Larissa Aspects and Design-By-Contract

David Stauch, Karine Altisen, Florence Maraninchi

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 and a guarantee
- If the assumption holds when the method is called, the guarantee holds when the method returns
- Example :

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

Aspects Modify Contracts

– Adding an aspect may invalidate the contract of a method

```
pointcut pcm(int i) :
  execution(int c.m(int)) && args(i);

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

Aspects Modify Contracts

– Adding an aspect may invalidate the contract of a method

```
pointcut pcm(int i) :
  execution(int c.m(int)) && args(i);

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

– Sometimes, a new contract may 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**

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

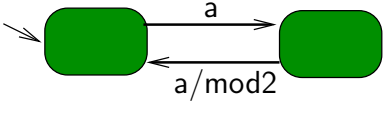
– **Goal : find a way to build C' automatically from C and asp, for Argos and Larissa aspects**

Outline

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

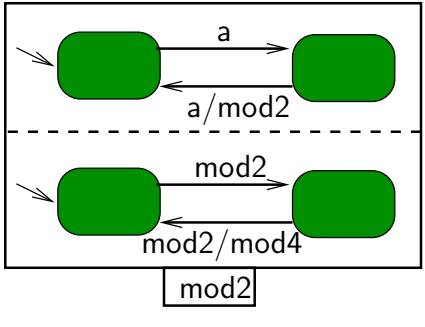
Argos

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



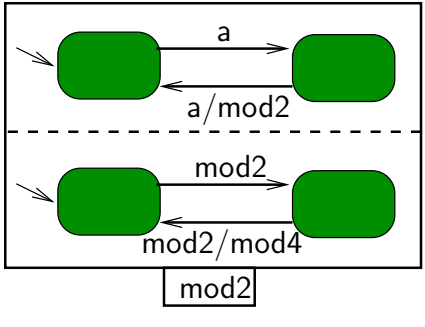
Argos

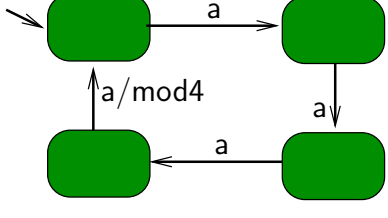
- Hierarchical, synchronous automata language
- Basic element : complete and deterministic Mealy automata with Boolean signals
- Interface : a set of inputs, a set of outputs
- Operators : parallel product, encapsulation



Argos

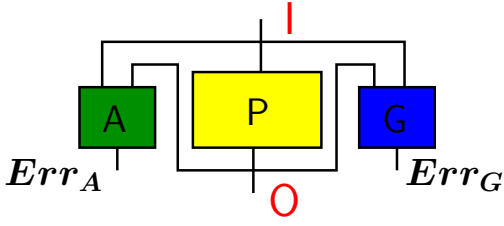
- Hierarchical, synchronous automata language
- Basic element : complete and deterministic Mealy automata with Boolean signals
- Interface : a set of inputs, a set of outputs
- Operators : parallel product, encapsulation
- Compiled into flat automata





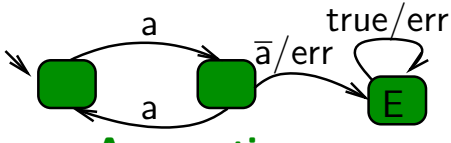
Contracts for Reactive Systems

- **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**
- Assumptions must not constrain outputs
- guarantees must not constrain inputs

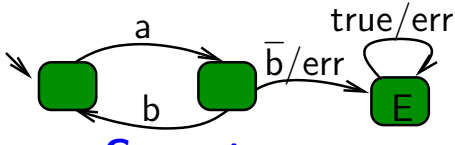


Expressing Contracts with Observers

- **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



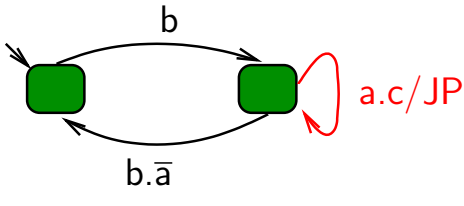
Guarantee

Larissa

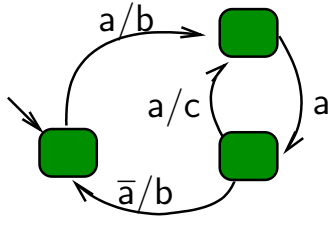
- Aspect language for Argos
- Consists of pointcuts and advice :
  - a join point is a transition
  - pointcuts select transitions in automata
  - advice replaces these transitions
- This cannot be done with the existing operators
- We want to preserve semantic properties, e.g. preservation of equivalence

