Interference of Larissa Aspects

David Stauch, Karine Altisen, Florence Maraninchi

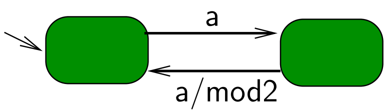
Verimag, Grenoble, France

Outline

- Reactive systems are systems which are in constant interaction with their environment
- Cross-cutting concerns exist in reactive systems, but existing aspect languages cannot be used
- Larissa is an aspect language for the synchronous programming language Argos
- This talk :
  - Sequential weaving in Larissa causes aspect interference problems
  - Joint weaving resolves these problems
  - We can define sufficient conditions to prove non-interference of aspects

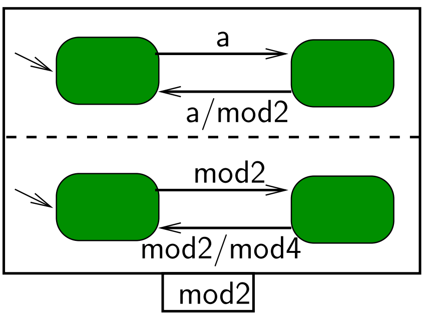
Argos

- Synchronous automata language
- Basic element : complete and deterministic Mealy automata
- Interface : set of inputs and set of outputs



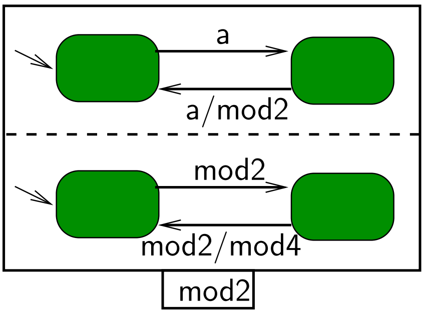
Argos

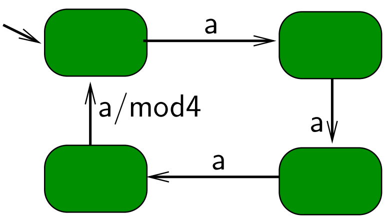
- Synchronous automata language
- Basic element : complete and deterministic Mealy automata
- Interface : set of inputs and set of outputs
- Operators : parallel product, encapsulation



Argos

- Synchronous automata language
- Basic element : complete and deterministic Mealy automata
- Interface : set of inputs and set of outputs
- Operators : parallel product, encapsulation
- Operators are transformations into flat automata



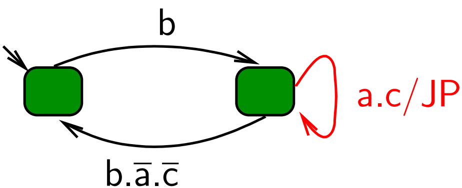


Larissa

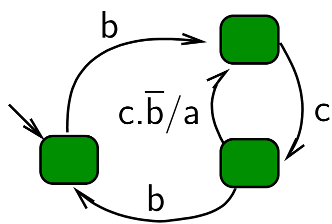
- **Aspect language for Argos**
- **Modularizes recurrent cross-cutting concerns in Argos**
- **Consists of pointcuts and advice :**
  - **pointcuts select transitions in automata**
  - **advice replaces these transitions**
- **This cannot be done with the existing operators**
- **We want to preserve semantic properties, e.g. preservation of trace equivalence**

Pointcuts

- Observer automata which take as inputs the inputs and outputs of the program
- Output JP is emitted when the program is in a join point, i.e. it takes a join point transition
- Independent of the implementation of the program



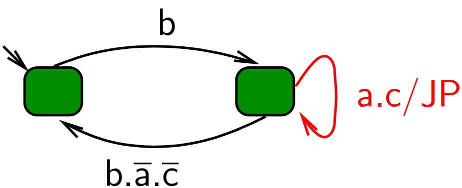
pointcut



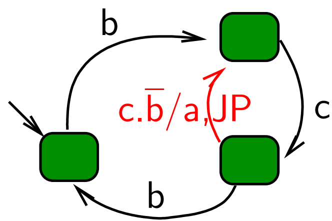
base program

Pointcuts

- Observer automata which take as inputs the inputs and outputs of the program
- Output JP is emitted when the program is in a join point, i.e. it takes a join point transition
- Independent of the implementation of the program



