leads to a single state**



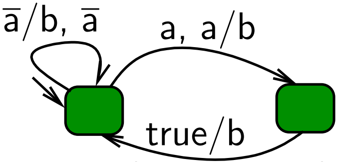
Pointcut

Example – Weaving Aspects

- **Example aspect : advice output b, trace a**
- **Trace leads to a single state**



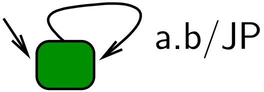
Pointcut



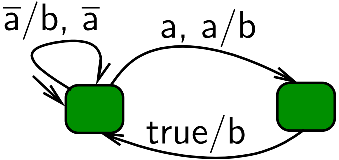
$gen_{\mathbf{G}}$(**Guarantee**)

Example – Weaving Aspects

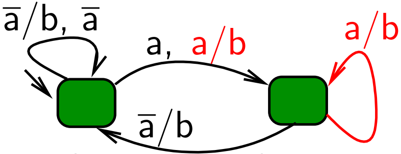
- **Example aspect : advice output b, trace a**
- **Trace leads to a single state**



Pointcut



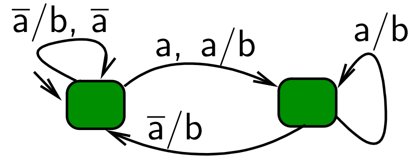
$gen_{\mathbf{G}}$(**Guarantee**)



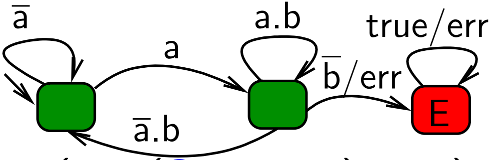
$gen_{\mathbf{G}}$(**Guarantee**)$\lhd$ **asp**

Transforming Generators into Observers

- Guarantee generators have no error transitions
  - create error transition for input/output combination that are not produced



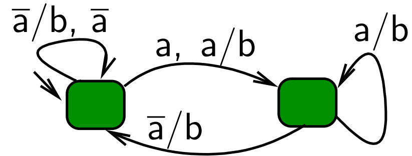
$gen_{\mathbf{G}}$(**Guarantee**)$\lhd$ **asp**



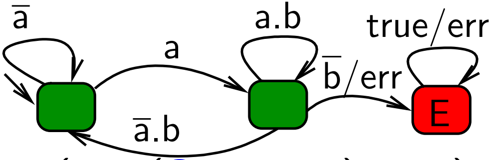
$obs_{\mathbf{G}}(gen_{\mathbf{G}}$(**Guarantee**)$\lhd$ **asp**)

Transforming Generators into Observers

- **Guarantee generators have no error transitions**
  - **create error transition for input/output combination that are not produced**
- **Obtained observer may be non-deterministic, but can be determinized**



$gen_{\mathbf{G}}(\mathbf{Guarantee}) \lhd \mathbf{asp}$



$obs_{\mathbf{G}}(gen_{\mathbf{G}}(\mathbf{Guarantee}) \lhd \mathbf{asp})$

Assumption Weaving

- **Assumption : $gen_{\mathsf{A}}, obs_{\mathsf{A}}$ different from $gen_{\mathsf{G}}, obs_{\mathsf{G}}$**
  - **contract (A,G) can be written (true,A ⇒G), thus A negated guarantee**
  - **need to be conservative : assumption must reject all possibly undefined behaviors, guarantee must produce all possible behaviors**
  - **goal : make G as strict as possible, A as liberal as possible**
- **We have shown**