Pointcuts

- **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**



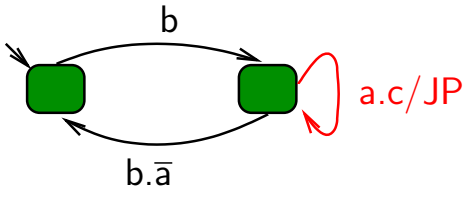
pointcut



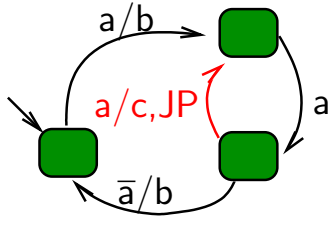
base program

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**



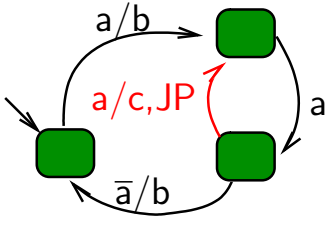
pointcut



join point program

toInit Advice

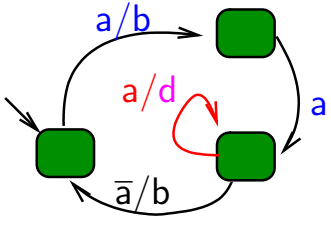
- When a join point is passed, program execution is changed :
  - emit some outputs O
  - go to some target state
  - target state defined by a finite input trace
- Example advice : trace a.a, advice output d



join point program

toInit Advice

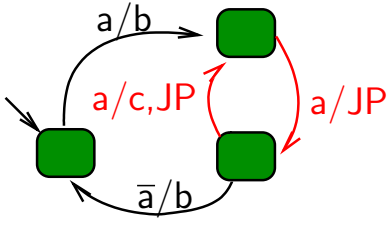
- When a join point is passed, program execution is changed :
  - emit some outputs O
  - go to some target state
  - target state defined by a finite input trace
- Example advice : trace a.a, advice output d



woven program

toCurrent Advice

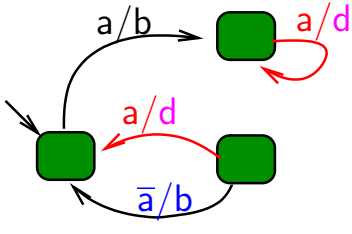
– As toInit, but execute the trace from the source state of the join point transition
– Example advice : trace $\overline{a}$, advice output d



join point program

toCurrent Advice

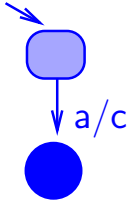
– As toInit, but execute the trace from the source state of the join point transition
– Example advice : trace $\overline{a}$, advice output d



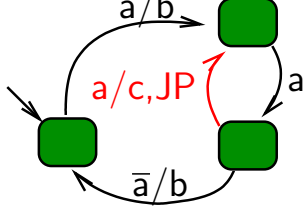
woven program

Advice Program

– **Replace join point transition by an automaton**
– **Example : toInit advice, trace a.a, output d, inserted automaton**



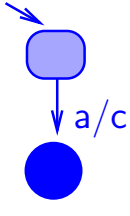
inserted automaton



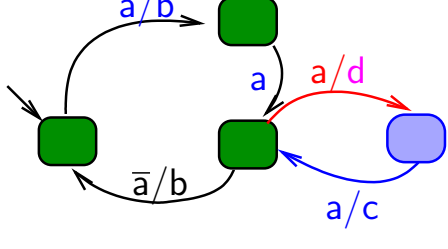
join point program

Advice Program

– **Replace join point transition by an automaton**
– **Example : tolnit advice, trace a.a, output d, inserted automaton**



inserted automaton



woven program

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**

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

– **Simulate the effect of the aspect on the program as far as possible on the assumption and the guarantee**