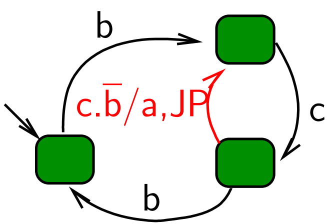
pointcut



join point program

Advice

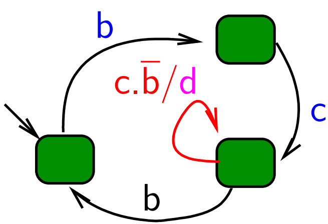
- When a join point is passed, program execution is changed :
  - emit outputs O
  - go to some target state
  - target state defined by a finite input trace, executed from the initial state
- Example advice : trace b.c, advice output d



join point program

Advice

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

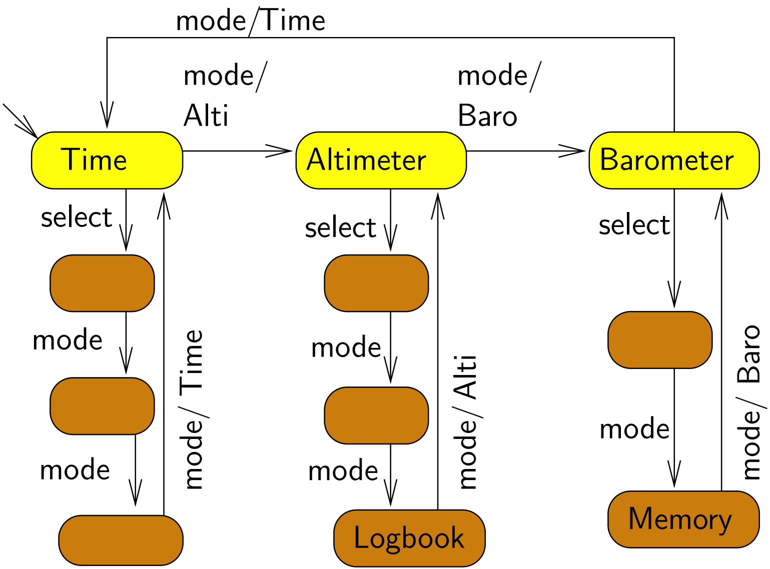


woven program

Example : Suunto Wristwatch

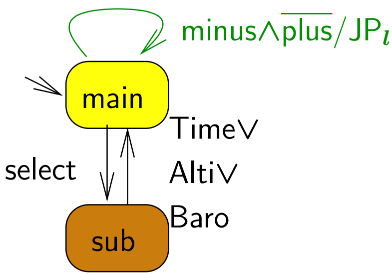
- **Model the interface of a complex wristwatch**
- **Functionalities : watch, altimeter, barometer**
- **Each functionality has a main mode and some submodes**
- **Four buttons :** `mode`, `select`, `minus`, `plus`

Model in Argos : watch



Two Shortcut Aspects

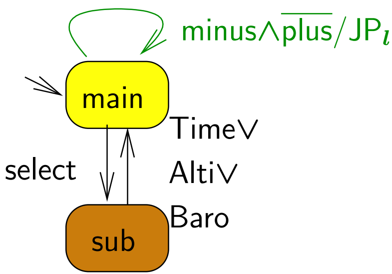
- `minus` **and** `plus` **buttons are used as shortcuts in the main modes**
- **Pressing** `minus` **goes to the Logbook mode**
- **aspect LB with trace** `mode.select.mode.mode`
- **output** Logbook



