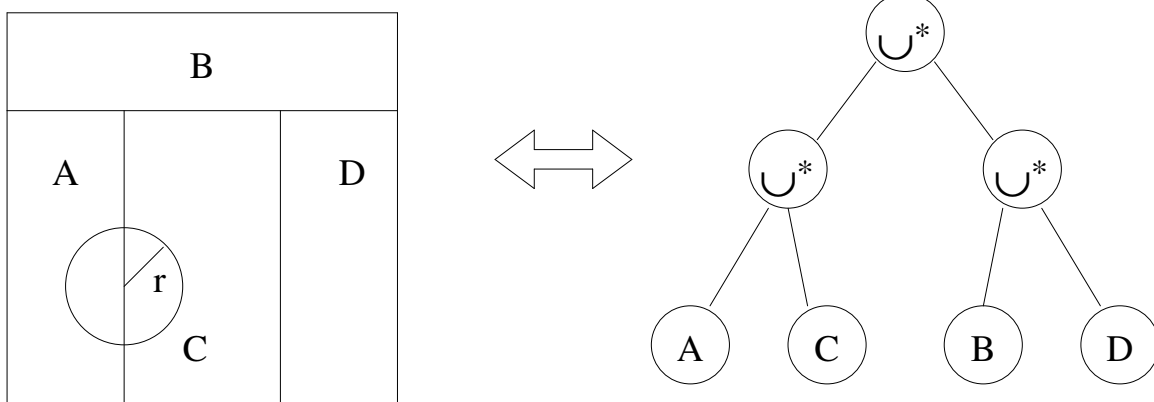


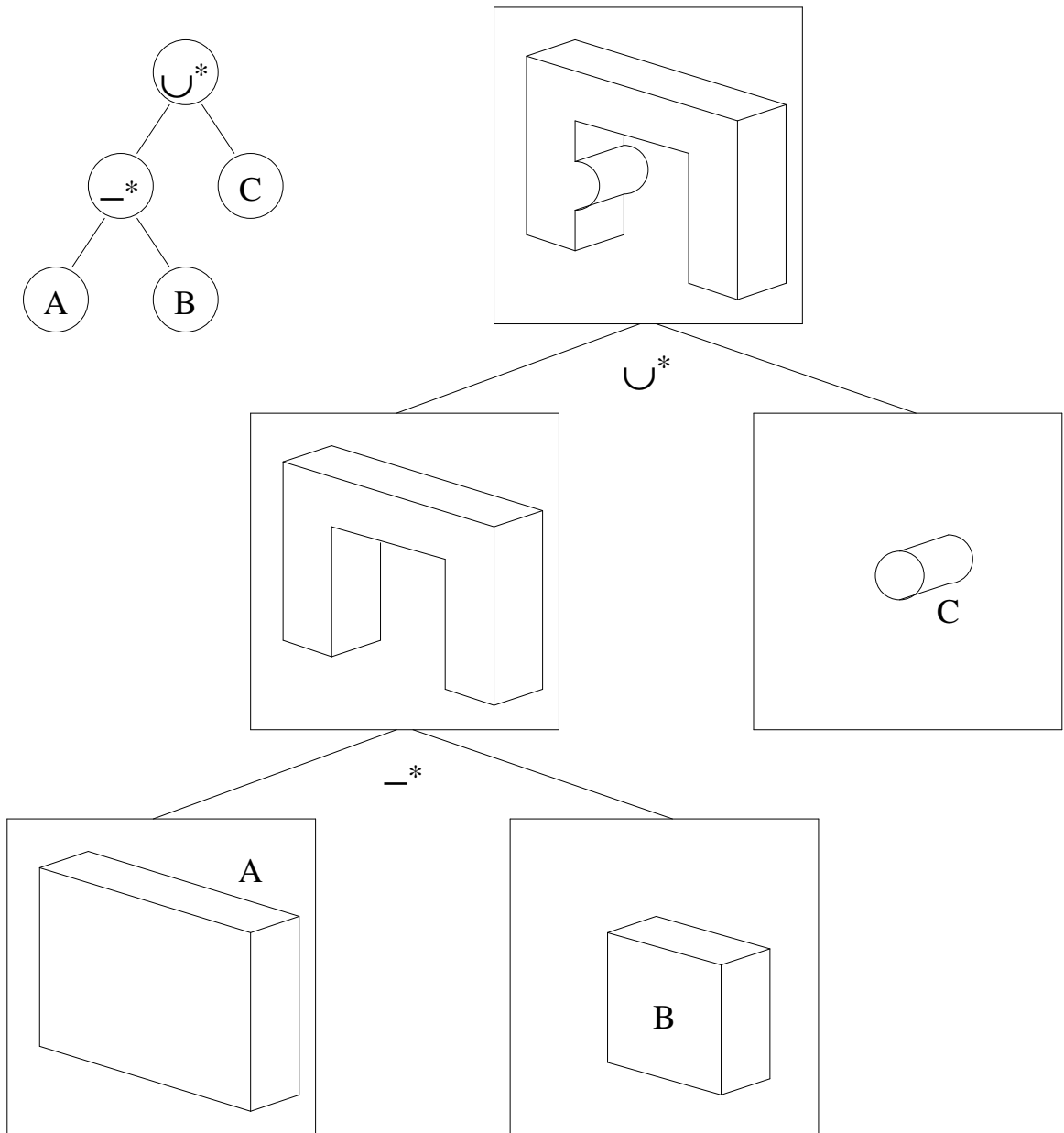
10.10 Constructive Solid Geometric (CSG)

