A Methodology to Validate Interactive Storytelling Scenarios in Linear Logic

Kim Dung Dang, Ronan Champagnat, Michel Augeraud

A Methodology to Validate Interactive Storytelling Scenarios in Linear Logic

Kim Dung Dang, Ronan Champagnat, and Michel Augeraud

University of La Rochelle - L3i, Avenue Michel Crépeau, 17042 La Rochelle, France
{kim_dung.dang, ronan.champagnat, michel.augeraud}@univ-lr.fr

Abstract. Debugging is one of the main requirements for Interactive Storytelling (IS) authoring tools. During the authoring phase, authors have to specify large numbers of rules and actions as well as consider many possible paths. As a consequence, flaws may happen and finding them "by hand" is complex. Therefore the validation of an IS becomes a crucial issue and automatic assistance in this process is needful. Originated from those requirements, we propose, within the framework of this paper, a methodology using Linear Logic, based on analyzing automatically the resource allocation mechanisms, that helps authors derive a valid scenario of an IS. To do this, we model a scenario by a Linear Logic sequent, then prove the received sequent, which allows building and examining automatically all the possible branches in the scenario, thereby authors may guarantee that all the decisions (that may be made while unfolding the IS) lead to satisfactory endings of their goals. The paper ends with an example on an extract of an educational game to illustrate the methodology.

Keywords: Interactive Storytelling (IS), Linear Logic, validation of scenario, debugging, proof graph.

1 Introduction

Interactive Digital Storytelling, especially the gaming sector, nowadays, is considered as one of popular research directions that have had important and successful contribution to the edutainment domain [3, 10, 14, 13, 8]. [30] demonstrated that, thanks to the effects of interactivity, games brought a learning support that was more efficient than the traditional supports (non-interactive formats such as text, oral presentation, video). Indeed, the learner, from a passive role (sustains an available course), becomes active (can control the course through her/his actions during the game), and so her/his involvement in the learning process is reinforced.

However, the unfolding of a game (the unfolding of the story corresponding with the game) and its level of interactivity have commonly been thought to be opposite [18]. The first relates to a designer's control on the game s/he has created as the second relates to a player's control on the game s/he is playing. Research to deal with this opposition, in general, is divided into two major families: scenario-driven

approach (discourse point of view) and emergent narrative theory (character point of view) [9].

The first family [23, 31, 26, 22] aims to guarantee that the story development is coherent and leads to authors' desired effects. And hence when a player's action deviates from the pre-computed story plan, the system either replans (gets the story back on track), or makes the player's action have no effect on the story progress. As a consequence, the player cannot direct the story unfolding in a considerable way. On the opposite, the emergent narrative theory [2, 4, 28, 27] gives complete freedom to the player, who may deeply influence, through her/his actions, the evolution of the virtual world where s/he has been immersed. This means that the story will emerge from the player's interaction with the game, and the unfolding of the story is not based on any specific structure. However, its foremost limit is the quality of the created course of events, in terms of consistency and pertinence, which highly depends on the player and therefore cannot be guaranteed.

In previous works [5, 6], we showed how to use Linear Logic to model an IS thanks to which the strong points of both the discourse point of view and the character point of view are combined (*i.e.* the story development is coherent and leads to authors' desired effects while still ensuring player's freedom and her/his determinant role on the story's unfolding). These are really necessary, especially in the edutainment domain, as through the effects of interactivity, it is more interesting and efficient for the player/learner to comprehend knowledge/educational lessons transmitted by authors, and at the same time messages that authors have inserted into the game as well as the coherence of the created course of events are also guaranteed.

In [9], we studied the principle for a "good" IS controller which allows creating a discourse made by the author intention, the state of the system and player's action choices. We also gave a set of objectives that such an IS controller should satisfy: the player does not feel constrained by the game but s/he can determine its evolution; the virtual world must provide a coherent environment that is appropriate for player's actions; the progress of the game has to respect a structure of discourse (a common structure of a discourse is made of introduction; stating problems; solving them step by step; conclusion) which has been pre-defined by experts of the domain (authors/designers).

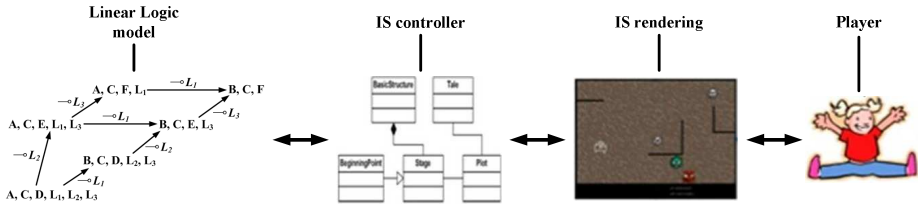

Fig. 1. Architecture of the system using Linear Logic to manage interactive video games.

As a result, a system to manage interactive video games with the assistance of Linear Logic that ensures the foregoing is needful. Indeed, we have realized, in [7], such a system whose architecture is composed of three components: Linear Logic model, IS controller and IS rendering (see Fig. 1). The IS rendering directs the

"rendering process" of the game (which interacts immediately with the player). The IS controller aims to manage "intelligently" the unfolding of the game and to ask the IS rendering to show suitable interfaces for the game at each step, by taking into account the suggestions of the Linear Logic model, the state of the system and player's action choices (transmitted to the IS controller via the IS rendering). More concretely, it will play the role of an expert system, and so adapt the progress of the game to each player via its action choices. The *Linear Logic model* component stores a sequent that models the situation of the game at each moment (it is updated after each step; at the initial moment it models the initial situation of the game and so determines its scenario). The automatic reasoning of the sequent (the automatic sequent proof) assists directly the IS controller in managing the unfolding of the game.

