JML-based Verification of Liveness Properties on a Class in Isolation

SAVCBS 2006

Julien Groslambert Jacques Julliand Olga Kouchnarenko

November 10-11th 2006
Portland, Oregon.

LIFC - University of Franche-Conté

Motivations

Formal Verification of Conformity between

- Requirements
- Implementation source code

Requirements $\xleftrightarrow{\text{Conformity ?}}$ Implementation

Requirements

- Absence of null pointer exception.
- Class Invariance.
- Temporal behavior.

Leavens and Al.
Annotations for Java.
Well tool supported.

Temporal properties

+ are expressible in JML.
− need a tedious work for annotating.

Motivations

Formal Verification of Conformity between

- Requirements
- Implementation source code

Requirements ⟷ Conformity ? ⟷ Implementation

Requirements

- Absence of null pointer exception.
- Class Invariance.
- Temporal behavior.

JML Annotations

- Leavens and Al.
- Annotations for Java.
- Well tool supported.

Temporal properties

+ are expressible in JML.
− need a tedious work for annotating.

Motivations

Requirements

- Absence of null pointer exception.
- Class Invariance.
- Temporal behavior.

JML Annotations

- Leavens and Al.
- Annotations for Java.
- Well tool supported.

Temporal properties

+ are expressible in JML.
– need a tedious work for annotating.

⇒ **Automatic annotation generation** from high level temporal properties.

Motivations

Requirements

- Absence of null pointer exception.
- Class Invariance.
- Temporal behavior.

JML Annotations

- Leavens and Al.
- Annotations for Java.
- Well tool supported.

Temporal properties

+ are expressible in JML.
− need a tedious work for annotating.

⇒ **Automatic annotation generation** from high level temporal properties.

Motivations

Requirements

- Absence of null pointer exception.
- Class Invariance.
- Temporal behavior.

JML Annotations

- Leavens and Al.
- Annotations for Java.
- Well tool supported.

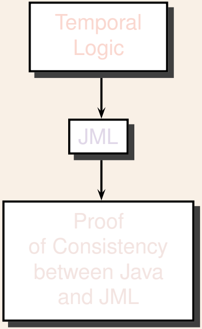
Temporal properties

+ are expressible in JML.
– need a tedious work for annotating.

⇒ **Automatic annotation generation** from high level temporal properties.

Proposed Approach - Huisman Trentelman [AMAST'02]

1. Expressing **security properties** from requirements
2. Translating properties into a **annotation language** for the implementation
3. Verifying the properties on the **code**

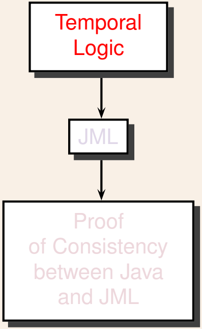


Focus of the talk

Extension of the approach to Liveness Properties

Proposed Approach - Huisman Trentelman [AMAST'02]

1. Expressing **security properties** from requirements
2. Translating properties into a **annotation language** for the implementation
3. Verifying the properties on the **code**

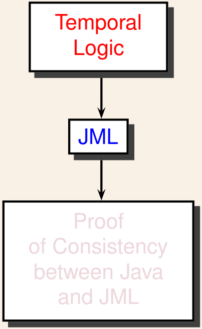


Focus of the talk

Extension of the approach to Liveness Properties

Proposed Approach - Huisman Trentelman [AMAST'02]

1. Expressing **security properties** from requirements
2. Translating properties into a **annotation language** for the implementation
3. Verifying the properties on the **code**

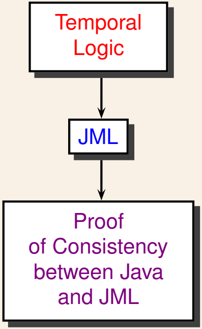


Focus of the talk

Extension of the approach to Liveness Properties

Proposed Approach - Huisman Trentelman [AMAST'02]

1. Expressing **security properties** from requirements
2. Translating properties into a **annotation language** for the implementation
3. Verifying the properties on the **code**

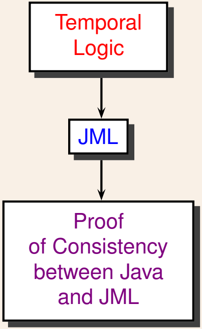


Focus of the talk

Extension of the approach to Liveness Properties

Proposed Approach - Huisman Trentelman [AMAST'02]

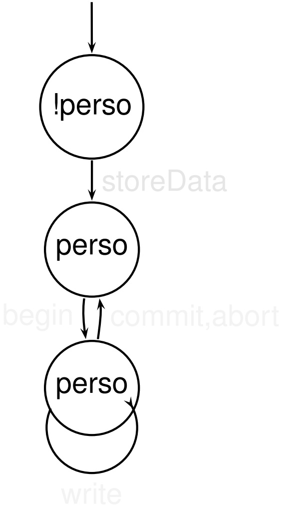
1. Expressing **security properties** from requirements
2. Translating properties into a **annotation language** for the implementation
3. Verifying the properties on the **code**



Focus of the talk

Extension of the approach to Liveness Properties

Running Example: A Transaction System

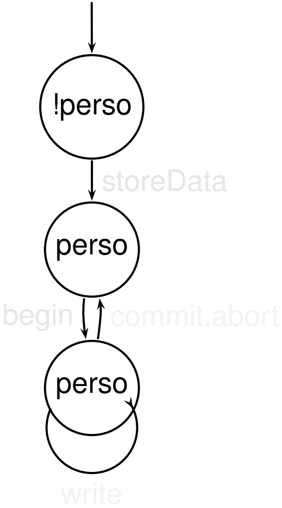


Behavior

Two steps:

- Personalization.
  - storeData: fix the size of the buffer.
- Use.
  - begin a transaction.
  - write a modification.
  - commit transaction.
  - abort transaction.

