THE DEFINITIVE, PEER-REVIEWED AND EDITED VERSION OF THIS ARTICLE WILL APPEAR IN:
Yue K, Krishnamurti R, Forthcoming, "Developing a tractable shape grammars" *Environment and Planning B: Planning and Design*, vol. 42, 2015

Developing a tractable shape grammar

Ramesh Krishnamurti [(1)]

ramesh@cmu.edu, Carnegie Mellon University, Pittsburgh, PA 15213-3890

Kui Yue

kuiyue@microsoft.com, Microsoft, Redmond, WA 98052

Abstract

Previously, we examined tractable parametric shape grammars (Yue and Krishnamurti, 2013), and developed a general paradigm for implementing classes of such grammars (Yue and Krishnamurti, 2014). A tractable shape grammar has polynomial computing complexity, and is specified in a way that is readily transformable to a computer program. By contrast, traditionally, shape grammars have been typically developed without a computer implementation in mind, either requiring ambiguity to be clarified, or it is hardly possible for the grammar to be implemented by a polynomial algorithm. Each tractable shape grammar is tied to a particular framework, which is backed by a data structure and supports a meta-language. In this paper, we illustrate the development of tractable shape grammars by transforming a shape grammar developed, essentially, in traditional fashion for the Baltimore Rowhouse (Hayward 1981; Hayward and Belfoure, 2005). The development is for a specific application context, namely, to determine the interior layout of a building given its external features; and the process serves as a general strategy for developing tractable shape grammars.

Keywords: shape grammars, implementation, Baltimore rowhouse

[(1)] Author for correspondences

1 Introduction

Tractable shape grammars have polynomial time and language space complexity (Yue and Krishnamurti, 2013). Grammars exhibit a variety of characteristics thereby rendering impossible a single uniform shape grammar interpreter. Instead, a strategy for implementing tractable shape grammars has been advocated, in which the grammar is specified within a specific representational framework (Yue and Krishnamurti, 2014). Each framework has its own underlying data structure, a set of basic manipulation algorithms, and a meta-language for describing shape rules. In that paper, three distinct frameworks were considered: *rectangular*, for grammars that are primarily directed at generating plans; *polygonal*, for designs essentially determined by subdivision; and *graph*, for shapes specified by topological relationships. Other frameworks are possible. This paper completes the sequence, in which we consider the development of a simple but exemplar tractable shape grammar, namely, for the Baltimore Rowhouse on the rectangular framework.

The development proceeds in the following manner. Firstly, shape rules that focus on capturing the style of the Baltimore Rowhouse are developed. Secondly, the shape rules are recast in terms of the chosen implementation framework, incorporating knowledge about constraints into the rules so that the entire grammar is tractable. Thirdly, the implementation of the interpreter is outlined.

2 Creating a shape grammar

The process has three steps: identifying a set of patterns that most succinctly constitutes the objects; formalizing the patterns as a set of shape rules; and organizing the shape rules so that the grammar generates as many valid designs as possible while producing as few invalid design as possible.

We look to real examples in order to find patterns within a set of designed objects. However, it should be noted that examples alone do not suffice; factors that motivate the designs must be considered. Such information helps to identify a minimal set of patterns that characterizes the design process. Patterns that have been so identified translate into shape rules.

The goal is to employ as few rules as possible to create as many valid designs as possible while keeping the rules as simple as possible. The criteria by which one establishes how well a grammar meets this goal is subjective, although it is typically not hard to identify the better solution from among possible candidates. At present, one cannot always determine *a priori* whether a grammar correctly generates all valid objects of a type; likewise, one cannot always determine a priori whether a grammar creates valid design objects. As a result, one generally evaluates the validity of a shape grammar through trial and error: applying every possible sequence of rules to the initial shape. In reality, the number of configurations generated by a sufficiently powerful grammar is so large that one cannot test every possible design. It is instructive to note the parallels and agreement between the above observations and the well-known fact that parsing a configuration against a shape grammar is computationally unsolvable in general (Gips, 1975; Stiny, 1975).

3 The Baltimore Rowhouse

The rowhouse became the dominant house type in Baltimore after its adoption in the eighteenth century (Hayward and Belfoure, 2005). Earlier rowhouses were found in other American cities like Boston, Philadelphia, New York, Richmond, and St. Louis; but few are like those in Baltimore in that the spirit and identity of the city are closely tied to this particular architectural form, whence the name—the Baltimore Rowhouse. The rowhouse

had been persistently and tirelessly developed across two centuries, blossoming with prosperity prior to World War II, suffering from discrimination of postwar planners, and their recent redemption as humanely scaled housing. The two-story, three-bay house was an English invention in the beginning, plain in design without useless ornamentation, representing an efficient development policy that proved viable over decades of use. This house form had been modified across time to meet the needs of different population groups of the city. Those for the wealthy were architect-designed; those for everyone else were built on speculation and, for the most part, designed by the builders themselves. To attract customers, and to make their product stand out among the thousands of rowhouses available, builders kept up with the latest styles, making modifications to cornice designs, window treatments, and the brick façade itself, adding bay windows, peaked roofs, stick-style porches, and carved or modeled embellishments. Across two centuries, the rowhouse history of Baltimore involves both changes and lack of changes; the changes relate to the development of the city, and the lack of changes forms the style of the Baltimore Rowhouse. Figure 1 shows the photographic images of rowhouses from the Federal Hill district of Baltimore.

There are two main resources used to develop the rowhouse grammar: the article *Urban Vernacular Architecture in Nineteenth-Century Baltimore* by Hayward (1981) as the primary source providing detailed information about rowhouse morphology; and the monograph *The Baltimore Rowhouse* by Hayward and Belfoure (2005) as a secondary source providing more detailed discussion of the cultural factors that have influenced the morphology. We focus on the first floor configuration based on the information available although the mechanism can apply to developing the grammar for the other floors.

(a) 1-11 East Montgomery Street
(1 is on the right)

(b) 202-208 East Montgomery Street

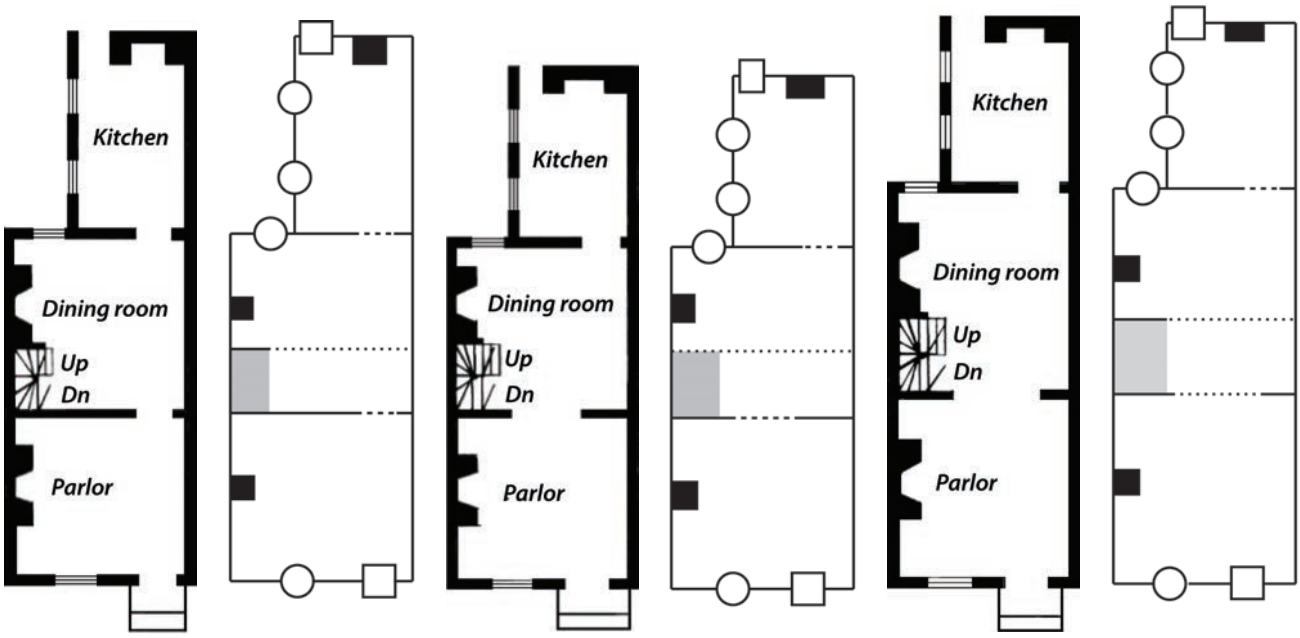
(c) 815-829 South Charles Street
(815 is on the left)

(d) 3-25 East Wheeling Street

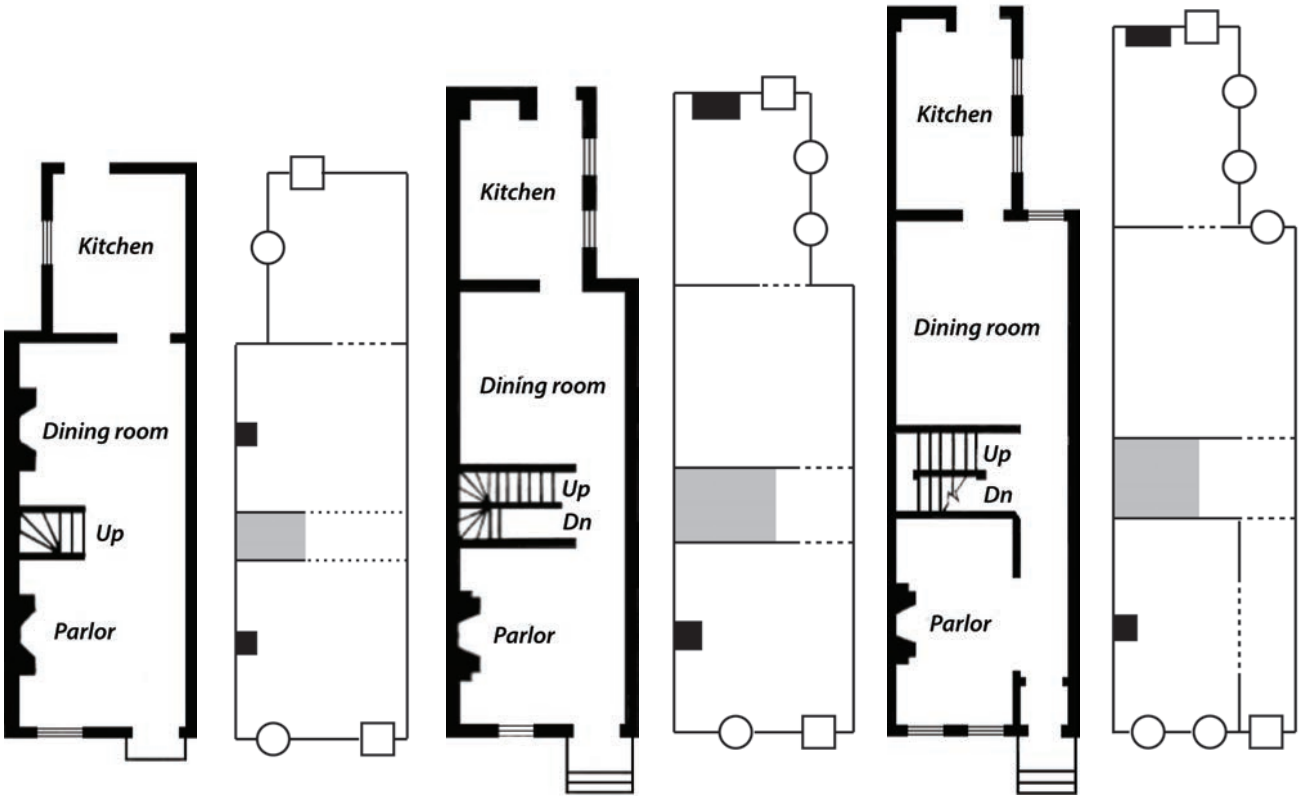
Figure 1 Photographic images of Baltimore rowhouses (Source: Kui Yue)

3.1. Abstract shape representation

To identify patterns in the rowhouses, we employ a specific shape representation, which is essentially an abstracted form of the actual plan. See Figure 2. The representation emphasizes topological information, e.g., relationship between spaces, rather than details