The works described in this paper present a methodology for authors to validate their IS model which is represented by the Linear Logic sequent mentioned above at the moment before it becomes the input of the *Linear Logic model* component (*i.e.* in the scenario building phase). More concretely, the goal of the paper is to propose a methodology based on analyzing automatically the resource allocation mechanisms, that allows authors to derive a valid scenario of an IS which guarantees that all the decisions (that may be made during unfolding a game) lead to satisfactory endings of authors' goals. To this purpose, our idea is to build and examine automatically the proof graph of the sequent representing all the possible branches in the current scenario, then to eliminate and/or to modify manually inappropriate ones (which do not direct toward successful endings because of problems on the resource allocation mechanisms), this process is repeated until authors receive the most pertinent scenario for their aims. Fig. 2 describes the modeling process of an IS till its scenario is validated.

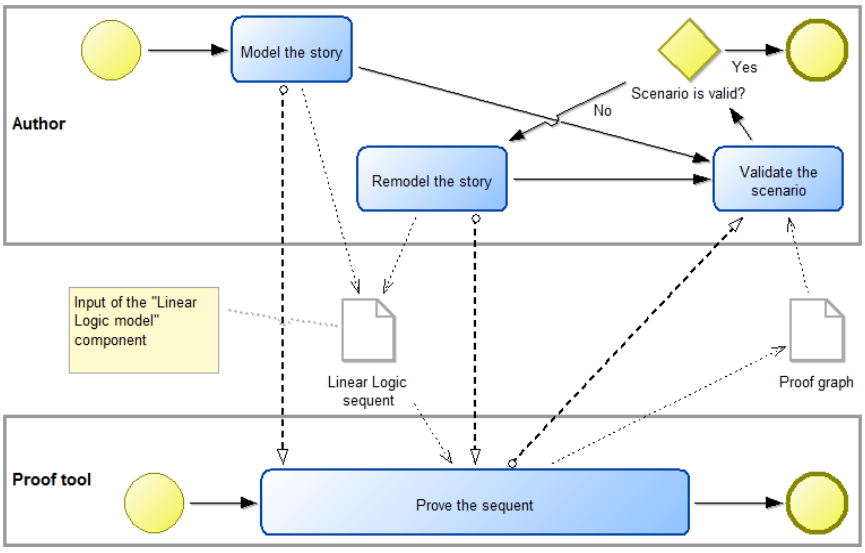

Fig. 2. Modeling process of an IS until its scenario is validated
(built according to the *BPMN – Business Process Modeling Notation* standard [35]).

The paper begins with a presentation of related works. Then we offer a brief introduction to Linear Logic and explain the method of IS modeling by means of Linear Logic regarding both the discourse point of view and the character point of view. After that, we show in detail our solution for validating a scenario of a story and give an example on an extract of an educational game to illustrate it. In these sections, we will present and explain three main contributions of the paper: (1) introduce the connectives & and ⊕ to model the choices of the player and of the IS controller; (2) assign a priority order to each event/action in the story; and the most important one, (3) propose a novel sequent proof algorithm that is based absolutely on the meaning of the Linear Logic connectives. Lastly, we discuss about issues which should be settled and future works to be done on the Linear Logic approach for IS modeling.

However, above all, we define some important notions that are used in our approach:

- A *story* (a *game*) is a set of entities, of events/actions and of constraints that solves a set of problems, describes an evolution concerning a set of characters and/or of objects. It consists in starting from an initial situation, then in solving the given set of problems in order to reach a final situation that corresponds with one of satisfactory endings of authors' goals.
- A *discourse* is an ordered sequence of events/actions that is a possible unfolding of a story. Therefore a same story can generate various discourses. This consists in scheduling the events/actions corresponding with the story.
- A *scenario* is a set of all the possible discourses for a story. If we change anything in the story then we will receive a new scenario.

2 Related Works

Pizzi and Cavazza presented in [24] an authoring approach which evolved from the development of proof-of-concept prototypes, due to the ability of IS systems to directly formalize narrative actions. They have also built tools that provide the representation of all the possible unfolding plans ensuring thus a greater visibility on the whole story space, in order to control the progress of the generated content. Its rationale is to support the authoring of a complex planning domain, by checking its completeness and its consistency. Moreover, the combinatorial aspect of content generation can quickly overflow the amount of possible paths that may be exploited if done using a brute force approach. For that reason, as an alternative to the offline generation of a complete narrative, an interactive mode allows a step-by-step generation of a solution including the visualization of all possible outcomes. After each action is selected by the user/author, the system automatically offers a list of possible subsequent actions/operators. Every operator in the planning domain is tested for applicability and whenever preconditions of an operator are satisfied, the operator will be applied to create a new state that may further be extended. For efficiency purposes, and in order to avoid redundancy, the system only develops new states that diverge significantly from their parent state. The whole tree can then be automatically scanned and all the possible solutions can be listed and visualized. The system also provides the user/author with the possibility of interaction at any time to change the

current plan. Finally, several analysis tools may allow the validation of the generated narrative content. Thus, in short, Pizzi and Cavazza have used the direct analysis of actions' sequence to evaluate consistency and so could validate preliminarily the generated narrative.

Vega and Natkin, in [29], considered the ability to describe non-deterministic structure of the game narration, gave a semi formal approach and proposed the use of Petri Nets for game analysis and specification. Their approach leaded to define and determine the dynamic structure of a game from a bottom up method: from transactions to levels, thanks to which it was possible to guarantee that the actions carried out by the player remained coherent within the framework of the game. To this purpose, they used techniques of subnet construction so that if each of these subnets was coherent then the whole also remained coherent. Their proposal was illustrated on the well known game *Myst* and could be considered as a starting point for a Game Design method and an authorware tool.

KANAL [20] is a tool which helps users to create sound plans by simulating them, checking for a variety of errors, and presenting the results in an accessible format that allows users to see an overview of the plan steps or timelines of objects in the plan. Indeed, this tool checks process models specified by users, reports possible errors and provides specific suggestions to users about how to fix those errors. Therefore, it helps users building or modifying process models by detecting invalid statements and pointing out what additional knowledge needs to be acquired and what existing knowledge needs to be modified. However, the pity that it is not designed for validating interactive scenarios in the gaming sector.

