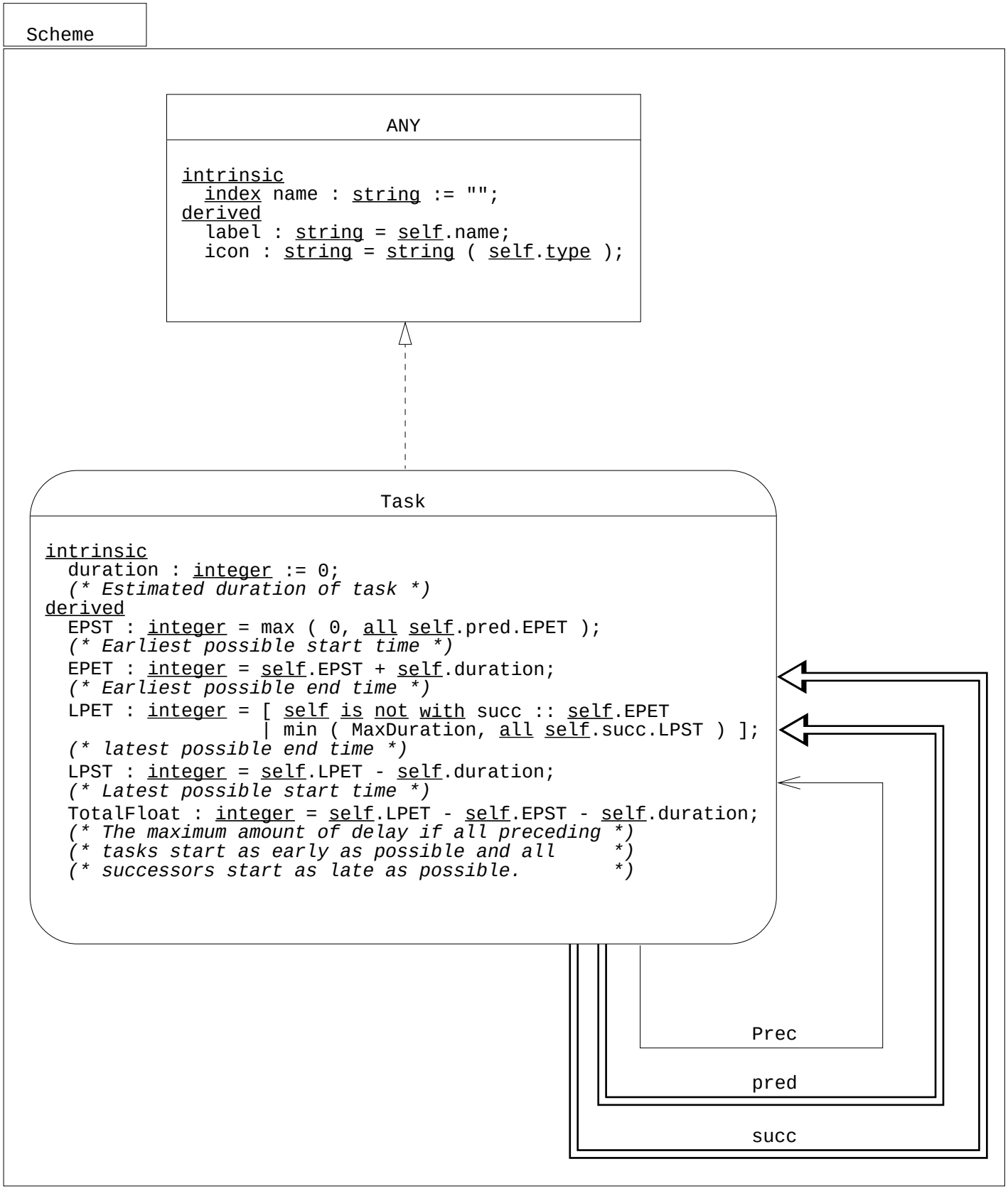
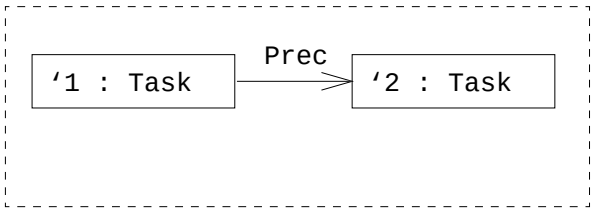
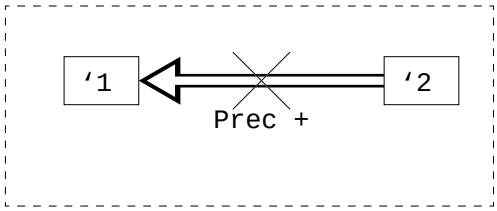