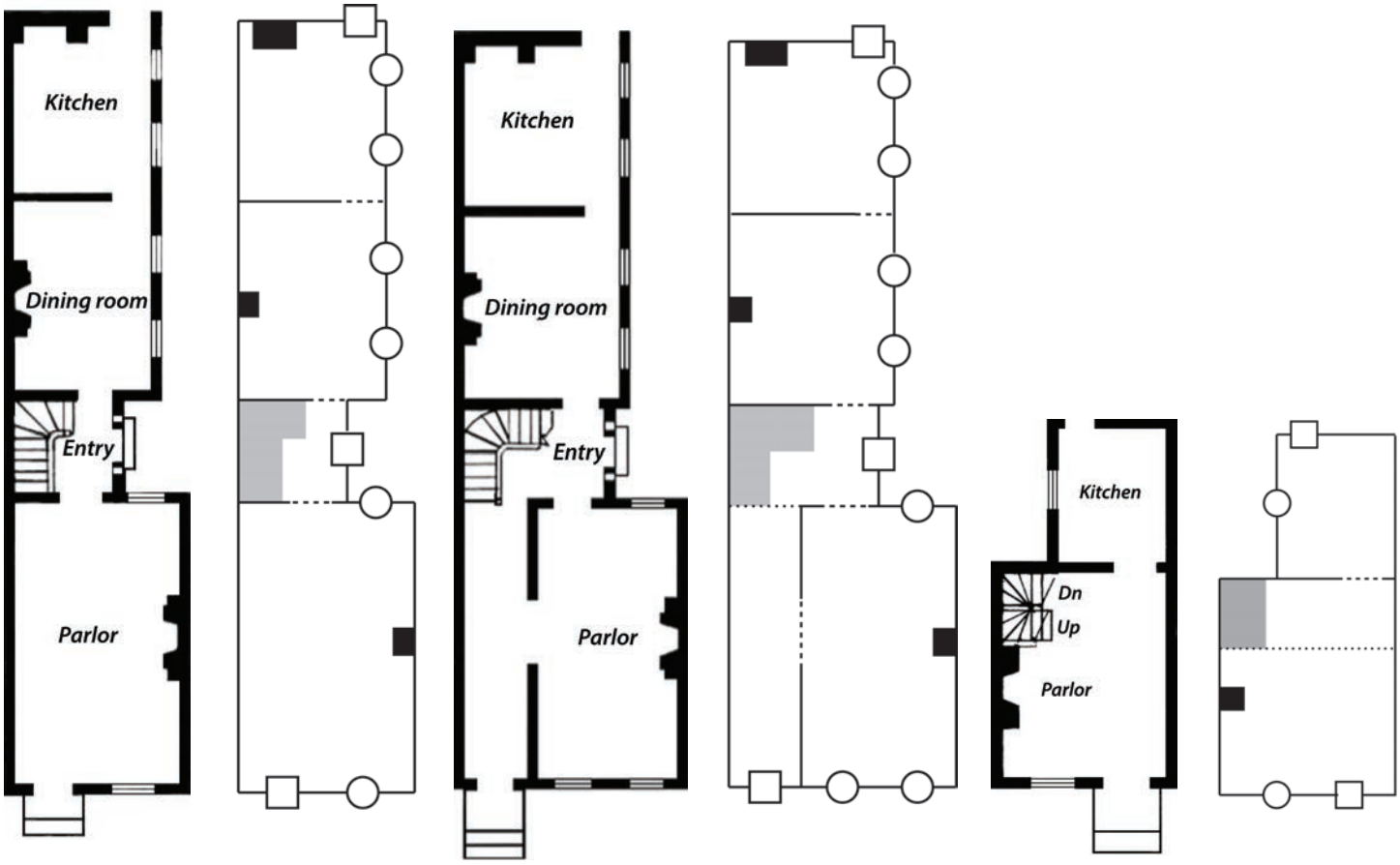


(a) 821 S Charles Street (b) 43 E Hamburg Street (c) 21 E Wheeling Street

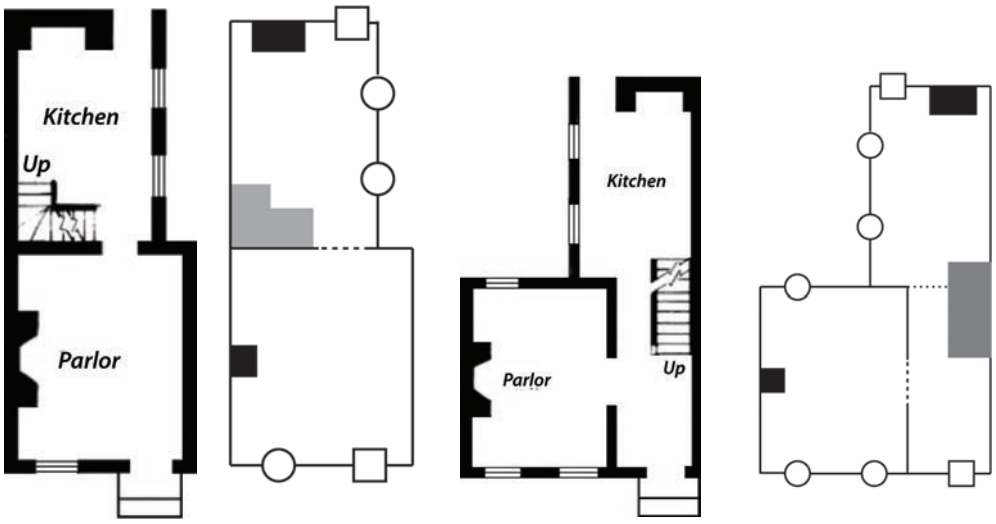


(d) 1028 Patapasco Street (e) 401 Grindall Street (f) 1029 S Hanover Street

Figure 2 Sample shape representation for the Baltimore Rowhouse



(g) 208 E Montgomery Street (h) 236 E Montgomery Street (i) 14 W Cross Street



(j) 819 S Charles Street (k) 3 E Montogomery Street

Figure 2 (continued)

of the building itself. Spaces are simplified for clarity while certain building features, e.g., wall thickness, are eliminated from consideration. The representation emphasizes topological information, e.g., relationship between spaces, rather than details of the building itself. Spaces are simplified for clarity while certain building features, e.g., wall thickness, are eliminated from consideration. Pertinent exterior features are represented through graphic icons: a window is represented by a circle; a door, by a hollow rectangle; a fireplace as a solid rectangle; and a staircase as a solid grey area. Interior features, such as doorways between rooms, are shown as dashed lines. For the purposes of this paper, other interior features are not incorporated.

3.2. Variation in interior configuration

Hayward (1981) suggests that the Baltimore Rowhouse show little morphological variation. The lack of significant variation is clearly visible when comparing the three buildings in Figure 2a~c. The building in Figure 2a, located at 821 South Charles Street, constructed in 1818, is of the 'two-and-a-half-story federal style.' The building in Figure 2b, located at 43 East Hamburg Street, constructed in 1838, is of a later variation of the federal style. The building in Figure 2c, located at 21 East Wheeling Street, constructed in 1850, is of the 'two-story-plus-attic Greek revival style.'

Although these three buildings were constructed decades apart, and nominally, of distinct styles, each follows the same basic plan. This is not to suggest that the rowhouse shows no variation. Rather, the variations are fairly uniform and follow well-defined patterns. At least five major variations across the entire corpus can be identified:

i) Rowhouses are divided into two blocks: a main block toward the street and a kitchen block toward the rear (Figure 3). The two blocks may be directly adjacent to one another, as diagrammed in (a) or they may connect to one another through a short corridor, as diagrammed in (b).

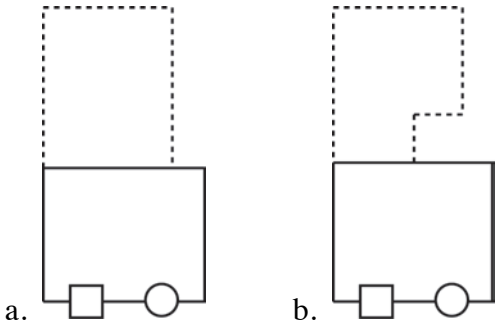


Figure 3 Block configurations

ii) The main block of a rowhouse is two or three bays wide. A bay, in this context, is defined by a single window or door on the front façade. See Figure 4. In a two-bay-wide house, as diagrammed in (a), the front door enters directly into a parlor. In a three-bay-wide house, as diagrammed in (b), the front door enters into a hallway, which is directly adjacent to a parlor.

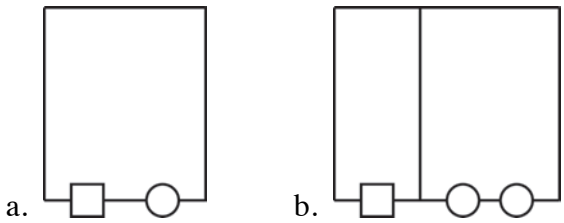


Figure 4 Width configuration

iii) The main block of a rowhouse is either one or two rooms deep. See Figure 5. In a two-room-deep main block, as diagrammed in (a), the front room is a parlor and the back room is a dining room. In a one-room-deep main block, as diagrammed in (b), the parlor may serve as a dining room.

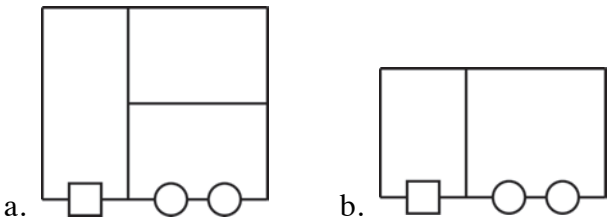


Figure 5 Depth configuration

iv) The main staircase exists in an assortment of locations within a rowhouse. These are diagrammed in Figure 6: in the parlor toward the back of the house; in the dining room toward the front of the house; between the dining and parlor; in the hallway occupying its entire width; in the hallway toward the outer side of the house; and in the kitchen block toward the front of the house.

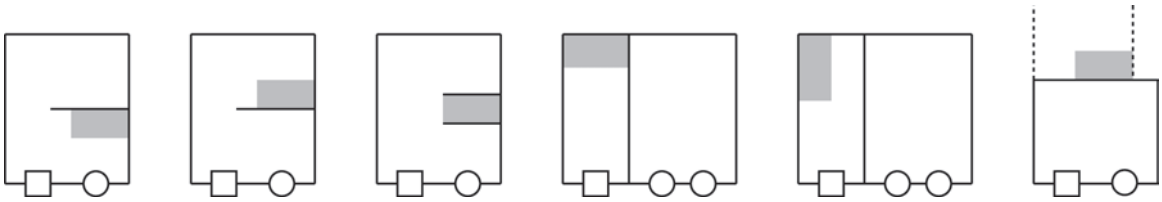


Figure 6 Stair configurations

v) Rowhouses follow an assortment of story and basement configurations: two full stories, but no attic or dormer story; two full stories and a dormer story; two full stories and an attic; with a full basement partially underground; with a full basement entirely underground; and with no basement.

3.3. Identified patterns

0 shows the patterns that were identified. Of the different patterns visible within a rowhouse, stairs present the most intriguing set of combinations. In general, stairs exist in a distinct space that can take one of two forms, a literal room, separated

Pattern	**Illustration in Figure 2**
Style	
Federal	a, b, j, k
Greek Revival	c, g
Italianate	e, f, h
Division between main block and kitchen block	
Front and back portions connected by mutual wall – more common	a~f, i~k
Front and back portions connected by a corridor – less common	g, h
Isolated front portion – relatively rare	
Overall width	
Two bays width – more common	a~e, g, i, j
Three bays width – less common	f, h, k
Entryway configuration	
Enter into parlor – all two-bay-wide houses follow this pattern	a~e, g, i, j
Enter into dedicated hallway that runs full depth of front block	h, k
Enter into dedicated hallway that runs partial depth of front block	f
Location of dedicated dining room	
In main block – more common	a~f
In kitchen block – less common	g, h
Depth of front portion	
One space deep –parlor (g,h) or combined parlor dining room (i-k)	g, h, i~k
Two spaces deep –parlor and dining room	a~c
Three spaces deep –parlor, dedicated stair, and dining room	d~f
Stair location	
In the front division	
On the other side of the front entrance	
Between the separate parlor and dining room	d~f
Within combined parlor dining room, toward the back	i
Within separate dining room, toward the front	a~c
On the same side of the front entrance	
In the hallway	k
In the back division	
Within the kitchen, toward the front	j
In the connection between front and back	
On the same side as the front entrance	g, h
Stair shape	
U-shaped	a~f, i
L-shaped	g, h, j
Straight, bound by a wall on one side	k
Above-ground floor variations	
Two stories	d, e
Two full stories and a ‘half’ dormer storey	a, b, j, k
Two full stories and an attic	c, i
Three stories	g, h

Figure 7 Patterns in the Baltimore Rowhouse

from other rooms by walls, or in a 'phenomenal' room, which exists within a literal room and is defined by the stair itself. Within the representations illustrated in Figure 2, the boundaries of phenomenal rooms are designated by dotted lines.

