I think there are a few things can be learned on our current framework:

Let's call a shape descriptor S give a fixed-size feature vector for each object (size, position, orientation etc.):
Let's call an action of the agent is to move(A, X): Move block A to position X

1. Learn "elementary" relationship of objects:
    - Unary relationships: attributive (size, color): this could be done easily by deixis. We might not want to learn this.
    - Binary relationships: relative distance "near" vs "far", EC relations "next to" vs "on", shaped-based relation "at one end of" vs "at the middle of". By learning from observation (LfO), the agent can learn these relations, both in the distinction sense and performing sense.

      Learn to perform actions:

      For example, let's learn to perform "Put A next to B". The location of X in action *move(A,X)* could be learned by 4 linear functions g1, .. g4:
      $X \sim \sum p_i * \text{Gaussian}(g_i(S(B)), \sigma)$

      Similarly for "Put A at one end of B", X could be learned by two linear functions g1 and g2 (we need to accommodate two ends of B if B is a long object):
      $X \sim p * \text{Gaussian}(g1(S(B)), \sigma) + q * \text{Gaussian}(g2(S(B)), \sigma)$

      The reason we use a Mixture of gaussian distributions instead of a deterministic move, is because:
      1. We have fuzzy inputs
      2. With probabilistic model, we can incorporate priors and bias, such as: we want to move A to the end that closer to the agent.
      3. Only with probabilistic model, we can handle relative relationship, such as "near" vs "far".

      Learning to distinguish between different relations could be handled by comparing probability of X given different models:

      P(X|near) vs P(X|far)
      P(X|next to) vs P(X|on) etc.

2. Learn an action that involves replicating path-movement of objects:
    - This is what I'm currently doing experiments on (in the paper submitted to AAAI Spring Symposium). It might not straightly related to the problem of building a staircase, for example, as building a staircase does not involve distinction of path-movement.

3. Learn a new category of objects:
    - Basic compound types: Let's assume that we want to teach the agent to learn to distinguish "a row" and "a column", and to create "a row" or "a column", and all the agent knows are concept of blocks, and simple elementary relationships "on" and "next to":

      Let's consider: learning to create *a row* with a few samples:
        - Let's say we have only 2 samples: one with two blocks, one with three blocks
        - For each sample, we have 2 representations, as sequence of actions, or as approximation of final configuration (convex hull).

- When the agent is asked to create 2-block row, it's straightforward, any action Move(B,NextTo(A)) leads to a row that matches the training sample.
- When the agent is asked to create 3-block row, when the agent adds the third block Move(C,NextTo(B)), it could add to three locations next to B (except the location where A occupies), corresponding to . However, only one location leads to correct row. Fortunately, it could asks: "Can I put C here" and pointing to unaccepted locations. The performer says "No, you can only put C there", and pointing to the correct location. The best thing the agent can do is to refine the model by considering the first two blocks as one object AB, and learn the same kind of model as the previous example of leaning "C at one end of AB". Let's call this refined model P, that can give position of C by a combined shape descriptor of A and B.

  The learning model for 3-block rows is: *put(B,NextTo(A)), put(C, P(AB))*

- When the agent is asked to create 4-block row in the form of "Add one more block" or "Do it again", it doesn't yet have a 4-block prototype, how can it generalize? Here we can use some heuristic methods: 1. it can try to do *put(D, P(AB))* which is invalid, because C is at the same place. 2. It can sift the "conditional" part to the last two blocks BC, and try *put(D, P(BC))*, which would create a correct row.

The same algorithm could be applied to a column, a zig-zag shape etc.