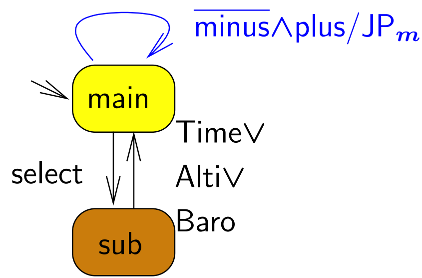
Two Shortcut Aspects

- `minus` **and** `plus` **buttons are used as shortcuts in the main modes**

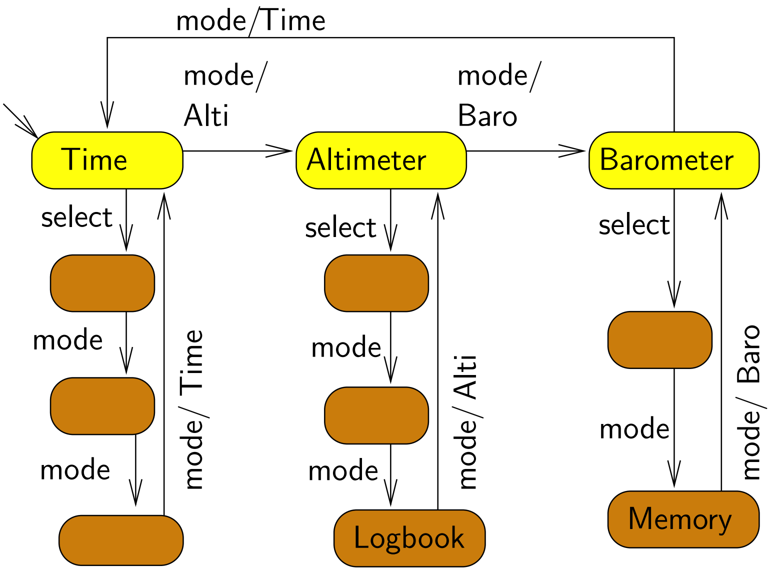
- **Pressing** `minus` **goes to the Logbook mode**
- **aspect LB with trace** `mode.select.mode.mode`
- **output** `Logbook`



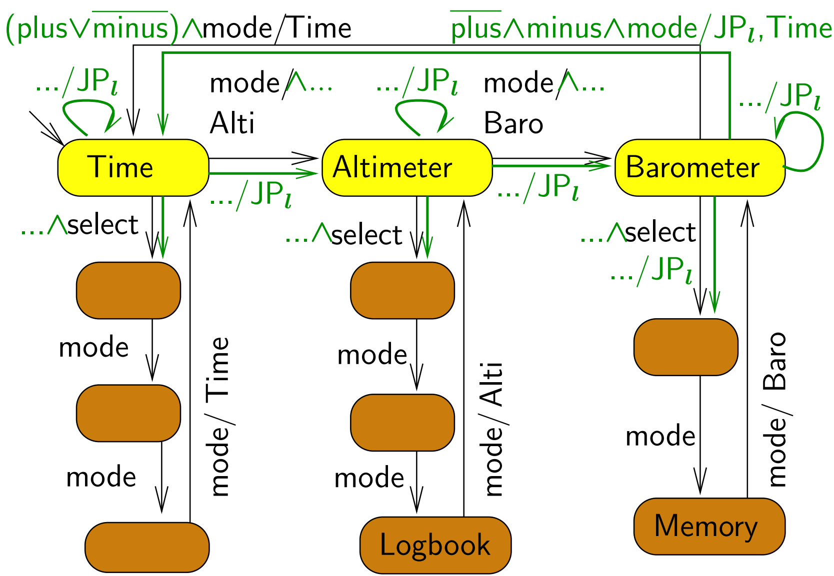
- **Pressing** `plus` **goes to the Memory mode**
- **aspect M with trace** `mode.mode.select.mode`
- **output** `Memory`



