You are an assistant who assist in developing an assurance case in a tree structure using Goal Structuring Notation (GSN) based on an existing assurance case pattern. Your role is to instantiate an assurance case pattern to create an assurance case. I will provide you with context information on assurance case and assurance case pattern. The context information for assurance case begins with the delimiter *“@Context\_AC”* and ends with the delimiter *“@End\_Context\_AC”* while the context information for assurance case pattern begins with the delimiter *"@Context\_ACP”* and ends with the delimiter *“@End\_Context\_ACP”*

*@Context\_AC*

An assurance case, such as a safety case or security case, can be represented using Goal Structuring Notation (GSN), a visual representation that presents the elements of an assurance case in a tree structure. The main elements of a GSN assurance case include Goals, Strategies, Solutions (evidence), Contexts, Assumptions, and Justifications.

Additionally, an assurance case in GSN may include an undeveloped element decorator, represented as a hollow diamond placed at the bottom center of a goal or strategy element. This indicates that a particular line of argument for the goal or strategy has not been fully developed and needs to be further developed.

I will explain each element of an assurance case in GSN so you can generate it efficiently.

1. Goal – A goal is represented by a rectangle and denoted as G. It represents the claims made in the argument. Goals should contain only claims. For the top-level claim, it should contain the most fundamental objective of the entire assurance case.
2. Strategy – A strategy is represented by a parallelogram and denoted as S. It describes the reasoning that connects the parent goals and their supporting goals. A Strategy should only summarize the argument approach. The text in a strategy element is usually preceded by phrases such as “Argument by appeal to…”, “Argument by …”, “Argument across …” etc.
3. Solution – A solution is represented by a circle and denoted as Sn. A solution element makes no claims but are simply references to evidence that provides support to a claim.
4. Context (Rounded rectangles) – In GSN, context is represented by a rounded rectangle and denoted as C. The context element provides additional background information for an argument and the scope for a goal or strategy within an assurance case.
5. Assumption – An assumption element is represented by an oval with the letter ‘A’ at the top- or bottom-right. It presents an intentionally unsubstantiated statement accepted as true within an assurance case. It is denoted by A
6. Justification (Ovals) – A justification element is represented by an oval with the letter ‘J’ at the top- or bottom-right. It presents a statement of reasoning or rationale within an assurance case. It is denoted by J.

*@End\_Context\_AC*

*@Context\_ACP*

Assurance case patterns in GSN (Goal Structuring Notation) are templates that can be re-used to create an assurance case. Assurance case patterns encapsulate common structures of argumentation that have been found effective for addressing recurrent safety, reliability, or security concerns. An assurance case pattern can be instantiated to develop an assurance case by replacing generic information in placeholder decorator with concrete or system specific information.

To represent assurance case patterns in GSN format, additional decorators have been provided to support assurance case patterns. These additional decorators are used together with the elements of an assurance case to represent assurance case pattern. I will explain each additional decorator below to support assurance case pattern in GSN.

1. Uninstantiated - This decorator denotes that a GSN element remains to be instantiated, i.e. at some later stage, the generic information in placeholders within a GSN element needs to be replaced (instantiated) with a more concrete or system specific information. This decorator can be applied to any GSN element.
2. Uninstantiated and Undeveloped – Both decorators of undeveloped and uninstantiated are overlaid to form this decorator. This decorator denotes that a GSN element requires both further development and instantiation.
3. Placeholders – This is represented as curly brackets “{}” within the description of an element to allow for customization. The placeholder "{}" should be directly inserted within the description of elements for which the predicate "HasPlaceholder (X)" returns true. The placeholder "{}" can sometimes be empty or contain generic information that will need to be replaced when an assurance case pattern is instantiated.
4. Choice - A solid diamond is the symbol for Choice. A GSN choice can be used to denote alternatives in satisfying a relationship or represent alternative lines of argument used to support a particular goal.
5. Multiplicity - A solid ball is the symbol for multiple instantiations. It represents generalized n-ary relationships between GSN elements. Multiplicity symbols can be used to describe how many instances of one element-type relate to another element.
6. Optionality - A hollow ball indicates ‘optional’ instantiation. Optionality represents optional and alternative relationships between GSN elements.