3 Brief Introduction to Linear Logic

Linear Logic is a substructural logic, introduced by Girard [11] as an executable formal model and a refinement of Classical Logic, where uses of the *Contraction* and *Weakening* rules are carefully controlled. In addition, it also considers atoms and formulas as resources that are consumed and/or produced. As a result, in Linear Logic, two instances of an atom (or of a formula) are different from one instance. Unlike Classical Logic, Linear Logic is not applied to determine whether an assertion is true or not, but it is employed to represent the validity of how resources (atoms and/or formulas) are used when proving an assertion. In other words, we are interested in writing the proof and in analyzing the resource choices made during this phase. Besides, Linear Logic is well suited to model the natural reasoning through the mechanism of sequent calculus introduced by Gentzen [17]. In addition, the linguistic theory uses a subset of Linear Logic (intuitionist multiplicative and non commutative), that corresponds to Lambek calculus [1]. Consequently, Linear Logic provides a semantic framework to model causality, automatic reasoning as well as resource allocation mechanisms for a story (for interested readers, [25] is a good reference to have more information on how to model resource allocation problems in Linear Logic, while [19] showed a complete computational interpretation of several fragments of Linear Logic and established exactly the complexity level of those fragments).

In order to model an IS, in this paper, we do not employ all the features of Linear Logic, but just the following connectives:

- *⊸: linear implication (imply)*, expresses the possibility of deduction. Example: "1$ ⊸ 1kg strawberries" means that we can give 1$ to buy 1kg strawberries.
- *⊗: multiplicative conjunction (times)*, expresses multisets of resources. Example: "1$ ⊸ 1kg strawberries ⊗ 1kg tomatoes" means that we can give 1$ to buy 1kg strawberries and 1kg tomatoes.
- *&: additive conjunction (with)*, expresses an external choice to the system (for instance coming from a player) if it is in the left part of the sequent (this connective does not make sense (and so is not used) if it is in the right part of the sequent in our approach). Example: "1$ ⊸ tea & coffee" means that we (the player) can choose tea or coffee when we give 1$ to an automatic machine.
- *⊕: additive disjunction (plus)*, expresses an internal choice to the system (for instance coming from an IS controller) if it is in the left part of the sequent. Example: "1$ ⊸ tea ⊕ 1$" means that it is the automatic machine which decides it will give us tea or return to us 1$ depending on the availability of tea in the machine, if this formula is in the left part of the sequent. If the connective ⊕ is in the right part of the sequent, it is only used to connect distinct consequents. For example, if the right part of a sequent is "tea ⊕ coffee", this means that if the sequent is provable, then from the left part of the sequent, we can receive either tea or coffee.

A sequent is an expression $\Gamma \vdash \Delta$, where Γ and Δ are sequences of atoms and/or of formulas; ⊢ (turnstile) is used to separate its left (antecedents/available resources) and right (consequents/conclusions) parts. For example, "A ⊗ (A ⊸ B) ⊢ B" (or "A, A ⊸ B ⊢ B") means the possibility to produce a copy of "B" by consuming the available resources "A" and "A ⊸ B" (we can substitute the connective ⊗ between two atoms, between two formulas, or between an atom and a formula in the left part of a sequent by the comma "," to be briefer). From the left part of a sequent, we may lead to many valid conclusions/consequents, and at the same time, a proof (how to reach a conclusion/consequent) is not unique, meaning that there exist many ways of reaching a same conclusion/consequent. In order to prove a sequent (*i.e.* showing that it is syntactically correct) we have employed the sequent calculus [17]. This consists in rewriting the sequent, by making a substitution of its formulas until obtaining initial sequents (a sequent is called an initial one if it is in the form "A ⊢ A" where A is any atom). Thus for a same sequent, there may be several successful proofs as well as several unsuccessful ones. As a result, the proof strategy becomes crucial in using Linear Logic to reason on the logic of discourse and on the resource allocation mechanisms for a story.

4 Interactive Storytelling Modeling by Means of Linear Logic

According to the Greimas' analysis [16], an abstract formula, called a *narrative program* (whose logic is close to the concept of *linear implication* in Linear Logic), is

used to represent an event/action and the progress of a story (an ordered sequence of events/actions) may be described by a *narrative program array*. More concretely, *"doing (action) is defined as a temporal succession from a state to another state, effected by any agent (the subject of doing) and affecting any patient (the subject of state). A state may be broken down into a subject of state and an object of state and the junction between them, which is either a conjunction (the subject is with the object) or a disjunction (the subject is without the object). […] The subject of doing may or may not correspond to the subject of state; in other words, what accomplishes the action may or may not be what is affected by it. […] A narrative program array is composed of at least two narrative programs between which at least one temporal relation (succession, simultaneity) or one logical relation (simple or reciprocal presupposition, mutual exclusion, comparing/compared, etc.) is identified"*.

Originated from those ideas, we have proposed a method to model an IS by means of Linear Logic, in which we take into account the states of the story, the (player and non-player) characters' states, player's action choices as well as action choices of the IS controller.

4.1 Method Description

Our method is briefly described as follows:

- Player and non-player characters are modeled by atoms in the left part of the sequent. An atom corresponds to a state of a character considering a certain point of view. Therefore a character at a moment can be modeled by various atoms which constitute the character's situation at that moment and are put in a state vector corresponding with the character. Thus, the size and the component of the state vector of a character may vary during the unfolding of the story.
- States of the story are modeled as atoms in the left part of the sequent. These atoms represent the discourse point of view (the authors' desired effects) in the modeling process.
- The availability of the characters' states and of the states of the story is considered as the availability of the corresponding atoms in the left part of the sequent.
- Player's action choices are represented by inputs. This means that the player decides her/his occurrence in the unfolding of the story by entering the inputs. These inputs are modeled as atoms in the left part of the sequent and will become available after being entered into the story by the player.
- An additive conjunction (disjunction) formula in the left part of the sequent represents action choices of the player (the IS controller) in the progress of the story.
- An event/action may modify the situation of a (or some) character(s) and/or the states of the story. This is similar to the working of the connective $\multimap$. As a result, we link a linear implication formula to an event/action of the story. Besides, in order to help authors control better the unfolding of a story, each event/action is assigned a priority order (the smaller the priority order is, the higher is the priority of an event/action). Therefore, the event/action with the

smallest priority order (the highest priority) will be selected if there are many satisfied events/actions at a same moment. For example, let us see the sequent A, D, A ⊸ B, D ⊸ E ⊢ B ⊗ E. It has two available atoms A and D, two events/actions A ⊸ B (priority order = 1) and D ⊸ E (priority order = 2). As the event/action A ⊸ B is more priority than the event/action D ⊸ E, although at this moment both the events/actions are satisfied, the event/action A ⊸ B will be executed before the event/action D ⊸ E. Thus the transformation order of the sequent is: (A, D, A ⊸ B, D ⊸ E ⊢ B ⊗ E) ⟶ (B, D, D ⊸ E ⊢ B ⊗ E) ⟶ (B, E ⊢ B ⊗ E).