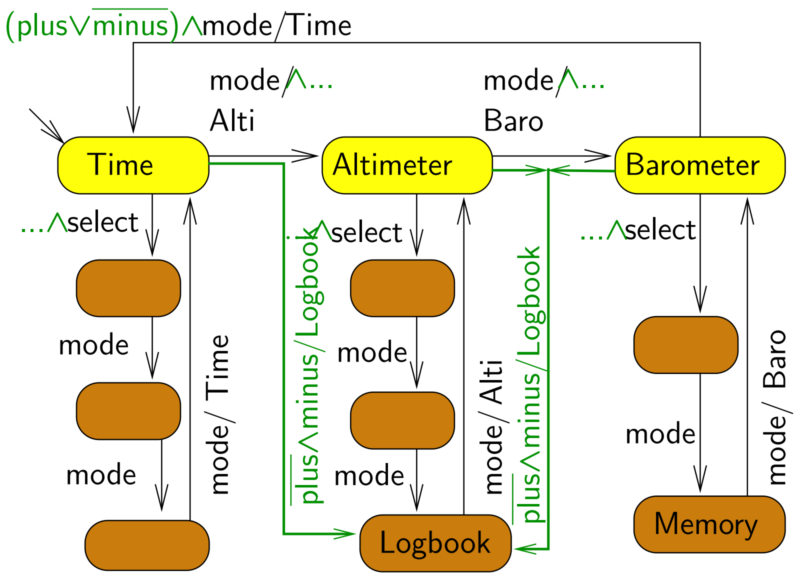
Weaving the First Aspect : watch◁LB



Weaving the First Aspect : watch◁LB

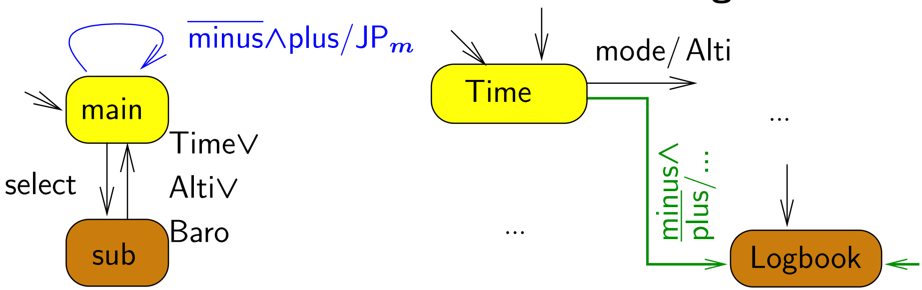


Weaving the First Aspect : watch◁LB



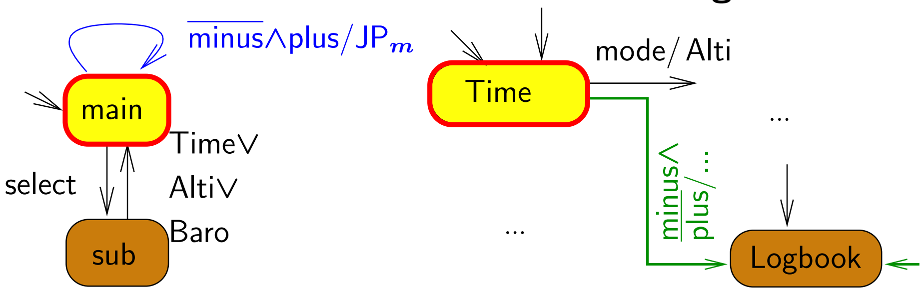
Weaving the Second Aspect : watch◁LB◁M

- Pointcut doesn't capture join points correctly
- When `minus` is pressed in a main mode, program goes to a submode but the pointcut stays in main mode
- Advice transitions are added to the Logbook mode



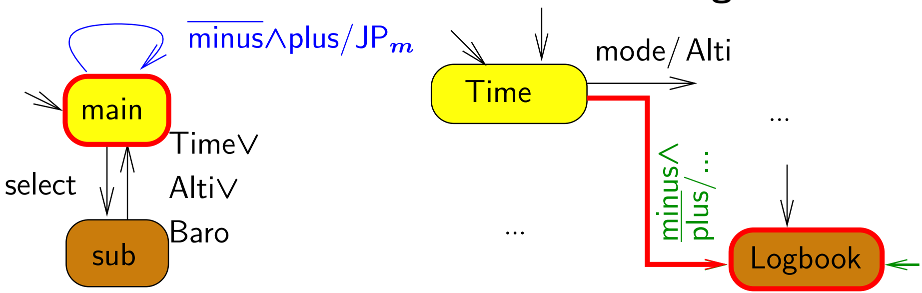
Weaving the Second Aspect : watch◁LB◁M

- **Pointcut doesn't capture join points correctly**
- **When `minus` is pressed in a main mode, program goes to a submode but the pointcut stays in main mode**
- **Advice transitions are added to the Logbook mode**