Running Example: A Transaction System

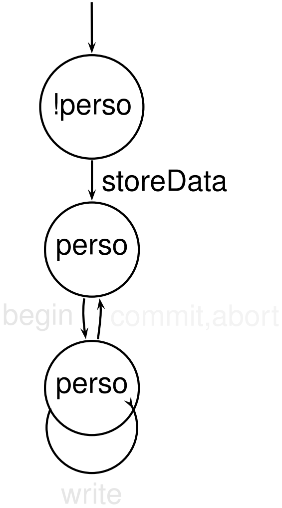


Behavior

Two steps:

- Personalization.
  - storeData: fix the size of the buffer.
- Use.
  - begin a transaction.
  - write a modification.
  - commit transaction.
  - abort transaction.

Running Example: A Transaction System

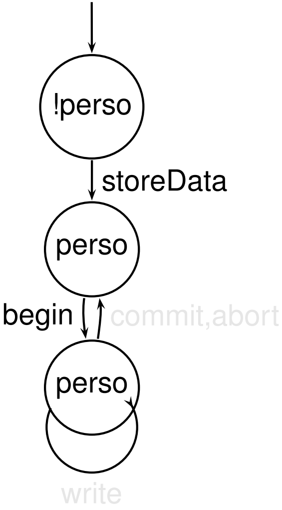


Behavior

Two steps:

- Personalization.
  - storeData: fix the size of the buffer.
- Use.
  - begin a transaction.
  - write a modification.
  - commit transaction.
  - abort transaction.

Running Example: A Transaction System

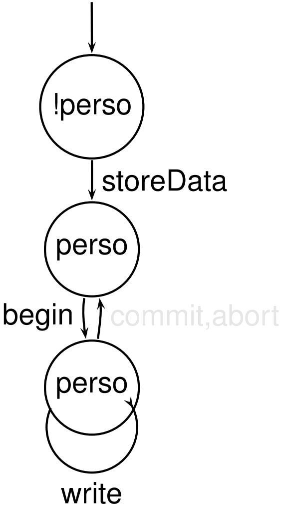


Behavior

Two steps:

- Personalization.
  - storeData: fix the size of the buffer.
- Use.
  - begin a transaction.
  - write a modification.
  - commit transaction.
  - abort transaction.

Running Example: A Transaction System

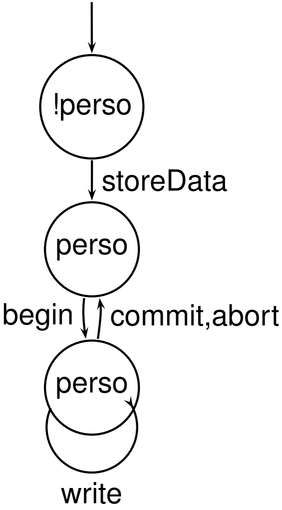


Behavior

Two steps:

- Personalization.
  - storeData: fix the size of the buffer.
- Use.
  - begin a transaction.
  - write a modification.
  - commit transaction.
  - abort transaction.

Running Example: A Transaction System



Behavior

Two steps:

- Personalization.
  - storeData: fix the size of the buffer.
- Use.
  - begin a transaction.
  - write a modification.
  - commit transaction.
  - abort transaction.

Running Example: A Transaction System

Fields

`perso`: boolean describing if the card is already personalized.

`len`: Integer representing the length of the Buffer.

`status`: byte array specifying the status of the system.

`buffer`: byte array specifying a temporal buffer.

`position`: integer representing the current position in the Buffer.

`trDepth`: boolean describing if there is a current transaction.

Example

```
public class Buffer {
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
... }
```

Running Example: A Transaction System

Fields

`perso`: boolean describing if the card is already personalized.

`len`: Integer representing the length of the Buffer.

`status`: byte array specifying the status of the system.

`buffer`: byte array specifying a temporal buffer.

`position`: integer representing the current position in the Buffer.

`trDepth`: boolean describing if there is a current transaction.

Example

```
public class Buffer {
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
... }
```

Running Example: A Transaction System

Fields

`perso`: boolean describing if the card is already personalized.

`len`: Integer representing the length of the Buffer.

`status`: byte array specifying the status of the system.

`buffer`: byte array specifying a temporal buffer.

`position`: integer representing the current position in the Buffer.

`trDepth`: boolean describing if there is a current transaction.

Example

```
public class Buffer {
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
... }
```

Running Example: A Transaction System

Fields

`perso`: boolean describing if the card is already personalized.
`len`: Integer representing the length of the Buffer.
`status`: byte array specifying the status of the system.
`buffer`: byte array specifying a temporal buffer.
`position`: integer representing the current position in the Buffer.
`trDepth`: boolean describing if there is a current transaction.