- Objects are created as *combination of simple primitives*, such as rectangular blocks or cylinders, via (*regularized*) Boolean set operations (\cup^* (*union*), \cap^* (*intersection*), $-^*$ (*difference*))
- The construction process is usually recorded in a CSG tree
- A CSG tree is an ordered binary tree: non-terminal nodes are operators (\cup^* , \cap^* , $-^*$), terminal nodes are primitives.

An example of 2D CSG object:



An example of 3D CSG object:



Non-terminal nodes are also called **composites**

Terminal nodes are also called **primitives**

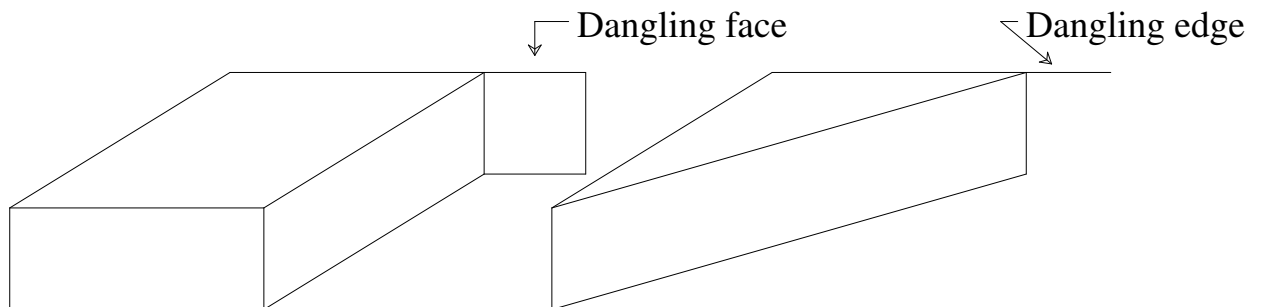
10.10.1 Regular Sets & Regular Set Operations

- A set X is a **regular set** if it equals the closure of its interior, i.e.,

$$X = Cl(Int(X))$$

where Cl and Int denote the closure and interior of a set, respectively.

Example: sets which are not regular



- The regularization of a set X is the closure of its interior, i.e.,

$$Re(X) = Cl(Int(X))$$

where Re denotes the regularization of a set.