Scheme
ANY
intrinsic
index name : string := "";
derived
label : string = self.name;
icon : string = string (self.type);
Task
intrinsic
duration : integer := 0;
(* Estimated duration of task *)
derived
EPST : integer = max (0, all self.pred.EPET);
(* Earliest possible start time *)
EPET : integer = self.EPST + self.duration;
(* Earliest possible end time *)
LPET : integer = [self is not with succ :: self.EPET
| min (MaxDuration, all self.succ.LPST)];
(* latest possible end time *)
LPST : integer = self.LPET - self.duration;
(* Latest possible start time *)
TotalFloat : integer = self.LPET - self.EPST - self.duration;
(* The maximum amount of delay if all preceding *)
(* tasks start as early as possible and all *)
(* successors start as late as possible. *)
Prec
pred
succ

```
spec SimplePNets

  section Scheme

    node class ANY
      [...]
    end;

    node type Task : ANY
      [...]
    end;

    edge type Prec : Task -> Task;
    (* precedence relationship *)

    path succ : Task -> Task =
      -Prec->
    end;

    path pred : Task -> Task =
      <-Prec-
    end;

  end;

  section HelperFunctions

    function MaxDuration : -> integer =
      100
    end;

    function min : ( v1, v2 : integer) -> integer =
      [ v1 > v2 :: v2
      | v1 ]
    end;

    function max : ( v1, v2 : integer) -> integer =
      [ v1 < v2 :: v2
      | v1 ]
    end;

  end;
```

```
section TaskOperations

  section EditOperations

    constraint
      for
```

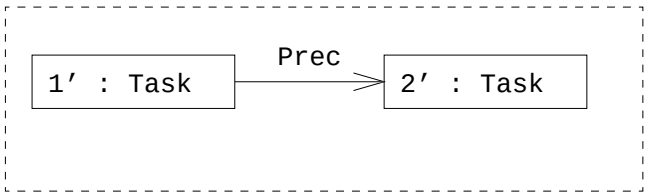



```
      ensure
```



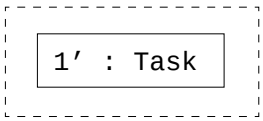

```
    end;
    (* Ensure acyclic task nets *)

    production CreateDiagram( StartName, StopName : string) =
```

```
      ::=
```



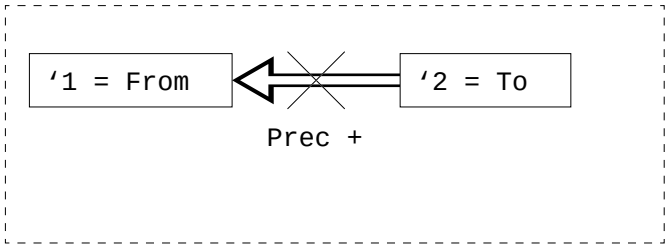

```
      transfer 1’.name := StartName;
               2’.name := StopName;
    end;

    production CreateTask( Name : string) =
```

```
      ::=
```

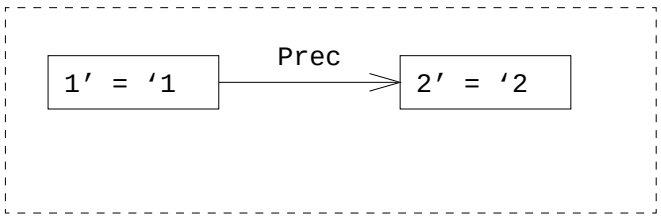



```
      transfer 1’.name := Name;
    end;
```

