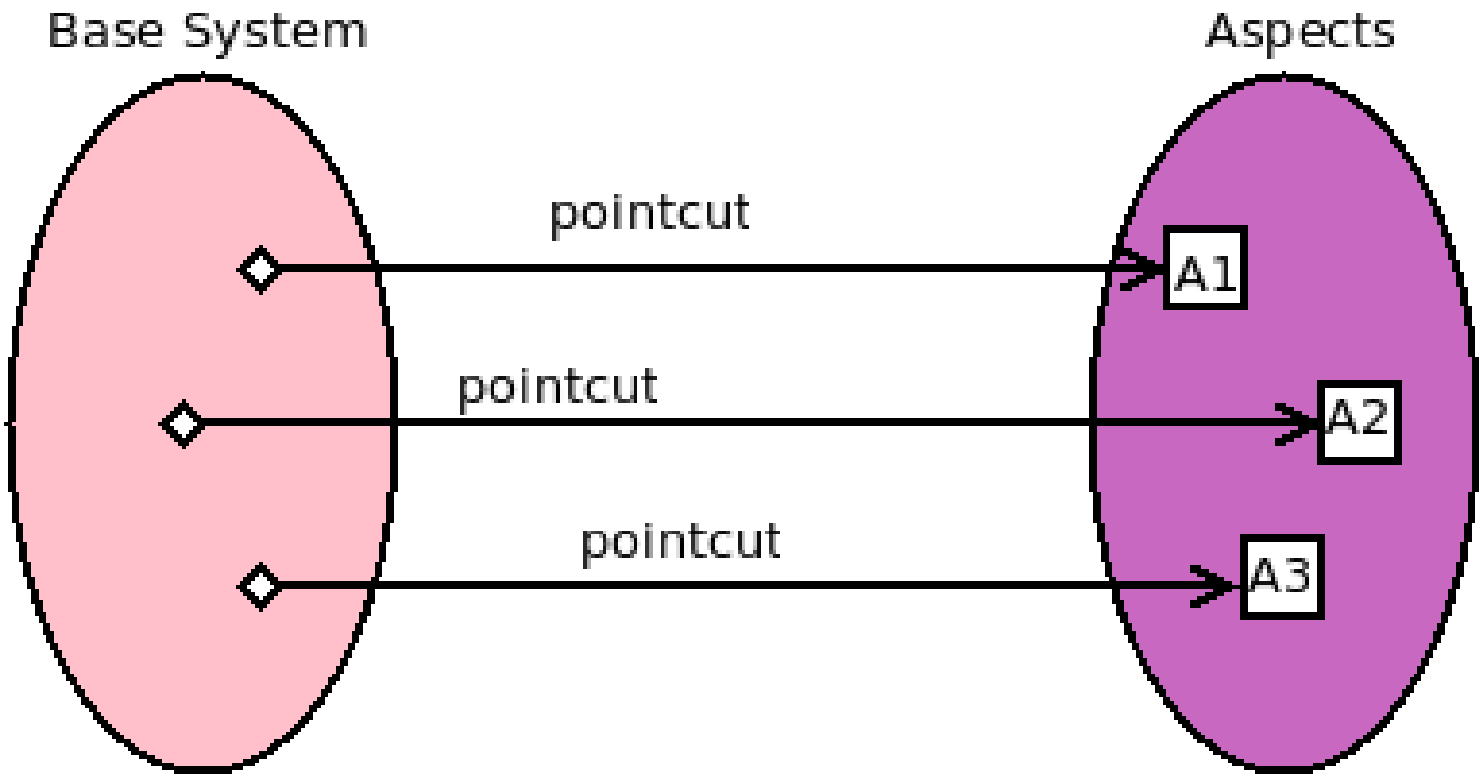
Compositional Verification of Events and Observers