- **Regularized set operators:**

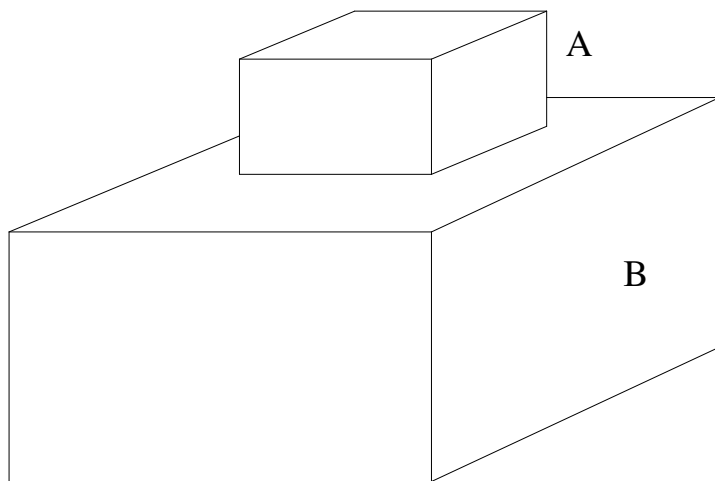
$$X \langle rop \rangle Y = Cl(Int(X \langle op \rangle Y))$$

"op": conventional set operator (\cup , \cap or $-$)

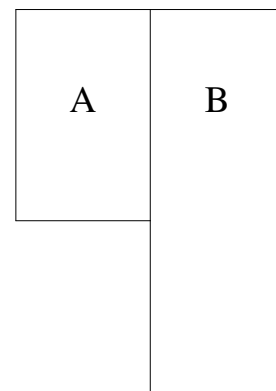
"rop": regularized set operator (\cup^* , \cap^* , $-^*$)

Example:

The regularized set intersection of A and B below is empty but the conventional intersection of A and B is not.



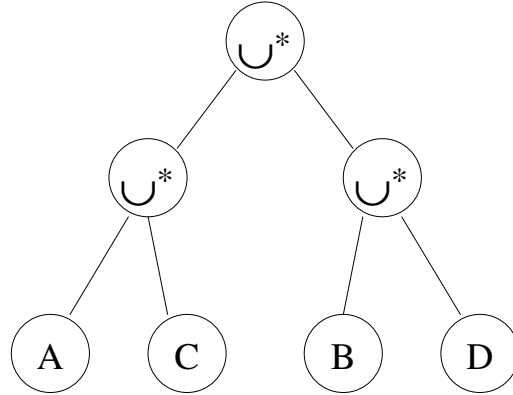
3D



2D

10.10.2 Data Structures

Composite: Name
Operator
L_T_PTR
R_T_PTR



primitive: Name
Type
Local_To_Global_Transformation
Global_To_Local_Transformation
Surface_PTRs

where

Name: string identifier, provided by user during dreation of the solid

Operator: \cup^* , \cap^* , or $-^*$

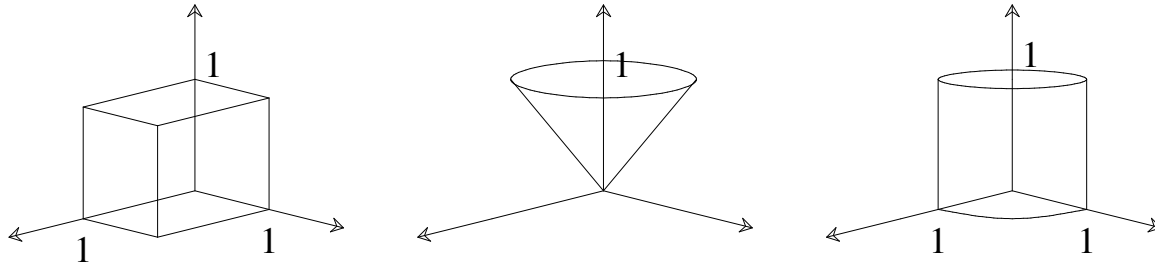
Type: block, cylinder, cone, sphere, wedge, ..., etc

Local_To_Global_Transformation: 4×4 matrix

Global_To_Local_Transformation: inverse of

Surface_PTRs:

Each primitive type has a local coordinate system:



To construct solids, we position primitives in proper location and scale and rotate them to get the desired size and orientation in global coordinate system.

For instance, to position a block in the global coordinate system, we need:

Reference point (R):

X direction coordinate (X):

Y direction coordiante (Y):

Z length:

These parameters are used to construct the Local_To_Global_Transformation so that a copy of the required primitive can be move from the local coordinate system to the desired location (R) in global coordinate system and then rotated and scaled to obtain the required size and orientation.