Example

```
public class Buffer {
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
... }
```

Running Example: A Transaction System

Fields

`perso`: boolean describing if the card is already personalized.
`len`: Integer representing the length of the Buffer.
`status`: byte array specifying the status of the system.
`buffer`: byte array specifying a temporal buffer.
`position`: integer representing the current position in the Buffer.
`trDepth`: boolean describing if there is a current transaction.

Example

```
public class Buffer {
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
... }
```

Running Example: A Transaction System

Fields

`perso`: boolean describing if the card is already personalized.

`len`: Integer representing the length of the Buffer.

`status`: byte array specifying the status of the system.

`buffer`: byte array specifying a temporal buffer.

`position`: integer representing the current position in the Buffer.

`trDepth`: boolean describing if there is a current transaction.

Example

```
public class Buffer {
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
... }
```

Running Example: A Transaction System

Fields

`perso`: boolean describing if the card is already personalized.
`len`: Integer representing the length of the Buffer.
`status`: byte array specifying the status of the system.
`buffer`: byte array specifying a temporal buffer.
`position`: integer representing the current position in the Buffer.
`trDepth`: boolean describing if there is a current transaction.

Example

```
public class Buffer {
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
... }
```

Running Example: A Transaction System

Methods

storeData to personalize the Transaction System.
begin to start a new transaction.
write to write in the current Buffer.
getBufferLess to get the Buffer free place
commit to valid the current transaction.
getStatus to get the current status of the transaction.
abort to abort the current transaction.

Example

```
public class Buffer {
...
 void storeData(int l){
  len = l;
  perso = true;
}
void begin()
    throws Exception{
  if (perso == false) {
    throw new Exception();
  }
  buffer = new byte[len];
  trDepth = true;
  }
...
```

Running Example: A Transaction System

Methods

`storeData` to personalize the Transaction System.

`begin` to start a new transaction.

`write` to write in the current Buffer.

`getBufferLess` to get the Buffer free place

`commit` to valid the current transaction.

`getStatus` to get the current status of the transaction.

`abort` to abort the current transaction.

Example

```
public class Buffer {
...
   void storeData(int l){
    len = l;
    perso = true;
  }
void begin()
    throws Exception{
  if (perso == false) {
    throw new Exception();
  }
  buffer = new byte[len];
  trDepth = true;
  }
...
```

Running Example: A Transaction System

Methods

`storeData` to personalize the Transaction System.

`begin` to start a new transaction.

`write` to write in the current Buffer.

`getBufferLess` to get the Buffer free place

`commit` to valid the current transaction.

`getStatus` to get the current status of the transaction.

`abort` to abort the current transaction.

Example

```
public class Buffer {
...
 void storeData(int l){
  len = l;
  perso = true;
}
 void begin()
     throws Exception{
   if (perso == false) {
     throw new Exception();
   }
   buffer = new byte[len];
   trDepth = true;
   }
...
```

Running Example: A Transaction System

Methods

`storeData` to personalize the Transaction System.

`begin` to start a new transaction.

`write` to write in the current Buffer.

`getBufferLess` to get the Buffer free place

`commit` to valid the current transaction.

`getStatus` to get the current status of the transaction.

`abort` to abort the current transaction.

Example

```
public class Buffer {
...

 void write(byte b){
buffer[position] = b;
position++;
}

int getBufferLess(){
return len
    - buffer.length;
}
```

Running Example: A Transaction System

Methods

`storeData` to personalize the Transaction System.

`begin` to start a new transaction.

`write` to write in the current Buffer.

`getBufferLess` to get the Buffer free place

`commit` to valid the current transaction.

`getStatus` to get the current status of the transaction.

`abort` to abort the current transaction.

Example

```
public class Buffer {
...

 void write(byte b){
buffer[position] = b;
position++;
}

int getBufferLess(){
return len
    - buffer.length;
}
```

Running Example: A Transaction System

Methods

`storeData` to personalize the Transaction System.

`begin` to start a new transaction.

`write` to write in the current Buffer.

`getBufferLess` to get the Buffer free place

`commit` to valid the current transaction.

`getStatus` to get the current status of the transaction.

`abort` to abort the current transaction.

Example

```
public class Buffer {
...

  void commit(){
status = buffer;
position = 0;
trDepth = false;
}
byte [] getStatus(){
return status;
}
void abort(){
position = 0;
trDepth = false;
}
```

Running Example: A Transaction System

Methods

`storeData` to personalize the Transaction System.

`begin` to start a new transaction.

`write` to write in the current Buffer.

`getBufferLess` to get the Buffer free place

`commit` to valid the current transaction.

`getStatus` to get the current status of the transaction.

`abort` to abort the current transaction.

Example

```
public class Buffer {
...

  void commit(){
status = buffer;
position = 0;
trDepth = false;
}
byte [] getStatus(){
return status;
}
void abort(){
position = 0;
trDepth = false;
}
```

Running Example: A Transaction System

Methods

`storeData` to personalize the Transaction System.

`begin` to start a new transaction.

`write` to write in the current Buffer.

`getBufferLess` to get the Buffer free place

`commit` to valid the current transaction.

`getStatus` to get the current status of the transaction.

`abort` to abort the current transaction.