- An outcome (goal/conclusion/consequent) of a story is an authors' desired ending. It corresponds to an atom, or a set of atoms (connected between them by the connective ⊗) in the right part of the sequent. Outcomes of the story are connected by the connective ⊕.
- A proof expresses the actions/events (linear implication formulas) to be executed, to reach a consequent of a sequent (may be successful or unsuccessful), and hence it is equivalent to a discourse which is an ordered sequence of events/actions that is a possible unfolding of the story.
- From a sequent, we are able to build all the ways of writing proofs. Therefore it corresponds to a scenario which is a set of all the possible discourses for a story.

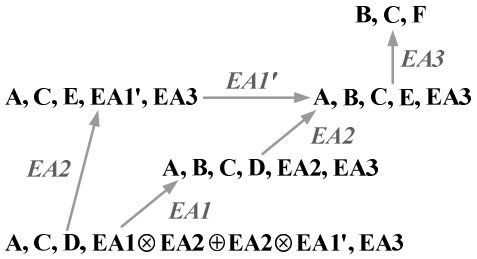

Fig. 3. All the possible discourses in the scenario corresponding to the sequent A, C, D, EA1 ⊗ EA2 ⊕ EA2 ⊗ EA1', EA3 ⊢ B ⊗ C ⊗ F where EA1 (priority order = 1): A ⊗ C ⊸ A ⊗ B ⊗ C; EA2 (priority order = 2): D ⊸ E; EA1' (priority order = 3): A ⊗ C ⊸ A ⊗ B ⊗ C; EA3 (priority order = 4): A ⊗ C ⊗ E ⊸ C ⊗ F; each node corresponds to the left part of the sequent at a moment; each edge is one executed event/action.

For instance, let us consider the sequent A, C, D, EA1 ⊗ EA2 ⊕ EA2 ⊗ EA1', EA3 ⊢ B ⊗ C ⊗ F where EA1 (priority order = 1): A ⊗ C ⊸ A ⊗ B ⊗ C; EA2 (priority order = 2): D ⊸ E; EA1' (priority order = 3): A ⊗ C ⊸ A ⊗ B ⊗ C; EA3 (priority order = 4): A ⊗ C ⊗ E ⊸ C ⊗ F. The scenario corresponding to this sequent has six atoms/states A, B, C, D, E, F in which A, C, D are available; four events/actions EA1, EA1', EA2, EA3 in which the priority of EA1 > EA2 > EA1' > EA3; one outcome B ⊗ C ⊗ F including three atoms/states B, C and F; one choice made by the IS controller (choose either EA1 ⊗ EA2 meaning that the sequent becomes A, C, D, EA1, EA2, EA3 ⊢ B ⊗ C ⊗ F, or EA2 ⊗ EA1' meaning that the sequent becomes A, C, D, EA2, EA1', EA3 ⊢ B ⊗ C ⊗ F). All the possible

discourses in the scenario (two successful discourses/proofs/paths/branches) are given in Fig. 3 (simplified to the substitution of the events/actions): EA1 ⟶ EA2 ⟶ EA3 and EA2 ⟶ EA1' ⟶ EA3. Besides, it is easy to find that, the IS modeling by Linear Logic helps us know the shape of a scenario of a story as well as modify it simply.

4.2 Two Principal Improvements of the Modeling Method in Comparison with Previous Works

Within the framework of the previous works [5], we did neither employ the connectives & and ⊕ to represent action choices of the player and of the IS controller in the progress of the story, nor assign any priority order to the events/actions (their priority was the same). As a consequence, all the possible discourses in a scenario came from the possibilities of scheduling its events/actions (if there were many satisfied events/actions at a same moment, each of them would be selected one after another). For example, let us see the sequent A, C, D, EA1, EA2, EA3 ⊢ B ⊗ C ⊗ F where EA1: A ⊗ C ⊸ A ⊗ B ⊗ C; EA2: D ⊸ E; EA3: A ⊗ C ⊗ E ⊸ C ⊗ F. The scenario corresponding to this sequent has six atoms/states A, B, C, D, E, F in which A, C, D are available; three events/actions EA1, EA2, EA3 (their priority is the same); one outcome B ⊗ C ⊗ F including three atoms/states B, C and F. All the possible discourses (possibilities of choice) in the scenario are given in Fig. 4 (simplified to the substitution of the events/actions): EA1 ⟶ EA2 ⟶ EA3, EA2 ⟶ EA1 ⟶ EA3 and EA2 ⟶ EA3 (two successful discourses and one unsuccessful discourse).

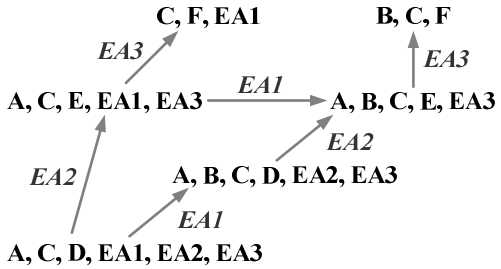

Fig. 4. All the possible discourses in the scenario corresponding to the sequent A, C, D, EA1, EA2, EA3 ⊢ B ⊗ C ⊗ F where EA1: A ⊗ C ⊸ A ⊗ B ⊗ C; EA2: D ⊸ E; EA3: A ⊗ C ⊗ E ⊸ C ⊗ F (within the framework of the previous works [5]).