10.10.3 Set Membership Classification

Most algorithms that operate on a CSG representation use the technique of divide-and-conquer.

The technique of divide-and-conquer will be used in finding solutions to the following problems:

- (1) Given a regular set X (candidate set) and a regular set S (reference set), find out if X a member of S .
- (2) Given a regular set X and a regular set S , divide X into parts that are inside, on the boundary of, and outside S .

These are the so-called *membership classification problems*

Membership classification is the fundamental building block of algorithms in solid modeling

The membership classification function $M[\cdot, \cdot]$ is defined as follows:

$$M[X, S] = (X_{inS}, X_{onS}, X_{outS})$$

where

S : reference set
 X : condidate set

$$X_{inS} = X \cap^* (Int(S))$$

$$X_{onS} = X \cap^* B(S)$$

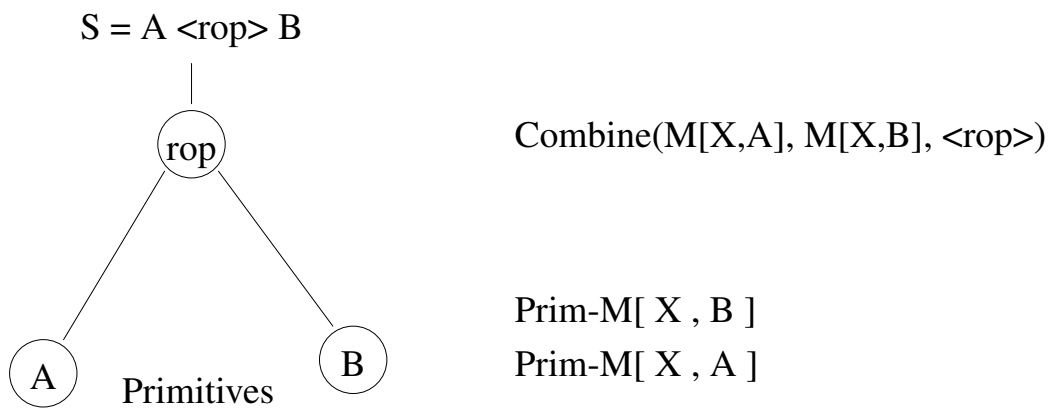
$$X_{outS} = X \cap^* C(S)$$

The classification process is usually performed using the technique of divide-and-conquer, i.e., if S is not a primitive, we classify X with respect to the subtrees of S and then "combine" the results to get the classification of X with respect to S . problem of classifying.

If S is primitive, the problem can not be decomposed further, a "primitive evaluator" is used.

```
procedure M[X, S]
  if ( S is a primitive ) then
    return Prim - M[X, S]
  else
    combine(M[X, LT(S)], M[X, RT(S)], Root(S))
```

Example:



The primitive classification procedure "Prim-M" and the marriage procedure "Combine" must be designed.

10.10.3.1 Point Membership Classification

P: a point, **S**: a CSG object

```
procedure PMC[P, S]  
  if ( S is a primitive ) then  
    return Prim - PMC[P, S]  
  else  
    Combine(PMC[P, LT(S)], PMC[P, RT(S)], Root(S))
```

Prim-PMC classifies a point against a primitive. This is the trivial part.

"Combine" procedure is to determine whether a point is "in", "on", or "out" a CSG tree (object)

$$S = A < \text{rop} > B$$

by combining the classification of the point with respect to *A* and *B*.

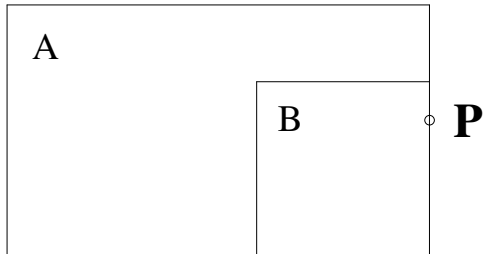
This can be done by using a look-up table. The tables for \cup^* , \cap^* , and $-^*$ are shown below.

