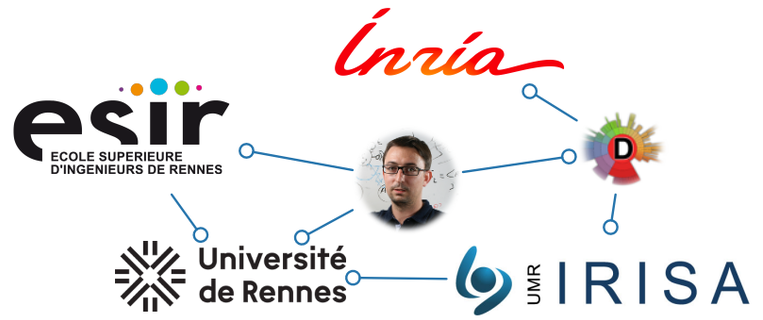
VALIDATION & VERIFICATION

TEST QUALIFICATION

UNIVERSITY OF RENNES, ESIR, 2023-2024

BENOIT COMBEMALE
FULL PROFESSOR, UNIVERSITY OF RENNES, FRANCE

HTTP://COMBEMALE.FR
BENOIT.COMBEMALE@IRISA.FR
@BCOMBEMALE



Test criteria

- Software testing aims at finding faults
- Yet, we don't know what are faults
  - `if (days > 366)` a fault?
- But we still need to test for a purpose
- Test criteria provide unambiguous objectives for test generation
  - Hopefully leading to test cases that find faults

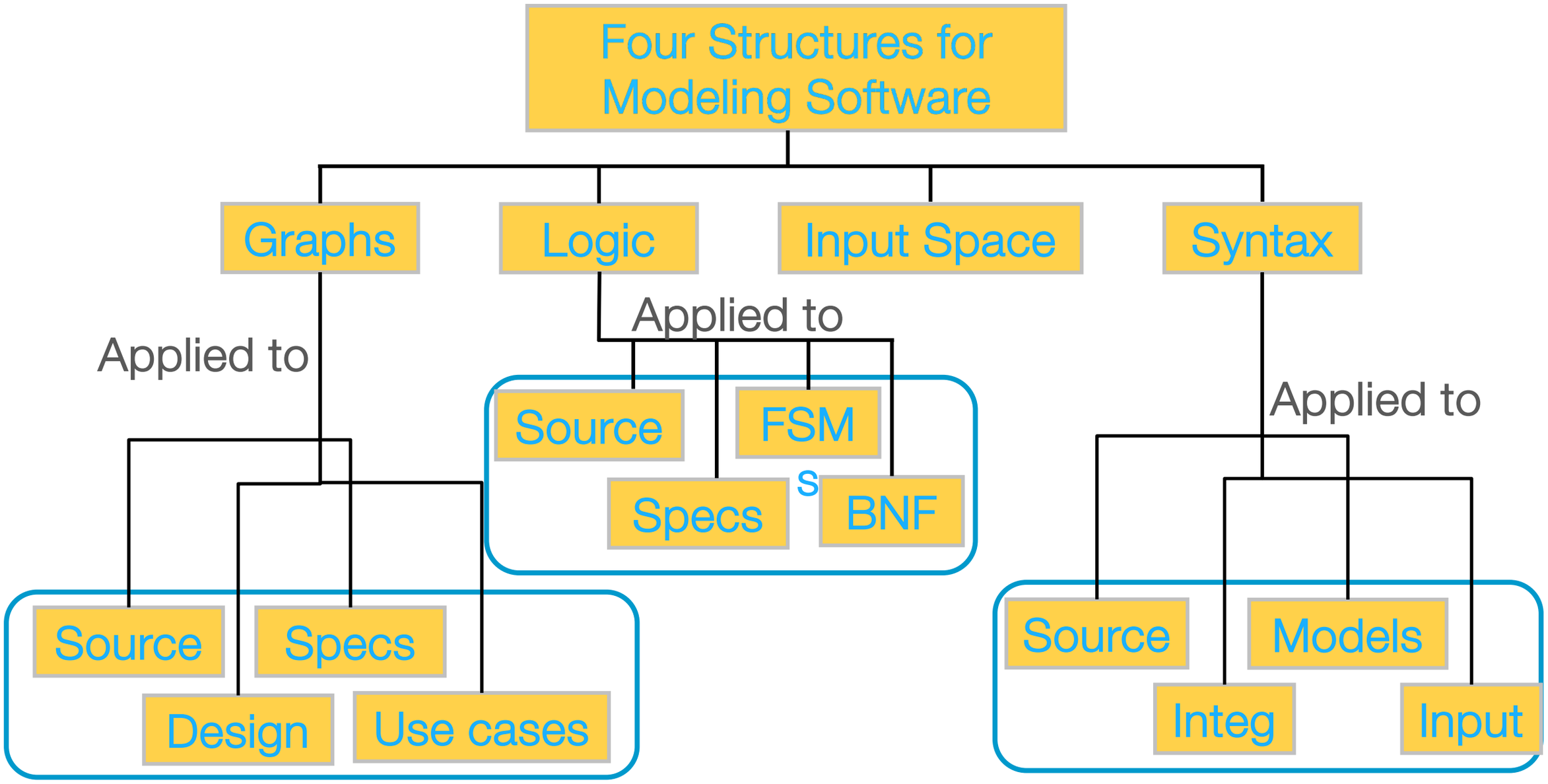
Test criteria

- Different view on software testing
- Software testing aims at defining a model of the software
  - build the model
  - test criterion defines a set of test requirements
  - generate tests that fulfill the requirements

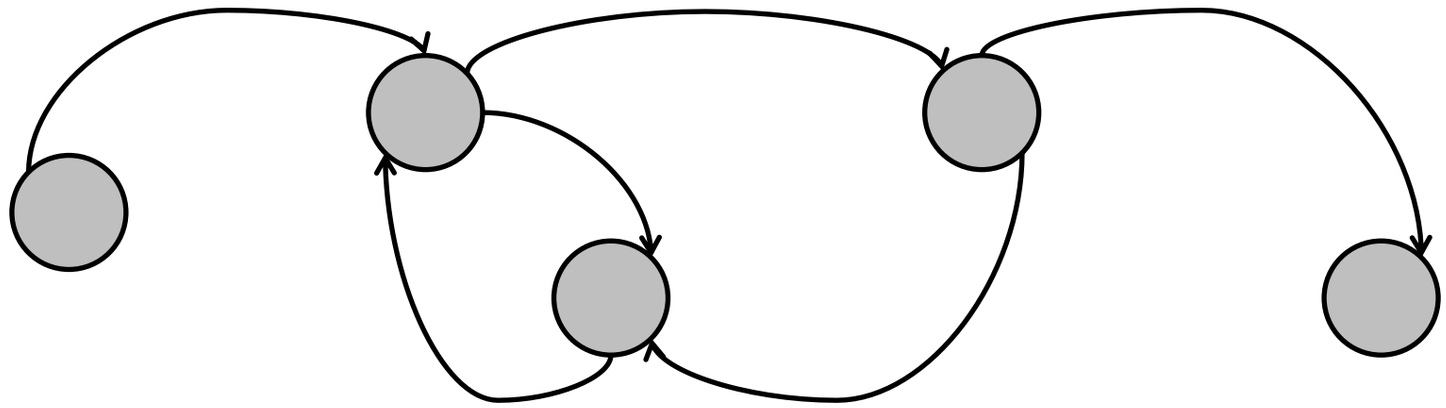
Test models and criteria

- Large numbers of models
  - control / data flow
  - flow of pages in a web site
  - API
  - …
- Large number of criteria
  - cover the paths
  - cover the input space
  - cover all conditions to execute a function
  - …

Four structures



Graphs



- Can capture
  - control flow in a function
  - method calls in a program
  - dependencies in a class diagram
  - states and transitions
  - flow in a web site
  - …

Logic

((a>b) or (b==c)) and z

- Formula are found in
  - Decisions in a program
  - Pre and post conditions
  - Conditions to access a specific resource or state
  - …

Input space

- Input space modeling
  - identify the parameters
  - select a finite number of values for each parameter
  - select combinations
- Unit level
  - pgcd(int x, int y)
  - x : {0,1, -1, 1000}; y : {0,1, -1, 1000}
  - Test data: (0,0), (0,1), (0,-1)…
- System level
  - fields in a web form
  - actions in a GUI

Syntax

for ((<EXPR1> ; <EXPR2> ; <EXPR3>)) ;
do <LIST> ;
done

- Syntactic description of inputs
  - command-line programs
  - configurable systems
  - inputs for parsers, compilers
  - …

White and black box criteria

- Test model depends on available elements
- White-box: the code is available
  - control / data flow in the program
  - method calls
  - parameter types
- Black-box: a form of specification is available
  - user-level inputs
  - GUIs
  - documentation