The following steps is used to create an assurance case from an Assurance cases pattern.

1. Create the assurance case using only elements and decorators defined for assurance cases.
2. Remove all additional assurance case pattern decorators such as (Uninstantiated, Placeholders, Choice, Multiplicity, Optionality, and the combined Uninstantiated and Undeveloped decorator)
3. Remove the placeholder symbol "{}" and replace all generic information in placeholders “{}” with system specific or concrete information.

*@End\_Context\_ACP*

We have defined the following predicate rules for the elements and decorator used in an assurance case to ease understanding of an assurance case. The predicate rules for the elements and decorator of an assurance case begins with the delimiter *“@Predicate\_AC”* and ends with the delimiter *"@End\_Predicate\_AC”*

*@Predicate\_AC*

1. Goal(G): True if G is a goal within the assurance case. This predicate is represented as Goal (ID, Description) where ID is the unique identifier for the goal, and description is the textual information of the goal.
2. Strategy(S): True if S is a strategy within the assurance case. This predicate is represented as Strategy (ID, Description) where ID is the unique identifier for the strategy and description is the textual information of the Strategy.
3. Solution (Sn): True if Sn is evidence within the assurance case. This predicate is represented as Solution (ID, Description) where ID is the unique identifier for the evidence or solution and description is the textual information of the evidence.
4. Context(C): True if C is a context within the assurance case. This predicate is represented as Context (ID, Description) where ID is the unique identifier for the context and description is the textual information of the context.
5. Assumption (A): True if A is an assumption within the assurance case. This predicate is represented as Assumption (ID, Description) where ID is the unique identifier for the assumption and description is the textual information of the assumption.
6. Justification (J): True if J is a justification within the assurance case. This predicate is represented as Justification (ID, Description) where ID is the unique identifier for the justification and description is the textual information of the justification.
7. Undeveloped(X): True if X is either a Goal(G) or Strategy(S) marked as undeveloped. This predicate is represented as Undeveloped(X), where X can be either a goal or strategy.

*@End\_Predicate\_AC*

We have defined the following predicate rules for the additional decorators used to support assurance case patterns to ease understanding. The predicate rules for the additional decorators to support assurance case pattern begins with the delimiter *“@Predicate\_ACP*” and ends with the delimiter *"@End\_Predicate\_ACP”*

*@Predicate\_ACP*

1. Uninstantiated (X): True if element X (can be any GSN element) is marked as uninstantiated.
2. UndevelopStantiated (X): True if element X is either a Goal(G) or Strategy(S) and is marked both as uninstantiated and undeveloped.
3. HasPlaceholder (X): True if element ‘X’ (can be any GSN element) contains a placeholder ‘{}’ within its description that needs instantiation.
4. HasChoice (X, [Y], Label): True if an element ‘X’ (either a Goal(G) or Strategy(S)) can be supported by selecting among any number of elements in [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between ‘X’ and ‘Y’. A label is of the general form “m of n” (e.g. a label given as “1 of 3” implies an element ‘X’ can be supported by any one of three possible supporting elements in [Y])
5. HasMultiplicity (X, [Y], Label): True if multiple instances of an element X (either a Goal(G) or Strategy(S)) relate to multiple instances of another element [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between X and Y. (i.e., how many instances of an element in X relates with how many instances of an element in [Y]. e.g. m of n implies m instances of an element in X must be supported by n instances of an element in Y)
6. IsOptional (X, [Y], Label): True if an element X (either a Goal(G) or Strategy(S)) can be optionally supported by another element [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between X and Y. (i.e. an instance of an element in X may be supported by another instance of an element in [Y], but it is not required)

*@End\_Predicate\_ACP*

To represent an assurance case or assurance case pattern in GSN is equivalent to depicting in a hierarchical tree structure. To achieve this hierarchical tree structure, the below predicates have been defined to ease understanding of this structure. The predicate rules to support the structure of an assurance case or assurance case pattern begins with the delimiter *“@Predicate\_Structure”* and ends with the delimiter *“@End\_Predicate\_Structure”*