Cynthia Disenfeld and Shmuel Katz

Technion - Israel Institute of Technology

March 21, 2011

"Standard" Aspect Model



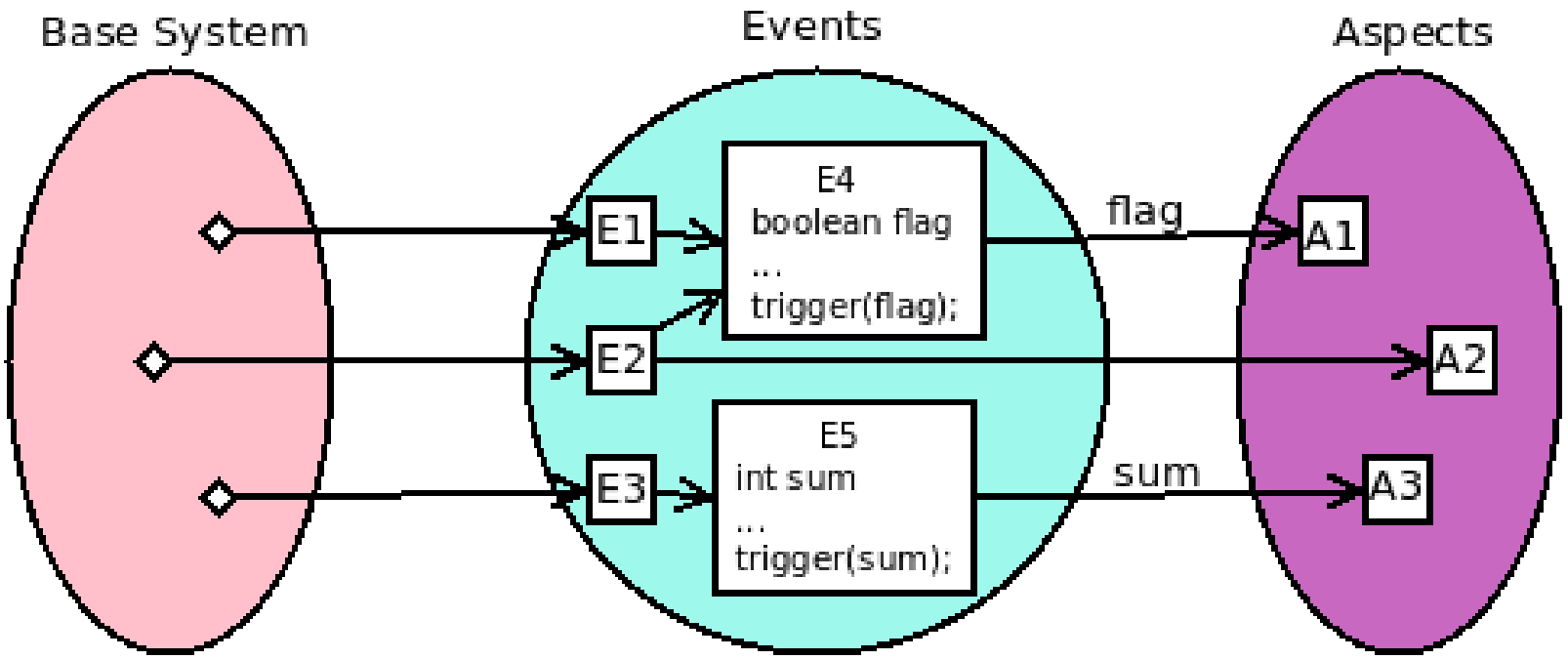
Pointcuts determine the places where aspects should be woven.
Aspect advices consist of the combination of Pointcut + Response.

Motivation for change

Problems of the previous model:

- Aspect advices include both when they should be woven, and what to do. This reduces modularity and reuse.
- Pointcuts express the current state of the system, including the current call stack. However, pointcuts cannot directly express a sequence of events that have occurred.
- The parameters exposed by a pointcut may only be: arguments, target and the current aspect. No other information can be exposed.
- Aspects are directly related to low level events in the code, instead of considering an abstraction of when they should respond.

Events and Observers



Events [1] collect information and detect interesting situations where aspects should be woven.

Events can be composed of lower level events.

Observer aspects collect information in the base system, and they may afterwards interact with other aspects.

[1] will be presented in Modularity Visions Track - on Wednesday.

Example

```
event LowActivity(P product, int purchases){
  int LOWER_BOUND = 100;
  Info purchaseInfo = new Info();
  after(Purchase purchase): RelevantPurchase(purchase){
    purchaseInfo.increase(purchase.product());
  }
  when(P product): call(P.timeDone()) && target(product) {
    if (purchaseInfo.count(product) < LOWER_BOUND) {
      trigger(product, purchaseInfo.count(product));
    }
    purchaseInfo.reset(product);
  }
}
```

RelevantPurchase is a lower level event.