Example

```
public class Buffer {
...

  void commit(){
status = buffer;
position = 0;
trDepth = false;
}
byte [] getStatus(){
return status;
}
void abort(){
position = 0;
trDepth = false;
}
```

JML Class Specification

JML Annotations

Example

```
public class Buffer {
        ...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Example

```
public class Buffer {
        ...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Example

```
public class Buffer {
//@ invariant position >= 0;
       ...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Invariant

Properties that have to be true in all *visible* states:

- Before invocation of a method
- After invocation of a method

Example

```
public class Buffer {
//@ invariant position >= 0;
       ...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Example

```
public class Buffer {
//@ invariant position >= 0;
      ...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Constraint

Property linking two visible states.

- `old` keyword.
- `for` keyword.

Example

```
public class Buffer {
//@ invariant position >= 0;
/*@ contraint perso ==>
 @ len == \old(len);
 @*/
    ...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Constraint

Property linking two visible states.

- `old` keyword.
- `for` keyword.

Example

```
public class Buffer {
//@ invariant position >= 0;
/*@ contraint perso ==>
@ len == \old(len);
@*/
/*@ contraint
 @ position >= \old(position);
 @ for write;
 @*/
...
boolean perso = false;
int len;
byte [] status;
byte [] buffer;
int position = 0;
boolean trDepth = false;
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Method Specification

1. Precondition.
2. Postcondition.
3. Exceptional Postcondition.
4. Frame Condition.

Example

```
void begin() throws Exception{
...
}
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Method Specification

1. Precondition.
2. Postcondition.
3. Exceptional Postcondition.
4. Frame Condition.

Example

```
/*@
@ requires trDepth == false;
@ requires perso == true;
@*/
void begin() throws Exception{
...
}
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Method Specification

1. Precondition.
2. Postcondition.
3. Exceptional Postcondition.
4. Frame Condition.

Example

```
/*@
@ requires trDepth == false;
@ requires perso == true;
@ ensures trDepth == true;
@*/
void begin() throws Exception{
...
}
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Method Specification

1. Precondition.
2. Postcondition.
3. Exceptional Postcondition.
4. Frame Condition.

Example

```
/*@ normal_behavior
@ requires trDepth == false;
@ requires perso == true;
@ ensures trDepth == true;
@ also
@ exceptional_behavior
@ requires perso == false;
@ signals (Exception e) true;
@*/
void begin() throws Exception{
...
}
```

JML Class Specification

JML Annotations

Main JML Annotations.

1. Class invariant specification.
2. History constraint specification.
3. Method Specification.

Method Specification

1. Precondition.
2. Postcondition.
3. Exceptional Postcondition.
4. Frame Condition.

Example

```
/*@ normal_behavior
@ requires trDepth == false;
@ requires perso == true;
@ assignable buffer;
@ ensures trDepth == true;
@ also
@ exceptional_behavior
@ requires perso == false;
@ assignable \nothing;
@ signals (Exception e) true;
@*/
void begin() throws Exception{
...
}
```

Modular Reasonning

Reasonning Modularly consists in

1. Establishing a property of a class in isolation assuming some hypothesis of the program using the class.
2. Verifying the hypothesis on the program.

Java Class

Java Program
Using
the class

Modular Reasonning

Reasonning Modularly consists in

1. Establishing a property of a class in isolation assuming some hypothesis of the program using the class.
2. Verifying the hypothesis on the program.

Java Class

Java Program
Using
the class

Verification on isolation

Hypothesis

Reasonning Modularly consists in

1. Establishing a property of a class in isolation assuming some hypothesis of the program using the class.
2. Verifying the hypothesis on the program.

Java Class

Java Program Using the class

Verification of the Hypothesis

Modular Reasonning - Example: Method Correctness

Design by contract approach.

Java Class Contract

- Assumes the Precondition.
- Establishes the Postcondition.

Example

```
/*@ requires trDepth;
  @ requires buffer != null;
  @ requires position > 0;
  @ requires position < buffer.length;
  @ ensures position == \old(position)+1;
  @*/
void write(byte b){
buffer[position] = b;
position++;
}
```

Java Program using the class

- Assumes the Postcondition.
- Establishes the Precondition.

Example

```
a.storeDate(4);
a.begin();
// assert precondition
a.write(7);
// assume postcondition
```

Temporal Properties

Examples of Temporal Requirements for the Buffer

1. The application can be personalized only once.
2. The status is always the same unless a `commit` happens.
3. A `begin` must inevitably been followed by a `commit` or an `abort`.

Temporal Properties

Examples of Temporal Requirements for the Buffer

1. The application can be personalized only once.
2. The status is always the same unless a `commit` happens.
3. A `begin` must inevitably been followed by a `commit` or an `abort`.

Temporal Properties

Examples of Temporal Requirements for the Buffer

1. The application can be personalized only once.
2. The status is always the same unless a `commit` happens.
3. A `begin` must inevitably been followed by a `commit` or an `abort`.

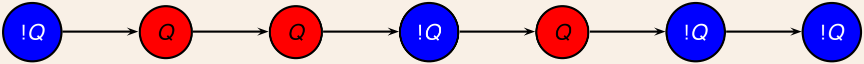
A Loop Modality for Expressing Liveness Properties

Definition (Loop Primitive)

Loop(Q) A state where Q is satisfied must be inevitably followed by a state where Q is not satisfied.

$$\forall i.((i \geq 0 \land \sigma_i \models Q) \;\Rightarrow\; (\exists j.j > i \land \sigma_j \models \neg Q)).$$

Illustration



Example

A `begin` must inevitably been followed by a `commit` or an `abort`.

$$\text{Loop}(\texttt{TrDepth == true})$$

Modular Reasonning - Application to liveness properties

Java Class contract

- Assumes a Progress Hypothesis
- Establishes the Liveness on a class in isolation.