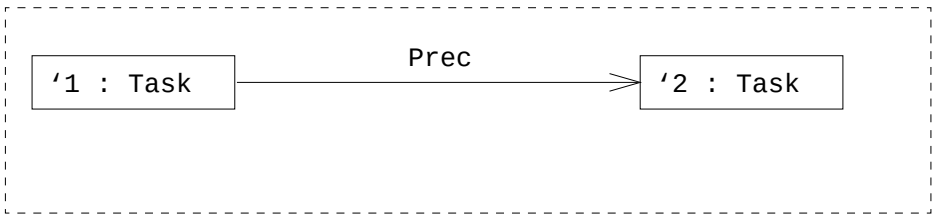
```
production CreatePrecedence( From, To : Task) =
```



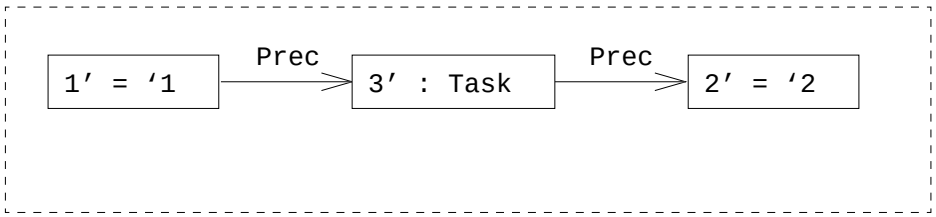
::=




```
end;

production InsertTask( FromName, NewName, ToName : string) =
```

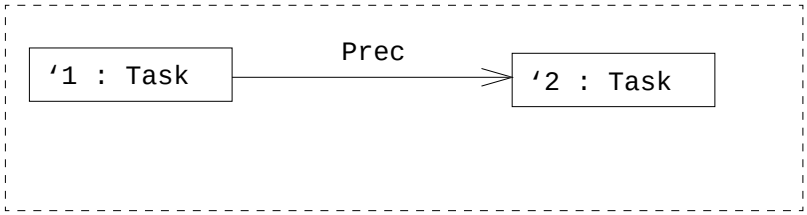


::=

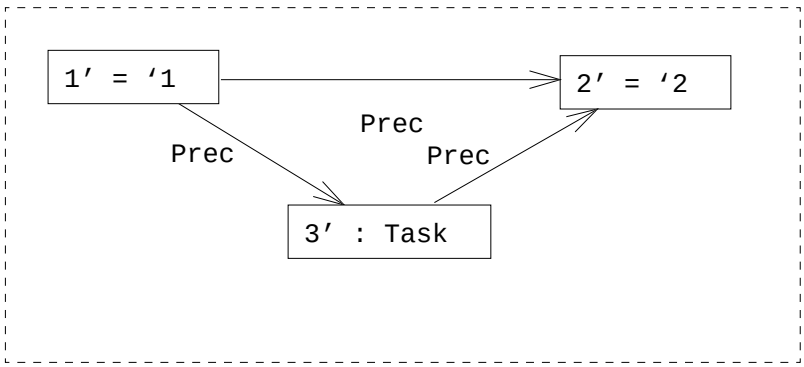



```
  condition ‘1.name = FromName;
            ‘2.name = ToName;
  transfer 3’.name := NewName;
end;
```

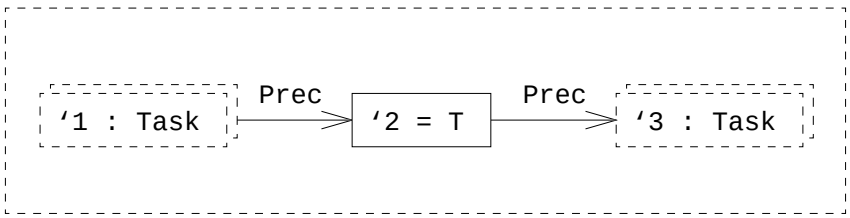
```
production ParallelTask( FromName, NewName, ToName : string)
=
```