Goal

The goal is to present a modular verification procedure for events, which takes advantage of the fact that they **do not change the base system** in their execution. Their guarantees must be strong enough to be used by higher level events and aspects. A clear way to specify event assumptions and guarantees should be found.

Specification

For every event, we want to verify two things:

1. The event is triggered in the correct places.
2. The event's exposed information is correct.

Note: The assumption that events do not change the base system can be checked syntactically, so we do not need to prove that using formal methods such as model checking.

Specification of LowActivity

The specification is based in the *assume-guarantee* model.

- Assumption: every period for considering the activity of a product ends, i.e. always in the future *timeDone* will occur : GF*timeDone* .
- Assumption on used event detectors: *RelevantPurchase* is triggered when appropriate and exposes the needed information.

Specification of LowActivity - Guarantee

Intuitively the guarantee of LowActivity is simple: The event is triggered if and only if the sum of relevant purchases for a product in a certain period is lower than the lower bound.
However, this property gets hard to express in temporal logic, given that each possible sequence of relevant purchases between 0 and 99 must be considered to express that the event is triggered.
Taking a smaller model in which the lower bound $= 3$, we define the formula that expresses that the event is detected if and only if a period has finished and there have been at most two relevant purchases.

Example Temporal Logic

- Guarantee:

$$\mathsf{G}(\mathit{lowActivity\_trigger} \implies \mathit{timeDone}) \wedge \quad (1)$$

$$\mathsf{G}(((\mathbf{start} \vee \mathbf{timeDone}) \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur})\mathsf{U}(\mathbf{timeDone} \wedge \mathbf{lowActivity\_trigger}))) \quad (2)$$

$$\vee ((\mathbf{start} \vee \mathbf{timeDone}) \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur})\mathsf{U}(\quad (3)$$

$$\mathbf{relPur} \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur})\mathsf{U}(\mathbf{timeDone} \wedge \mathbf{lowActivity\_trigger}))))) \vee \quad (4)$$

$$((\mathbf{start} \vee \mathbf{timeDone}) \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur})\mathsf{U}(\mathbf{relPur} \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur}) \quad (5)$$

$$\mathsf{U}(\mathbf{relPur} \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur})\mathsf{U}(\mathbf{timeDone} \wedge \mathbf{lowActivity\_trigger}))))))) \vee \quad (6)$$

$$(\mathbf{relPur} \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur})\mathsf{U}(\mathbf{relPur} \wedge \mathsf{X}((\neg \mathit{timeDone} \wedge \neg \mathit{relPur})\mathsf{U}(\quad (7)$$

$$\mathbf{relPur} \wedge \mathsf{X}(\neg \mathit{timeDone}\mathsf{U}(\mathbf{timeDone} \wedge \neg \mathbf{lowActivity\_trigger}))))))))) \quad (8)$$

Example Temporal Logic

The formula expresses:

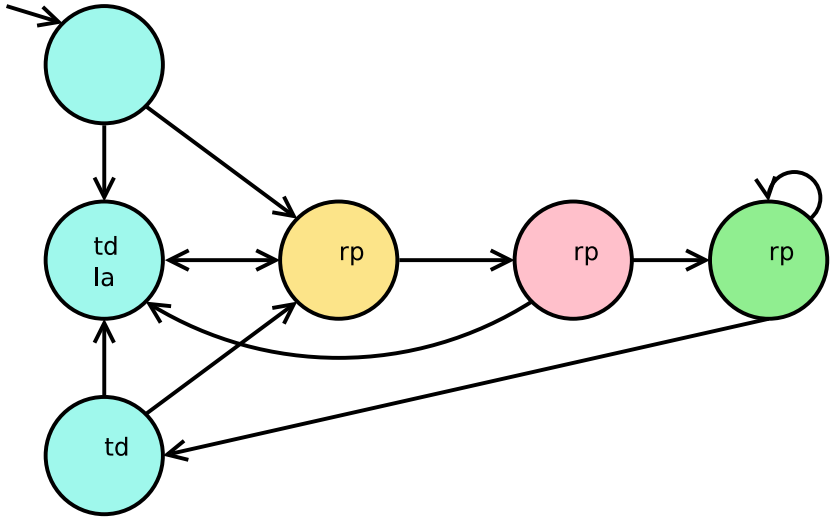
- line 1 The event may only be triggered when a period has finished (*timeDone*).
- line 2 When no relevant purchase has been made in a period the event is triggered.
- lines 3-4 When exactly one relevant purchase has been made, the event is triggered.
- lines 5-6 When exactly two relevant purchases have been made, the event is triggered.
- lines 7-8 When three or more relevant purchases have been made, the event is not triggered.