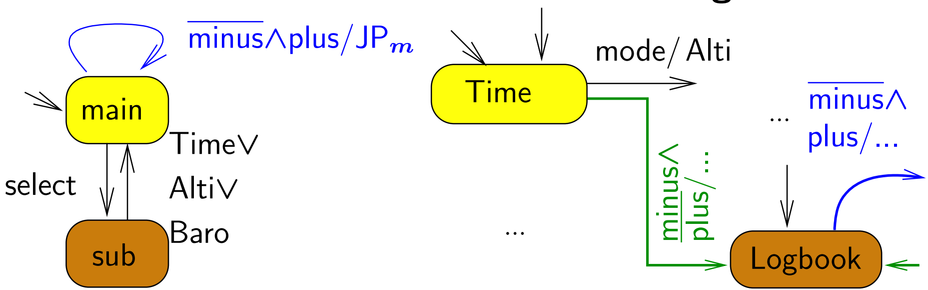
Weaving the Second Aspect : watch◁LB◁M

- Pointcut doesn't capture join points correctly
- When `minus` is pressed in a main mode, program goes to a submode but the pointcut stays in main mode
- Advice transitions are added to the Logbook mode



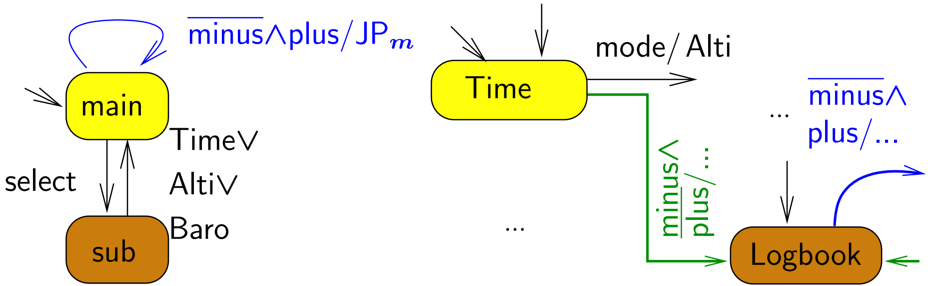
Weaving the Second Aspect : watch◁LB◁M

- Pointcut doesn't capture join points correctly
- When `minus` is pressed in a main mode, program goes to a submode but the pointcut stays in main mode
- Advice transitions are added to the Logbook mode



Weaving the Second Aspect : watch◁LB◁M

- **Pointcut doesn't capture join points correctly**
- **When `minus` is pressed in a main mode, program goes to a submode but the pointcut stays in main mode**
- **Advice transitions are added to the Logbook mode**



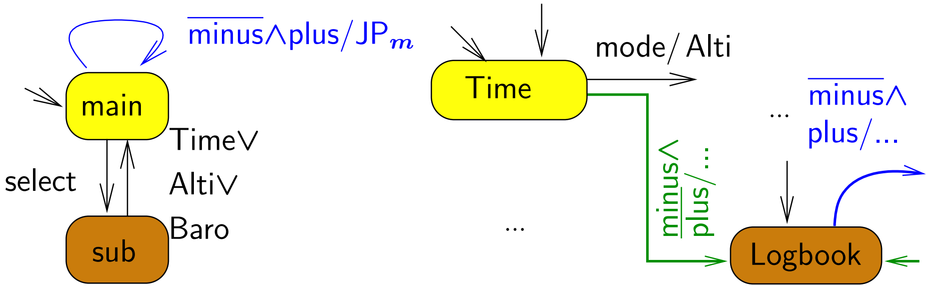
- **Problem : pointcut was written for the base program, not for the woven program watch◁LB**

Weaving the Second Aspect : watch◁LB◁M

– **Pointcut doesn't capture join points correctly**

– **When `minus` is pressed in a main mode, program goes to a submode but the pointcut stays in main mode**

– **Advice transitions are added to the Logbook mode**



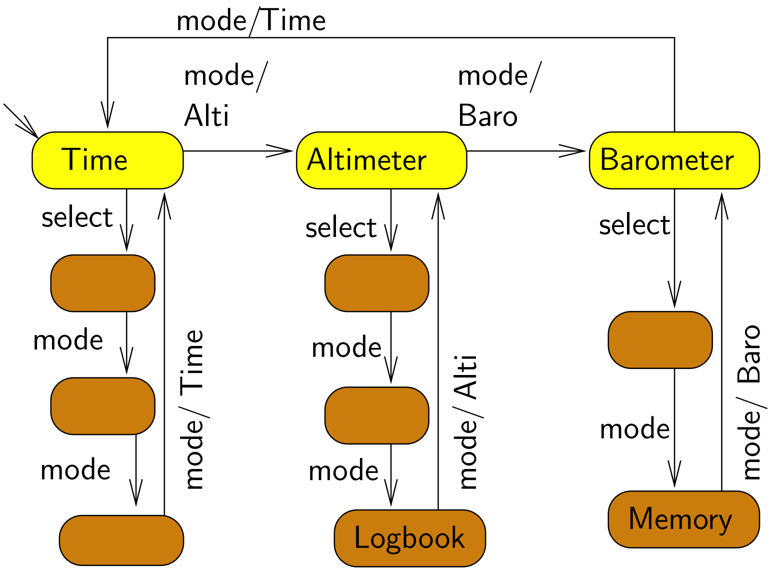
– **Problem : pointcut was written for the base program, not for the woven program watch◁LB**