3.4. The Baltimore Rowhouse grammar [(2)]

The Rowhouse grammar comprises 52 shape rules that generate first floor configurations with features of stairs, fireplaces, windows, exterior doors and interior doors. The shape rules are given in the Appendix. It should be noted that the shape grammar description contained therein is nonstandard. There is redundancy in the grammar. For this we make no apologies, as our ultimate objective is the implementation of a grammar as a generative device, more so, than for its value as an explanatory device. The derivation for 236 East Montgomery Street is shown in Figure 8.

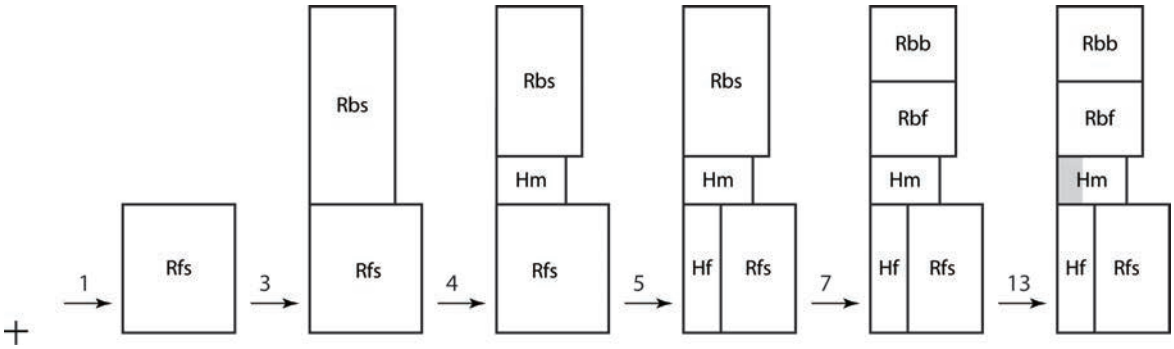


Figure 8 Derivation of 236 East Montgomery Street by the rowhouse grammar

4 A tractable rowhouse grammar

A shape grammar is not tractable without explicitly quantifying conditions on parameters. Quantification eliminates the kind of ambiguity necessary for implementation. Even when shape rules are well quantified, shape recognition may be computationally

[(2)] The Baltimore Rowhouse grammar described in this paper is derived from a version developed by Casey Hickerson, a member of the AutoPILOT project team.

intractable. The sub-frameworks (Yue and Krishnamurti, 2014) offer a way of ensuring tractability of the grammar. As with any traditional shape grammar the focus here is on generating all possible designs without necessarily fully specifying all conditions under which shape rules apply. This is particularly evident when shape grammars are applied in specific situations. Desired features posit constraints over possible designs; this further posits constraints on which shape rules apply when comparing the conditions on the shape rules against the constraints on a current configuration.

A tractable encoding of the rowhouse grammar enables effective and efficient use of the grammar. To distinguish, the original and tractable versions of the grammar are respectively referred to as the *old* and *new rowhouse grammar*. In making the old grammar new, for ease, we consider only a subset of the corpus, namely, working-class rowhouses, in the process excluding large luxurious rowhouses, which were included in the original old grammar. Unlike their luxurious counterparts a working-class rowhouse usually has a unique staircase on the first floor (Hayward, 1981). All rowhouses in Figure 1 fall into this category.

4.1. An application context

The shape grammar interpreter was originally developed for a specific problem context (Yue, 2009), namely, to determine the interior layout of a building given three pieces of information: i) the footprint and number of stories of a building; ii) a reasonably complete set of exterior features, e.g., windows, chimneys and surrounding buildings; and iii) a shape grammar, which describes the building style. Clearly, the implementation of a grammar interpreter is essential to solving this problem. For the remainder of this paper we consider the grammar interpreter in the context of this problem.

For the specific problem context, not all rules were needed and thus, were not encoded, although the general approach itself does not preclude any rule. The features

that are assumed given *a priori* such as windows and exterior doors dictate which rules are relevant. Thus, shape rules to generate windows and exterior doors are not needed and hence, were not considered. For convenience, as the mechanism to generate a fireplace is essentially identical to that of generating an interior door or staircase, tractable fireplace rules are omitted. The new tractable shape grammar is constrained to the allowable transformations, which for the grammar are translation, horizontal reflection, and a combination of the two. Shape rule application is sequential.

4.2. New tractable shape grammar for the Baltimore Rowhouse

The encoded new tractable shape grammar is based on the rectangular framework (Yue and Krishnamurti, 2014) and comprises five phases: block generation: rules (1~4); space generation: rules (5~10); stair generation: rules (11~16); space modification: rules (17~20); and interior door generation: rules (21~26). We describe the rule encodings for each phase.

Initial Shape

Figure 9 shows two possible initial shapes [(3)] that lead to either a two-block or three-block rowhouse design. A set of input dimensions describes the basic building footprint, which is given as a list of rectangular blocks. All lines are aligned to the *X*- or *Y*-directions. The line at the bottom corresponds to the front of the building.

[(3)] From pre-processing the feature input. See Section 5 for more details.

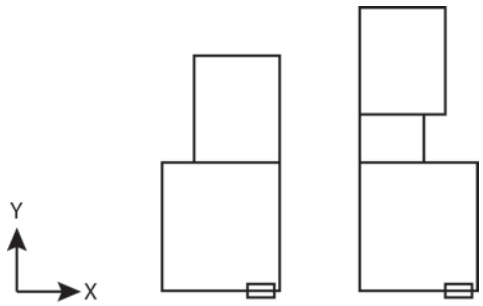
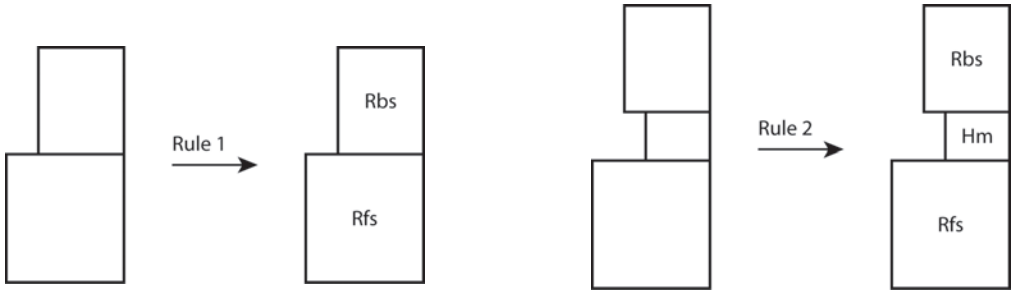


Figure 9 Initial shapes

Block Generation

Rules 1 and 2 assign names to the front, back, and middle blocks. createRoom is a meta-language function within the rectangular framework (Yue and Krishnamurti, 2014).




```
// Rule 1
if (numOfBlocks() == 2) {
        createRoom( rect = getBackBlock(), name = 'Rbs' )
        createRoom( rect = getFrontBlock(), name = 'Rfs' )
}
```

```
// Rule 2
if (numOfBlocks() == 3) {
        createRoom( rect = getBackBlock(), name = 'Rbs' )
        createRoom( rect = getMidBlock(),  name = 'Hm' )
        createRoom( rect = getFrontBlock(), name = 'Rfs' )
}
```

Figure 10 Naming the blocks: rules 1 and 2

Rowhouse blocks are either left- or right- aligned, which is captured by the Boolean, *isRightAligned*. Likewise, the front door being to the right is captured by the Boolean, *isFrontDoorRight.* These attributes are set by rules 3 and 4 respectively.

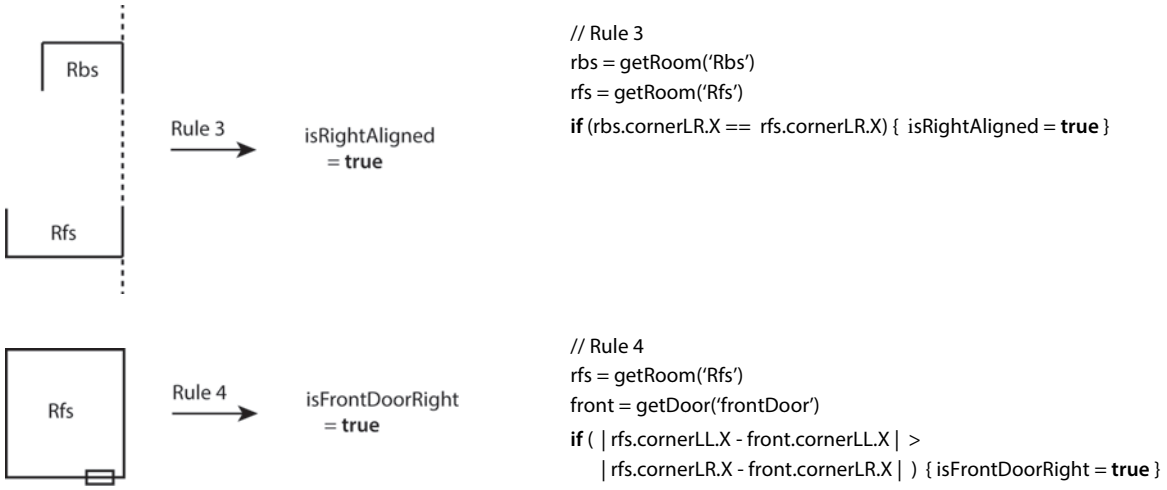


Figure 11 Right aligning rowhouse blocks: rules 3 and 4

Space Generation

The front block is divided into two public rooms as shown in rules 5 and 6. Additionally, if block depth permits (≥ 29'-4"), a staircase area is introduced (rule 5).

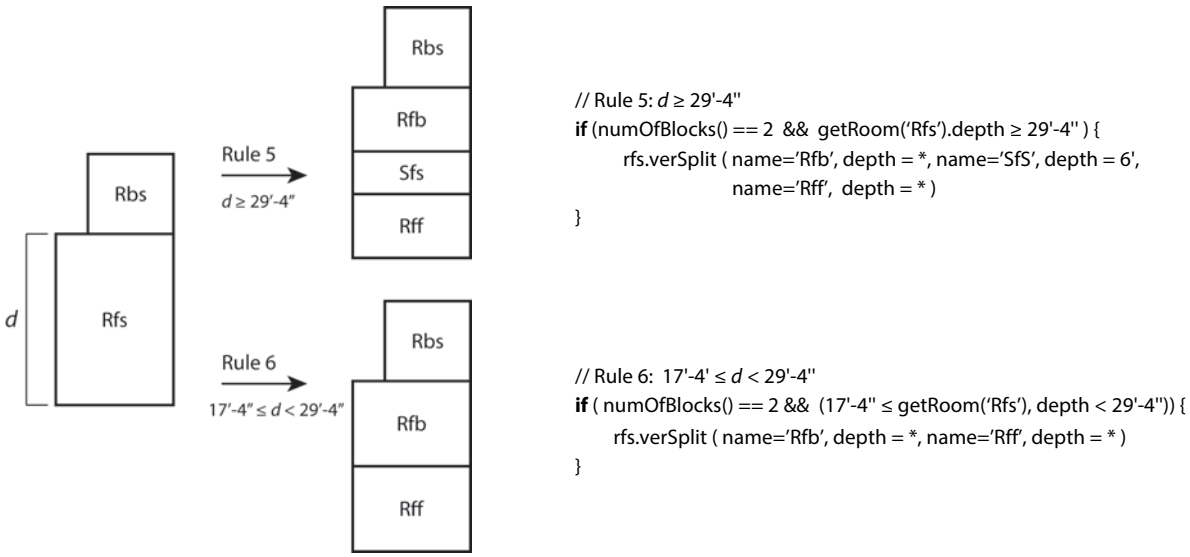
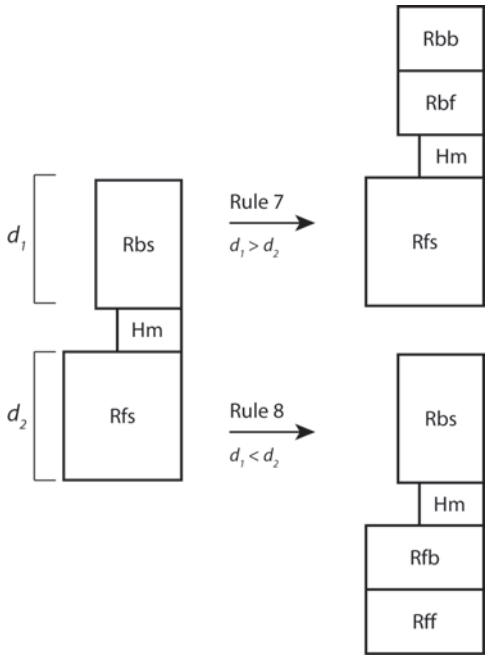


Figure 12 Creating public rooms in the front in a 2-block layout: rules 5 and 6

Likewise, in a three-block design, the front or back block is divided into two rooms depending on which has more depth. That is, combined depth of the subdivided room is always larger than undivided room. These are captured by rules 7 and 8.