WHITE BOX CRITERIA

Test structurel

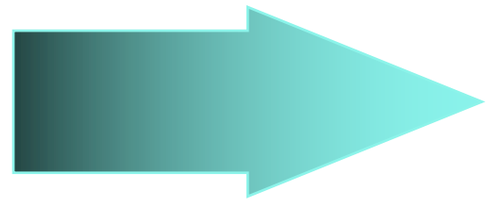
- A partir d'un modèle du code
  - modèle de contrôle (conditionnelles, boucles...)
  - modèle de données
    - modèle de flot de données (définition, utilisation...)
- Utilisation importante des parcours de graphes
  - critères basés sur la couverture du code

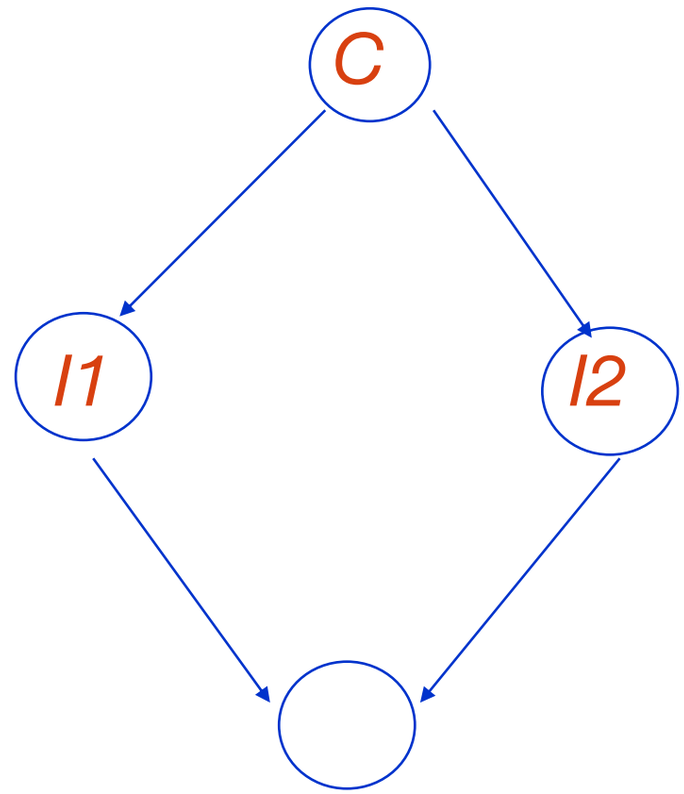
Le test structurel

- Les tests sont construits en exploitant
  - la structure de l'architecture : test d'intégration
  - la structure du programme : test unitaire structurel
- Quel est le rôle de la structure (par rapport à une boîte noire)
  - Offrir des informations sur la manière dont le système est réalisé (le comment)
  - Des éléments qui doivent être couverts par les tests
    - Critères de couverture

Le test unitaire structurel: abstraire pour obtenir un critère formel

si C alors I1
sinon I2





Le test unitaire structurel

- Graphe de Contrôle (langages procéduraux/actionnels)
  - But : représenter tous les chemins d'exécution potentiels
  - Graphe orienté : avec un noeud d'entrée E et un nœud de sortie S

    (nœud fictif ajouté si plusieurs sorties)
  - Sommets :

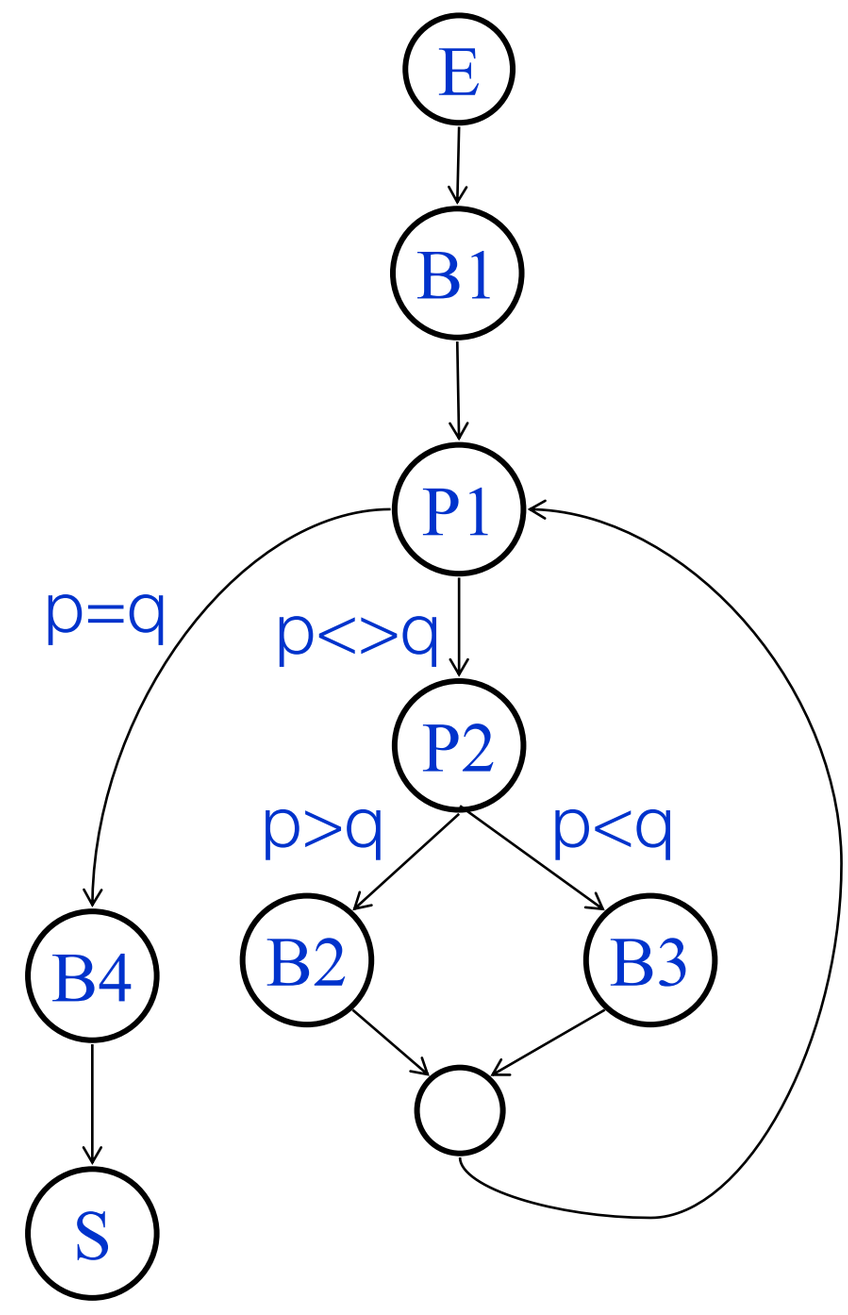
    blocs élémentaires du programme, ou prédicats des conditionnelles /boucles, ou nœuds de jonction "vide" associé à un noeud prédicat

    Bloc élémentaire : séquence maximale d'instructions séquentielles
  - Arcs :

    enchaînement d'exécution possibles entre deux sommets

Exemple de test unitaire structurel

```
pgcd: integer is
local p,q : integer;
do
    read(p, q)                          B1
    while p<> q do                      P1
        if p > q                        P2
        then p := p-q                   B2
        else q:= q-p                    B3
        end -- if
    end -- while
    result:=p                           B4
end-- pgcd
```



Le test unitaire structurel

- Question : Sur ce modèle imaginer un critère de couverture
  - minimal
  - maximal

Le test unitaire structurel

- Chemin: suite d'arcs rencontrés dans le graphe, en partant de E et finissant en S.
  - en général représenté sans perte d'information par une liste de sommets
- Prédicat de chemin: conjonction des prédicats (ou de leur négation) rencontrés le long du chemin.
  - Pas toujours calculable

E B1 P1 P2 B2 S : $p_1=q_1$ & $p_0 > q_0$ (p_i = ième valeur de p)

E B1 P1 P2 B3 S : $p_1=q_1$ & $p_0 <= q_0$ (p_i = ième valeur de p)