– **watch◁LB◁M is not equivalent to watch◁M◁LB**

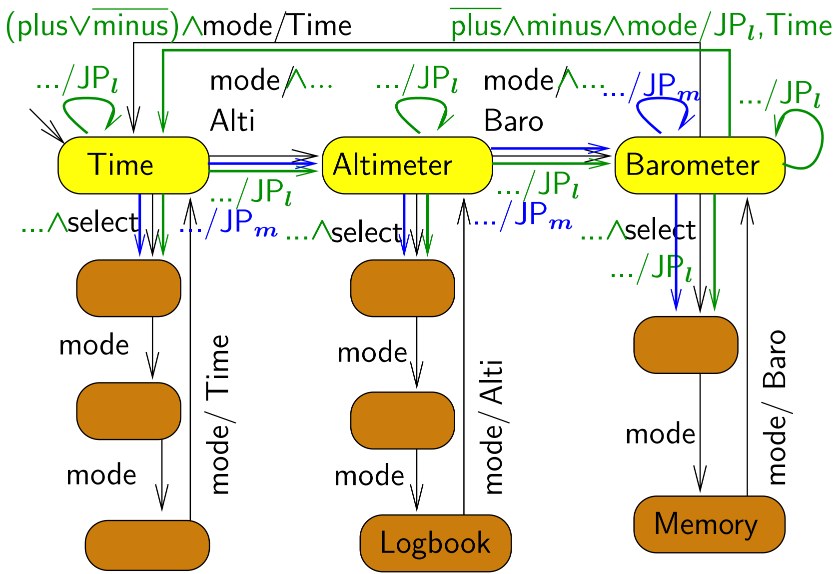
Joint Weaving

- **Idea : weave aspects jointly into the program**
- **select join points for all aspects first, then apply advice**
- **let P be a program and $A_1, \ldots, A_n$ aspects with pointcuts $\mathsf{PC}_1 \ldots \mathsf{PC}_n$**
- **calculate $P \triangleleft (A_1, \ldots, A_n)$**
  - **compute parallel product of $\mathsf{PC}_1 \ldots \mathsf{PC}_n$**
  - **apply product to program and determine join point transition**
  - **sequentially apply advice in reverse order**

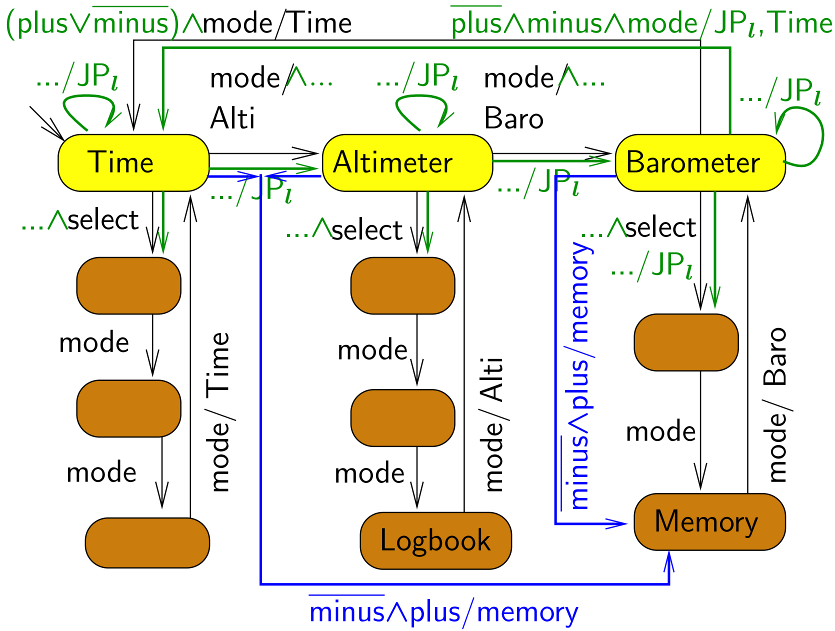
Application to the Example : watch◁(LB,M)