However, such modeling of an IS has met the following inconveniences (which will be surmounted through the two main improvements of the IS modeling method described in this paper):

- authors cannot determine that which possibilities of choices in the scenario are decided by the player, which ones are decided by the IS controller. For instance, in Fig. 4, at the first node of the graph (A, C, D, EA1, EA2, EA3), there are two possibilities of choices (execute either EA1 or EA2) but we cannot know who (the player or the IS controller) will decide that. The use of

the connectives & and ⊕ in this paper allows authors to deal with that ambiguity;

- it is difficult to express the following kind of choice: choose one event/action in a list of events/actions and after executing the chosen event/action, it is not allowed to execute the remaining events/actions in the list. For instance, let us see the sequent in Fig. 4, we cannot determine: choose either EA1 or EA2 and after executing EA1 (EA2), it is not allowed to execute EA2 (EA1), because after executing EA1, we always have the possibility of executing EA2 and vice versa. The use of the connectives & and ⊕ in this paper allows authors to determine that, indeed, we can modify the sequent as follows: A, C, D, (EA1 & EA2), EA3 ⊢ B ⊗ C ⊗ F;
- it is complex for authors to define the priority order in choosing the events/actions. For instance, for the sequent in Fig. 4, we cannot express that EA1 is more priority than EA2 and so it has to be chosen at the first node of the graph (A, C, D, EA1, EA2, EA3). This also makes the authors' control on the possibilities of choices in the scenario limited, more concretely, there may exist other possibilities besides authors' intentional discourses. For instance, let us reconsider the sequent in Fig. 4, if authors desire that EA3 is executed last (after both EA1 and EA2), although the discourse EA2 ⟶ EA3 is not in their intention, it still can happen. In this paper, by proposing a priority order between the events/actions (therefore the event/action with the highest priority will be selected if there are many satisfied events/actions at a same moment), this weak point is solved. For example, the sequent in Fig. 3 only includes two discourses EA1 ⟶ EA2 ⟶ EA3 and EA2 ⟶ EA1' ⟶ EA3 as the priority of EA3 is lowest.

5 A Methodology to Validate a Scenario Modeled by a Linear Logic Sequent

5.1 Problem to Settle

As presented above, a Linear Logic sequent models a scenario of a story and a way of proving the sequent expresses a discourse. Thus we can validate the scenario by building and examining automatically the proof graph of the sequent which includes all the possible discourses in the scenario. As each discourse corresponds with a choice made either by the player or by the IS controller in the story unfolding, the problem to settle in this paper is as follows: We will automatically build the proof graph of the sequent, each branch of which is one possibility of choice in the scenario. These choices come from the player (using the connective & in the left part of the sequent), and/or from the IS controller (using the connective ⊕ in the left part of the sequent). The final received result of each discourse is compared with each outcome/goal of the story, thereby we will receive all the discourses (branches/paths) leading to unsuccessful and/or successful endings as well as some important statistical

information on the scenario such as: number of discourses, number of successful/unsuccessful discourses, number of successful discourses for each outcome/goal. Thanks to these feedbacks, authors may quickly have an overall view on the current scenario, then they can improve it if necessary and so receive a new sequent/scenario (this process is repeated in the scenario building phase until authors receive a scenario that is the most pertinent for their aims, in other words, until the scenario is validated – see Fig. 2).

For example, let us consider a scenario corresponding to the sequent A, C, EA1 & (EA2 ⊕ EA3) ⊢ B ⊗ F ⊕ A ⊗ F where EA1: A ⊗ C ⊸ B ⊗ F; EA2: A ⊸ E; EA3: C ⊸ F. The player can choose between two possibilities: either EA1 or EA2 ⊕ EA3. If s/he chooses EA2 ⊕ EA3, then the IS controller will decide which event/action (either EA2 or EA3) will happen. Thus the proof graph of the sequent includes three discourses/branches/paths (to be briefer we keep the labels EA1, EA2 and EA3 if they are not executed):

(1) Execute EA1: (A, C, A ⊗ C ⊸ B ⊗ F & (EA2 ⊕ EA3) ⊢ B ⊗ F ⊕ A ⊗ F) ⟶ (B, F ⊢ B ⊗ F ⊕ A ⊗ F)
(2) Execute EA2: (A, C, EA1 & ((A ⊸ E) ⊕ EA3) ⊢ B ⊗ F ⊕ A ⊗ F) ⟶ (E, C ⊢ B ⊗ F ⊕ A ⊗ F)
(3) Execute EA3: (A, C, EA1 & (EA2 ⊕ (C ⊸ F)) ⊢ B ⊗ F ⊕ A ⊗ F) ⟶ (A, F ⊢ B ⊗ F ⊕ A ⊗ F)

As there are two outcomes/goals B ⊗ F and A ⊗ F, we compare the final received result of each discourse with each of these outcomes. Lastly, we have the following conclusion on the scenario:

- if the player chooses EA1 (discourse 1), this discourse will go to one successful ending of the goal of the story (B ⊗ F);
- if the player chooses EA2 ⊕ EA3
    - if the IS controller decides that EA2 will happen (discourse 2), this discourse cannot go to any successful ending of the goal of the story;
    - if the IS controller decides that EA3 will happen (discourse 3), this discourse will go to one successful ending of the goal of the story (A ⊗ F);
- in short, the scenario has three discourses: two successful discourses (one discourse reaches the outcome B ⊗ F, one discourse reaches the outcome A ⊗ F) and one unsuccessful discourse.

Thanks to these feedbacks, authors can improve the current scenario if necessary (so receive a new sequent/scenario) and then repeat the validation process until obtaining the best possible scenario (*i.e.* the scenario that is the most pertinent for their aims).

We can find that the Linear Logic sequent proof plays a key role in our approach. The next section will speak of some existing solutions to prove a sequent and explain why they are not really suitable for the problem mentioned above.

5.2 Some Existing Linear Logic Sequent Proof Solutions

Proving a sequent consists in rewriting the sequent, by making a substitution of its formulas until obtaining initial sequents. As for a same sequent, there may be several successful proofs as well as several unsuccessful ones, the proof strategy becomes crucial in using Linear Logic to reason on the logic of discourse and on the resource allocation mechanisms for a story. We introduce here some existing solutions concerning this question.