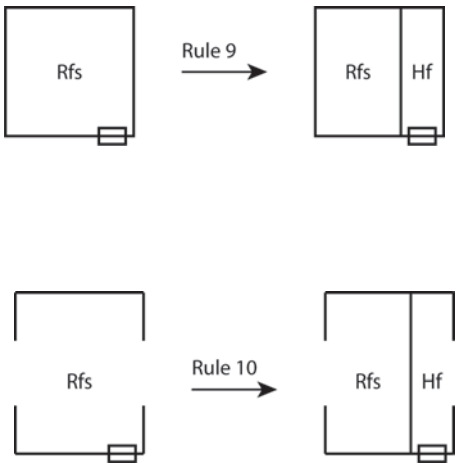



```
// Rule 7: d1 > d2
if ( numOfRooms() == 3 &&
    getRoom('Rbs').depth > getRoom('Rfs').depth ) {
        rbs.verSplit ( name='Rbb', depth =*, name='Rbf',  depth =* )
}
```

```
// Rule 8: d1 <  d2
if ( numOfRooms() == 3 &&
    getRoom('Rbs').depth < getRoom('Rfs').depth ) {
        rbs.verSplit ( name='Rfb', depth =*, name='Rff',  depth =* )
}
```

Figure 13 Creating public rooms in a 3-block layout: rules 7 and 8

Rules 9 and 10 add a hallway centered about the front door provided the front block is 3-bays wide and the front space has not yet been divided.




```
// Rule 9
if (numberOfBays() == 3  &&  roomExists('Rfs'))  {
      rfs = getRoom('Rfs')
      w = hallwayWidth( getDoor('frontDoor'), rfs)
      rfs.horSplit ( name='rfs', width=*, name='Hf',  width=w )
 }

// Rule 10
if (numberOfBays() == 3  &&  roomExists('Rfs'))  {
      rooms = getRoomsBetween('Rfb', 'Rff')
      w = hallwayWidth( getDoor('frontDoor'), getRoom('Rfs'))
      foreach (room in rooms) {
            room.horSplit( name=room.getName, width=*,
                        name='tmp', width=w)
            hf.merge(getRoom('tmp))
      }
      hf.name('Hf')
}
```

Figure 14 Adding a hallway centered about the front door: rules 9 and 10

It is instructive to note that the shape rules of the new shape grammar quantitatively specify the conditions that apply to the configuration or to the rule. Some conditions are

straightforward, for example, the number of spaces in terms of blocks (rules 1 and 2), a value in a specific range (rules 5 and 6), and a relationship of two or more values (rules 7 and 8). Others require not only reasoning based on common design knowledge, but also certain threshold values, statistically determined. The following illustrates the complexity, using as exemplars, the rules for generating staircases.

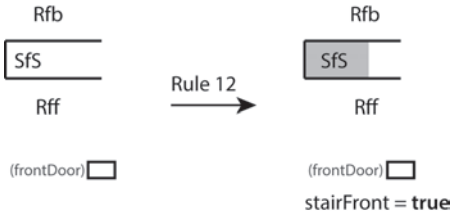
Staircase Generation

Figure 15 gives the staircase generation rules (11~16). The rules are not necessarily mutually exclusive. For example, for layouts with room *Rfs* and *Rbs*, where no exclusive condition has been specified as to when to apply each rule, both rules 11 and 16 are applicable. Since we are considering only the working-class rowhouse, each has a single staircase on its first floor. Therefore, for each layout, just one of the shape rules for generating staircases applies and only once.

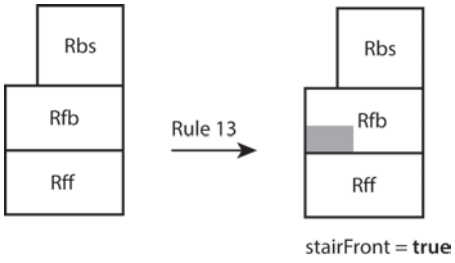
If there is a staircase room labeled *SfS*, then rule 12 applies. As a result, an implicit condition for Rule 11, 13, 14, 15, and 16 is that the current layout has no staircase room labeled *SfS*.

Rule 14 adds a staircase to a hallway. Obviously, the hallway needs to be wide enough to hold the staircase, hence the width of the front block. From the samples (see Figure 16), 18 ft is a good threshold value to distinguish whether or not rule 14 can apply. To ensure the exclusive application of rule 14, an implicit condition for rules 11, 13, 15 and 16 is that the width of the front block is smaller or equal to 18 ft.

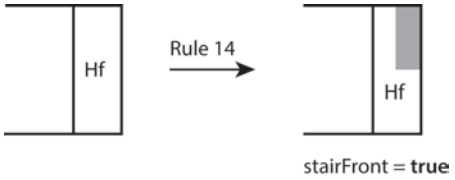

```
// Rule 11
if ( not stairExists() &&  roomExists('Rfs') &&
    roomNotExist(['SfS','Rfb', Hm']) && getFrontBlock().width ≤ 18' &&
    getKitchenArea() ≤ 130 ) {
        room ('Rfs').addStaircase ( position='crossFrontDoor',
                width=4, depth=6, getFrontDoor() )
        stairFront = true
}
```



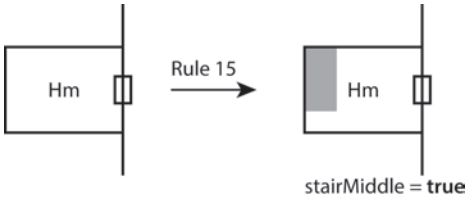

```
// Rule 12
if (not stairExists()  &&  not roomExists('SfS') &&
    roomExists('Rfb') &&  getFrontBlock().width ≤ 18') {
        room('Rfb').addStaircase( position = 'bottom&crossFrontDoor',
                width = 6, depth = 4, getFrontDoor())
        stairFront = true
}
```



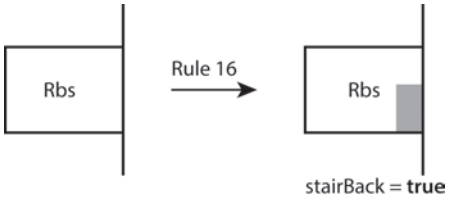

```
// Rule 13
if (not stairExists()  &&  not roomExists('SfS') &&
    roomExists('Rfb') &&  getFrontBlock().width ≤ 18') {
        room('Rfb').addStaircase( position = 'bottom&crossFrontDoor',
                width = 6,  depth = 4, getFrontDoor())
        stairFront = true
}
```




```
// Rule 14
if (not stairExists() && roomExists('Hf') &&
   not roomExists('SfS') &&  getFrontBlock().width > 18' )  {
        room('Hf').addStaircase( position = 'nextToExterior',
                width = room('Hf').width/2)
        stairFront = true
}
```




```
// Rule 15
if (not stairExists() && roomExists('Hm') &&
    roomsNotExist(['SfS', 'Rfb']) && getFrontBlock().width ≤ 18') {
        room('Hm').addStaircase(position = 'straightSide',
                depth = room('Hm').depth)
        stairMiddle=true
}
```




```
// Rule 16
if (not stairExists()  &&  roomExists('Rbs') &&
    roomsNotExist(['SfS', 'Rfb', 'Rfb']) && getFrontBlock().width ≤ 18'
    && getKitchenArea() > 130) {
        room('Rbs').addStaircase( position = 'bottom&crossFrontDoor',
                width = 4,  depth = 6, getFrontDoor() )
        stairBack= true
}
```

Figure 15 Staircase generation rules 11~16

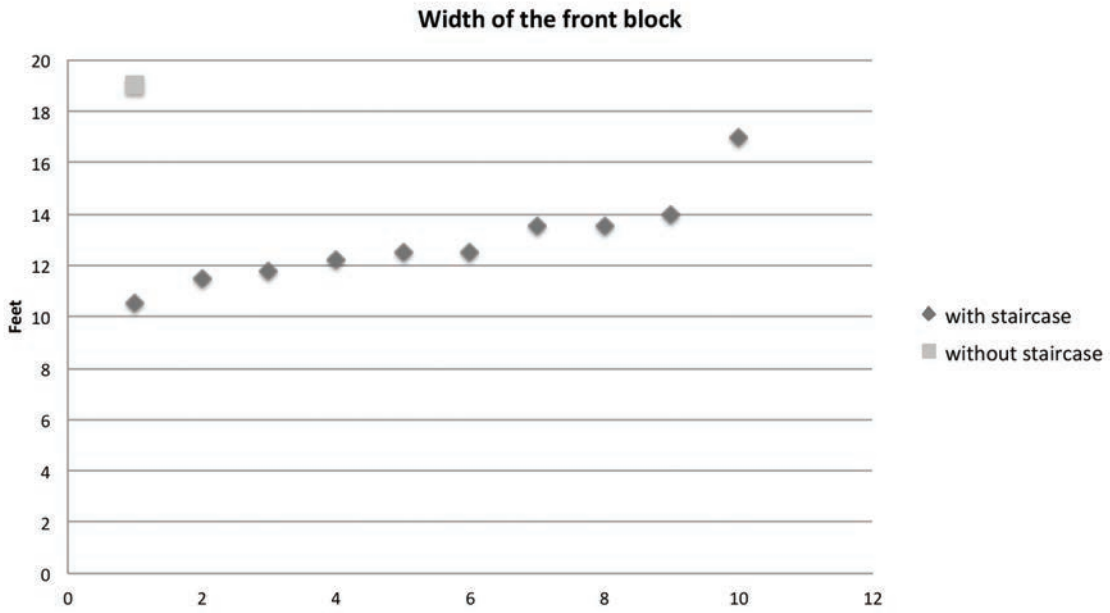


Figure 16 Quantifying the shape rules generating staircases

For rules 11, 13, 15, and 16, if there is an *Rfb* room in the layout, then rule 13 should be applied to add a staircase there. Accordingly, an implicit condition for rule 11, 15 and 16 is that there is no *Rfb* room. For rules 11, 15, and 16, if there is a middle block *Hm*, rule 15 should be applied to add a staircase in the middle block. Thus, an implicit condition for rules 11 and 16 is that there is no *Hm* room.

It remains to distinguish between rules 11 and 16. The implicit conditions added by rules 12, 13, 14, and 15 can be summarized as: if there are only a *Rfs* room (the front block) and a *Rbs* room (the black block) in the current layout, then possibly rules 11 and 16 can be applied. Rule 16 adds a staircase to an *Rbs* room, which is actually a kitchen. Therefore, the kitchen space has to be large enough to hold a staircase as well as function as a kitchen.

In the sample available, only one uses rule 11 and one uses rule 16. Because of this, the related statistical data for all samples is computed as a reference: the average area of kitchens without a staircase is 127.7 ft^2, the minimum is 92.8 ft^2, and the maximum is

185.4 ft^2. The area of a staircase is about 26~30 ft^2. The kitchen area of the case that uses rule 11 is 94.4 ft^2, and the kitchen area of the case that uses rule 16 is 165.5 ft^2. The average of these two cases is about 130 ft^2, which is close to the average of kitchens without staircases. So, 130 ft^2 is used as the threshold value. As a result, an added condition for rule 16 is that the area of kitchen is greater than 130 ft^2. An additional condition for rule 11 is that the area of the kitchen is smaller or equal to 130 ft^2. Figure 17 gives a summary of implicit conditions to make rules for generating staircases exclusive.

	Rule 12	*Rule 14*	*Rule 13*	*Rule 15*	*Rule 11*	*Rule 16*
Rule 12	With 'SfS'	No 'SfS'	No 'SfS'	No 'SfS'	No 'SfS'	No 'SfS'
Rule 14		Front block width > 18'	Front block width ≤ 18'	Front block width ≤ 18'	Front block width ≤ 18'	Front block width ≤ 18'
Rule 13			With 'Rfb'	No 'Rfb'	No 'Rfb'	No 'Rfb'
Rule 15				With 'Hm'	No 'Hm'	No 'Hm'
Rule 16					Kitchen ≤ 130 ft^2	Kitchen > 130 ft^2

Figure 17 Implicit conditions to make staircase rules exclusive

Space Modification

These rules create openings. Rules 17 and 18 open shared walls between the staircase area and neighboring rooms, respectively the front hallway or public rooms in the front, in which case there is no hallway sharing a wall with the stair area. Rules 19 and 20 create an opening between the front hallway and the middle or back blocks.