Java Program Contract

- Assumes the liveness on a class in Isolation.
- Establishes a Progress Hypothesis.

Satisfaction of the liveness by the whole program.

Progress Hypothesis

Definition (Progress Hypothesis $PH(Q, M)$)

$$(\mathsf{F}^{\infty}\mathrm{pre}(M)) \vee (\mathsf{G}^{\infty}\neg Q)$$

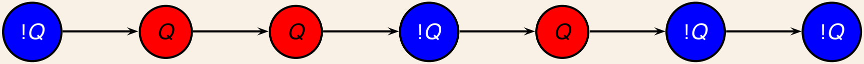
where $\mathrm{pre}(M)$ denotes the predicate $\bigvee_{m \in M} \mathrm{pre}(m)$.

Progress methods are infinitely often called.
The program stay in a state satisfying $\neg Q$.

Variant Introduction

Need a variant V like a proof termination.

- Given by the user
- Well founded $\Rightarrow$ Expression from a subset of the class variables to the positive integers
- must decrease for each method invocation until Q



Example

Loop(`TrDepth`) The variant V is `getBufferLess()`.

Annotation for Loop

Annotations

$$\texttt{//@ invariant } V \texttt{ >= 0;} \qquad (\mathcal{A}_1)$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{ < \textbackslash old(}V\texttt{) for } M; \qquad (\mathcal{A}_2)$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{ <= \textbackslash old(}V\texttt{) ;} \qquad (\mathcal{A}_3)$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigvee_{m \in M} \text{requires}(m) \qquad (\mathcal{A}_4)$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigwedge_{m \in M_C} (\text{requires}(m) \texttt{ ==> } !\text{diverges}(m)); \qquad (\mathcal{A}_5)$$

Example (Loop(`TrDepth`))

Annotation for Loop

Well-foundation of V

`//@ invariant` V `>= 0;` $(\mathcal{A}_1)$

`//@ constraint` Q `==>` V `< \old(`V`) for` M; $(\mathcal{A}_2)$

`//@ constraint` Q `==>` V `<= \old(`V`) ;` $(\mathcal{A}_3)$

`//@ invariant` Q `==>` $\bigvee_{m \in M} \mathrm{requires}(m)$ $(\mathcal{A}_4)$

`//@ invariant` Q `==>` $\bigwedge_{m \in M_C} (\mathrm{requires}(m) \Longrightarrow !\mathrm{diverges}(m));$ $(\mathcal{A}_5)$

Example (Loop(`TrDepth`))

```
//@ invariant getBufferLess() >= 0
```

Annotation for Loop

Each progress method decreases V

$$\texttt{//@ invariant } V \texttt{ >= 0;} \qquad (\mathcal{A}_1)$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{ < \textbackslash old(} V \texttt{) for } M; \qquad (\mathcal{A}_2)$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{ <= \textbackslash old(} V \texttt{) ;} \qquad (\mathcal{A}_3)$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigvee_{m \in M} \text{requires}(m) \qquad (\mathcal{A}_4)$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigwedge_{m \in M_C} (\text{requires}(m) \text{ ==> !diverges}(m)); \qquad (\mathcal{A}_5)$$

Example (Loop(`TrDepth`))

```
/*@ constraint trDepth
 @ ==>  getBufferLess() <\ old(getBufferLess()) for
storeData,begin,abort,commit,write;
```

Annotation for Loop

Each method does not increase V

$$
\begin{array}{ll}
\texttt{//@ invariant } V \texttt{ >= 0;} & (\mathcal{A}_1) \\
\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{ < \textbackslash old(} V \texttt{) for } M; & (\mathcal{A}_2) \\
\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{ <= \textbackslash old(} V \texttt{) ;} & (\mathcal{A}_3) \\
\texttt{//@ invariant } Q \texttt{ ==> } \bigvee_{m \in M} \text{requires}(m) & (\mathcal{A}_4) \\
\texttt{//@ invariant } Q \texttt{ ==> } \bigwedge_{m \in M_C} (\text{requires}(m) \texttt{ ==> } !\text{diverges}(m)); & (\mathcal{A}_5)
\end{array}
$$

Example (Loop(`TrDepth`))

```
//@ constraint trDepth
 @ ==>  getBufferLess() <= \ old(getBufferLess()) ;
```

Annotation for Loop

No dead-lock for the class

$$\texttt{//@ invariant } V \texttt{ >= 0;} \tag{$\mathcal{A}_1$}$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{< \textbackslash old(}V\texttt{) for } M; \tag{$\mathcal{A}_2$}$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{<= \textbackslash old(}V\texttt{) ;} \tag{$\mathcal{A}_3$}$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigvee_{m \in M} \text{requires}(m) \tag{$\mathcal{A}_4$}$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigwedge_{m \in M_C} (\text{requires}(m) \texttt{ ==> } !\text{diverges}(m)); \tag{$\mathcal{A}_5$}$$

Example (Loop(`TrDepth`))

```
/*@ invariant trDepth ==> ( perso == false ||
  @ (trDepth == false && perso == true) ||
  @ (trDepth == true && perso == true
  @ && position < len)) @*/
```

