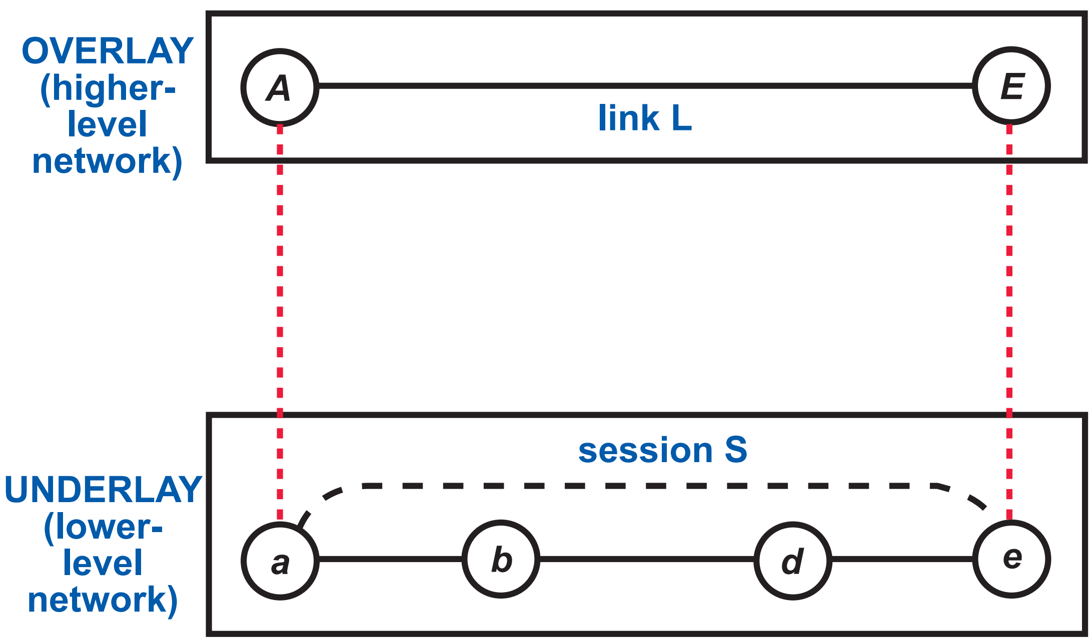
Exercise (exam of 28/6/2017)

- Observe the following figure. It describes a communication system composed of an overlay network linking *nodes* A and E. This overlay is a virtual network built on top of an underlay network. In the figure, the underlay network is composed of four nodes (a, b, d, and e) and link L exploits the links between a, b, d and e in the underlay network to ensure that A and E can communicate.



E is attached to e, or . . .

e is the location of E.

L uses S, or . . .

S implements L.

Exercise (cont.)

- Consider the following Alloy signatures:

```
sig Network {
  uses: lone Network
}{ this not in uses }

sig Node {
    belongsTo: Network,
    isLinkedTo: some Node,
    isAttachedTo: lone Node
}{ this not in isAttachedTo and
   this not in isLinkedTo }
```

Exercise (cont.)

- **A)** Explain the meaning of these signatures with respect to the figure above and indicate which elements in the figure are not explicitly modeled by the two signatures.

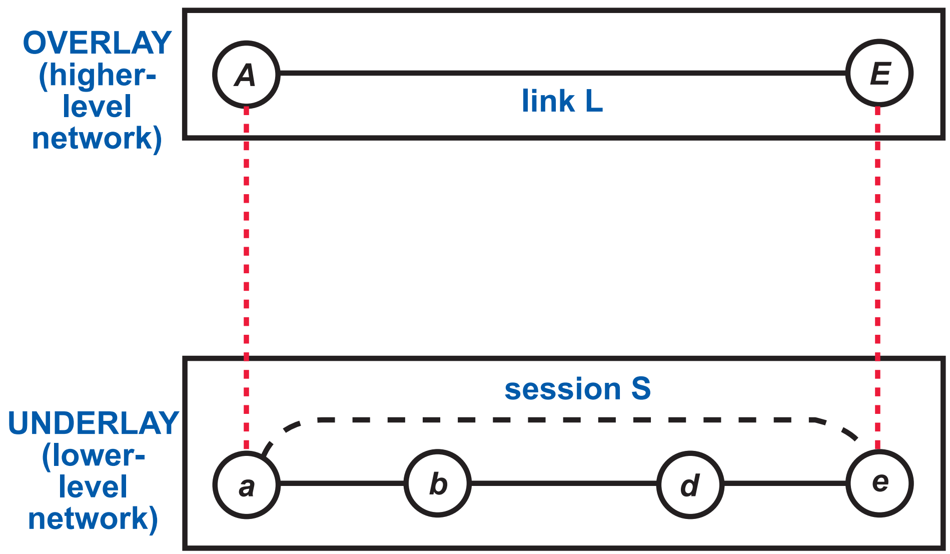
- **B)** Write facts to model the following constraints:
  - Linked nodes have to be in the same network
  - A node belonging to a certain network can only be attached to nodes of the corresponding underlay network
  - If a network is an overlay one, then there should not be nodes in this network that are not attached to other nodes
  - A network should always contain some nodes

- **C)** Write the predicate `isReachable` that, given a pair of `Nodes`, `n1` and `n2`, is true if there exists a path that from `n2` reaches `n1`, possibly passing through any intermediate node.