Petri Nets

It has been shown [12, 21] that there is an equivalence between a Linear Logic sequent and a Petri net (with some restrictions). Thus the proof of a Linear Logic sequent corresponds to the firing of a sequence of transitions in its equivalent Petri net, starting from the initial marking and going to the expected marking. Our previous works in [5] has implemented this Linear Logic sequent proof strategy in order to generate the set of discourses for the given scenario/sequent. However they have encountered some problems:

- Among the possibilities of choice verified while proving a sequent, there may be many discourses/branches that the verification is useless when we consider the different priority order between the events/actions. For example, let us see the sequent A, C, D, EA1, EA2, EA3 $\vdash$ B $\otimes$ C $\otimes$ F where EA1 (priority order = 1): A $\otimes$ C $\multimap$ A $\otimes$ B $\otimes$ C; EA2 (priority order = 2): D $\multimap$ E; EA3 (priority order = 3): A $\otimes$ C $\otimes$ E $\multimap$ C $\otimes$ F. If we use a Petri net corresponding to the sequent to prove it like [5] did, the Petri net will verify all three discourses/branches EA1 $\rightarrow$ EA2 $\rightarrow$ EA3, EA2 $\rightarrow$ EA1 $\rightarrow$ EA3 and EA2 $\rightarrow$ EA3 (see Fig. 4) because for the previous works, the priority order of EA1, EA2 and EA3 is not considered. However, this sequent, in reality, only includes one successful discourse EA1 $\rightarrow$ EA2 $\rightarrow$ EA3 as the priority of EA1 > EA2 > EA3. Therefore it is useless to verify the discourses/branches in the proof graph of a sequent that do not respect the priority order between its events/actions (such as EA2 $\rightarrow$ EA1 $\rightarrow$ EA3 and EA2 $\rightarrow$ EA3 in the considered example), which results in quickly the combinatorial explosion as the complexity of the scheduling algorithm of events/actions is exponential.
- The transformation from a Linear Logic sequent into a Petri net has also brought us some restrictions. Indeed, we have neither been able to use all the four connectives ($\otimes$, &, $\oplus$, $\multimap$) and nor been able to model complex events/actions in the previous works [5]. More concretely, the left part of the sequent has only accepted the connectives $\otimes$, $\multimap$; its right part has only accepted the connectives $\otimes$, $\oplus$; and it has only been possible for us to model simple events/actions corresponding with a unique format of the linear implication formulas (A1 $\otimes$ A2 $\otimes$ … $\otimes$ An $\multimap$ B1 $\otimes$ B2 $\otimes$ … $\otimes$ Bm, where Ai, Bj are atoms), in other words, we have not been able to model more complex events/actions, such as A $\multimap$ (B $\multimap$ C) or (A $\multimap$ B) $\multimap$ (C $\multimap$ D). As a consequence, the transformation from a Linear Logic sequent into a Petri net has reduced the power of Linear Logic and so it has been impossible for us to

model complex systems. Besides, the "reciprocal transformation" between a sequent and its corresponding Petri net (almost) after each step during unfolding the story causes a "waste" of time and computer resources to process. Thus, another solution which calculates directly on Linear Logic models (does not have to base on Petri nets) is necessary.

- Besides, in order to avoid the combinatorial explosion, we have applied a dynamic analysis which consisted in generating each local part of the proof graph by considering a limited window of events/actions while unfolding the story. Nevertheless, that strategy has had several weak points: (1) it always eliminates unsuccessful discourses just after detecting them during executing the game, as a consequence, it reduces the scenario's variety; (2) it does not guarantee that the unfolding of the story is always successful. For instance, let us see the sequent A, EA1, EA2, EA3, EA4, EA5 $\vdash$ D where EA1: A $\multimap$ B; EA2: A $\multimap$ C; EA3: A $\multimap$ D; EA4: B $\multimap$ E; EA5: E $\multimap$ F. We consider in this example a window of size 3 (*i.e.* a window of 3 events/actions). In the beginning, after verifying the whole of the possible discourses made up of 3 events/actions, the IS controller detects that if the player chooses either EA1 or EA3, there is not any problem, but if s/he chooses EA2, it is an unsuccessful discourse. Therefore, it eliminates EA2 (unfortunately this makes the scenario's variety reduced), the sequent becomes A, EA1, EA3, EA4, EA5 $\vdash$ D. Then for example, the player executes EA1, so the sequent becomes B, EA3, EA4, EA5 $\vdash$ D. The IS controller continues to verify the whole of the possible discourses made up of 3 events/actions, it finds that now there is only one discourse which is unsuccessful. However, this moment is too late to change and so it is impossible for the IS controller to do anything. Thus the dynamic analysis by considering a limited window of events/actions mentioned in [5] has not guaranteed that the unfolding of the story was always successful. As a result, we should propose another approach to solve this inconvenience, and at the same time to maintain the scenario's variety (*i.e.* not have to eliminate any discourse).

Other Linear Logic Sequent Proof Tools

In addition to Petri nets, in order to prove a Linear Logic sequent, we may also employ Lolli [33], Lygon [34], LINK prover [15], and especially Linear Logic Theorem Prover *llprover* [32]. However, they do not respond exactly to our requirement:

- These available tools can only be used to show whether or not a sequent is proved, which does not offer enough necessary information. Concretely, they cannot divide the sequent into branches and hence, we cannot know how many branches there are, which branches are successful and which branches are unsuccessful (our purpose is not only to conclude if a sequent is true or false but also to build its proof graph which corresponds with all the possible discourses in the given scenario).
- They do not consider the priority order of the linear implication formulas (similarly to our previous works in [5]). As a consequence, among the possibilities verified while proving a sequent, there may be many branches

that the verification is useless, besides, in some cases, the received result is not correct. For example, let us see the sequent A, B, EA1, EA2 ⊢ E where EA1 (priority order = 1): A ⊸ C; EA2 (priority order = 2): B ⊸ D. These tools will verify all two branches EA1 ⟶ EA2 as well as EA2 ⟶ EA1 (because they do not care the priority order between EA1, EA2) and conclude that the sequent is false. However, this sequent, in reality, only includes one unsuccessful discourse EA1 ⟶ EA2 because EA1 is more priority than EA2. Therefore it is useless to verify the branch EA2 ⟶ EA1. Another example, the sequent A, B, EA1, EA2 ⊢ C ⊗ D where EA1 (priority order = 1): B ⊸ D; EA2 (priority order = 2): A ⊗ B ⊸ B ⊗ C, according to these tools, is true because they find the successful branch EA2 ⟶ EA1 but in reality, this sequent is false because it only includes one unsuccessful discourse EA1 ⟶ EA2 in the corresponding scenario.