E B1 P1 $(P2\ B2)^n$ S : $p_n=q_n$ & $(p_i > q_i\ i \in [0..n])$

Chemins infaisables : problème en général indécidable

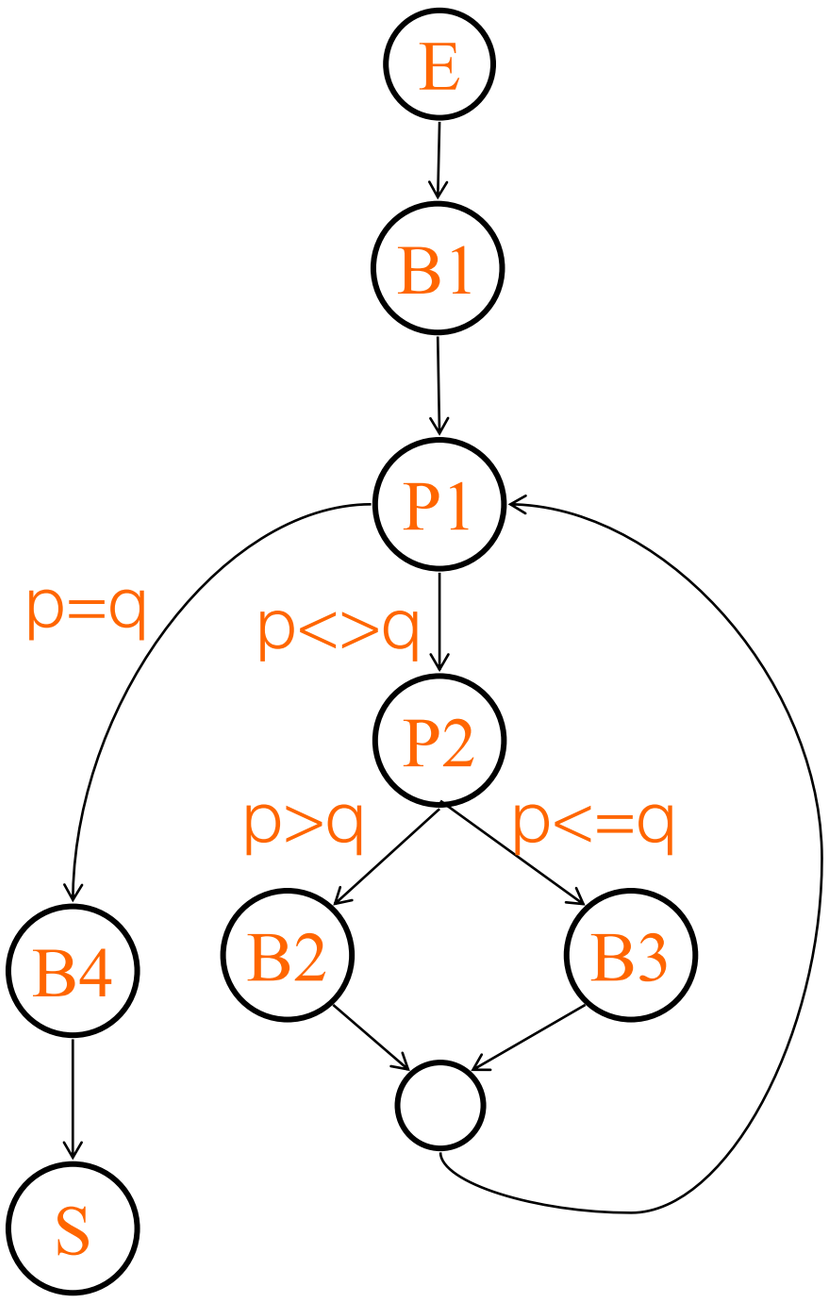
Le test unitaire structurel

- Critère maximal
  - Couverture des chemins

- « Tous les chemins »
  - Nombre très souvent infini
  - Non atteignable
- « Tous les k-chemins »
  - Les chemins qui passent au maximum k fois dans le corps des boucles
- « Tous les chemins élémentaires »
  - 0 à 1 passage dans les corps de boucles

- Question: est-ce faisable ?

Exemple de test unitaire structurel



Tous les noeuds:

(E, B1, P1, P2, B2, P1, B4, S)
(E, B1, P1, P2, B3, P1, B4, S)

Tous les arcs : idem

Tous les chemins élémentaires (1-chemin) :

idem + (E, B1, P1, B4, S)

Tous les 2-chemins :

idem +
(E, B1, P1, P2, B2, P1, P2, B2, P1, B4, S)
(E, B1, P1, P2, B2, P1, P2, B3, P1, B4, S)
(E, B1, P1, P2, B3, P1, P2, B2, P1, B4, S)
(E, B1, P1, P2, B3, P1, P2, B3, P1, B4, S)

Exercice: fonction d'ajout dans une liste triée

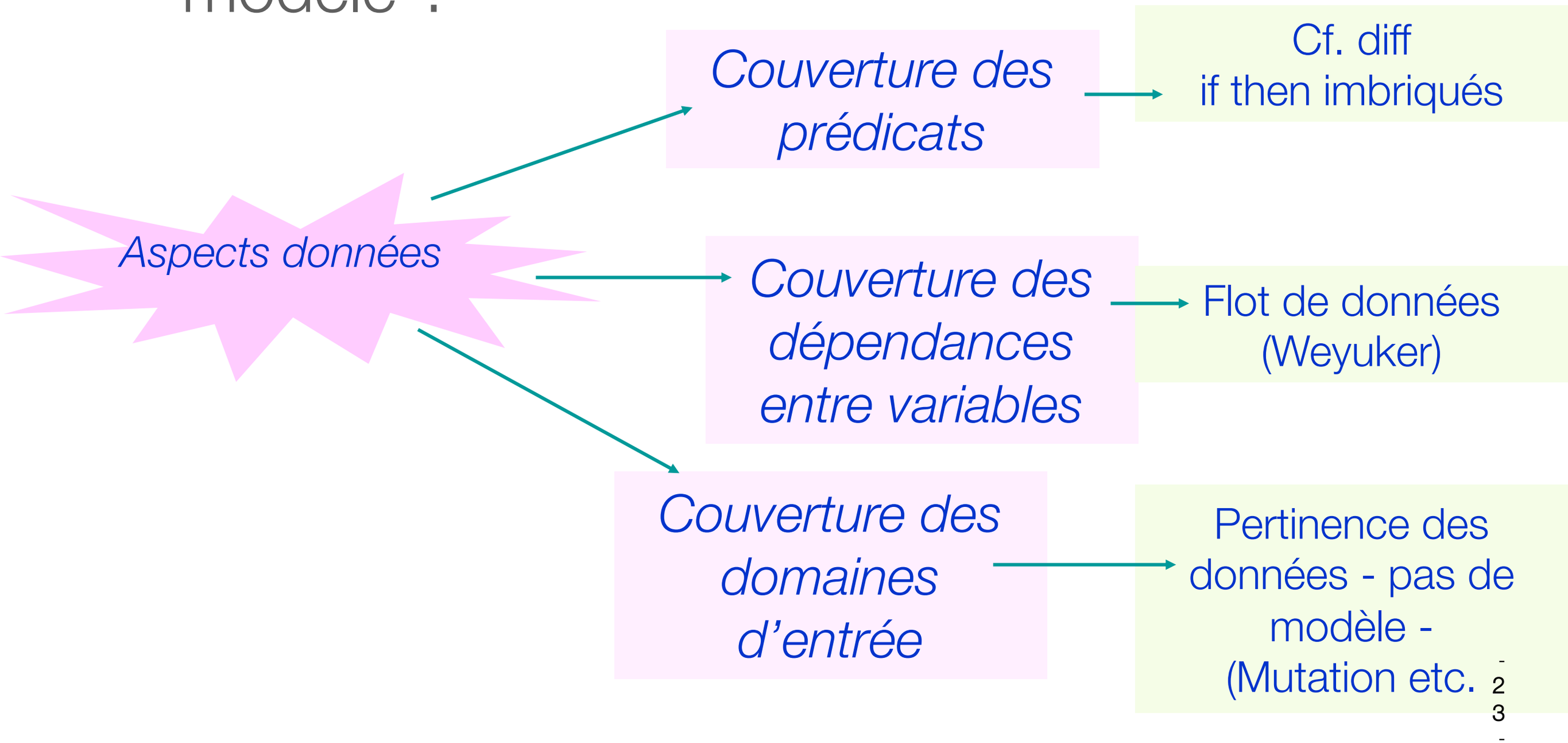
```
action Insère (x : entier, donnée-résultat L : liste)
lexique
    courant, précédent, nouveau : liste ;
    continue : booléen ;
algorithme
courant = L ;                                   B1
précédent = nil ;                               B1
continue = (courant != nil) ;                   B1
tant que continue et (courant != nil)           P1
faire
  si courant.info >= x alors                    P2
  continue = faux                               B2
  sinon
  précédent = courant                           B3
  courant = courant.suiv                        B3
  fsi
fait
allouer(nouveau)                                B4
nouveau.info = x                                B4
nouveau.suiv = courant                          B4
si précédent == nil alors L = nouveau           P3 B5
sinon précédent.suiv = nouveau                  B6
fsi
fin algorithme
```

Avantages et limites

- Critère systématique
- Grands nombre de chemins
- Chemins infaisables : problème en général indécidable

Test unitaire structurel

- Question : qu'est-ce que ne capture pas ce modèle ?