Solution, Part A

```
sig Network {
  uses: lone Network
}{ this not in uses }
```

```
sig Node {
  belongsTo: Network,
  isLinkedTo: some Node,
  isAttachedTo: lone Node
}{ this not in isAttachedTo and
   this not in isLinkedTo }
```



E is attached to e, or . . .

e is the location of E.

L uses S, or . . .

S implements L.

Solution, Part B

```
// Linked nodes have to be in the same network
fact linkedNodesInTheSameNetwork {
  all disj n1, n2: Node |
    n1 in n2.isLinkedTo implies
                        #(n1.belongsTo & n2.belongsTo) > 0
}

// A node belonging to a certain network can only be
// attached to nodes of the corresponding underlay network
fact isAttachedToInConnectedNetworks {
  all disj n1, n2: Node |
     n1 in n2.isAttachedTo implies
                   #(n1.belongsTo & n2.belongsTo.uses) > 0
}
```

Solution, Part B (cont.)

```
// If a network is an overlay one,
// then there should not be nodes in this network
// that are not attached to other nodes
fact overlayNodeShouldBeAttached {
  all ntw: Network |
    some ntw2: Network | ntw2 in ntw.uses
    implies
    all n: Node | n.belongsTo = ntw implies
                                        n.isAttachedTo != none
}

// A network should always contain some nodes
fact notEmptyNetwork {
  all ntw: Network | some n: Node | n.belongsTo = ntw
}
```

Solution, Part B
(alternative formulation of some facts)

```
// Linked nodes have to be in the same network
fact linkedNodesInTheSameNetwork {
  all disj n1, n2: Node |
    n1 in n2.isLinkedTo implies n1.belongsTo = n2.belongsTo
}


// A node belonging to a certain network can only be
// attached to nodes of the corresponding underlay network
fact isAttachedToInConnectedNetworks {
  all disj n1, n2: Node |
     n1 in n2.isAttachedTo implies
                         n1.belongsTo in n2.belongsTo.uses
}
```

Solution, Part C

```
//n1 is reachable from n2
pred isReachable[n1: Node, n2: Node] {
  n1 in n2.^isLinkedTo
}
```

Exercise (exam of 13/2/2017)

- Consider construction cubes of three different sizes, small, medium, and large. You can build towers by piling up these cubes one on top of the other respecting the following rules:
  - A large cube can be piled only on top of another large cube
  - A medium cube can be piled on top of a large or a medium cube
  - A small cube can be piled on top of any other cube
  - It is not possible to have two cubes, A and B, simultaneously positioned right on top of the same other cube C

Exercise (cont.)

- **Question 1**: Model in Alloy the concept of cube and the piling constraints defined above.
- **Question 2**: Model also the predicate `canPileUp` that, given two cubes, is true if the first can be piled on top of the second and false otherwise.
- **Question 3**: Consider now the possibility of finishing towers with a top component having a shape that prevents further piling, for instance, a pyramidal or semispherical shape. This top component can only be the last one of a tower, in other words, it cannot have any other component piled on it.
  Rework your model to include also this component. You do not need to consider a specific shape for it, but only its property of not allowing further piling on its top. Modify also the `canPileUp` predicate so that it can work both with cubes and top components.

Solution, Question 1

```
abstract sig Size{}
one sig Large extends Size{}
one sig Medium extends Size{}
one sig Small extends Size{}

sig Cube {
  size: Size,
  cubeUp: lone Cube
}{ cubeUp != this }

fact noCircularPiling {
  no c: Cube | c in c.^cubeUp
}

fact pilingUpRules {
  all c1, c2: Cube | c1.cubeUp = c2 implies
          ( c1.size = Large or
            c1.size = Medium and (c2.size = Medium or c2.size = Small) or
            c1.size = Small and c2.size = Small )
}

// it is still possible for a cube to be on top of two different cubes
// this is not explicitly ruled out by the specification
```

Solution, Question 2

```
pred canPileUp[cUp: Cube, cDown: Cube] {
    cDown != cUp
    and
    ( cDown.size = Large
      or
      cDown.size = Medium and
                 (cUp.size = Medium or cUp.size = Small)
      or
      cDown.size = Small and cUp.size = Small )
}
```

Solution, Question 3

```
// modified signatures
abstract sig Block {}
sig Top extends Block {}
sig Cube extends Block {
  size: Size,
  cubeUp: lone Block
}{ cubeUp != this }

pred canPileUp[bUp: Block, bDown: Block] {
    bDown != bUp and
    bDown in Cube and
    ( bUp in Top
      or
      bDown.size = Large
      or
      bDown.size = Medium and
              (bUp.size = Medium or bUp.size = Small)
      or
      bDown.size = Small and bUp.size = Small )
}
```