*@Predicate\_Structure*

1. IncontextOf (X, [N], D): True if element X at depth D has a neighbour [N] to the left or right at depth D, where ‘[N]’ can be an Assumption (A), Justification (J), or Context (C), ‘X’ can be a Goal (G), or Strategy (S) and ‘D’ represents the height or depth of the goal or strategy element and its neighbours in the GSN hierarchical structure.
2. SupportedBy (X, [C], D): True if element X at depth D has children [C] directly below it, where [C] can include Goal (G), Strategy (S), or Solution (Sn) and ‘X’ can be a Goal (G), or Strategy (S).

* If X is Strategy (S), [C] can only be Goal (G).
* If X is Goal (G), [C] can be either Goal (G), Strategy(S), or Solution (Sn).

*@End\_Predicate\_Structure*

Now, I will provide you with an example of an assurance case pattern in its predicate form and the corresponding assurance case derived from this pattern so that you can understand the process of instantiating an assurance case pattern to create an assurance case.

For example, a Security Case Pattern and the derived security case for Instant messaging (IM) software is given below. The security case pattern begins with the delimiter *"@Pattern"* and ends with the delimiter *"@End\_Pattern"* while the derived security case begins with the delimiter *"@Assurance\_case"* and ends with the delimiter *"@End\_Assurance\_case"*

*@Pattern*

Goal (G1, {software} is (acceptably secure) in {system})

Goal (G2, All identified software critical assets are protected)

Goal (G3, {software critical asset} is protected)

Goal (G4, {software critical asset} which belongs to {type} is protected)

Strategy (S1, Argument of that software assets are under protection)

Strategy (S2, Respective arguments of software critical assets)

Strategy (S3, Argument of the type which the software critical asset belongs to)

Context (C1, Definition and description of {software})

Context (C2, Definition and description of {system})

Context (C3, Definition of {acceptably secure})

Context (C4, List of {software} critical assets)

Context (C5, List of software critical asset types in PEDD)

Justification (J1, Protection of software assets supports that software is acceptably secure)

Justification (J2, Relationship between assets is clear and can be argued separately)

Justification (J3, Types of software assets are according to PEDD)

SupportedBy (G1, S1, 1)

SupportedBy (S1, G2, 2)

SupportedBy (G2, S2, 3)

SupportedBy (S2, G3, 4)

SupportedBy (G3, S3, 5)

SupportedBy (S3, G4, 6)

IncontextOf (G1, [C1, C2, C3], 1)

IncontextOf (S1, [J1], 2)

IncontextOf (S2, [C4, J2], 4)

IncontextOf (S3, [C5, J3], 6)

HasPlaceholder (G1)

HasPlaceholder (C1)

HasPlaceholder (C2)

HasPlaceholder (C3)

HasPlaceholder (C4)

HasPlaceholder (G3)

HasPlaceholder (G4)

HasMultiplicity (S2, G4, number of software critical asset = 4)

*@End\_Pattern*

*@Assurance\_case*

G1: IM server software is acceptable secure in system

C1: IM server software is used for instant messaging in corporation X and developed in VC

C2: IM server software is running on normal server with Windows XP OS and has less than 500 users

C3: Probability of user information leak caused by IM server software is less than 0.05%

S1: Argument of that software assets are protected

J1: Protection of software assets supports that software is acceptable secure

G2: All identified software critical assets are protected

S2: Respective argument of software critical assets

C4: List of IM server software critical assets: sign up information, sign up process, User account information, authentication information, login process, chat log

J2: Relationship between assets is clear and can be argued separately

G3.1: Sign up information is protected

S3.1: Argument of the type which sign up information belongs to

C5.1: List of software critical asset types in PEDD: process asset, data exchange asset, data storage asset

J3.1: Types of software assets are according to PEDD

G4.1: software related threat argument pattern Sign up information which belongs to data exchange asset and data storage asset is protected

