LAG Grammar Semantics

- **PROG**: A LAG strategy or procedure, containing a set of instructions (programming constructs) defining the user and presentation adaptation in an adaptive hypermedia environment.

- **DESCRIPTION**: The description of PROG; contains a natural language description of the behavior of the adaptive strategy; it serves as the label (meta-description) for the whole strategy. It is important, as laic (non-programmer) authors should be able to extract from it the necessary elements to make a decision about using this adaptation or not.

- **VARIABLES**: The variables of PROG; contains the list of variables that are used in the adaptive strategy. This information can be used by a laic (non-programmer) author to decide what attributes of the GM (goal and constraints model) should be filled-in for this strategy.

- **INITIALIZATION**: The static initialization part of PROG; in this part, the initial experience of the user, when entering the adaptive environment, is described. This is useful so that a user doesn’t enter a void environment. Here, all the default decisions are set. Adaptive environments which are adaptable but not adaptive can only render this part.

- **IMPLEMENTATION**: The dynamic implementation part of PROG; in this part, the interactivity between the adaptive environment and the user is described (for instance, the effect of user clicks).

- **STATEMENT**: The LAG language is a simple language built of a number of programming constructs, or statements, as follows:
  - **IFSTAT**: This statement is similar to IF statements in other programming languages, and is used for condition-action rules; the exact syntax is given in the grammar. This is the basic building block of the adaptation language. Any other (higher level) building block is translatable to it, as all adaptive hypermedia environments use this as the basis of adaptation.
  - **WHILESTAT**: This statement is similar to WHILE statements in other programming languages, and is used for loops; the exact syntax is given in the grammar.
  - **FORSTAT**: This statement is similar to WHILE statements in other programming languages, and is used for loops; the exact syntax is given in the grammar.
  - **BREAKSTAT**: This statement is similar to BREAK statements in other programming languages; the exact syntax is given in the grammar. It is used to exit a FOR or a WHILE loop. It is currently not available for the MOT2AHA conversion.
  - **GENSTAT**: This statement uses the hierarchical structure in the DM (domain model) and GM (goal and constraints model) for adaptive navigation. It specifies that more general concepts, higher in the hierarchy than the current concept, will be displayed to the user, given that the condition(s) is (are) fulfilled. It is currently not available for the MOT2AHA conversion; instead, the child-parent relation can be used.
  - **SPECSTAT**: This statement uses the hierarchical structure in the DM (domain model) and GM (goal and constraints model) for adaptive navigation. It specifies that more specific concepts, lower in the hierarchy than the current concept, will be displayed to the user, given that the condition(s) is (are) fulfilled. It is currently not available for the MOT2AHA conversion; instead, the child-parent relation can be used.
  - **ACTION**: This is part of the basic building block of condition-actions. It can be used by itself, as if the condition attached to it would be set to TRUE. This statement is the only one that allows specification of updates and changes of visible (such as the current screen) or invisible (such as the user knowledge) variables.
  - **CONDITION**: this part of a statement can appear in various constructs, for ultimately triggering condition-action rules. A condition can be a change in a variable, but also an event, such as the user access to a certain concept. The condition is specified by a prerequisite, or a set of prerequisites, of which enough of them should be fulfilled.
    - **enough**: this is a part of a condition statement; it specifies that a number of conditions should be fulfilled. How many of these conditions are enough is specified by the VALUE.
  - **ATTRIBUTE**: can appear in a condition or an action; can be a generic attribute of a DM (domain map), GM (goal and constraints map), UM (user map) or PM (presentation map) (e.g., DM.Concept.knowledge); or can be a specific attribute (e.g., ‘\Neural Networks Map\Learning\Introduction\Weight’.show). Generic
attributes are used in adaptive strategies that can be applied on (almost) any DM and GM. When specific attributes are used, the whole strategy becomes only applicable on a specific, given DM and/or GM (such as in our example, only on the “Neural Networks Map” domain model).

- **ATTR**: the variables in LAG follow the naming in LAOS (as shown by the syntax). There are a series of standard, predefined names, such as: title | keywords | text | introduction | conclusion | exercise | Relatedness | type (inherited from the DM – domain model); child | parent (inherited from both DM and GM – goal and constraints model); label| weight | order (inherited from the GM); next | ToDo | menu | show | access | visited (inherited from the PM – presentation model).
  - **Title**: title of a concept.
  - **Keywords**: keywords list of a concept.
  - **Text**: main body of a concept.
  - **Introduction**: introductory (hyper-)text of a concept.
  - **Conclusion**: concluding (hyper-)text of a concept.
  - **Exercise**: an exercise referring to the contents of a concept.
  - **Relatedness**: a (typed) relation of a concept with another concept, other than the hierarchical relation. Can be used to present all related concepts of the current concept.
  - **Type**: an attribute of a concept attribute (e.g., the attribute ‘Title’ is of type ‘Title’). Can be used to present only attributes of a given type.
  - **Child**: any sub-concept of the given concept in the concept hierarchy (concepts below in this hierarchy), as well as all attributes of a group concept.
  - **Parent**: the super-concept of the given concept in the concept hierarchy (concept above in this hierarchy)
  - **Label**: the textual label used in the GM for giving extra presentational information about a concept (such as pedagogical information – e.g., this concept is for beginners, thus it is labeled ‘beg’)
  - **Weight**: the numerical label used in the GM for giving extra presentational information about a concept (such as pedagogical information – e.g., this concept is for readers of the short version of 30% of the material, thus it is labeled ‘30’); it can be used in combination with Label to give more detailed extra presentational behaviour.
  - **Next**: (currently,) it can be used (only) in the Initialization part to turn the “Next” button on or off. PM.GM.next = false turns it off (default is true). (for beginners, it might be enough to have a next button without other confusing menus or lists)
  - **ToDo**: (currently,) it can be used (only) in the Initialization part to make the “ToDo” list of concepts still to visit visible or not. PM.GM.ToDo = false turns it off (default is visible).
  - **Menu**: (currently,) it can be used (only) in the Initialization part to make the “Menu” hierarchical list of concepts visible or not. PM.GM.menu = false turns it off (default is visible). (for global learners, for instance, it is advisable to have a menu)
  - **Show**: this is a reserved variable (from the PM – presentation model) used to show a concept to the user, when true. By default, all concepts’ show variable is set to false, so they are invisible. If no concepts’ show variable is turned to true in the initialization part of the adaptive strategy, the user will enter a blank screen.
  - **Access**: this is a reserved variable (from the PM – presentation model) used to specify that a concept is currently accessed by the user.
  - **Visited**: this is a reserved variable (from the PM – presentation model) used to specify that a concept has been accessed by the user in the past.