Annotation for Loop

No divergence for the class

$$\texttt{//@ invariant } V \texttt{ >= 0;} \qquad (\mathcal{A}_1)$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{< \textbackslash old(}V\texttt{) for } M; \qquad (\mathcal{A}_2)$$

$$\texttt{//@ constraint } Q \texttt{ ==> } V \texttt{<= \textbackslash old(}V\texttt{) ;} \qquad (\mathcal{A}_3)$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigvee_{m \in M} \text{requires}(m) \qquad (\mathcal{A}_4)$$

$$\texttt{//@ invariant } Q \texttt{ ==> } \bigwedge_{m \in M_C} (\text{requires}(m) \texttt{ ==> } !\text{diverges}(m)); \qquad (\mathcal{A}_5)$$

Example (Loop(`TrDepth`))

obvious Annotation

Soundness of the method

Theorem

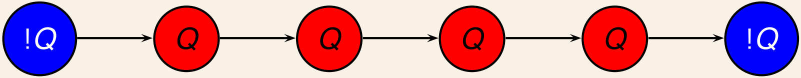
$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Intuition.



- $\mathcal{A}_4$. No dead-lock for the class
- $\mathcal{A}_5$. No divergence for the class
- $\mathcal{A}_2$. Each progress method decreases V
- $\mathcal{A}_3$. Each method does not increase V
- Decrease of the variant
- $\mathcal{A}_1$.Well-foundation of V

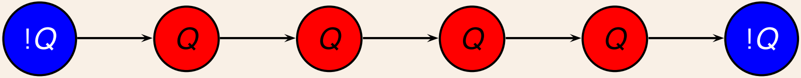
□

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Intuition.



- $\mathcal{A}_4$. No dead-lock for the class
- $\mathcal{A}_5$. No divergence for the class
- $\mathcal{A}_2$. Each progress method decreases V
- $\mathcal{A}_3$. Each method does not increase V
- Decrease of the variant
- $\mathcal{A}_1$.Well-foundation of V

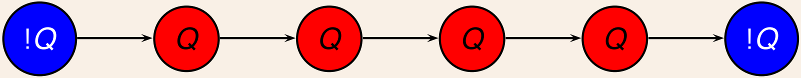
□

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Intuition.



- $\mathcal{A}_4$. No dead-lock for the class
- $\mathcal{A}_5$. No divergence for the class
- $\mathcal{A}_2$. Each progress method decreases V
- $\mathcal{A}_3$. Each method does not increase V
- Decrease of the variant
- $\mathcal{A}_1$.Well-foundation of V

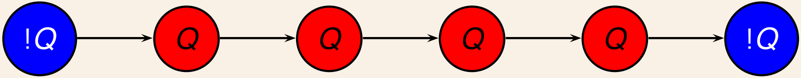
□

Soundness of the method

Theorem

If $C : \mathcal{A}_{1-5} \wedge PH(Q, M)$ then $Loop(Q)$.

Intuition.



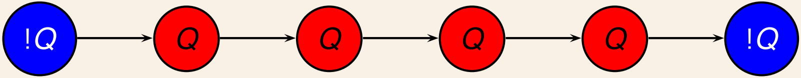
- $\mathcal{A}_4$. No dead-lock for the class
- $\mathcal{A}_5$. No divergence for the class
- $\mathcal{A}_2$. Each progress method decreases V
- $\mathcal{A}_3$. Each method does not increase V
- Decrease of the variant
- $\mathcal{A}_1$.Well-foundation of V

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Intuition.



- $\mathcal{A}_4$. No dead-lock for the class
- $\mathcal{A}_5$. No divergence for the class
- $\mathcal{A}_2$. Each progress method decreases V
- $\mathcal{A}_3$. Each method does not increase V
- Decrease of the variant
- $\mathcal{A}_1$.Well-foundation of V

□

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Intuition.

$$!Q \longrightarrow Q \longrightarrow Q \longrightarrow Q \longrightarrow Q \longrightarrow !Q$$

- $\mathcal{A}_4$. No dead-lock for the class
- $\mathcal{A}_5$. No divergence for the class
- $\mathcal{A}_2$. Each progress method decreases V
- $\mathcal{A}_3$. Each method does not increase V
- Decrease of the variant
- $\mathcal{A}_1$.Well-foundation of V

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Proof.

PH $\quad C :: \mathcal{A}_{1-5} \quad$ Loop(Q)

□

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

Proof.

	PH	$C :: \mathcal{A}_{1-5}$	Loop(Q)
Trace Semantics	Σ_{PH}	$\Sigma_{\mathcal{A}_{1-5}}$	$\Sigma_{Loop(Q)}$

$\square$

Soundness of the method

Theorem

$$\text{If} \quad C : \mathcal{A}_{1-5} \wedge PH(Q, M) \quad \text{then} \, Loop(Q).$$

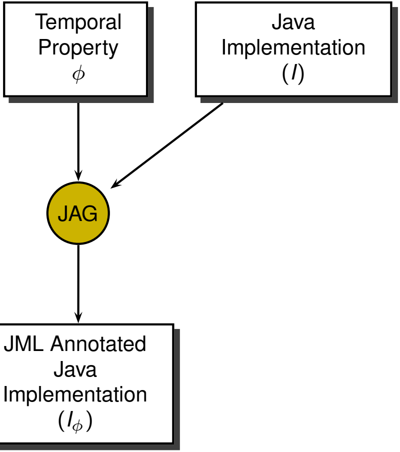
Proof.