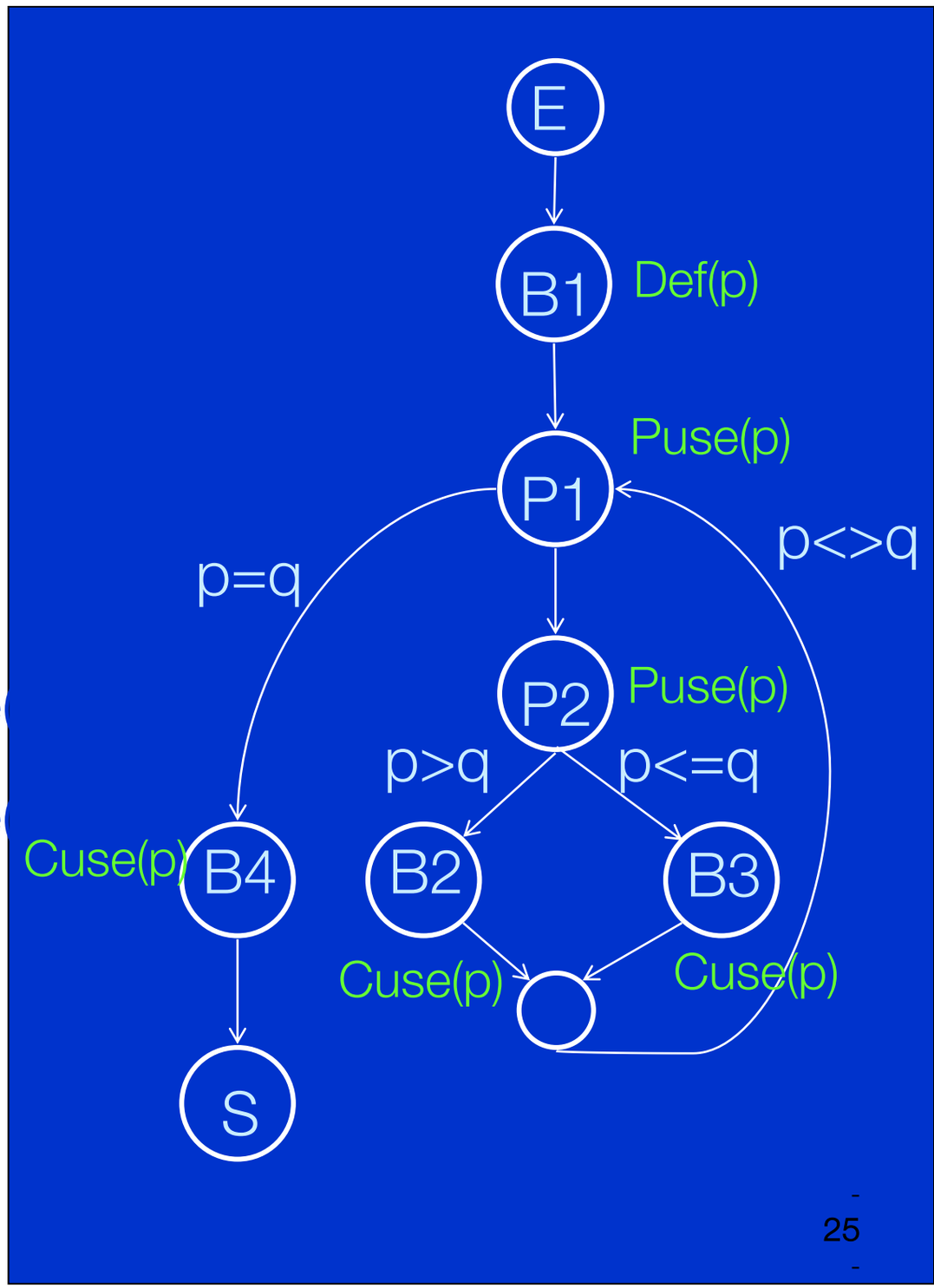


Le test unitaire structurel

- Graphe de Flot de Données (Weyuker)
  - But : représenter les dépendances entre les données du programme dans le flot d'exécution.
  - Graphe de contrôle décoré d'informations sur les données (variables) du programme.
  - Sommets : idem GC +
    - une **définition** (= affectation) d'une variable v est notée def(v)
    - une **utilisation** d'une variable est v notée P_use(v) dans un prédicat et C_use(v) dans un calcul.

Le test unitaire structurel

```
pgcd: integer is
local p,q : integer;
do
    read(p, q)                  B1- Def(p), Def(q)
    while p<> q do              P1 - Puse(p), Puse(q)
        if p > q                P2- Puse(p), Puse(q)
        then p := p-q           B2- Def(p), Cuse(q), Cuse
        else q:= q-p            B3 - Def(q), Cuse(p),Cuse
        end -- if
    end -- while
    result:=p                   B4- Cuse(p),
end-- pgcd
```



Paires DU

Variable	Paires DU
p	(B1,P1) (B1,P2) (B1,B2) (B1,B3) (B1,B4) (B2,B2)(B2,B3) (B2,B4)
q	(B1,P1) (B1,P2) (B1,B2) (B1,B3) (B3,B3)(B3,B2)

Chemins DU

Paires DU	Chemin DU
(B1,P1)	[B1,P1]
(B1,P2)	[B1,P1,P2]
(B1,B2)	[B1,P1,P2,B2]
(B1,B3)	[B1,P1,P2,B3]
(B1,B4)	[B1,P1,B4]
(B2,B2)	
(B2,B3)	[B2,P1,P2,B3]
(B2,B4)	[B2,P1,B4]

Paires DU	Chemin DU
(B1,P1)	[B1,P1]
(B1,P2)	[B1,P1,P2]
(B1,B2)	[B1,P1,P2,B2]
(B1,B3)	[B1,P1,P2,B3]
(B3,B3)	
(B3,B2)	[B3,P1,P2,B2]

Chemins DU

12 chemins DU pour le pgcd, seulement 8 uniques

[B1,P1]	[B1,P1,B4]
[B1,P1,P2]	[B3,P1,P2,B2]
[B1,P1,P2,B2]	[B2,P1,P2,B3]
[B1,P1,P2,B3]	[B2,P1,B4]

★ Forcent à ne pas rentrer dans la boucle

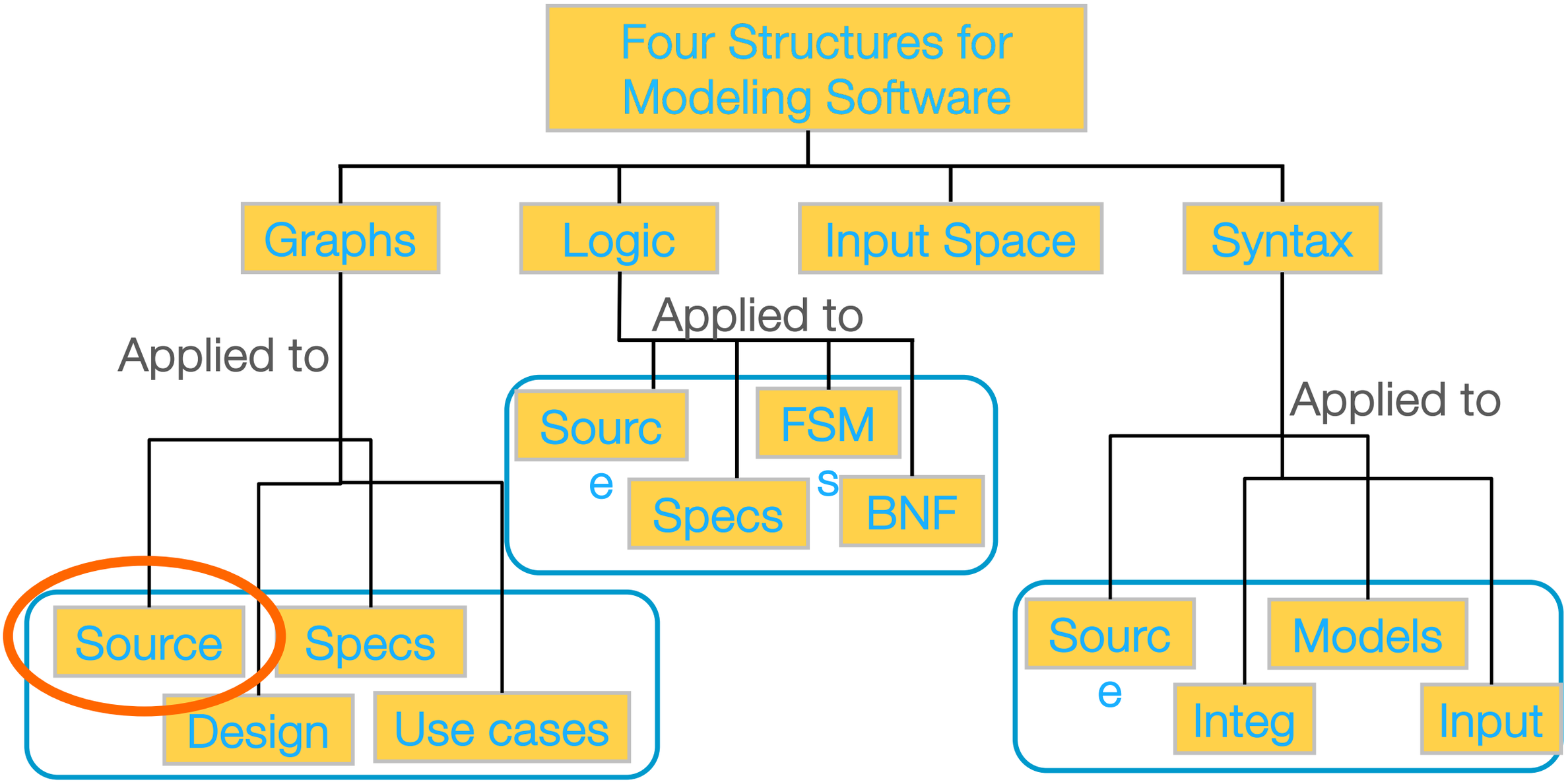
✦ Forcent à entrer au moins une fois dans la boucle