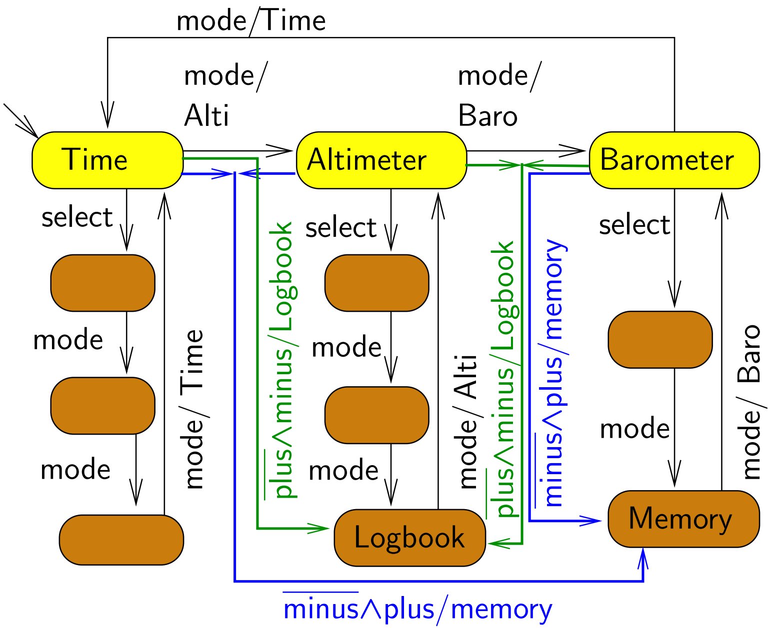
Application to the Example : watch◁(LB,M)



Application to the Example : watch◁(LB,M)



Application to the Example : watch◁(LB,M)

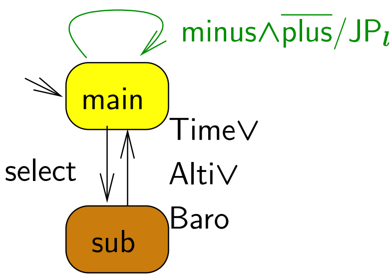


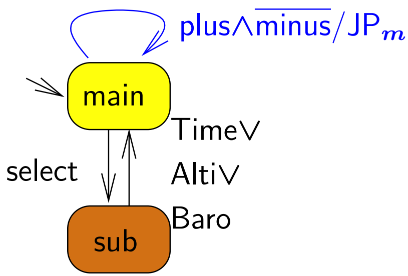
Interference

- **watch$\lhd$(LB,M) is equivalent to watch$\lhd$(M,LB)**
- **We say that two aspects $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$ interfere iff $P \lhd (\mathcal{A}_1 \ldots \mathcal{A}_i, \mathcal{A}_{i+1} \ldots \mathcal{A}_n)$ is not trace equivalent to $P \lhd (\mathcal{A}_1 \ldots \mathcal{A}_{i+1}, \mathcal{A}_i \ldots \mathcal{A}_n)$**
- **Jointly woven Larissa aspects still interfere, if they have the same join points.**

Interfering aspects

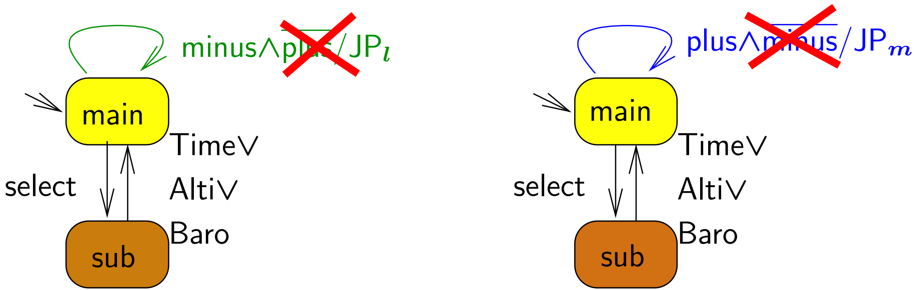
– **If we modify the pointcuts slightly, the shortcut aspects interfere**