	PH	$\wedge\ C :: \mathcal{A}_{1-5}$	$\Rightarrow$ Loop(Q)
Trace Semantics	Σ_{PH}	$\cap\ \Sigma_{\mathcal{A}_{1-5}}$	$\subseteq \Sigma_{Loop(Q)}$

□

JAG: General Approach

- JAG 0.1 input: JTPL Properties [Huisman Trentelman - AMAST'02].
- Generates JML annotations ensuring the JTPL properties.
- Annotations traceability



JAG: Example of JTPL Properties

Example (Unique personalization)

after storeData **normal**
  always (perso == true
    and storeData **not enabled**)

Example (Command not enabled before personnalization)

always (begin **not enabled**
  and commit **not enabled**)
    unless storeData **normal**

Example (Begin eventually followed by Commit or Abort)

after begin **normal**
  (**eventually** commit **called**),
    abort **called**)

JAG: Example of JTPL Properties

Example (Unique personalization)

after storeData **normal**
  always (perso == true
    and storeData **not enabled**)

Example (Command not enabled before personnalization)

always (begin **not enabled**
  and commit **not enabled**)
    unless storeData **normal**

Example (Begin eventually followed by Commit or Abort)

after begin **normal**
  (**eventually** commit **called**),
    abort **called**)

JAG: Annotation Generation

Example (JTPL Property)

after begin **normal**
(**eventually** commit **called**),
abort **called**)

Annotation Generation

- Declaration of a ghost variable.
- Assignement of the ghost variable.
- Loop(`Witness`).

Example (Java Code)

```
public class Buffer {
//@ ghost witness = false;
boolean perso = false;
...
void begin(){
...
//@ set witness = true;
}
void commit(){
//@ set witness = false;
...
}
void abort(){
//@ set witness = false;
...
}
```

JAG: Annotation Generation

Example (JTPL Property)

after begin **normal**
(**eventually** commit **called**),
abort **called**)

Annotation Generation

- Declaration of a ghost variable.
- Assignement of the ghost variable.
- Loop(`Witness`).

Example (Java Code)

```
public class Buffer {
//@ ghost witness = false;
boolean perso = false;
...
void begin(){
...
//@ set witness = true;
}
void commit(){
//@ set witness = false;
...
}
void abort(){
//@ set witness = false;
...
}
```

JAG: Annotation Generation

Example (JTPL Property)

after begin **normal**
(**eventually** commit **called**),
abort **called**)

Annotation Generation

- Declaration of a ghost variable.
- Assignement of the ghost variable.
- Loop(`Witness`).

Example (Java Code)

```
public class Buffer {
//@ ghost witness = false;
boolean perso = false;
...
void begin(){
...
//@ set witness = true;
}
void commit(){
//@ set witness = false;
...
}
void abort(){
//@ set witness = false;
...
}
```

JAG: Annotation Generation

Example (JTPL Property)

after begin **normal**
(**eventually** commit **called**),
abort **called**)

Annotation Generation

- Declaration of a ghost variable.
- Assignement of the ghost variable.
- Loop(`Witness`).

Example (Java Code)

```
public class Buffer {
//@ ghost witness = false;
boolean perso = false;
...
void begin(){
...
//@ set witness = true;
}
void commit(){
//@ set witness = false;
...
}
void abort(){
//@ set witness = false;
...
}
```

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools
  - Runtime Verification (jmlc - Iowa State University)
  - Symbolic Animation, Test Generation (JML-TT - LIFC, Tobias, Jartege - LSR)
  - JML annotations consistency (JML2B - LIFC)
  - Consistency of JML annotations with Java code
    - ESC-Java
    - Krakatoa - LRI
    - Jack - INRIA Sophia

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools
  - Runtime Verification (jmlc - Iowa State University)
  - Symbolic Animation, Test Generation (JML-TT - LIFC, Tobias, Jartege - LSR)
  - JML annotations consistency (JML2B - LIFC)
  - Consistency of JML annotations with Java code
    - ESC/Java
    - Krakatoa - LRI
    - Jack - INRIA Sophia

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools
  - Runtime Verification (jmlc - Iowa State University)
  - Symbolic Animation, Test Generation (JML-TT - LIFC, Tobias, Jartege - LSR)
  - JML annotations consistency (JML2B - LIFC)
  - Consistency of JML annotations with Java code
    - ESC-Java
    - Krakatoa - LRI
    - Jack - INRIA Sophia

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools
  - Runtime Verification (jmlc - Iowa State University)
  - Symbolic Animation, Test Generation (JML-TT - LIFC, Tobias, Jartege - LSR)
  - JML annotations consistency (JML2B - LIFC)
  - Consistency of JML annotations with Java code
    - ESC-Java
    - Krakatoa - LRI
    - Jack - INRIA Sophia

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools
  - Runtime Verification (jmlc - Iowa State University)
  - Symbolic Animation, Test Generation (JML-TT - LIFC, Tobias, Jartege - LSR)
  - JML annotations consistency (JML2B - LIFC)
  - Consistency of JML annotations with Java code
    - ESC-Java
    - Krakatoa - LRI
    - Jack - INRIA Sophia