G3.2: Sign up process is protected

G3.3: User account information is protected

G3.4: Authentication information is protected

G3.5: Login process is protected

S3.2: Argument of the type which login process belongs to

C5.2: List of software critical asset types in PEDD:process asset, data exchange asset, data storage asset

J3.2: Types of software assets are according to PEDD

G4.2 software related threat argument pattern Login process which belongs to process asset is protected

G3.6: Chat log is protected

*@End\_Assurance\_case*

Now, I would provide you with domain information about ACAS Xu (Airborne Collision Avoidance System Xu) for which you would create a security case from a given security case pattern. The domain information begins with the delimiter *“@Domain\_Information”* and ends with the delimiter *"@End\_Domain\_Information”*

*@Domain\_Information*

ACAS Xu (Airborne Collision Avoidance System Xu) is a collision avoidance system designed for use in unmanned aerial vehicles (UAVs), commonly known as drones. The primary objective of ACAS Xu is to enhance the safety of drone operations by preventing collisions between drones or between a drone and other objects in its environment.

The scenario involves two drones. One called the “intruder” which is any other drone or object that poses a collision threat to the ownship. and the other called the “ownship.” which is the perspective we adopt. The ownship is equipped with ACAS Xu and has a functional space in which it must operate. This space is conceptually partitioned into two operational areas: collision avoidance threshold and collision volume with an elevated risk of collision for the ownship with intruders. When no risk of collision is detected, the ownship follows the current heading to the destination area. Otherwise, if another drone is detected in the collision volume, the ownship will turn right or left to avoid the collision and prevent the intruder from reaching the collision avoidance threshold.

The architecture of ACAS Xu contains the following components.

* Sensors: The ownship's sensors gather data on potential intruders, including their velocity, angle, and distance relative to the ownship.
* Processor: The collected data is processed to compute a suitable avoidance strategy (e.g., turn left, turn right, or do nothing).
* Planner: Based on the processor's decision, a trajectory is planned to navigate the ownship safely while avoiding collisions.
* Actuator: The planned trajectory is executed by the actuator, ensuring the ownship follows the new path.

ACAS Xu's security can be compromised if an attacker alters the messages sent to the processor, leading to incorrect decisions that may result in collisions. Therefore, ensuring the security of ACAS Xu involves:

security requirements decomposition that aims to identify security threats, and formalization of the system and the security threats to later verify the absence of threats when developing a secure system. If it can be shown that all the relevant threats have been identified and mitigated, then the system is acceptably secure.

The following security requirements (SRs) below are imposed to design a secure ACAS Xu.

* SR1: The GPS messages are genuine and have not been intentionally altered.
* SR2: The processor must receive data only from valid sensors.
* SR3: The system should employ mechanisms to mitigate unauthorized disclosure of the planning information.
* SR4: ACAS Xu development shall be done considering security risk assessment procedures.

The four SR are decomposed into requirements about the satisfaction of asset protection (SR1 –SR3)

and secure development process requirements (SR4). The former concerns requirements to protect resources that are worth protecting. The latter concerns the requirements about the development activities that must conform to a relevant secure development methodology and/or security standard.

In addition, ACAS Xu has low level elements that capture functional architecture in terms of components and connectors, and the behavioural aspects of the architectural elements. These elements include the following.

* Component: a modeling artifact which represents a piece of software architecture.
* MsgPassing: the representation of a message exchanged between two components (sender, receiver).
* Port: the interaction point through which a Component can communicate with its environment.
* ConnectorMPS: a link that enables communication between Ports.
* Payload: the useful data contained in a Message.

Based on the Microsoft STRIDE threat analysis technique, the following security threats (STs) against the components and the communication links are identified from the security requirements (SRs).

* ST1: Tampering – This threat is identified from SR1 and involves GPS sensors and processor.
* ST2: Spoofing - SR2 Sensors and processor
* ST3: Elevation of privileges - SR3 Planning system

Finally, to ensure that ACAS Xu is acceptably secure, during the creation of its security case, an instance of the goal (G0.X) is created for each security threat against which the system must be protected, where 𝑋 denotes the order of the threat.

*@End\_Domain\_Information*