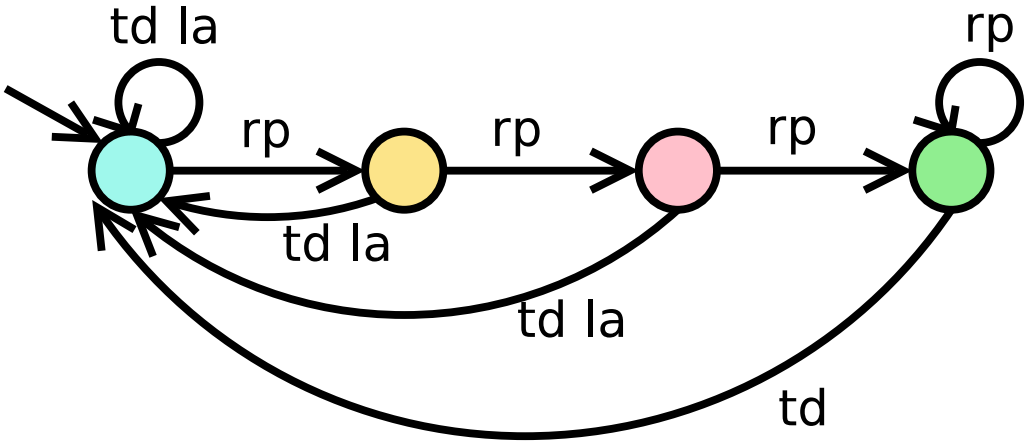
Other specification languages

Expressing exactly when an event is detected requires analyzing every possible sequence of lower level events.
Temporal logic, though precise, becomes quickly unreadable.
State machines, automatas, regular expressions, Kripke models, Moore machines serve better for this goal.