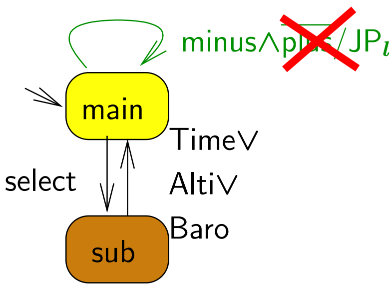
Interfering aspects

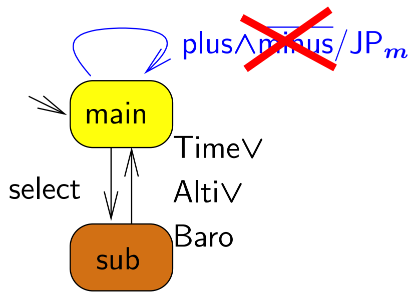
– **If we modify the pointcuts slightly, the shortcut aspects interfere**



Interfering aspects

- If we modify the pointcuts slightly, the shortcut aspects interfere
- Both pointcuts select the transitions with minus∧plus as join points, but only one advice can execute
- Thus, the aspects interfere



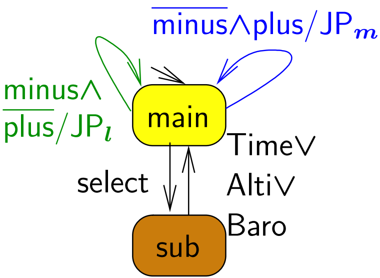


Strong Non-Interference

- Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two aspects with pointcuts PC_1 and PC_2 with join point signals JP_1 and JP_2
- Strong non-interference : $\mathcal{A}_1$ and $\mathcal{A}_2$ never interfere, regardless of the program they are applied to.
- Theorem 1 : If the product of PC_1 and PC_2 contains no transition that emits JP_1 and JP_2, then the two aspects are strongly non-interferent.
- Theorem 1 describes a sufficient, but not a necessary condition

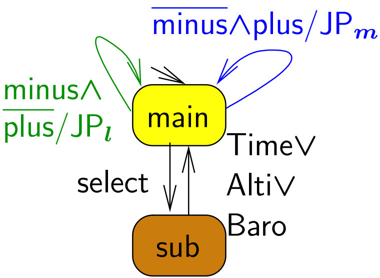
Shortcut aspects

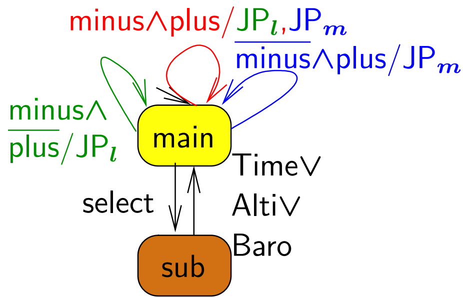
- Calculate the product of the pointcuts of the shortcut aspects
- For the original aspects, no transition emits both JP_l and JP_m
- the aspects are strongly non-interferent



Shortcut aspects

- Calculate the product of the pointcuts of the shortcut aspects
- For the original aspects, no transition emits both JP_l and JP_m
- the aspects are strongly non-interferent
- For the modified shortcut aspects, there is such a transition
- Tells us where the aspects interfere





Weak Non-Interference

- Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two aspects with pointcuts PC_1 and PC_2 with join point signals JP_1 and JP_2
- Weak non-interference : $\mathcal{A}_1$ and $\mathcal{A}_2$ do not interfere when they are applied to a program P
- Theorem 2 : If after the application of the product of PC_1 and PC_2 to P, no transition emits JP_1 and JP_2, then the two aspects are weakly non-interferent for P
- Theorem 2 describes a sufficient, but not a necessary condition

Conclusion

- Extended Larissa with joint weaving mechanism
- Joint weaving was easy to add, because join point selection and advice weaving were already separated
- Sufficient condition for non-interference
- Conditions are cheap to calculate, included in weaving
- Precise way to calculate non-interference : prove semantic equivalence
  - very expensive for larger automata
  - only possible for Boolean signals
- Perspective : extend Larissa to valued signals