⯃ Forcent à entrer au moins deux fois dans la boucle

Données de test

- Donnée de test (4,4)
  - Chemin couvert B1, P1, B4
  - Paires DU couvertes [B1,P1] [B1,P1,B4]
- Donnée de test (4,2)
  - Chemin couvert B1, P1, P2, B2, P1, B4
  - Chemins DU couverts [B1,P1,P2] [B1,P1,P2,B2] [B2,P1,B4]
- Donnée de test (2,4)
  - Chemin couvert B1, P1, P2, B3, P1, B4
  - Chemins DU couverts [B1,P1,P2] [B1,P1,P2,B3]
- Donnée de test (6,4)
  - Chemin couvert B1, P1, P2, B2, P1, P2, B3, P1, B4
  - Chemin DU [B2,P1,P2,B3]
- Donnée de test (4,6)
  - Chemin couvert B1, P1, P2, B3, P1, P2, B2, P1, B4
  - Chemin DU couvert [B3,P1,P2,B2]

Four structures



Couverture des prédicats

```
if (A && (B || C))
```

- Critère pour valider les différentes combinaisons de valeurs pour A, B et C

Critère MC/DC

- Modified Condition/Decision Criterion
- A pour but de démontrer l'action de chaque condition sur la valeur de vérité de l'expression (appelée décision)
- Pour satisfaire MC/DC il faut
  - Pour chaque condition, il faut générer 2 cas de test tels que la décision change alors que toutes les autre conditions sont fixées

Exemple

if (A && (B || C))

Pour A A=0 B=1 C=1 Dec=0

A=1 B=1 C=1 Dec=1

Pour B A=1 B=1 C=0 Dec=1

A=1 B=0 C=0 Dec=0

Pour C A=1 B=0 C=1 Dec=1

A=1 B=0 C=0 Dec=0

=> 5 cas de test pour satisfaire MC/DC

Exercice

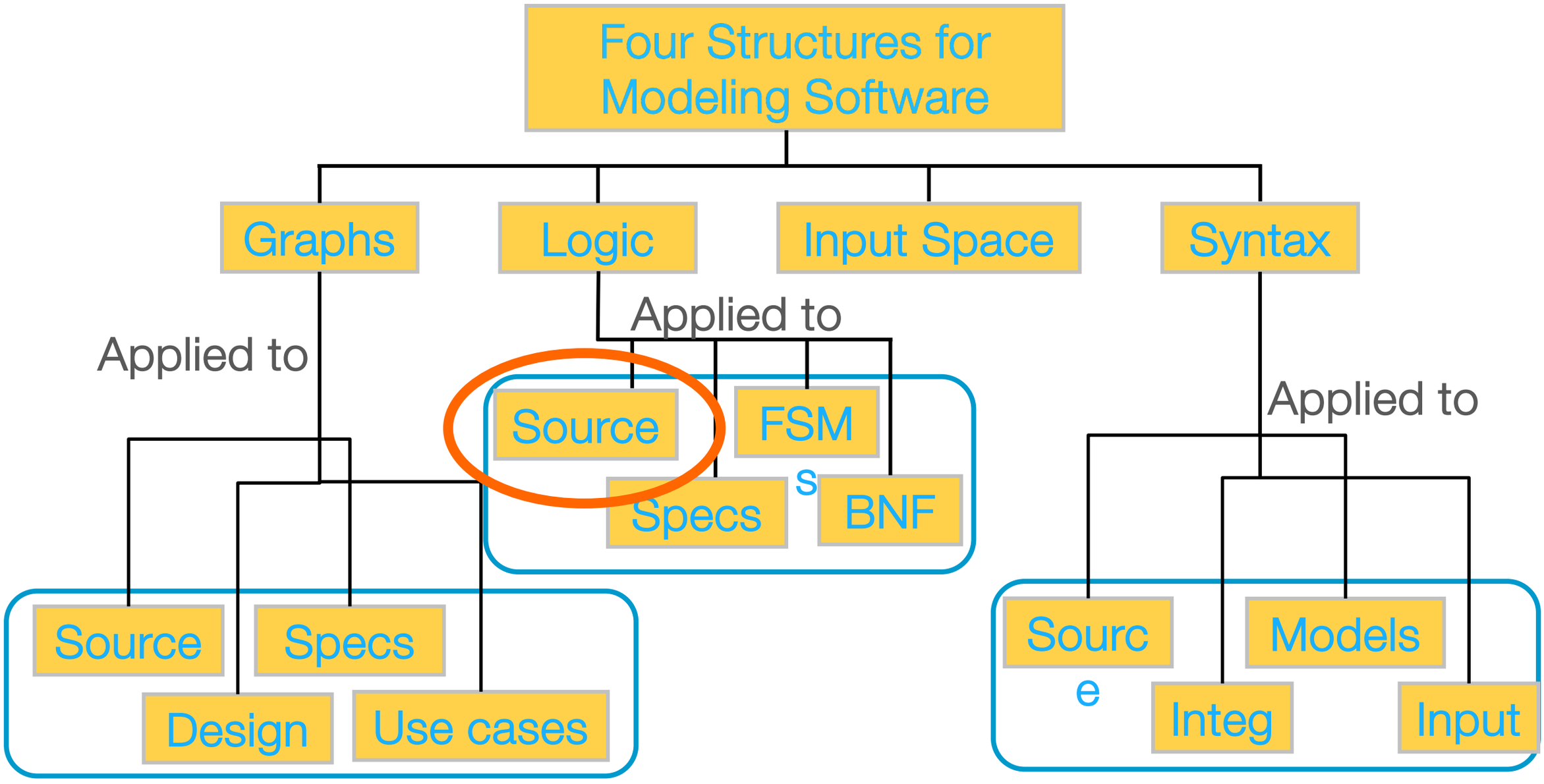
```
public Cell getNextCell(int nb_prey, int nb_predator) {
    Cell result;
    if ( (nb_predator >= 1 && nb_predator <= 3)
                                        && nb_prey > nb_predator) {
        result = new Predator();
        ((Herd)result).setPopulation(Cell.MAX_POP/2);
    }
    else if ((nb_prey >= 1 && nb_prey <= 3) && nb_predator == 0) {
            result = new Prey();
            ((Herd)result).setPopulation(Cell.MAX_POP/2);
        }
        else {
            result = new EmptyCell();
        }
    return result;
}
```

Exercice

```
(nb_prey >= 1 && nb_prey <= 3) &&
 nb_predator == 0)
```

Combien de données de test pour MC/DC?

Four structures



BLACK BOX CRITERIA

Black-box criteria

- Based only on a functional description of the program
  - the specification
- It captures multiple information
  - input domain
  - usage scenario
  - response time
  - HCI
  - ...

Black-box criteria: need a model

- Textual requirements
- Interface description
  - API
  - GUI
- UML
  - Use cases
  - State machines / sequence diagrams
- Formal specification

Input domain

- Different levels of abstraction
  - type of a method's parameters (API)
  - pre condition on a method
  - set of commands at the system level
  - ...
- Impossible to explore exhaustively
  - Partitioning / limit testing
  - Combinatorial testing
  - Root cause analysis
  - Random generation (fuzzing)

Input space partitioning

- Partition the input domain D
  - $B_q = b_1, b_2, \ldots, b_Q$
- Blocks must not overlap
  - $\forall\ i \neq j,\ b_i, b_j \in B_q,\ b_i \cap b_j = \varnothing$
- The union of block must cover D
  - $\cup b_n = D$
- For testing: choose one value in each block

Input space modeling

- step 1: choose a granularity
  - Example: method, whole system, …
- step 2: identify all parameters
  - Examples method param. and attributes, all inputs, data, signals for whole system
- step 3: model the input domain
  - Partition and identify relations between data
- step 4: select data and combinations
  - Example: boundary values
  - All combinations impossible

Exemple

- Le programme lit trois nombres réels qui correspondent à la longueur des côtés d'un triangle. Si ces trois nombres ne correspondent pas à un triangle, imprimer un message d'erreur. S'il s'agit d'un triangle, le programme détermine s'il est isocèle, équilatéral ou scalène et si son plus grand angle est aigue, droit ou obtus.

Exemple (analyse partitionnelle)

	aigu	obtus	droit
scalène	**6,5,3**	**5,6,10**	**3,4,5**
isocèle	**6,1,6**	**7,4,4**	**√2,2,√2**
équilatéral	**4,4,4**	**impossible**	**impossible**

Exemple (test aux limites)

1, 1, 2	**non triangle**
0, 0, 0	**un seul point**
4, 0, 3	**une des longueurs est nulle**
1, 2, 3.00001	**presque triangle**
0.001, 0.001, 0.001	**très petit triangle**
88888, 88888, 88888	**très grand**
3.00001, 3, 3	**presque équilatéral**
2.99999, 3, 4	**presque isocèle**
3, 4, 5.00001	**presque droit**
3, 4, 5, 6	**quatre données**
3	**une seule donnée**
5, 5, A	**une lettre**
	pas de donnée
-3, -3, 5	**données négatives**
...	