Technical Overview

- However, aspects cannot be applied directly to observers
- Transform observers into generator automata
- Apply aspect to generators
- Transform generators with aspects back to observers
- $\mathbf{A}' = obs_{\mathbf{A}}(gen_{\mathbf{A}}(\mathbf{A}) \triangleleft \mathbf{asp})$
- $\mathbf{G}' = obs_{\mathbf{G}}(gen_{\mathbf{G}}(\mathbf{G}) \triangleleft \mathbf{asp})$
- Then,

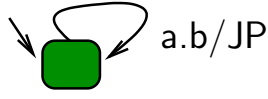
$$P \models (\mathbf{A}, \mathbf{G}) \Rightarrow P \triangleleft \mathbf{asp} \models (\mathbf{A}', \mathbf{G}')$$

Guarantee vs. Assumption

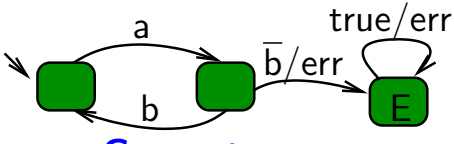
- Guarantee generators have no error transitions
  - correspond to input/output combinations that don't exist
  - error transitions are added by obs_{G} for input/output combinations that cannot be produced
- Assumption generators have error transitions
  - input/output combinations may be produced by a different environment
  - aspect can remove these error transitions
  - obs_{A} does not add error transitions

Example – Guarantee

– **Example aspect : advice output b, trace a**



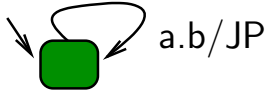
Pointcut



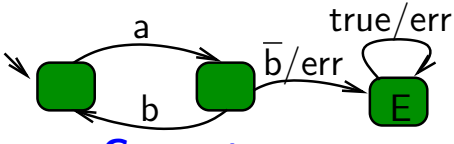
Guarantee

Example – Guarantee

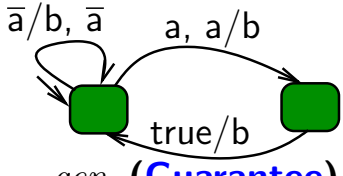
– **Example aspect : advice output b, trace a**



Pointcut



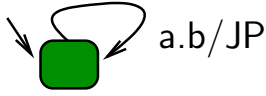
Guarantee



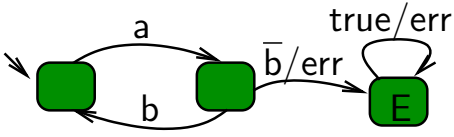
$gen_{\mathbf{G}}$**(Guarantee)**

Example – Guarantee

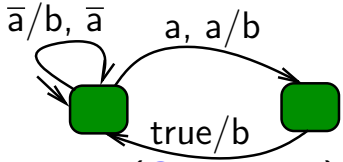
– **Example aspect : advice output b, trace a**



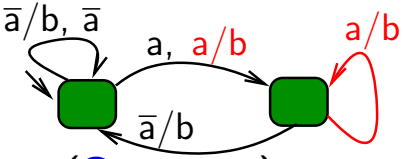
Pointcut



Guarantee



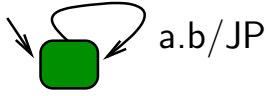
$gen_{\mathbf{G}}$**(Guarantee)**



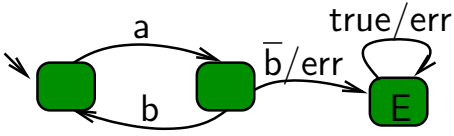
$gen_{\mathbf{G}}$**(Guarantee)**◁ **asp**

Example – Guarantee

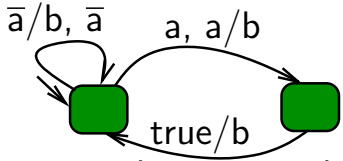
– **Example aspect : advice output b, trace a**



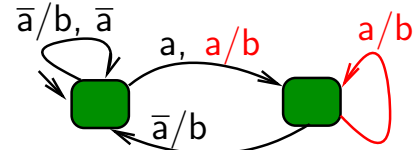
Pointcut



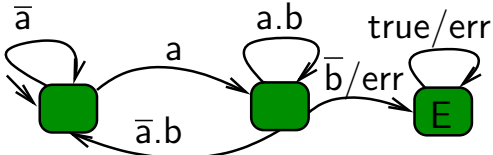
Guarantee



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



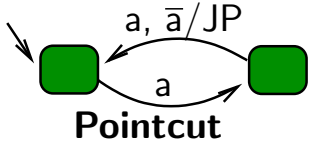
$gen_{\mathbf{G}}$**(Guarantee)**◁ **asp**