\cap^*	in	on	out
in	in	on	out
on	on	on/out	out
out	out	out	out

\cup^*	in	on	out
in	in	in	in
on	in	in/on	on
out	in	on	out

$-^*$	in	on	out
in	out	out	out
on	on	on/out	out
out	in	on	out

Ambiguous cases: **P** is "on" *A* and "on" *B* in both cases



(a)



(b)

(1) $S = A \cap^* B$

P is "on" $S = A \cap^* B$ in case (a)

P is "out" $S = A \cap^* B$ in case (b)

(2) $S = A \cup^* B$

P is "on" $S = A \cap^* B$ in case (a)

P is "in" $S = A \cap^* B$ in case (b)

(3) $S = A -^* B$

P is "out" $S = A \cap^* B$ in case (a)

P is "on" $S = A \cap^* B$ in case (b)

Remedy:

using the *regular neighborhood method*

10.10.3.2 Line Membership Classification

L : a line, S : a CSG object

procedure $LMC[L, S]$

for each primitive P in S do

 intersect L with P

 enter point(s) of intersections into $PList$

if $PList$ contains more than one point then

 order the points by sorting them along L

Add the endpoints of L into $PList$ at the ends

for each segment* in $PList$ do

$m \leftarrow$ midpoint of the segment

case $PMC[m, S]$ of

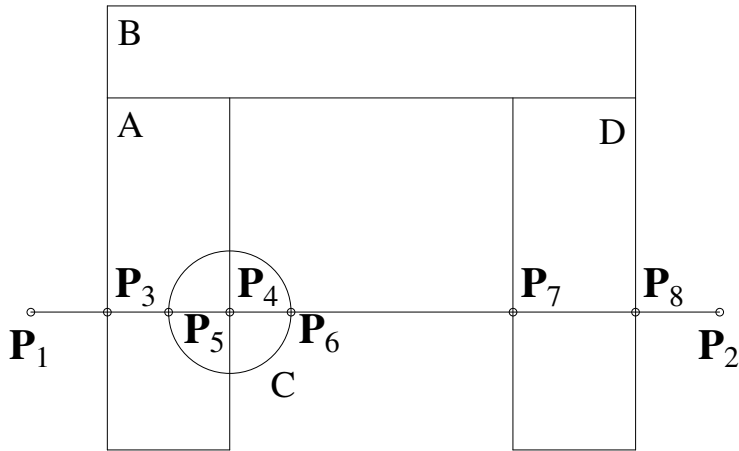
 "in": add segment to $in - list$

 "on": add segment to $on - list$

 "out": add segment to $out - list$

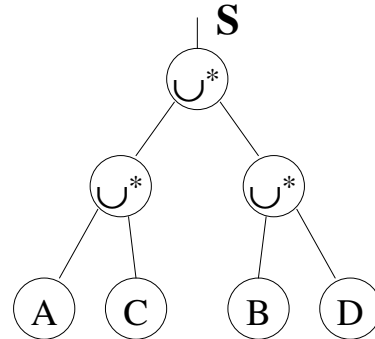
* segment: a portion of L defined by two consecutive points of $PList$

Example for LMC



$$S = A \cup^* B \cup^* C \cup^* D$$

$$L = [P_1, P_2]$$



L intersects A at: P_3, P_4

L intersects B at: null

L intersects C at: P_5, P_6

L intersects C at: P_7, P_8

After sorting, $PList = P_3, P_5, P_4, P_6, P_7, P_8$

After adding P_1 and P_2 into $PList$, we have

$$P_1, P_3, P_5, P_4, P_6, P_7, P_8, P_2$$

Mid-point of	$[P_1, P_3]$	is	"out"	S
	$[P_3, P_5]$		"in"	
	$[P_5, P_4]$		"in"	
	$[P_4, P_6]$		"in"	
	$[P_6, P_7]$		"out"	
	$[P_7, P_8]$		"in"	
	$[P_8, P_2]$		"out"	

Hence, in-list = $[P_3, P_5], [P_5, P_4], [P_4, P_6], [P_7, P_8]$
 on-list = *null*
 out-list = $[P_1, P_3], [P_6, P_7], [P_8, P_2]$