Combinatorial integration testing

- Partitions are identified for each characteristic
- Test input is a combination of characteristics
  - Number of combinations grows with the number of characteristics
- Combinatorial integration testing
  - Select a subset of combinations
  - Focus on specific interactions among values for characteristics

Combinatorial integration testing

- $X_1, \ldots, X_n$ n characteristics
- $\forall\ i \in [1..n]\ X_i \subset \{V_{i1}, \ldots, V_{im}\}$
  - m can be different for every Xi
- A test configuration is a set of values for each X_i
- Pairwise testing
  - A set TC of test configurations such that all values for every pair of characterisitics are in one configuration
  - $\forall\ X_j, X_k \mid \forall\ X_{ja}, X_{kb} \mid \exists\ c \in TC \mid TC \subset X_{ja}, X_{kb}$
- T-wise
  - Generalization to all tuples of characteristics

Combinatorial integration testing

- Families of algorithm for automatic generation of t-way combinations
  - Greedy algorithms
  - Algebraic approach
  - Meta-heuristic search
- Limited management of constraints
  - E.g. a child will pay only cash

Fuzzing

- Proposed as a student assignment in 1988
- Feeds a program/function with random input
- Observes the occurrence of undesirable behavior

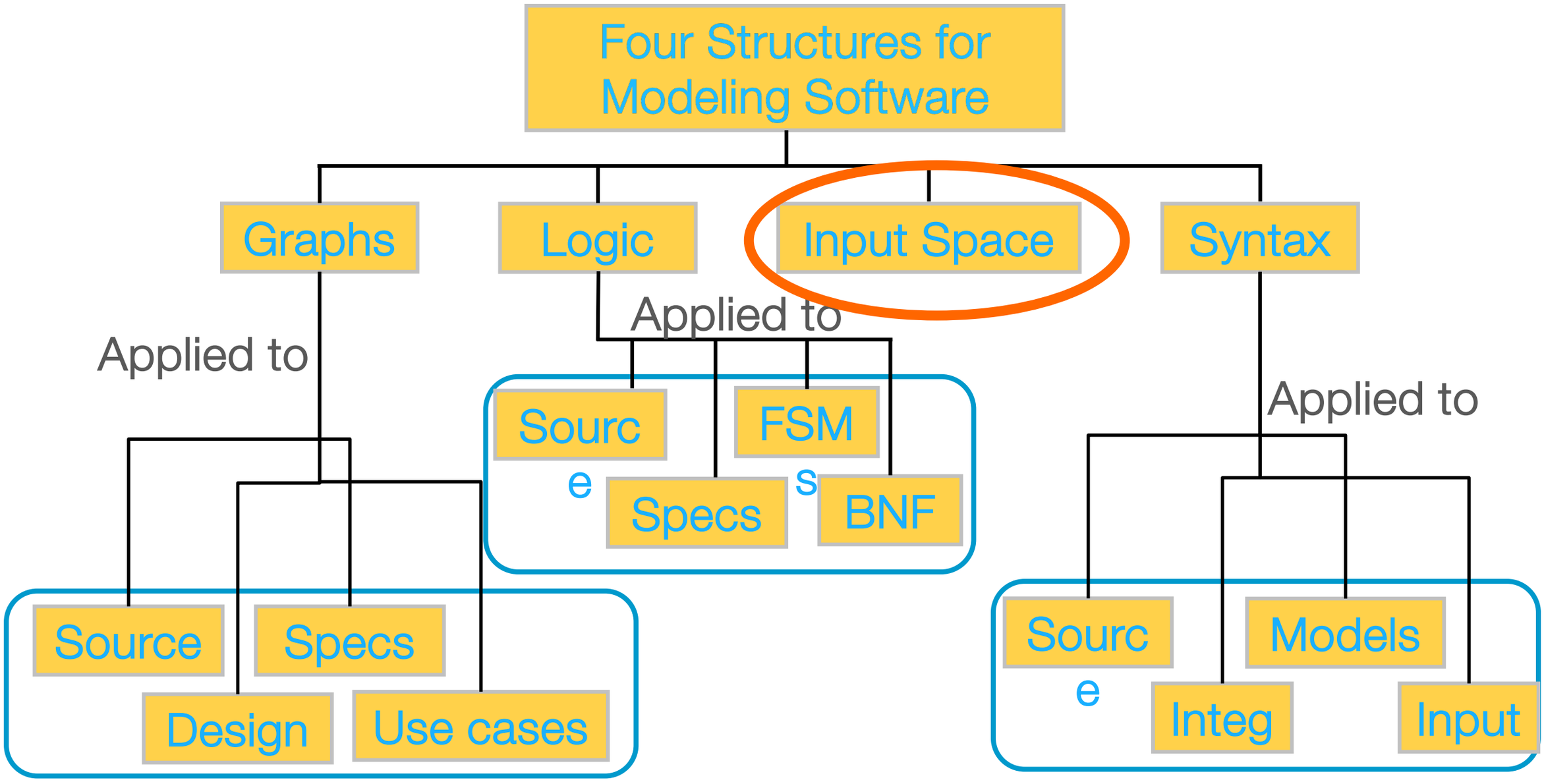
Fuzzing: What to observe?

- The presence of memory leaks.
- Crashes, (segmentation faults).
- Program performance
- Security issues

Fuzzers in practice

- AFL (American Fuzzy Loop) http://lcamtuf.coredump.cx/afl/
  - Security oriented
  - Finds inputs executing new paths
- jsfunfuzz, Langfuzz
  - More than 300 vulnerabilities in Firefox
- Grammarinator https://github.com/renatahodovan/grammarinator
  - Uses ANTLR grammar specifications

Four structures



Black-box testing

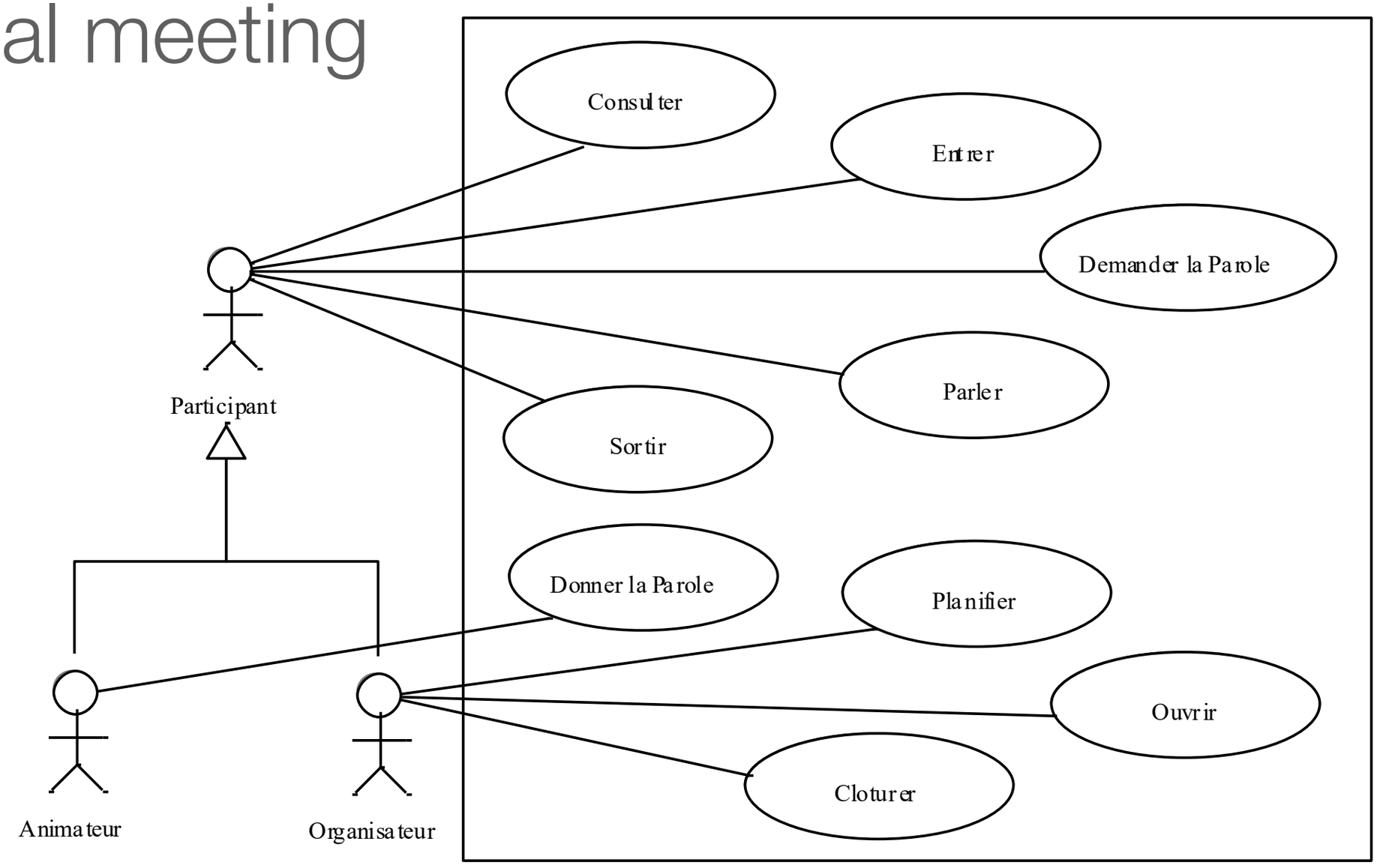
- Other models
  - more sophisticated data model (stub database)
  - GUI model
    - event flow
    - behavioral model

