Programming-in-the-many: SLIME*: checks*

Summer 2002

Karsten Stahl Martin Steffen

Institut. für Informatik u. Prakt. Mathematik

Christian-Albrechts Universtität zu Kiel

Offered functionality

- well-formed checking
  - sfc non-null
  - initial step contained in the set of steps.
  - all steps and transitions are non-null.
  - no step occurs twice (by name)
  - no transition has a non-existing step as source or target (by name)
  - no transition has at the same time more that one source and more than one target.
- type checking

Design

- Implemented as Visitor-pattern to the abstract syntax
- consists of
  - `absynt.Visitor` contains the visitor interfaces for each clause of the abstract syntax as inner classes
  - each class honors an `accept`-method [a]
- visitors hidden from the check-clients
- pros/cons:
  - extensible
  - separation of concerns, clean
  - higher static safety
  - slower

[a] Slight oo-hack (overriding) done to cater for additional absynt-classes cur-

Visitor interfaces, types clause

as inner class of `absynt.Visitors`:

```
 public interface IType{
//    public Object forDoubleType() throws Exception;
    public Object forIntType() throws Exception;
    public Object forBoolType() throws Exception;
    public Object forUndefType() throws Exception;
    public Object forUnitType() throws Exception;
 }
```

Visitor: abstract syntax

```
public abstract class Type extends Absynt implements Serializable {
  public Object accept (Visitors.IType ask) throws Exception
 { throw new Exception("This␣type␣does␣not␣accept␣visititors␣(yet).");

 public abstract boolean equals (Type t);
```

Visitors: abstract syntax (2)

```
public class BoolType extends Type implements Serializable {

  /**
     visitor acceptor
  */
  public Object accept (Visitors.IType ask) throws Exception {
    return ask.forBoolType();
  }

  public boolean equals (Type t) {
    return (t instanceof BoolType);
  }
```

Visitor: client

```
class ExprV implements Visitors.IExpr{
 public Object forB_Expr(Expr l, int o, Expr r) throws CheckException {
       // binary expressions
       try {
          Type t_l = (Type)(l.accept(this));
          Type t_r = (Type)(r.accept(this));
          if ((o == Expr.LEQ) | (o == Expr.GEQ) | (o == Expr.LESS) | (o == Expr.GR
             if ((t_l instanceof IntType) && (t_r instanceof IntType))
                return new BoolType();
             else throw new TypeMismatch();
```