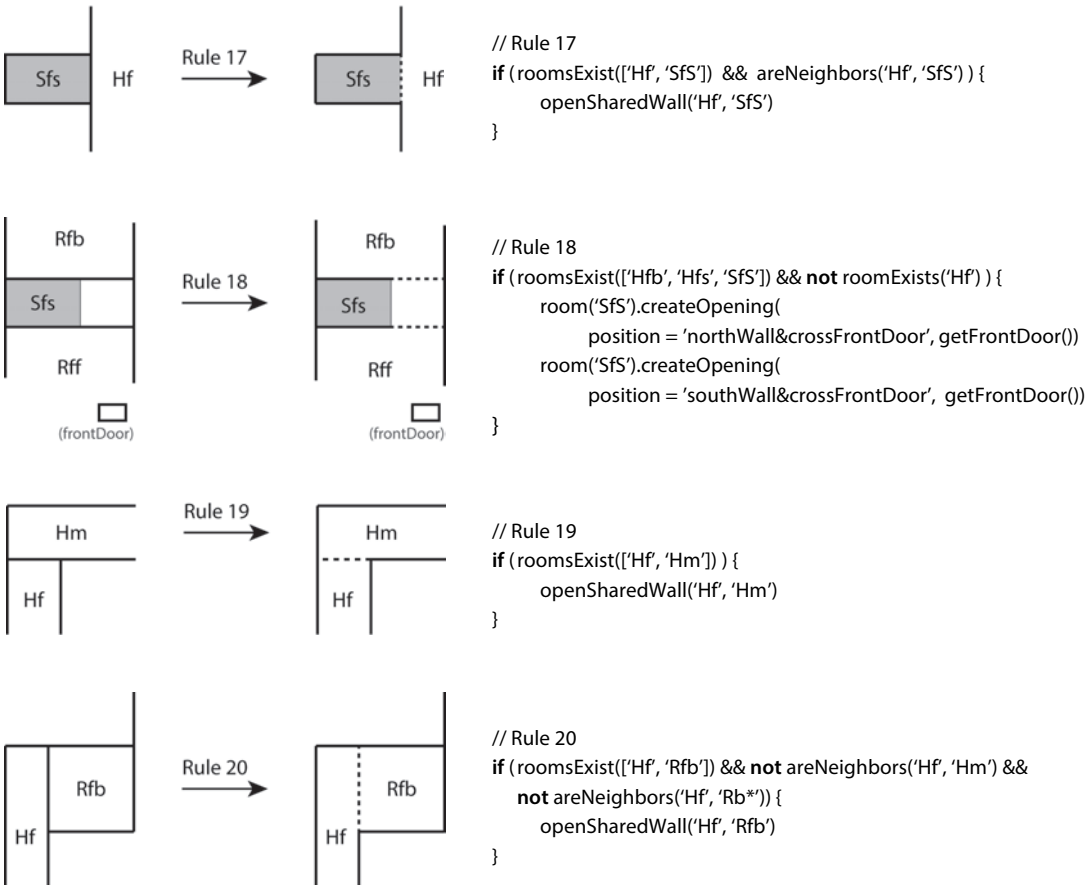
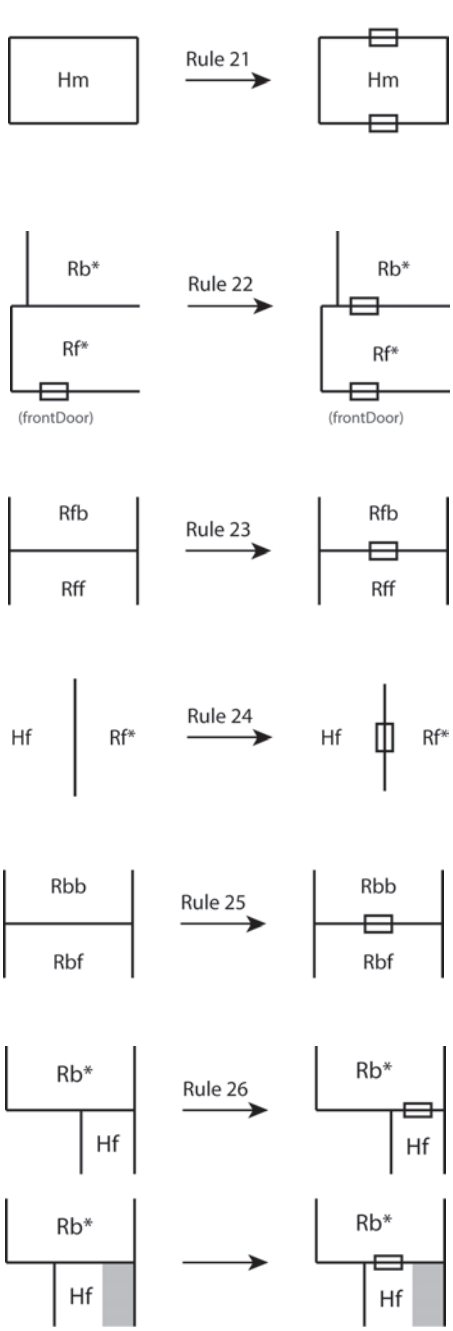


Figure 18 Space modification rules

Interior Door Generation

Rule 21 adds doors to the front and back of the hallway in the middle block. Implicitly, this hallway must exist and its shared walls are nonempty, that is, without doors. This is implied by the Boolean attribute 'connected', the value of which is returned by the 'get' function. Implicit with the connected' attribute is the fact that when true the shared wall is non-empty and has a door. Additionally, the wall must be at least 3' wide in order to insert a doorway. The remaining rules introduce openings in a shared wall between two disconnected areas. Wall length of at least 3' is implicitly guaranteed. Rule 26 has two variants one of which considers the situation when there is a staircase in the hallway, which affects the placement of the door.




```
// Rule 21
if ( not get('connected') && roomExists('Hm') ) {
        a = sharedWall('Hm', 'Rf*')
        b = sharedWall('Hm', 'Rb*')
        if ( a.type() ≠ EMPTY &&  a.length() ≥ 3') {  a.addMidDoor() }
        if ( b.type() ≠ EMPTY && b.length() ≥ 3' ) { b.addMidDoor() }
}
```

```
// Rule 22
if ( not get('connected') && not roomExists('Hm') &&
    (not roomExists('Hf') || ( roomExists('Hf') &&
     not areNeighbors('Hf', 'Rb*'))) {
        sharedWall('Rb*','Rf*').addSideDoor(position= 'frontDoorSide',
                getFrontDoor() )
 }
```

```
// Rule 23
if ( not get('connected') &&  roomsExist(['Rff', 'Rfb']) &&
     areNeighbor('Rfb', 'Rff') ) {
        sharedWall('Rfb', 'Rff').addMidDoor()
}
```

```
// rule 24
if (not get('connected') && roomExists('Hf'))  {
        sharedWall('Hf', 'Rf*).addMidDoor()
}
```

```
// Rule 25
if ( not get('connected') && roomsExist(['Rbf', 'Rbb'])  {
        sharedWall('Rbf', 'Rbb').addMidDoor()
}
```

```
// Rule 26
if ( not get('connected') && roomsExist(['Rb*', 'Hf']) {
        sharedWall('Rb*', 'Hf').addDoor()
}
```

Figure 19 Interior door insertion rules

Figure 20 shows the same derivation as Figure 8 using the rules of the new rowhouse grammar.

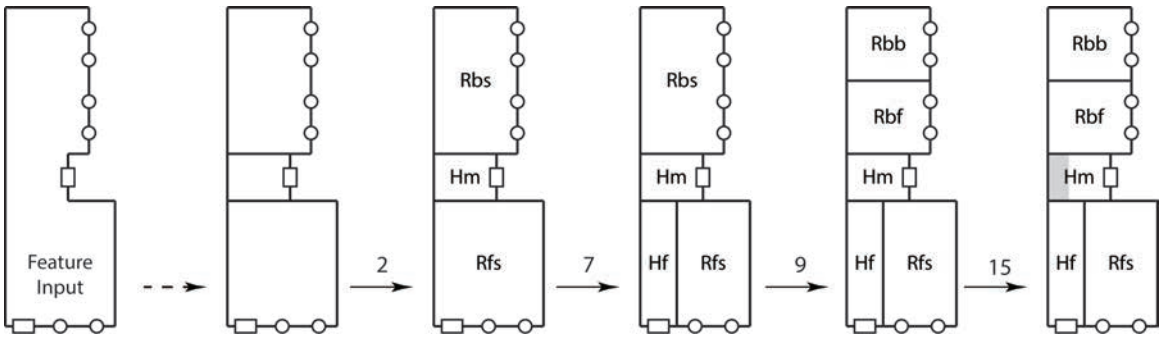


Figure 20 Derivation of 236 East Montgomery Street by the new rowhouse grammar

5 Implementation

A computer implementation of a shape grammar essentially enumerates all possible designs in the language of the grammar. This enumeration, alternatively, the derivation structure of the shape grammar, can be viewed as a tree structure. Valid designs correspond to specific nodes of the tree. Such nodes are mostly leaf nodes, although certain internal nodes may correspond to possible designs—an internal node usually corresponds to a design of a smaller size perhaps with unresolved labels or markers, whilst a leaf node represents a finished design.

Closer examination of the rules indicates that, as a pure shape grammar, there are parameters that would need to be set, typically, at the start. Note that the initial shape corresponds to one of two kinds. For example, in the case of the problem of determining the layout from footprint, this is specified by the feature input. That is, the depth and width of the initial shape are parameters that are feature or user specified. Moreover, there are block generation rules to indicate whether designs are two- or three-blocks deep. When the rules are applied exhaustively, a shape grammar generates, as a tree, the entire layout space for the building style. See Figure 21.

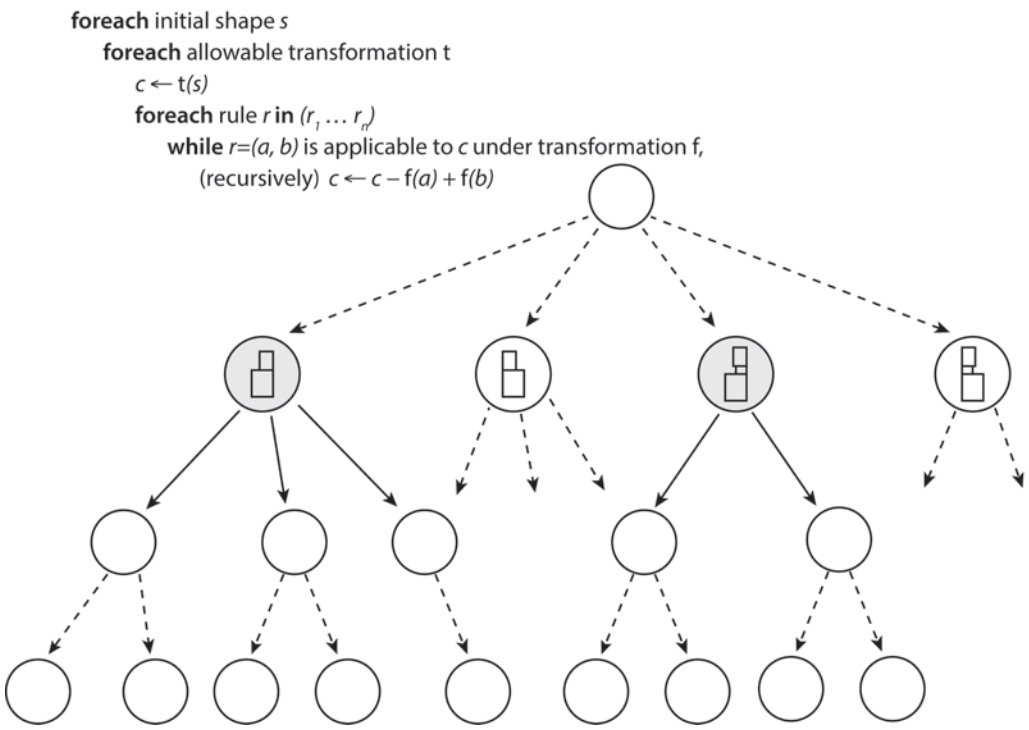


Figure 21 The layout tree of the Baltimore Rowhouse grammar

The depth and width dimensions and block type are part of the input—whether feature or user supplied—and therefore, are considered as constraints. For layout determination, in order to estimate an interior layout, we have to establish the connection between the design space of the grammar and input features so that designs consistent with the input features can be 'picked out'. The approach begins with an initial layout estimate based on the constraints given by the input. Spatial and topological constraints from this estimate are then employed to prune the layout tree, and 'fix' possible open terms in the current configuration. The layouts that remain correspond to possible required layouts.

The initial shape is a shape from the pre-processing of the features input instead of a point on a two-dimensional Cartesian coordinate system as implied by the grammar—in particular, it is a basic footprint with or without augmentation by windows and doors. To be exact, the initial shape contains a list of rectangular blocks, as well as bounds on

windows and doors. Such an initial shape helps avoid the complexity of pruning and fixing the underlying layout tree. In the sequel, tree pruning and initial layout estimation are discussed. The building input features for the Baltimore Rowhouse used in the description below were taken directly from existing drawings in (Hayward, 1981).