Data model

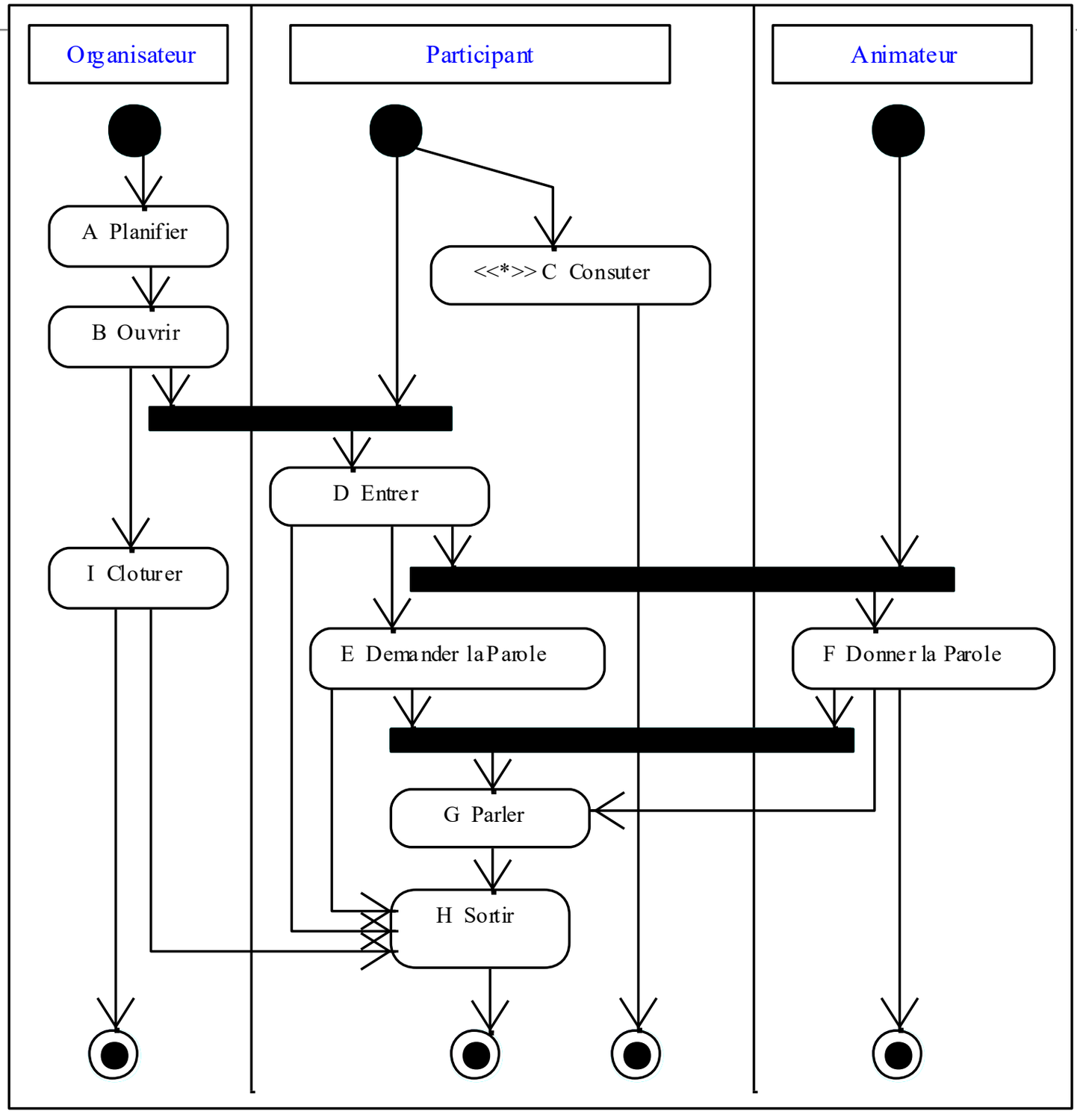
- GEDIS: http://www.gedis-studio.com/
- Simulate databases
- Model of data
  - fields
  - values (list, file, etc.)
  - constraints (e.g.: dependencies between values)
- Objectives for the generation
  - number of data
  - profiles on data types