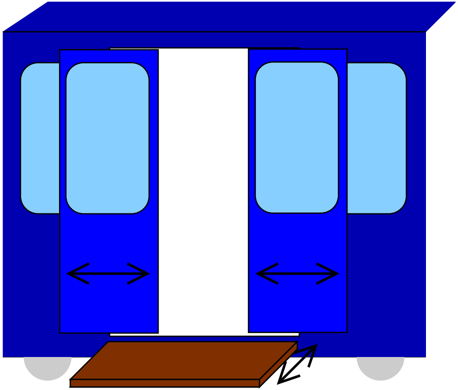
$$
\begin{aligned}
& P \models (\mathsf{A}, \mathsf{G}) \\
\Rightarrow\ & P \lhd \mathsf{asp} \models (obs_{\mathsf{A}}(gen(\mathsf{A}) \lhd \mathsf{asp}), obs_{\mathsf{G}}(gen(\mathsf{G}) \lhd \mathsf{asp}))
\end{aligned}
$$

Outline

- **Motivation**
- **Existing Work**
  - **Argos and Contracts**
  - **Larissa**
- **Contributions**
  - **Weaving Aspects in Contracts**
  - **Case Study**

Case Study - Tramway Door Controller (1)

- **Tramway door controller example from Lustre tutorial**
- **Implement a basic door controller P in Argos**



- **Add support for a gangway with Larissa aspects**
  - **Aspect Ext extends the gangway**
  - **Aspect Ret retracts the gangway**
- **Use contract generation for modular verification**

Case Study - Tramway Door Controller (2)

- **Three safety properties SP must be verified :**
  - **doors must not be open outside a station**
  - **gangway must not be extended outside a station**
  - **gangway must only be moved if doors are closed**
- **Verify safety properties SP against a model E of the environment and the controller P**

Case Study - Tramway Door Controller (2)

- **Three safety properties SP must be verified :**
  - **doors must not be open outside a station**
  - **gangway must not be extended outside a station**
  - **gangway must only be moved if doors are closed**
- **Verify safety properties SP against a model E of the environment and the controller P**
- **First approach : verify woven program**
  - **Weave aspect into P, verify SP**

    $\mathsf{E} \parallel \mathsf{P} \triangleleft \mathsf{Ext} \triangleleft \mathsf{Ret} \models \mathsf{SP}$
  - **Took 11s using our implementation**

Case Study - Tramway Door Controller (3)

- **Second approach : model P with a contract (A,G), and verify modularly**
- **Goal : avoid verifying woven program P $\triangleleft$ Ext $\triangleleft$ Ret**

Case Study - Tramway Door Controller (3)

- Second approach : model P with a contract (A,G), and verify modularly
- Goal : avoid verifying woven program $P \triangleleft Ext \triangleleft Ret$
- Three separate steps :
  - verify P fulfills contract (<0.5s) : $P \models (A, G)$
  - weave aspects into A, verify environment verifies new assumption (<0.5s) : $E \models A \triangleleft Ext \triangleleft Ret$
  - weave aspect into G, verify new guarantee verifies safety properties (3.4s) : $E \parallel G \triangleleft Ext \triangleleft Ret \models SP$

Case Study - Tramway Door Controller (3)

- Second approach : model P with a contract (A,G), and verify modularly
- Goal : avoid verifying woven program P $\triangleleft$ Ext $\triangleleft$ Ret
- Three separate steps :
  - verify P fulfills contract (<0.5s) : P $\models$ (A, G)
  - weave aspects into A, verify environment verifies new assumption (<0.5s) : E $\models$ A $\triangleleft$ Ext $\triangleleft$ Ret
  - weave aspect into G, verify new guarantee verifies safety properties (3.4s) : E $\|$ G $\triangleleft$ Ext $\triangleleft$ Ret $\models$ SP
- Then, we know that E $\|$ P $\triangleleft$ Ext $\triangleleft$ Ret $\models$ SP holds, because of $P \models$ (A,G)$\Rightarrow P \triangleleft$ asp $\models$ (A',G')

Conclusion

- **Larissa : an aspect language with strong semantic properties**
- **Contracts : exploit semantic properties**
- **Approach has been validated on a medium-size example**
- **Further work :**
  - **Extend approach to valued signals**