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools
  - Runtime Verification (jmlc - Iowa State University)
  - Symbolic Animation, Test Generation (JML-TT - LIFC, Tobias, Jartege - LSR)
  - JML annotations consistency (JML2B - LIFC)
  - Consistency of JML annotations with Java code
    - ESC-Java
    - Krakatoa - LRI
    - Jack - INRIA Sophia

Feature of JAG 0.1

- Automatic generation of ghost variables for observing events and states.
- Automatic generation of invariants for safety properties [Huisman Trentelman AMAST'02]
- All liveness formulae of the language are rewrited in Loop primitive.
- Generation of standard JML file
  $\Rightarrow$ Compatibility with all other JML tools
  - Runtime Verification (jmlc - Iowa State University)
  - Symbolic Animation, Test Generation (JML-TT - LIFC, Tobias, Jartege - LSR)
  - JML annotations consistency (JML2B - LIFC)
  - Consistency of JML annotations with Java code
    - ESC-Java
    - Krakatoa - LRI
    - Jack - INRIA Sophia

Experiments

Case study: Demoney

- JavaCard Electronic Purse
- Over 500 lines of JML

Case study: Demoney

- Annotation Generation with JAG on the JML model and proof with JML2B [B'07].
- Generation of tests with JML-TT and verification of the annotations generated by JAG at the runtime [FATES'06]

1. Introduction
2. Verification of Liveness Properties with JML
3. The JAG Tool
4. Conclusion and Future Work

Conclusion

- Sound method for verifying liveness on a class in isolation with JML.
- Trace-based semantics framework for reasonning about Java/JML.
- Reusable Liveness Primitive Operator.
- Tool supported: JAG.
- Generation of standard JML annotations.
- Experiment on a Java Card Application.

Conclusion

- Sound method for verifying liveness on a class in isolation with JML.
- Trace-based semantics framework for reasoning about Java/JML.
- Reusable Liveness Primitive Operator.
- Tool supported: JAG.
- Generation of standard JML annotations.
- Experiment on a Java Card Application.

Conclusion

- Sound method for verifying liveness on a class in isolation with JML.
- Trace-based semantics framework for reasonning about Java/JML.
- Reusable Liveness Primitive Operator.
- Tool supported: JAG.
- Generation of standard JML annotations.
- Experiment on a Java Card Application.

Conclusion

- Sound method for verifying liveness on a class in isolation with JML.
- Trace-based semantics framework for reasonning about Java/JML.
- Reusable Liveness Primitive Operator.
- Tool supported: JAG.
- Generation of standard JML annotations.
- Experiment on a Java Card Application.

Conclusion

- Sound method for verifying liveness on a class in isolation with JML.
- Trace-based semantics framework for reasonning about Java/JML.
- Reusable Liveness Primitive Operator.
- Tool supported: JAG.
- Generation of standard JML annotations.
- Experiment on a Java Card Application.

Conclusion

- Sound method for verifying liveness on a class in isolation with JML.
- Trace-based semantics framework for reasonning about Java/JML.
- Reusable Liveness Primitive Operator.
- Tool supported: JAG.
- Generation of standard JML annotations.
- Experiment on a Java Card Application.

Work in Progress

- Verification of the progress of the program.
- Extension to other input/output languages
  - SPEC♯
  - B [B'07]
  - Aspect J
- Extension to other formalisms
  - LTL
  - Transition Diagram
  - Regular Expression
  - CaRet
- Temporal Properties conformance testing [FATES'06]
- Collaboration of proof and test techniques [JML-TT]
  - Generating specific test cases when the proof fails.
  - $\Rightarrow$ Trying to find a counter-example.

Work in Progress

- Verification of the progress of the program.
- Extension to other input/output languages
  - SPEC♯
  - B [B'07]
  - Aspect J
- Extension to other formalisms
  - LTL
  - Transition Diagram
  - Regular Expression
  - CaRet
- Temporal Properties conformance testing [FATES'06]
- Collaboration of proof and test techniques [JML-TT]
  - Generating specific test cases when the proof fails.

  ⇒ Trying to find a counter-example.

Work in Progress

- Verification of the progress of the program.
- Extension to other input/output languages
  - SPEC♯
  - B [B'07]
  - Aspect J
- Extension to other formalisms
  - LTL
  - Transition Diagram
  - Regular Expression
  - CaRet
- Temporal Properties conformance testing [FATES'06]
- Collaboration of proof and test techniques [JML-TT]
  - Generating specific test cases when the proof fails.
  - ⇒ Trying to find a counter-example.

Work in Progress

- Verification of the progress of the program.
- Extension to other input/output languages
  - SPEC♯
  - B [B'07]
  - Aspect J
- Extension to other formalisms
  - LTL
  - Transition Diagram
  - Regular Expression
  - CaRet
- Temporal Properties conformance testing [FATES'06]
- Collaboration of proof and test techniques [JML-TT]
  - Generating specific test cases when the proof fails.
  - ⇒ Trying to find a counter-example.

Work in Progress

- Verification of the progress of the program.
- Extension to other input/output languages
  - SPEC♯
  - B [B'07]
  - Aspect J
- Extension to other formalisms
  - LTL
  - Transition Diagram
  - Regular Expression
  - CaRet
- Temporal Properties conformance testing [FATES'06]
- Collaboration of proof and test techniques [JML-TT]
  - Generating specific test cases when the proof fails.

  $\Rightarrow$ Trying to find a counter-example.