Behavioral model – example 1

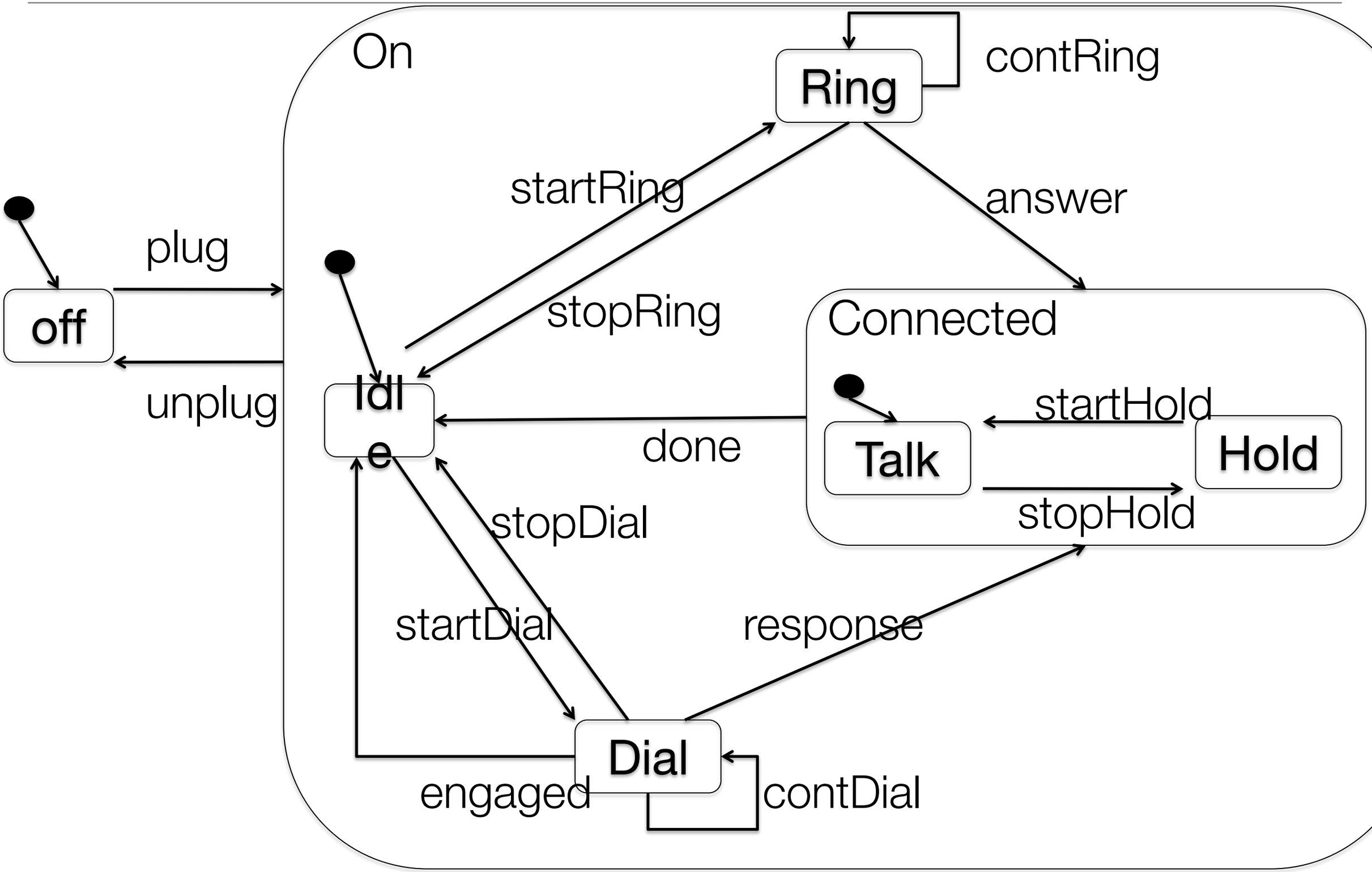
- Virtual meeting



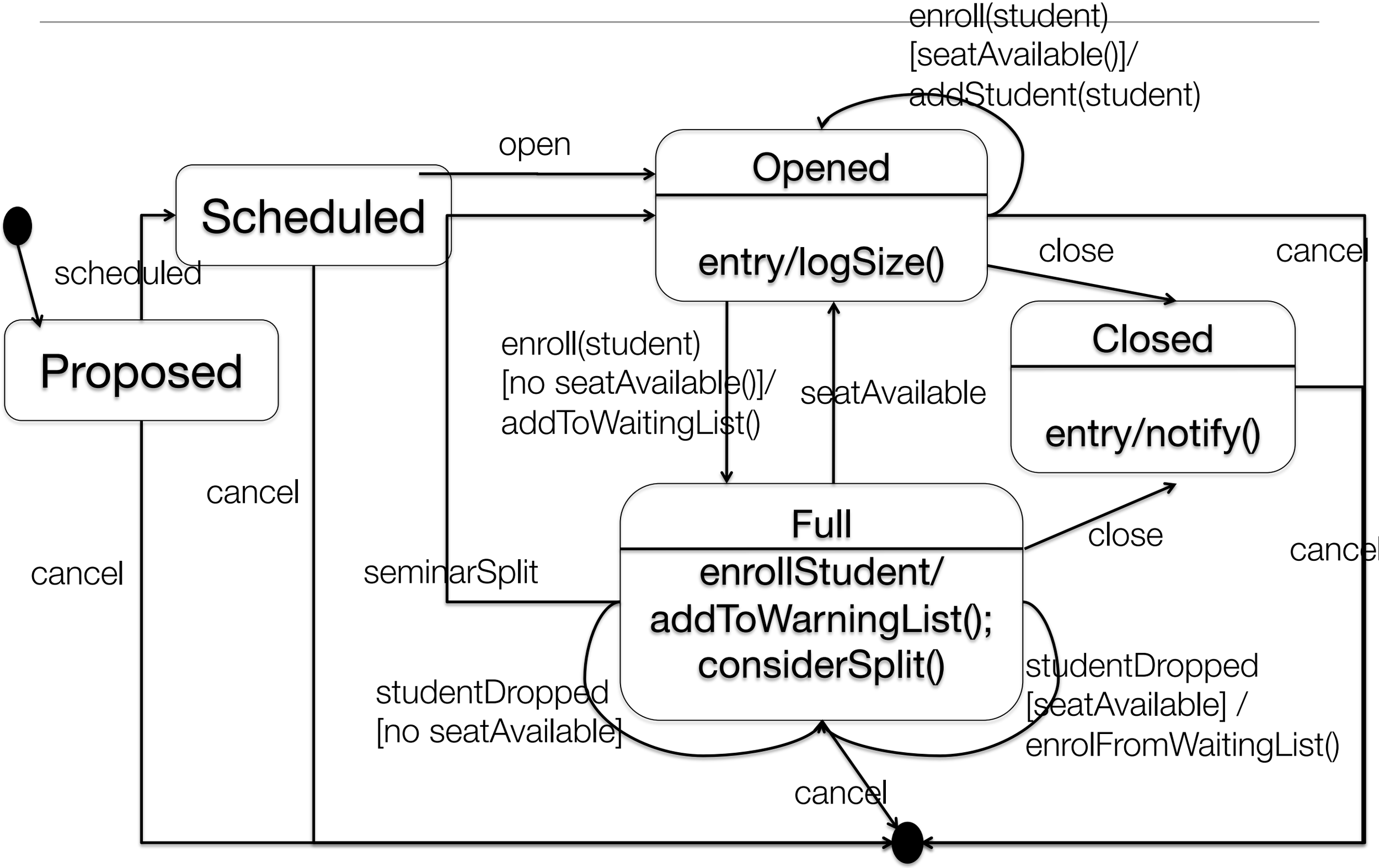
Behavioral model – example 1



Behavioral model – example 2



Behavioral model – example 3



Behavioral model – example 3

Seminar
capacity : int registered : int
enroll (studentNB : int) addStudent (studentNB : int) seatAvailable() : Boolean addToWarningList (studentNB : int) considerSplit() studentDropped() addToWaitingList() logSize() notify() enrollFromWaitingList (studentNB : int) open() close() cancel() schedule()

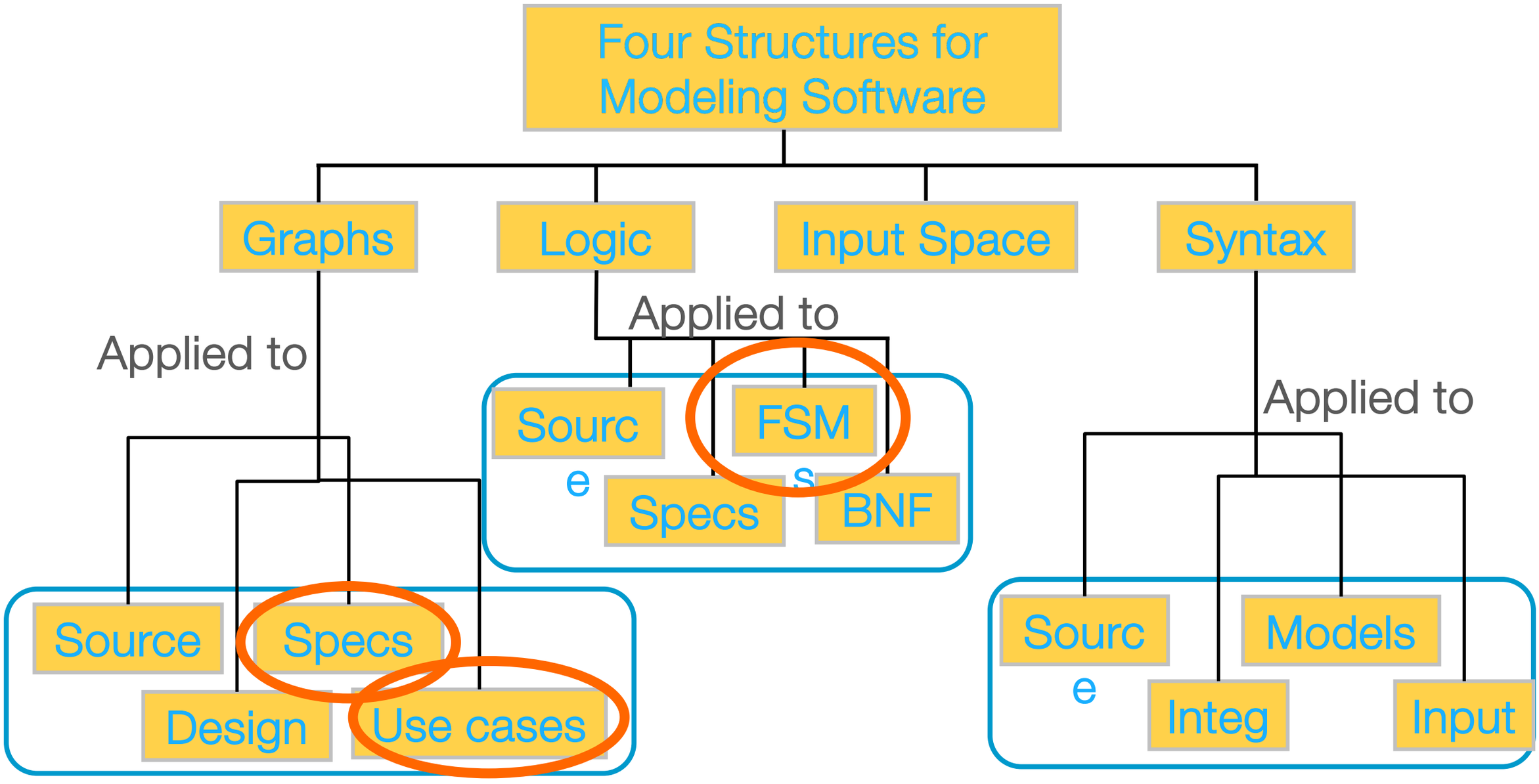
addStudent(studentNB : int)
post : registered = registered@pre +1

seatAvailable() : Boolean
post : result = (registered < capacity)

<u>s : Seminar</u>

capacity = 3
registered= 0

Four structures



Other models - environment

- For reactive systems, adaptive systems
- Simulate the environment
  - sequences of events
  - versatile conditions
- Test reaction abilities of the system

Structurel/fonctionnel: conclusion

- Les critères structurels et fonctionnels sont complémentaires
  - une erreur d'omission ne peut pas être détectée par le test structurel
  - du code mort ne peut pas être détecté par le test fonctionnel
- Au niveau unitaire
  - on commence par le test structurel
  - on complète par du test fonctionnel

Ce qu'il faut en retenir

- Aucun critère n'est meilleur qu'un autre

  ⇒ tous complémentaires

- Par contre :
  - Permet de quantifier la qualité des tests
  - Permet une génération automatique
  - Permet de savoir ce que l'on teste et donc d'automatiser la définition de l'oracle
- Exemple de contrainte (DO178-*C*)
  - Instruction or statement coverage
  - Decision Coverage
  - MC/DC Coverage