5.1. Space subdivision tree and the Baltimore Rowhouse

After applying the shape rules for several steps from the initial shape, the layout must be one of two shaded nodes or a horizontal reflection of the two shown in Figure 21. On the other hand, we can achieve the same results by decomposing the input into rectangles using space subdivision. See Figure 22.

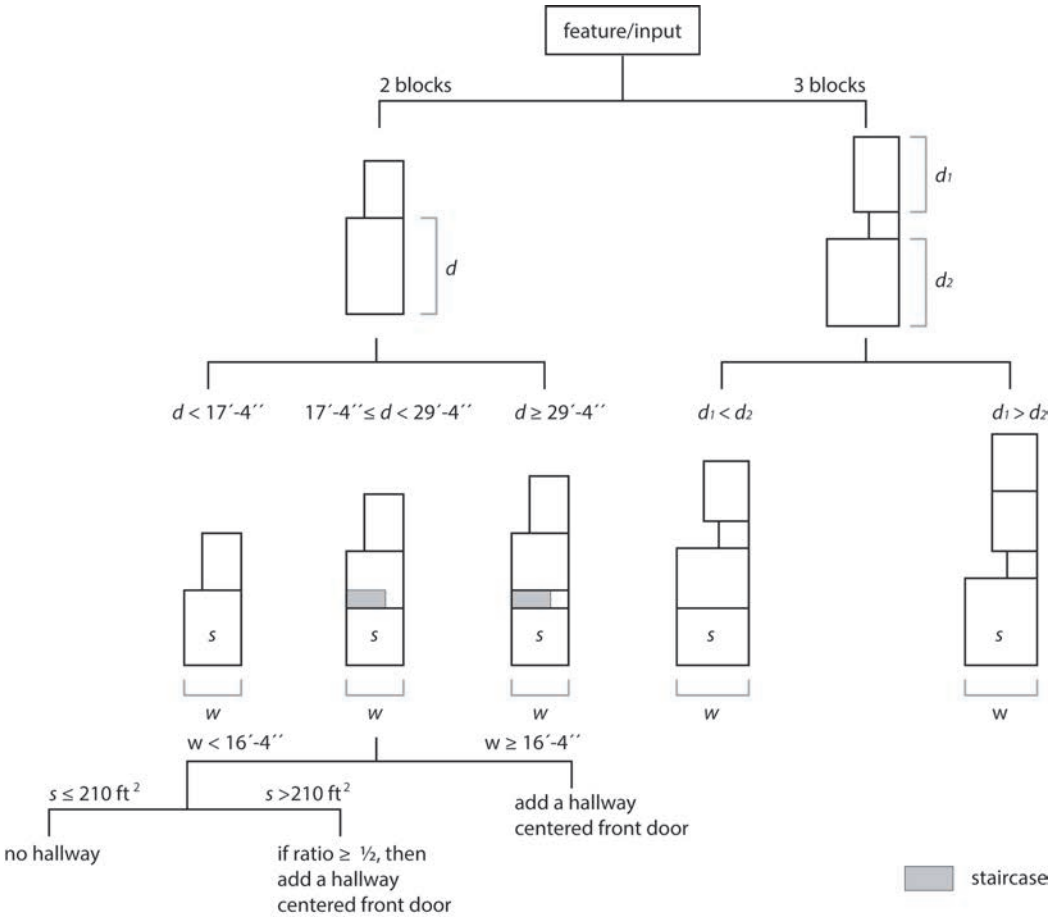


Figure 22 Space subdivision tree for the Baltimore Rowhouse

The first floor is typically decomposed into two or three rectangular blocks: a block containing a parlor towards the front, a block containing a kitchen towards the rear, and an optional, smaller central block that connects the two. In a three-block rowhouse, the central block contains a pantry or a stair, while the front and rear blocks are divided into one or two rooms. The kitchen is always the rear-most space while the parlor is the front-most space. The dining room usually appears in the front block behind the parlor or in the rear block forward of the kitchen. The two cases can be distinguished by comparing the depths of the front (d_2) and rear (d_1) blocks.

Two-block rowhouses are more involved. Depending on the depth (d) of the front block, it can contain a single room, or be divided into a parlor and dining room possibly separated by a staircase. If the front block comprises two rooms, the staircase can occupy an enclosed space or it can be open to one or both rooms. If the front block comprises a single room, the staircase may have multiple possible arrangements. These configurations are too complicated to be handled by the decision tree, which needs further refinement by using shape rules.

Regardless of whether the layout has two or three blocks, the front door enters into the front-most room or a dedicated hallway. This is determined from the width (w) and area (s) of the front-most room. Layout determination is a process of 'picking up' nodes from the layout tree that are consistent with input features. Pickup is typically achieved by tree pruning—eliminating nodes inconsistent with certain constraints with the remainder being the desired results. That is, variables (aka parameters) in the intermediate configurations have to be 'fixed' to match the input features at a certain stage. Parameters can be fixed at this step, and the desired layouts are then obtained simply by continued application of the shape rules. For the new tractable version, the grammar is designed to start from the rectangular decomposition of the footprint input so

that the parameter-fixing step is automatically handled. This situation illustrates the trade-off between pure shape rule application and the practicalities of problem solving.

5.2. Layout generation for the Baltimore Rowhouse

Layout generation is carried out in a single step. That is, layout generation becomes, simply, rule application on layouts resulting from the initial estimated layout. Figure 23 shows the screenshot of the computer implementation. On top, the left-side window shows a list of Baltimore rowhouses from a database, and the right-side window shows the shape rules. At the bottom, from left to right, the first window shows the tree structure of shape rule application. There is always at least one path in this window. By selecting an entry in the tree structure, the corresponding shape rule applied is highlighted in the shape rule window. The second, third and fourth windows respectively show the true layout, generated layout, and feature input. The rightmost window provides a three-dimensional view by extruding the two-dimensional generated layout using default values.

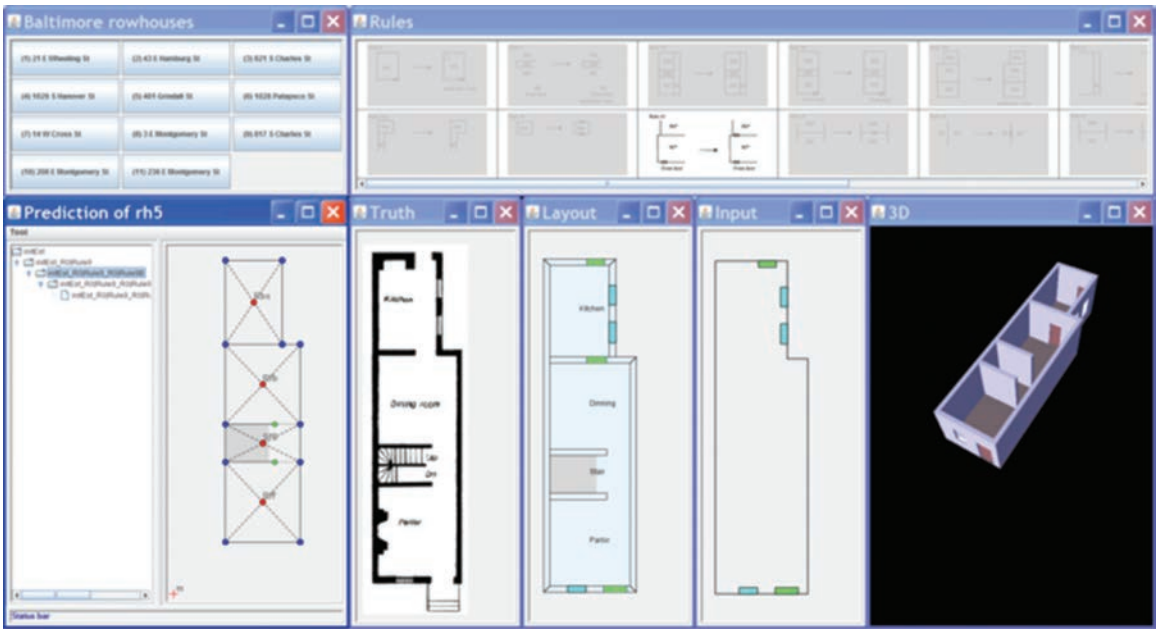


Figure 23 Screenshot of layout determination of the Baltimore Rowhouse

Figure 24 shows sample results from the layout determination for the Baltimore Rowhouse. For each rowhouse, two layouts are shown: on the left is the ground truth, the other, the generated layout. Additionally a rendered 3D model of the generated layout is given along with the derivation sequence. The efficacy of the approach is indicated by how nearly identical to the actual layout the generated layout is.

Figure 24 Layout results for the Baltimore Rowhouse

(Shown in order of ground truth, generated layout, and 3d model)

The layouts shown in Figures Figure 23 and Figure 24 all employ the following terminating labeling rules: Hm → Entry, Hf → Hallway, Rfs → Parlor, Rff → Parlor, Rfb → Dining, Rbf → Dining, Rbs → Kitchen, Rbb → Kitchen and SfS → Stair.

6 Discussion

We have described a strategy for developing and implementing a tractable shape grammar, based on the fact that the derivation of the language space of a shape grammar can be represented as a tree structure. Admittedly, the grammar is relatively simple, the rules conditioned by pragmatic considerations, and is context driven. Nonetheless, the implementation is based upon a framework, which comprises a data structure, underlying manipulation algorithms and a meta-language for specifying shape rules. Each framework offers a uniform approach to developing interpreters for a class of tractable shape grammars. In this paper, we have illustrated an implementation over the rectangular framework by developing a simple shape grammar for the Baltimore Rowhouse, and then encoding the rules so that the grammar is amenable to implementation.

In general it is neither essential to develop a completely new grammar nor does the grammar description have to be in the nonstandard form employed in this paper. Likewise, flexibility of the rules is not an essential for a grammar interpreter; that depends upon the nature of the application. On the other hand it is possible to take an existing and flexible shape grammar in standard description, for example, the Queen Anne grammar (Flemming, 1987), make it tractable, ready for implementation.

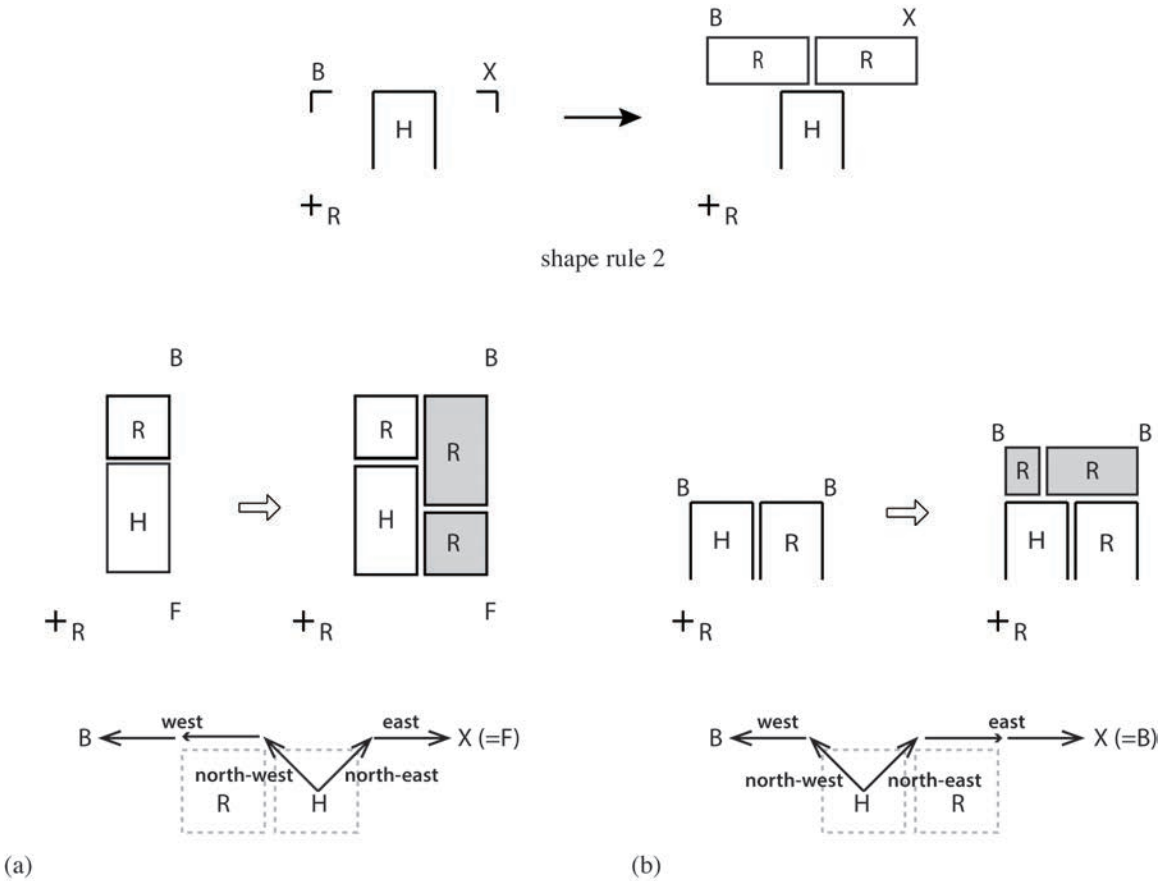
The Queen Anne House

Prior to developing the Rowhouse grammar, we tested the framework on the Queen Anne House. Our analysis, implementation and experimentation were limited to the

layout rules, namely, the first fifteen rules in Flemming (1987: Figures 4, 7 and 10). For reasons of tractability, Queen Anne shape rules may require additional constraints to be specified so that different possibilities are clarified.