- Besides the foregoing, Lolli, Lygon and the LINK prover do not handle completely all the four connectives (⊗, &, ⊕, ⊸); *llprover* includes all of them but it deals with the connective ⊕ in the left part of the sequent differently from our approach. More concretely, Lolli and Lygon do not consider the sequents containing ⊕ in the left part; the LINK prover is only applied to the multiplicative fragment of Linear Logic; for *llprover*, it deals with the connective ⊕ in the left part of a sequent as follows:

$$\frac{\Gamma 1, \Gamma 2, \Gamma 4 \vdash \Delta \qquad \Gamma 1, \Gamma 3, \Gamma 4 \vdash \Delta}{\Gamma 1, \Gamma 2 \oplus \Gamma 3, \Gamma 4 \vdash \Delta} \oplus L$$

  This means that *llprover* is interested in the truth of the sequent Γ1, Γ2 ⊕ Γ3, Γ4 ⊢ Δ and will conclude it is true if and only if both the subsequents Γ1, Γ2, Γ4 ⊢ Δ and Γ1, Γ3, Γ4 ⊢ Δ are true. If it finds that Γ1, Γ2, Γ4 ⊢ Δ is false, it will conclude the sequent Γ1, Γ2 ⊕ Γ3, Γ4 ⊢ Δ is false and does not prove Γ1, Γ3, Γ4 ⊢ Δ. This mechanism is not really identical to our approach. Indeed, we are only interested in the truth of the two subsequents, not in the truth of the original sequent. In other words, in any case we will consider both the subsequents and conclude the truth for each of them, not for the original sequent.

5.3 A Novel Algorithm to Build and Examine Automatically all the Possible Branches in a Scenario

The foregoing shows that the application of the solutions presented in Section 5.2 to deal with the problem posed in Section 5.1 is not really suitable. Thus, in order to settle directly and entirely this problem as well as overcome all the inconveniences mentioned above, we have proposed a novel algorithm of sequent proof that is based absolutely on the meaning of the Linear Logic connectives (does not have to base on any Petri net), which will be explained in detail in the following.

Before beginning, it is necessary to note that we do not intend to prove a Linear Logic sequent in general case, but just apply the algorithm to validate a scenario of a

story which is modeled by a sequent in our approach. As a consequence, within the framework of this paper, the following "components" in the syntax of Linear Logic are not considered (because they do not appear in the sequent received after the modeling process): linear negation (*e.g.* $A^{\perp}$); the connective & in the right part of the sequent; the connective ⅋; the constants 1, 0, ⊤, ⊥; the exponentials !, ?; and the quantifiers ∀, ∃. Besides, the proof of a Linear Logic sequent is a very complex question and results in combinatorial explosion, so we have proposed some constraints to make the algorithm simpler as well as to increase the processing speed of the program. We will present these constraints below together with the detailed description of the algorithm.

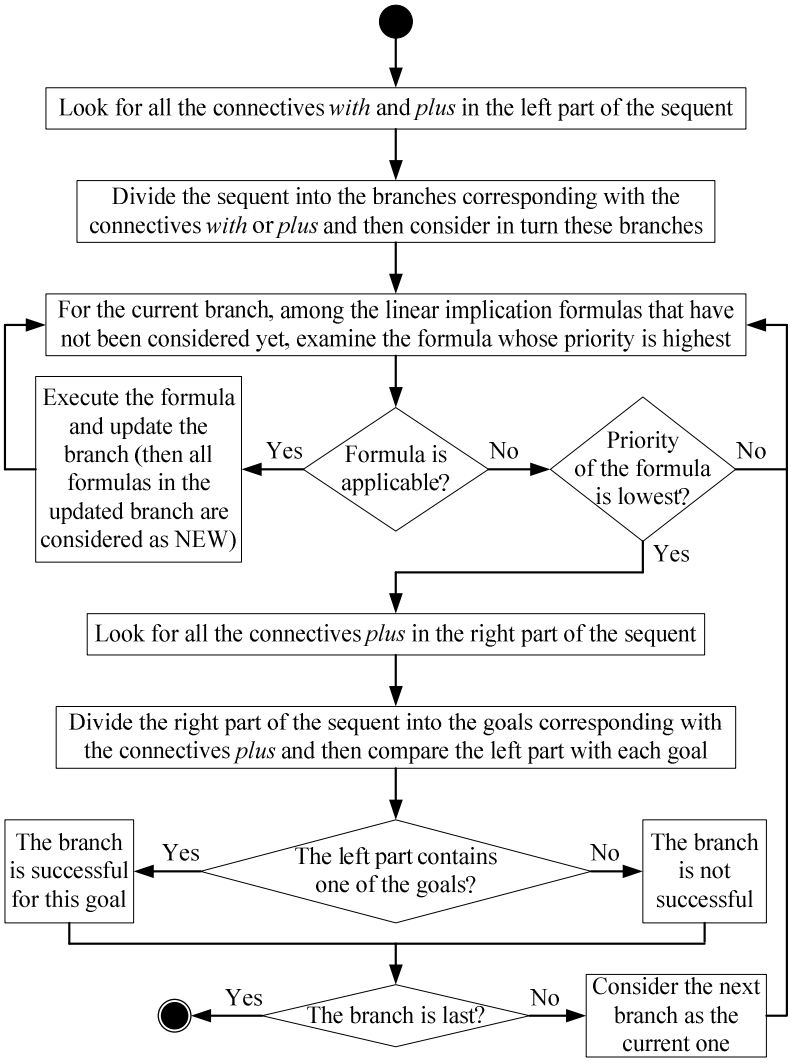

Fig. 5. Outline of the algorithm.