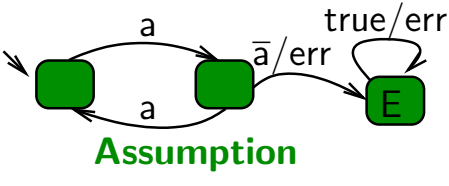


$obs_{\mathbf{G}}(gen_{\mathbf{G}}$**(Guarantee)**◁ **asp)**

Example – Assumption

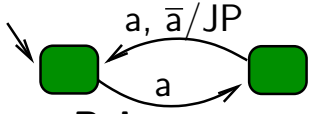
– **Example aspect : advice output b, trace a**



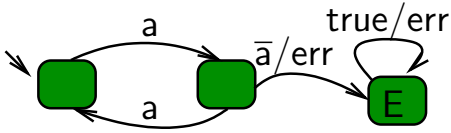


Example – Assumption

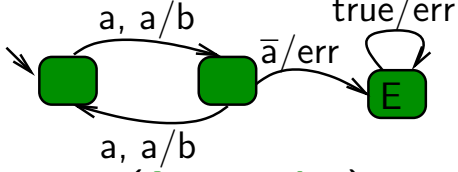
– **Example aspect : advice output b, trace a**



Pointcut



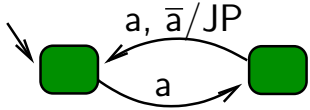
Assumption



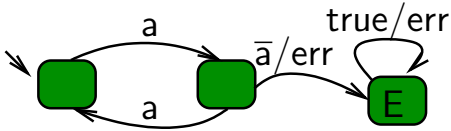
$gen_{\mathbf{A}}$**(Assumption)**

Example – Assumption

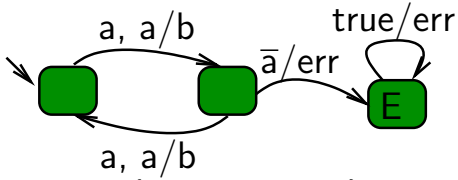
– **Example aspect : advice output b, trace a**



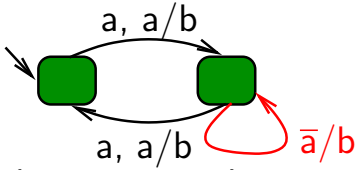
Pointcut



Assumption



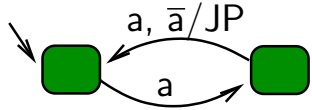
$gen_{\mathbf{A}}$**(Assumption)**



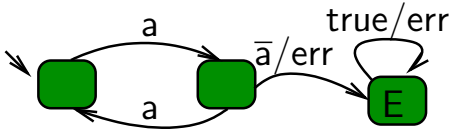
$gen_{\mathbf{A}}$**(Assumption)** $\triangleleft$ **asp**

Example – Assumption

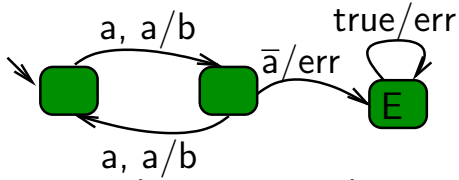
– **Example aspect : advice output b, trace a**



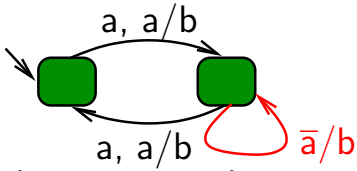
Pointcut



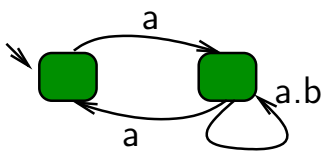
Assumption



$gen_{\mathbf{A}}$**(Assumption)**



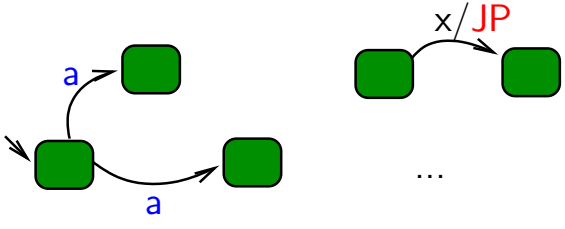
$gen_{\mathbf{A}}$**(Assumption)**◁ **asp**