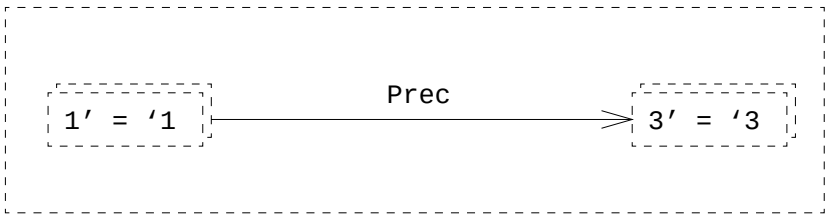

```
  ::=
```




```
  condition ‘1.name = FromName;
            ‘2.name = ToName;
  transfer 3’.name := NewName;
end;

production DeleteTask( T : Task) =
```

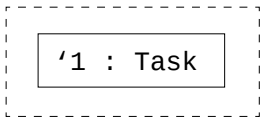



```
  ::=
```

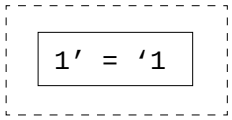



```
end;
```

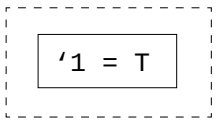
```
  production EnterDuration( Name : string ;
                            Duration :   integer) =
```



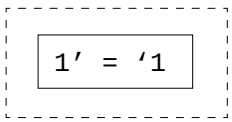

```
    ::=
```



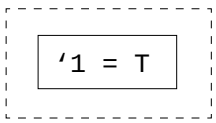

```
    condition ‘1.name = Name;
    transfer 1’.duration := Duration;
  end;

  production ChangeDuration( T : Task ;
                             Duration :   integer) =
```



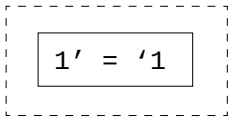

```
    ::=
```




```
    transfer 1’.duration := Duration;
  end;

  production ChangeName( T : Task ;
                         Name :   string) =
```

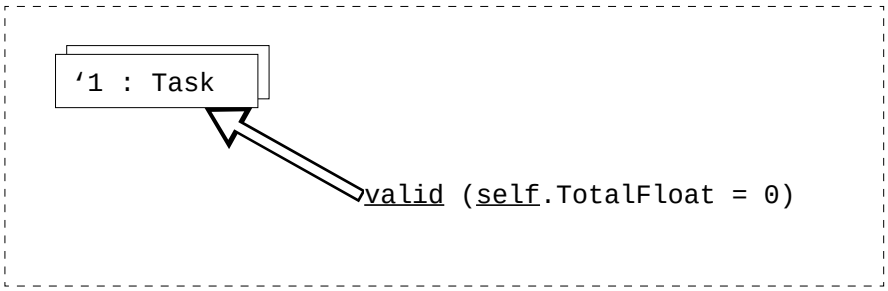



```
    ::=
```

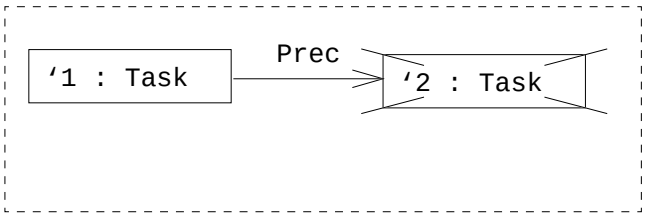



```
    transfer 1’.name := Name;
  end;

end;
```

```
  section AnalysisOperations
     test CriticalPath( out TSet : Task [1:n]) =
```




```
       return TSet := ‘1;

     end;

     test Duration( out T : integer) =
```




```
       return T := ‘1.EPET;

     end;

     query Analyse( out Path : Task [1:n] ; out Time :   integer)
     =
         CriticalPath ( out Path )
       & Duration ( out Time )
     end;

   end;

 end;

 transaction MAIN =
     CreateExampleNet
   & use CriticalPath : Task [0:n];
         Time : integer;
         R : Resource;
         Task1, Task2 : Task
     do
         Analyse ( out CriticalPath,
                   out Time )
     end
 end;

end.
```