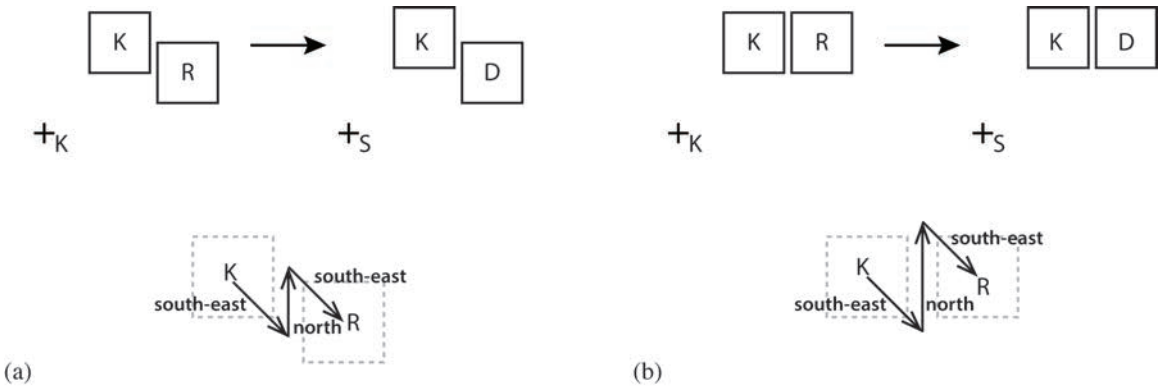
For example, in Figure 25, Rule 2, shown on top, is applicable to shapes (a) and (b). The application of the rule to shape (a) produces a reasonable layout, whereas to shape (b) might produce too small a room. In general, although dimensions are not important in implementing the Queen Anne grammar; yet, in order to eliminate such cases, pertinent understanding of dimension is essential. Rule 2 applies directly to shape (b) without transformation whereas it does so to shape (a) only under a 90º clockwise rotation, or equivalently, under a 90º counterclockwise rotation of the configuration (Kui and Krishnamurti, 2014). As the analysis illustrates, in both cases, label B is in a north-west westerly direction from the hall room node H; likewise, label X (B or F) is in a north-east easterly direction from H, which is captured in the meta-language description.

Another example is in the interpretation of Rule 8 shown in Figure 26(a). It would appear that two rooms have to partially overlap in order for this rule to apply. However, from the sample layouts shown in (Flemming, 1987), it is clear that the rule as shown in Figure 26(b) is also applicable. Other Queen Anne rules are similarly decompacted and implementation simply focuses on the task of coding.




```
// Rule 2: meta-language description
If ((layout.getStatusMarker().isType ('R')) && ( layout.existsRoomNodes ('H') == true))
{
    foreach ( allowableTransformations )
    {
        roomH = curLayout.getRoomNodes ('H');
        if ( lookForMarkerAtCornerAndExtension (roomH, NORTH_WEST, WEST, 'B')  &&
            lookForMarkerAtCornerAndExtension (roomH, NORTH_EAST, EAST, 'X') &&
            anySideOfNewRoomGreaterOrEqual( hallWayWidth )) {
             subDivideTheHallWay ();
        }
    }
}
```

Figure 25 Shape rule 2 (Flemming, 1987: Figure 4)—application, analysis and meta-language description




```
// Rule 8: meta-language description
if ((layout.getStatusMarker().isType ('K')) && (layout.existsRoomNodes ('K') == true))
{
    roomK = curLayout.getRoomNodes ('K');
    foreach ( allowableTransformations )
    {
        roomR =
        roomK.getNeighborNode(SOUTH_EAST).getNeighborNode(NORTH).getNeighborNode(SOUTH_EAST );
        if ((roomR  != null) && roomR.isType('R'))
        {
            roomR.setMarkerName ('D');
            curLayout.getStatusMarker().setMarkerName ('S');
        }
    }
}
```

Figure 26 Interpretation of shape rule 8 (Flemming, 1987: Figure 7)

The interface of the grammar implementation is shown in Figure 27. The top-left panel shows the layout tree generated by applying all the shape rules. The top-right panel is for layout display. When entries in the top-left panel are selected, the corresponding layout is displayed. The bottom panel is the status bar. Above the status bar is the rule panel, displaying the rules for the Queen Anne grammar. When an entry of the layout tree is selected, the current applicable shape rules are highlighted. When a rule is selected a larger display of the rule is shown. In this implementation, a total of 506 unique possible layouts were generated.

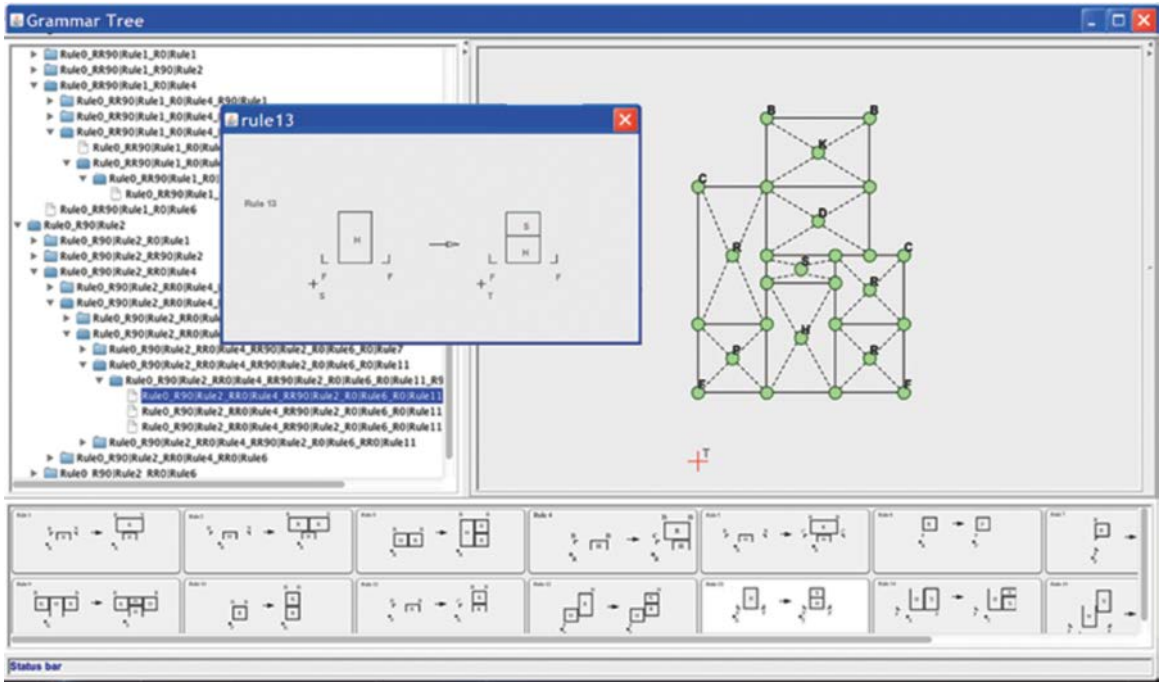


Figure 27 Screenshot of Queen Anne House grammar implementation

The work on Queen Anne Houses is reported in (Yue et al, 2012) albeit the main emphasis of that paper was in exploring artificial intelligence and constraint satisfaction techniques in order to estimate an initial interior layout for actual Queen Anne Houses in Pittsburgh, Pennsylvania based on their exterior features, less so on the details of a grammar implementation.

The successful implementation of Queen Anne rules provided a confident base upon which we set out to develop a tractable shape grammar for the Baltimore Rowhouse. The mainly restrictive less flexible nature of the rules is conditioned by the application context and the rowhouses themselves. Nonetheless, without taking advantage of any specific characteristic of a grammar and its language, the implementation structure is the same for both the Baltimore Rowhouse and Queen Anne grammars, indeed, for any grammar based on the rectangular framework—the essential difference lies in the encoding of the shape rules.

Acknowledgement

This research was supported in part by a grant from US Army Corps of Engineers, Engineer Research and Development Center – Champaign, IL. Any opinions, findings, conclusions or recommendations presented in this paper are those of the authors and do not necessarily reflect the views of CERL.

References

Flemming U (1987) "More than the sum of parts: the grammar of Queen Anne houses" *Environment and Planning B: Planning and Design,* 14**,** 323-350

Gips J, 1974 *Shape* Grammars *and Their Uses* PhD dissertation, Computer Science Department, Stanford University, Stanford, California

Hayward ME (1981) "Urban Vernacular Architecture in Nineteenth-Century Baltimore" *Winterthur Porfolio,* 16**,** 33-63

Hayward ME and Belfoure C (2005) *The Baltimore Rowhouse,* Princeton Architectural Press, New York

Yue K (2009) *Computation-Friendly Shape Grammars: With Application to Determining the Interior Layout of Buildings from Image Data*, PhD Thesis, Architecture, Carnegie Mellon University, September.

Yue K and Krishnamurti R (2013) "Tractable shape grammars" *Environment and Planning B: Planning and Design,* 40(4), 576-594

Yue K and Krishnamurti R (2014) "A paradigm for interpreting tractable shape grammars" *Environment and Planning B: Planning and Design,* 41(1), 110-137

Yue K, Krishnamurti R and Grobler F (2012) "Estimating the Interior Layout of Buildings Using a Shape Grammar to Capture Building Style" *Journal of Computing in Civil Engineering*, 26(1), 113-130

Stiny G, 1975 Pictorial *and formal aspects of shape and shape grammars and aesthetic systems* PhD dissertation, System Science, University of California, Los Angeles, CA

Appendix: The Baltimore Rowhouse Grammar

The shape rules for the Baltimore Rowhouse Grammar are organized into eight phases, progressing from major configurations that constrain the design process to minor configurations that follow logically from other configurations, namely: Block generation rules (1~4); Space generation rules (5~7); Stair generation (rules 8~17); Fireplace generation rules (18~22); Space modification: rules (23~24); Front door and window generation rules (25~29); Middle and back door and window generation rules (30~39); and Interior door generation rules (40~52). See Figure 28.

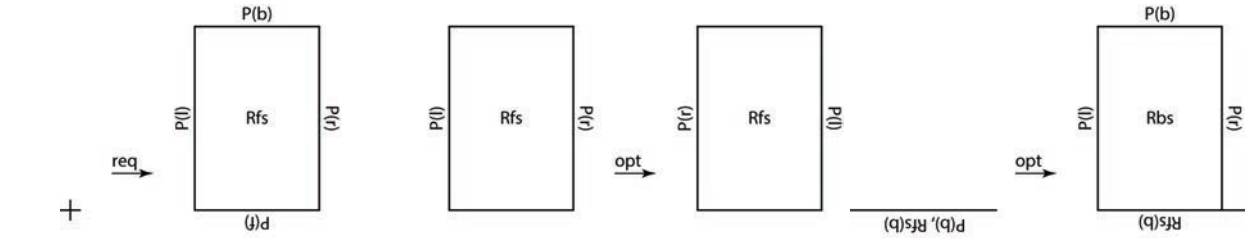
Rule description is nonstandard. Rules are marked as either required (*req*) or optional (*opt*). Required rules are applied *wherever* applicable whilst optional rules are applied at the interpreter's discretion. The decision whether to apply an optional rule directly impacts the overall design—in effect, the final design is determined by the set of optional rules that were applied. Whenever a rule is applied, it is applied exhaustively; that is, the rule is applied to every subshape that matches the rule's left-hand-shape. Moreover, rules are applied in sequence: only after Rule *n* has been applied exhaustively, can Rules *n+1* and greater be applied.

