```
Coq <
Coq < Inductive RULERG (MS : GmethPost) (MI : GmethInv ) :  Gstmt -> Gassertion -> Prop :=
Coq <
Coq <  | AffectRuleRG : forall   x e (post : Gassertion) ,
Coq <      (forall  (s1 s2: state) (g1 g2 : gState ),s2 = update s1 x (eval_expr s1 e)  -> g1 = g2
Coq <      RULERG MS  MI (GAffect x e) post
Coq <
Coq <  | IfRuleRG : forall   (e : expr) (stmtT stmtF: Gstmt ) ( post1  post2 post : Gassertion) ,
Coq <     ( forall ( s1 s2: state) (g1 g2 : gState ) event,  ( (not (eval_expr s1 e = 0)) -> post1
Coq <                                          (eval_expr s1 e = 0 ->  post2 s1 g1 event s2 g2 )
Coq <
Coq <     RULERG MS MI stmtT   post1   ->
Coq <     RULERG MS MI stmtF   post2   ->
Coq <
Coq <     RULERG MS MI (GIf e stmtT stmtF) post
Coq <
Coq <  | WhileRuleRG : forall   (st : Gstmt ) ( post post1   : Gassertion) e ( inv : Gassertion)
Coq <      (forall s1 s2 g1 g2 event, post1 s1 g1 event s2 g2   ->  post s1 g1  event s2 g2) ->
Coq <
Coq <      (* ( forall s p t  event1 event2,   eval_expr s e <>  0 -> inv s event1 p -> post1 p eve
Coq <      (forall s g, eval_expr s e = 0  -> post1 s g nil s g   ) ->
Coq <      RULERG MS MI st post1     ->
Coq <      RULETG MS st  inv  ->
Coq <      (forall  s1 s2 s3 g1 g2 g3 e1 e2,  ( inv s1 g1 e1 s2 g2  -> eval_expr s1 e <>  0 -> post
Coq <      RULERG MS  MI  (GWhile e st) post
Coq <
Coq <
Coq <  |  SeqRuleRG : forall  (stmt1 stmt2: Gstmt ) (post  post1 postT  postRst2 : Gassertion),
Coq <   ( forall s1 s2  e g1 g2 , ( post1 s1 g1 e s2 g2 -> post s1 g1  e s2 g2 )) ->
Coq <    (forall s1 s2 s3 e1 e2 g1 g2 g3,  postT s1 g1 e1 s2 g2 -> postRst2 s2 g2 e2 s3 g3 ->  pos
Coq <    RULERG MS MI stmt1  post1 ->
Coq <    RULETG  MS stmt1   postT ->
Coq <    RULERG MS MI stmt2 postRst2   ->
Coq <    RULERG MS MI (GSseq stmt1 stmt2) post
Coq <
Coq <  | SkipRuleRG :   forall  (post: Gassertion),
Coq <    ( forall (s1 s2: state ) g1 g2  ,  s1 = s2 -> g1 = g2 -> post s1 g1 nil s2 g2 ) ->
Coq <    RULERG MS MI GSkip  post
Coq <
Coq <  | SetRuleG :  forall   x e (post : Gassertion) ,
Coq <      (forall  (s1 s2: state) (g1 g2 : gState ),   g2 = gUpdate g1 x (gEval_expr  s1 g1 e) ->
Coq <      RULERG MS  MI (GSet x e) post
Coq <
Coq <
Coq <  | CallRuleRG : forall   ( mName : methodNames ) ( post   : Gassertion ) ,
Coq <    (forall (s1 s2 : state ) event g1 g2,  ( MI mName ) s1 g1 event s2 g2  -> post s1 g1  eve
Coq <    RULERG MS  MI (GCall mName )  post
Coq <
Coq <  | SignalRuleRG : forall   ( post   : Gassertion ) event,
Coq <     ( forall (s1 s2: state ) g1 g2 ,  s1 = s2 -> g1 = g2 -> post s1 g1 (event :: nil) s2 g2
```

```
Coq <     RULERG MS MI (GSignal event)  post.
Toplevel input, characters 2614-2615
>     ( forall (s1 s2: state ) g1 g2 ,  s1 = s2 -> g1 = g2 -> post s1 g1 (event :: nil) s2 g2 ) ->
>                                                                                              ^
Syntax error: [constr:binder_constr] or [constr:operconstr] expected after ':' (in [constr:opercons
```