Examples

- rp: RelevantPurchase
- td: timeDone
- la: lowActivity_trigger



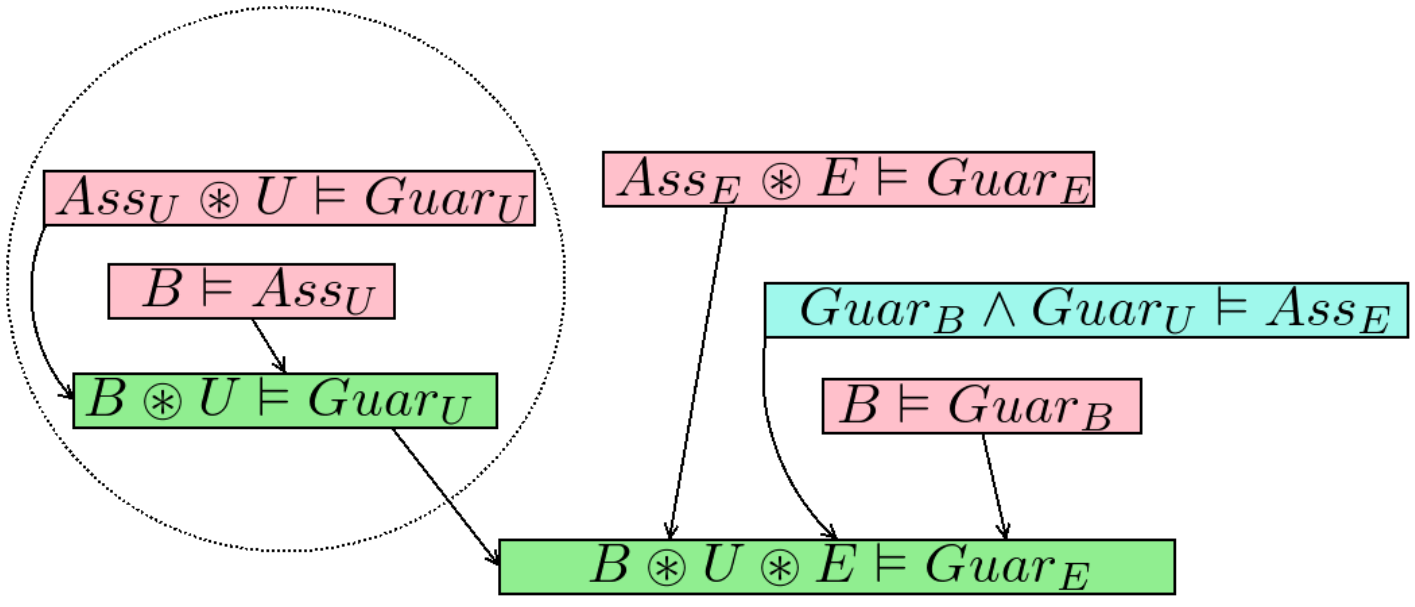


Verification Process

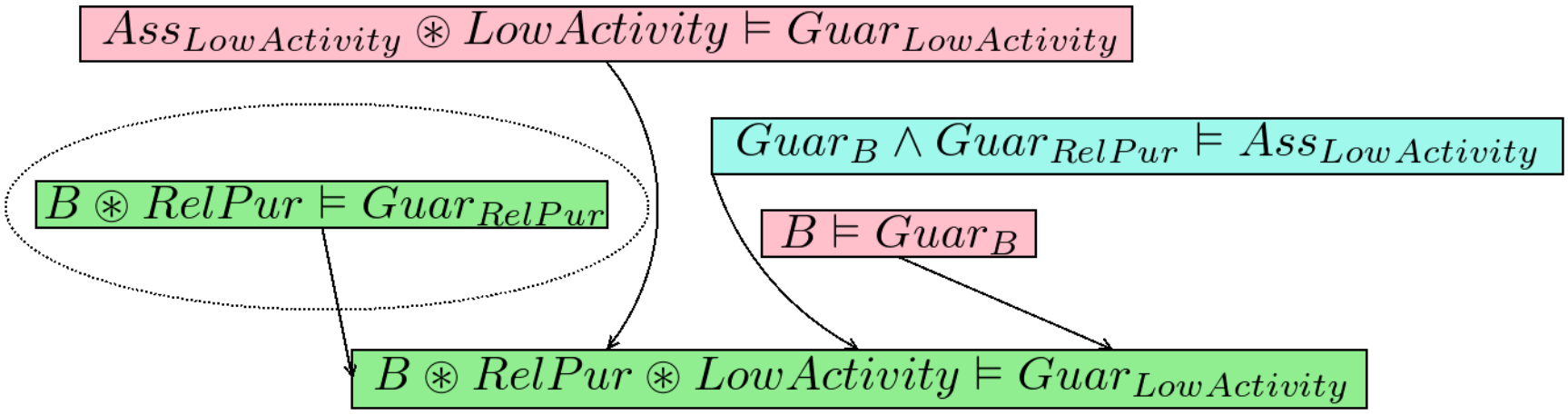
B: Base System

E: Event to be verified

U: E's used event declarations



Verification Process - LowActivity



Verification Process - LowActivity

To build a library of events, and prove that *LowActivity* is correct:

1. We consider that the assumption holds:
   $B \circledast RelPur \vDash Guar_{RelPur}$
2. Prove the correctness of the event:
   $Ass_{LowActivity} \circledast LowActivity \vDash Guar_{LowActivity}$
3. Verify that the guarantee of the used event detectors ($\{RelevantPurchase\}$), together with some guarantee of the base system is enough for the assumption of *LowActivity* to hold.

When the event is to be woven to a base system B:

1. Prove that the base system satisfies $Guar_B$

Conclusion: $B \circledast RelPur \circledast LowActivity \vDash Guar_{LowActivity}$

Events and Aspects

Events are easier to prove than aspects but still not trivial.
Given that during event evaluation the base system is guaranteed not to change, event evaluation can be considered as immediate.
Event guarantees are essential for showing aspects correct.

Example

Applying discounts based on *LowActivity*. The aspect that decides when to apply this discount must assume that the event is correct, and consider other conflicting discounts.

In ACM Digital Library and/or in the full version of the article: This example and another example of how the number of occurrences of commit is detected (using composition) are presented in detail.

Future work

- Combining Event verification with general Aspect verification.
- Finding expressive and natural specification languages and tools.

Conclusions

Events must be proven to be correct, so that other events and aspects can use their guarantees.
A modular verification method has been presented, where instead of weaving all the components and verifying correctness, the assume-guarantee model is used.
Alternative ways for specifying events have been demonstrated.
Temporal logic, Moore machines, Kripke models, etc.

Questions?