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, an Assurance Case Pattern for the Interpretability of a Machine Learning system and the derived assurance case is given below. The assurance case pattern begins with the delimiter *"@Pattern"* and ends with the delimiter *"@End\_Pattern"* while the derived assurance case begins with the delimiter *"@Assurance\_case"* and ends with the delimiter *"@End\_Assurance\_case"*

*@Pattern*

Goal (G1, Interpretability Claim. The {ML Model} is sufficiently {interpretable} in the intended {context})

Goal (G2, Right Method. The right {interpretability methods} are implemented, i.e. the correct information is faithfully being explained)

Goal (G3, Right Context. {Interpretations} produced in the {intended context})

Goal (G4, Right Format {Interpretability methods} are presented in the right {format} for the {audience})

Goal (G5, Right Time {Interpretations) produced at the {appropriate time})

Goal (G6, Right Setting {Interpretations) are available in the (right setting))

Goal (G7, Right Audience {Interpretations} produced for the {right audience})

Goal (G8, {Interpretability method} is right type e.g. local/global (i.e. the correct thing is being explained).)

Goal (G9, {Interpretability method} is suitably faithful to {ML model} process)

Strategy (S1, Argument based on the {essential aspects of interpretability})

Strategy (S2, Argument over {interpretability methods})

Context (C1, {ML Model})

Context (C2, {Interpretable})

Context (C3, (Context: setting time and audience})

Context (C4, (Essential aspects of interpretability})

Context (C5, {Interpretability methods})

Context (C6, {Format of interpretations})

SupportedBy (G1, S1, 1)

SupportedBy (S1, [G2, G3, G4], 2)

SupportedBy (G2, S2, 3)

SupportedBy (G3, [G5, G6, G7], 3)

SupportedBy (S2, [G8, G9], 4)

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

IncontextOf (S1, C4, 2)

IncontextOf (G2, C5, 3)

IncontextOf (G3, C6, 3)

IncontextOf (G4, C6, 3)

HasPlaceholder (G1)

HasPlaceholder (C1)

HasPlaceholder (C2)

HasPlaceholder (C3)

HasPlaceholder (C4)

HasPlaceholder (G2)

HasPlaceholder (G3)

HasPlaceholder (G4)

HasPlaceholder (C5)

HasPlaceholder (C6)

HasPlaceholder (S2)

HasPlaceholder (G5)

HasPlaceholder (G6)

HasPlaceholder (G7)

HasPlaceholder (G8)

HasPlaceholder (G9)

Uninstantiated (C1)

Uninstantiated (C2)

Uninstantiated (C3)

Uninstantiated (C4)

Uninstantiated (S1)

Uninstantiated (G2)

Uninstantiated (C5)

Uninstantiated (C6)

Uninstantiated (G3)

Uninstantiated (S2)

UndevelopStantiated (G4)

UndevelopStantiated (G5)

UndevelopStantiated (G6)

UndevelopStantiated (G7)

UndevelopStantiated (G8)

UndevelopStantiated (G9)

*@End\_Pattern*

*@Assurance\_case*

G1: Interpretability Claim - The ML system is sufficiently interpretable in the intended context.

C1: Dual NN system

C2: Interpretable = transparency of system logic

C3: Context: setting - retinal diagnosis pathway; time- alongside diagnosis predictions; audience - retinal/ clinicians.

S1: Argument based on the essential aspects of interpretability.

C4: Essential aspects of interpretability: method, context & format

G2: Right Method - The system structure and segmentation map provide transparency of the system logic and allow clinicians to understand decisions.

C5: System structure and segmentation map

S2: Argument over interpretability methods

G8: Interpretability method is right type (the correct thing is being explained).

G14: The system structure closely resembles the normal decision-process taken by clinicians (first producing the segmentation map then the diagnosis) (undeveloped)

G9: Interpretability method is suitably faithful to the system process.

G15: The interpretability method is the comprehensible structure of the system and the production of the segmentation map. (undeveloped)

G3: Right Context - Segmentation map produced in the retinal diagnosis pathway.

C6: Segmentation map

G5: Right Time - Segmentation map is produced alongside diagnosis prediction.

G11: Clinicians need an explanation alongside every diagnosis prediction. (undeveloped)

G6: Right Setting - Explanations are available in the clinical setting.

G12: Clearly clinicians need to be able to access these explanations within the clinical setting. (undeveloped)

G7: Right Audience - Explanations produced for the retinal clinicians.

G13: The clinicians need an explanation to understand and trust system predictions. (undeveloped)

G4: Right Format - The format of the interpretation is the transparent system logic, including the segmentation map

C6: Segmentation map

G10: The system structure, including production of the segmentation map, closely resembles the normal clinical decision-making process & offers comprehensible insight into system logic. (undeveloped)

*@End\_Assurance\_case*

Now, I would provide you with domain information about a safety critical system for which you would create a safety case from a given safety case pattern. The domain information begins with the delimiter *“@Domain\_Information”* and ends with the delimiter *"@End\_Domain\_Information”*

*@Domain\_Information*

The provided domain information describes a safety-critical system called the Generic Patient-Controlled Analgesia (GPCA) system, also known as an infusion pump. This system faces two operational hazards: "Overinfusion" and "Underinfusion," which can have dire consequences for patient safety.

The following potential causes for these hazards are identified:

1. Flow rate does not match the programmed rate.
2. Programmed rate too low
3. Dose limit exceeded due to excessive bolus requests.
4. Bolus volume/concentration too high

To mitigate these hazards, eight safety requirements (SR1.1, SR1.2, SR1.4, SR1.5, SR1.10, SR3.4.6, SR6.1.3, SR6.1.4) have been established. Some of the properties of these safety requirements include:

1. "Flow rate sensor is equipped."
2. "Period (to trigger the underinfusion alarm) is 15 minutes."
3. "Flow rate is less than 90% of the programmed rate setting."

These safety requirement properties are based on references such as "FDA standard," "Expertise opinion," and "Previous knowledge."

Additionally, In the safety case pattern for the GPCA system, relationships between different placeholder items within the safety case pattern such as the mitigation relationship between safety requirements and potential causes of hazards are supported using the Object Constraint Language (OCL).

OCL is a formal language that allows the expression of constraints on UML models. For example, OCL expressions like scenario.spec and operations such as allInstances() can be used to model relationships. An OCL expression like scenario.spec.allInstances() can represent all instances of spec related to a particular scenario.

*@End\_Domain\_Information*