Labels are used in two ways: to control where shape rules may apply, and to ensure that mutually exclusive rules cannot be applied to the same design. Spaces and stairs are labeled with two or three characters that indicate the general location of the space or stair within the house. For instance, *Rfb* indicates a room in the front block of the house that is oriented toward the back, a dining room. Walls are labeled using expressions of the form *x*(*y*) where *x* is a label for a space that the wall bounds and *y* is a one letter code indicating the side of the space the wall defines, namely, *f*(ront), *b*(ack), *l*(eft) and *r*(ight). For example, the front wall of the room labeled *Rfb* is labeled *Rfb*(*f*). Shared walls have multiple labels. Given a space, its wall labels can be easily reconstructed. Wall labels are omitted in the description except when needed or assigned, for example, perimeter walls, which are identified by the letter *P*.

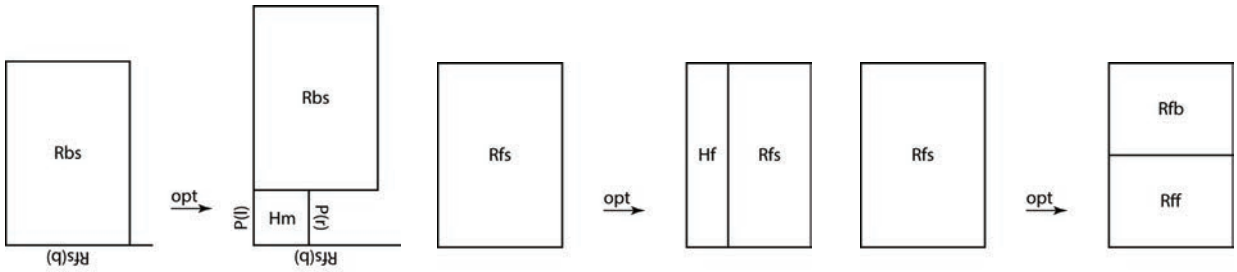
Within some rules, variables are used to match more than one label: the character * matches any string of characters while the string {*x* | *y*} matches the strings *x* or *y*. *Boolean* global labels are used to ensure that mutually exclusive rules are not applied with default value *false*.

Rule 18 ensures that there is at least one fireplace in the front block. Rule 22 ensures that there is always a fireplace in the back block.

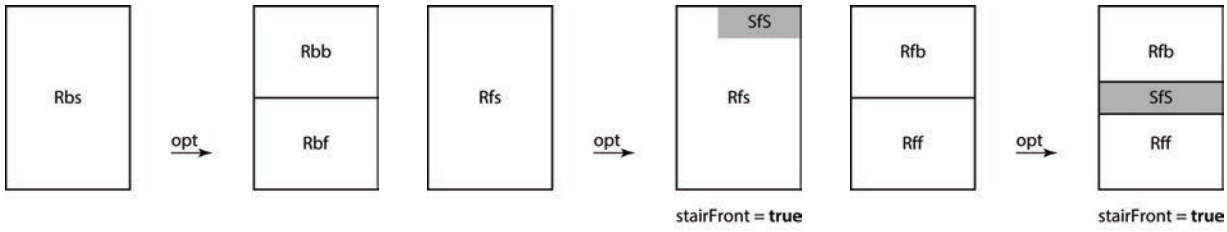
Lastly, we note that rule 46 is applicable only when the front hallway contains a full-width stair and when the front block contains a separate service stair.



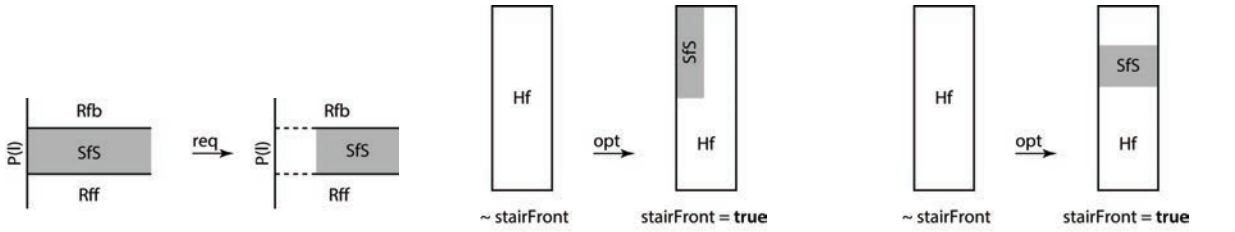
1. Generate the front block
2. Mirror the front block
3. Generate the back block



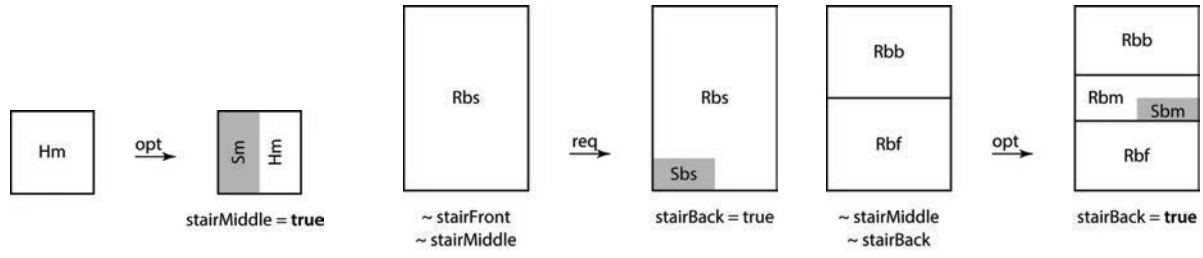
4. Generate the middle block
5. Generate a hallway in the front block
6. Generate two spaces in the front block



7. Generate two spaces in the back block
8. Generate stair at the back wall of a single-spaced front block
9. Generate stair between the two spaces of a double-spaced front block

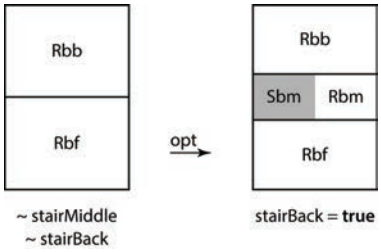


10. Modify the stair generated by Rule 9 if it runs the entire house width
11. Generate partial-width stair in the front hallway
12. Generate full-width stair in the front hallway

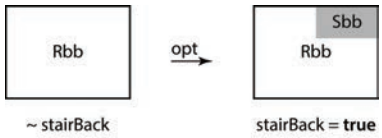


13. Generate stair in the middle block
14. Generate stair at the front of a single-spaced back block
15. Generate partial-width stair between two spaces of a double-spaced back block

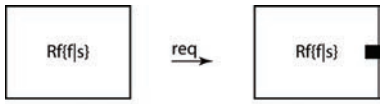
Figure 28 Baltimore Rowhouse Grammar



16. Generate full-width stair between the spaces of a double-spaced back block

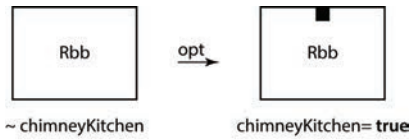


17. Generate accessory stair on the back wall of the back room of a back block

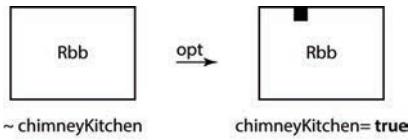


18. Generate required front-block fireplaces

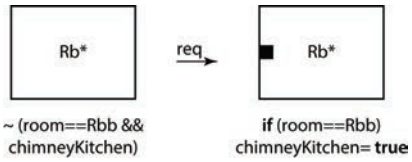
19. Generate optional front-block fireplaces



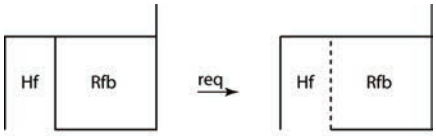
20. Generate back-block fireplaces



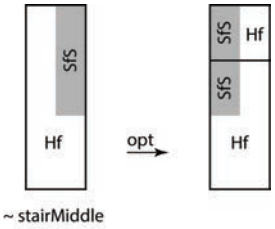
21. Generate back-block fireplaces on the back wall



22. Generate back-block fireplaces on a side wall



23. Modify the back room of a front block if the front hallway does not adjoin the middle or back block



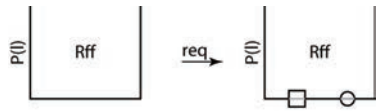
24. Generate a service stair behind a partial-width stair in the front hallway

25. Generate a hall way in the front of the back block, removing the fireplace

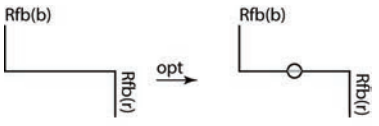
26. Generate the exterior door into front hallway of a three-bay configuration

27. Generate an entry vestibule in the front hallway of a three-bay configuration

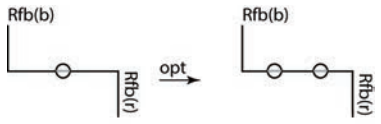
28. Generate the front windows of a three-bay configuration



29. Generate the front door and window for a two-bay configuration

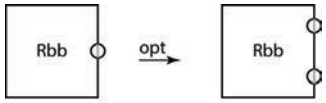


30. Generate a window on the back wall of the front block

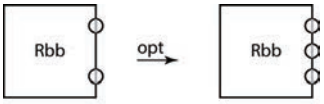


31. Add a second window on the back wall of the front block

32. Generate a window into the back block spaces

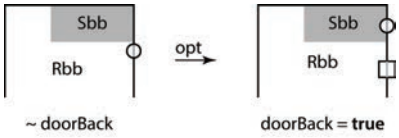


33. Add a second window in the back-block spaces

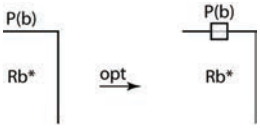


34. Add a third window in the back-block spaces

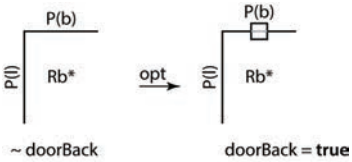
35. Generate an exterior door into the middle block



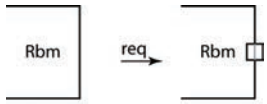
36. Generate an exterior door on the side wall of the back-most space when there is a stair on the back wall



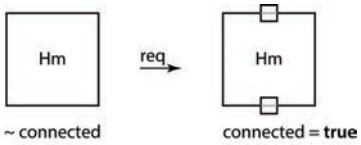
37. Generate an exterior door on the 'right' side of a back wall



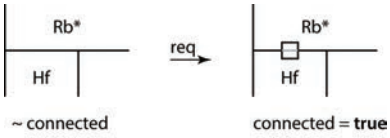
38. Generate an exterior door on the 'left' side of a back wall



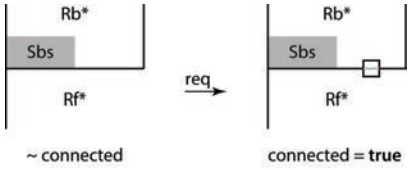
39. Generate an exterior door in a back block with partial-width stair



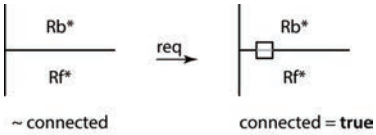
40. Generate interior doors connecting the front, middle and back blocks



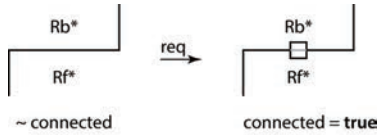
41. Generate an interior door between the front hallway and back block when there is no middle block



42. Generate an interior door between the front and back blocks when there is a stair on the front wall of the back block



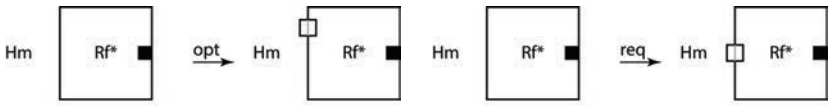
43. Generate a left-side interior door between the front and back blocks when there is no middle block nor front hallway



44. Generate a right-side interior door between front and back blocks when there is neither a middle block nor front hallway

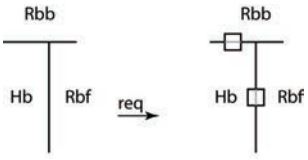
45. Generate an interior door between the front and back spaces in the front block

46. Generate interior doors between a front space and front hallways when them front block contains two divided hallways



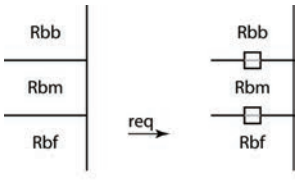
47. Generate asymmetric interior doors between hallway and spaces in front block

48. Generate symmetric interior doors between hallway and spaces in front block

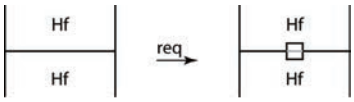


49. Generate interior doors when the back block has a hallway

50. Generate an interior door between the front and back spaces in the back block



51. Generate interior doors between front, middle and back spaces in the back block



52. Generate an interior door between adjacent front hallways (after Rule 46)