$obs_{\mathbf{A}}(gen_{\mathbf{A}}$**(Assumption)**◁ **asp)**

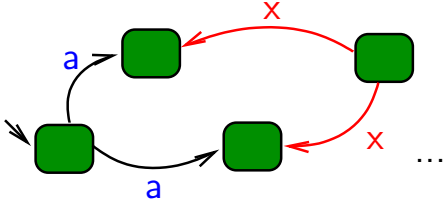
Some More Details (1)

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



Some More Details (1)

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



Some More Details (2)

- Observer may be non-deterministic
- Can be determinized, but differences between assumption and guarantee
- Assumption :
  - Assumption must reject a trace if there exists a program which is not defined for it
  - Thus, error transitions are given priority during determinisation
- Guarantee :
  - Guarantee must only reject a trace if no program can produce it
  - Thus, non-error transitions are given priority during determinisation

Case Study - Tramway Door Controller (1)

- Use contract generation for modular verification
- Tramway door controller example from Lustre tutorial
- Implement a basic door controller C in Argos
- Add support for a gangway with Larissa aspects
  - Aspect Ext extends the gangway before the door is opened
  - Aspect Ret retracts the gangway before the tram leaves the station

Case Study - Tramway Door Controller (2)

- Several safety properties must be verified : e.g. doors must not be open outside a station
- Verify safety properties P against a model E of the environment and the controller

Case Study - Tramway Door Controller (2)

- Several safety properties must be verified : e.g. doors must not be open outside a station
- Verify safety properties P against a model E of the environment and the controller
- First approach : verify woven program
  - Weave aspect into C, verify properties (11s)

    E, C$\triangleleft$ Ext $\triangleleft$ Ret $\models$ P

Case Study - Tramway Door Controller (2)

- Several safety properties must be verified : e.g. doors must not be open outside a station
- Verify safety properties P against a model E of the environment and the controller
- First approach : verify woven program
  - Weave aspect into C, verify properties (11s)
    E, C$\triangleleft$ Ext $\triangleleft$ Ret $\models$ P
- Second approach : use a contract (A,G) for C
  - verify C fulfills contract ($<$0.5s) : C $\models$ (A, G)
  - weave aspects into A, verify ($<$0.5s) :
    E $\models$ A $\triangleleft$ Ext $\triangleleft$ Ret
  - weave aspect into G, verify properties (3.4s) :
    E, G $\triangleleft$ Ext $\triangleleft$ Ret $\models$ P

Conclusion

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

$$P \models (\mathbf{A}, \mathbf{G})$$
$$\Rightarrow P \triangleleft \mathbf{asp} \models (obs_{\mathbf{A}}(gen(\mathbf{A}) \triangleleft \mathbf{asp}), obs_{\mathbf{G}}(gen(\mathbf{G}) \triangleleft \mathbf{asp}))$$

- **Approach has been validated on a medium-size example**
- **Further work :**
  - **Extend approach to valued signals**

Bibliography

- Argos
  - Argos : an Automaton-Based Synchronous Language, F. Maraninchi and Y. Rémond. Computer Languages no. 27, April 2001.
- Contracts for Argos
  - Logical-Time Contracts for Reactive Embedded Components. F. Maraninchi and L. Morel. 30eme Euromicro Conference, Component-Based Software Engineering Track (ECBSE) 2004.
- Larissa
  - K. Altisen, F. Maraninchi and D. Stauch. Aspect-oriented programming for reactive systems : Larissa, a Proposal in the synchronous framework. in Science of Computer Programming, Special Issue on Foundations of Aspect-Oriented Programming, 2006.
  - D. Stauch, K. Altisen and F. Maraninchi. Interference of Larissa Aspects. Foundations Of Aspect-oriented Languages(FOAL) , Bonn, Germany, 2006.
  - K. Altisen, F. Maraninchi and D. Stauch. Modular Design of Man-Machine Interfaces with Larissa. Software Composition, LNCS 4089, Vienna, Austria, 2006.