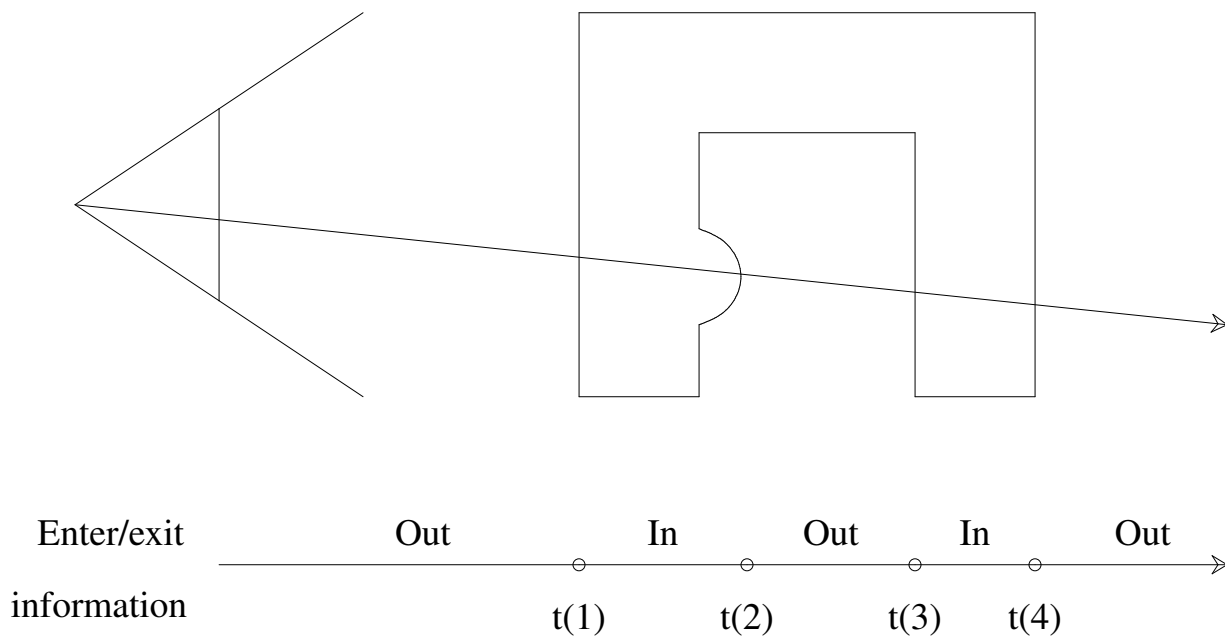
(If necessary, merge connecting segments in the same category into one segment)

After merging,

in-list = $[P_3, P_6], [P_7, P_8]$
on-list = *null*
out-list = $[P_1, P_3], [P_6, P_7], [P_8, P_2]$

10.10.4 Object Generation and Display

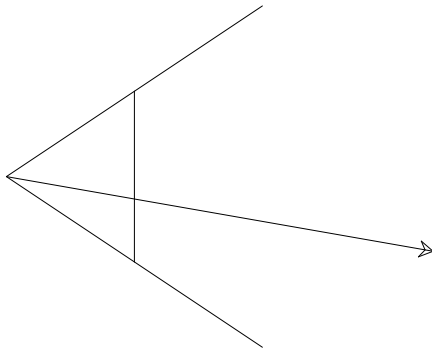
- Using a **ray tracing** based approach
- For a focal point (view point) and a rectangular pixel array (screen), generate a **ray** through each pixel and find the first surface the ray intersects.



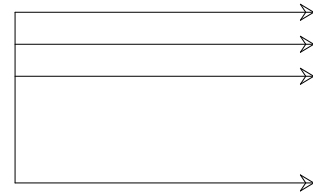
Example of a cast ray.

Definition of a ray:

$$(x, y, z) = (x_0, y_0, z_0) + t * (D_x, D_y, D_z)$$



Perspective



Parallel

For each ray, the following information will be returned:

Ray parameters: $t[1], t[2], \dots, t[n]$

Surface pointers: $S[1], S[2], \dots, S[n]$

where n is the number of ray-solid intersections.

The ordered list of ray parameters denotes the enter-exit points.

The list of surface pointers are pointers to the surfaces through which the ray passes.