Fig. 5 gives the outline of the algorithm thanks to which we can build and examine automatically the proof graph of a sequent, thereby receive all the discourses (branches/paths) leading to unsuccessful and/or successful endings as well as some important statistical information on the corresponding scenario such as: number of discourses, number of successful/unsuccessful discourses and number of successful discourses for each outcome/goal.

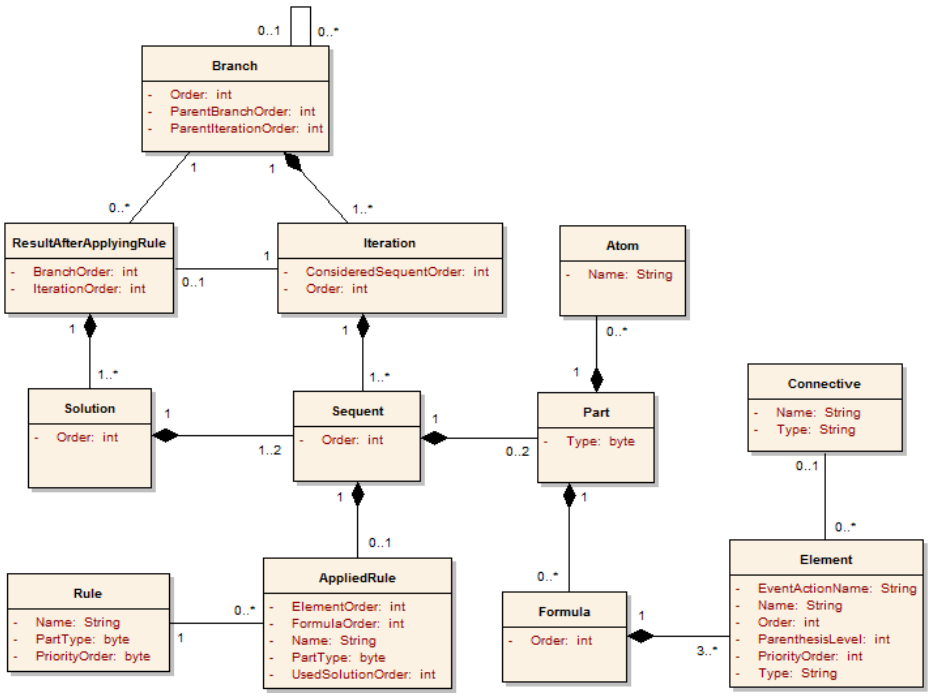

Fig. 6. Components used in the algorithm and their relation.

The components used in the algorithm and their relation are described in Fig. 6 which is concretely explained hereinafter:

- *Connective:* In the IS modeling by means of Linear Logic, we employ the four following connectives (all of them (⊗, &, ⊕, ⊸) are used in the left part of the sequent, but only two of them (⊗, ⊕) are used in the right part of the sequent because & and ⊸ do not make sense in our approach):
  - Name = times; Type = Multiplicative Conjunction
  - Name = with; Type = Addititive Conjunction
  - Name = plus; Type = Addititive Disjunction
  - Name = imply; Type = Linear Implication
- *Rule:* There are seven rules considered in the process of proving a sequent (the rule &R (applied to the connective & of a formula in the right part of the sequent) is not used)
  - Name = Multiplicative Conjunction; PartType = 1; PriorityOrder = 1: This rule is applied to the connective ⊗ of a formula in the left part

(PartType = 1) of the sequent. It is employed to rewrite a sequent (means substituting the connective ⊗ between two atoms, between two formulas, or between an atom and a formula in the left part by the comma ","). Its priority order is 1 (the highest priority), so this rule is always executed at first if possible.

$$\frac{\Gamma 1, \Gamma 2, \Gamma 3, \Gamma 4 \vdash \Delta}{\Gamma 1, \Gamma 2 \otimes \Gamma 3, \Gamma 4 \vdash \Delta} \otimes \mathrm{L}$$

- Name = Additive Conjunction; PartType = 1; PriorityOrder = 2: This rule is applied to the connective & of a formula in the left part of the sequent. There are two choices depending on the player.

$$\frac{\Gamma 1, \Gamma 2, \Gamma 4 \vdash \Delta}{\Gamma 1, \Gamma 2 \,\&\, \Gamma 3, \Gamma 4 \vdash \Delta} \,\&\mathrm{L} \qquad \frac{\Gamma 1, \Gamma 3, \Gamma 4 \vdash \Delta}{\Gamma 1, \Gamma 2 \,\&\, \Gamma 3, \Gamma 4 \vdash \Delta} \,\&\mathrm{L}$$

- Name = Additive Disjunction; PartType = 1; PriorityOrder = 3: This rule is applied to the connective ⊕ of a formula in the left part of the sequent. There are two choices depending on the IS controller.

$$\frac{\Gamma 1, \Gamma 2, \Gamma 4 \vdash \Delta}{\Gamma 1, \Gamma 2 \oplus \Gamma 3, \Gamma 4 \vdash \Delta} \oplus \mathrm{L} \qquad \frac{\Gamma 1, \Gamma 3, \Gamma 4 \vdash \Delta}{\Gamma 1, \Gamma 2 \oplus \Gamma 3, \Gamma 4 \vdash \Delta} \oplus \mathrm{L}$$

The rules &L and ⊕L are employed to divide a sequent into sub-branches. Their priority order is 2 and 3 respectively, so these rules are verified just after the rule ⊗L.

- Name = Linear Implication; PartType = 1; PriorityOrder = 4: This rule is applied to the connective ⊸ of a formula in the left part of the sequent to execute that linear implication formula. It is considered after the rules &L and ⊕L, so its priority order is 4.

$$\frac{\Gamma 1', \Gamma 4' \vdash \Gamma 2 \qquad \Gamma 1'', \Gamma 3, \Gamma 4'' \vdash \Delta}{\Gamma 1, \Gamma 2 \multimap \Gamma 3, \Gamma 4 \vdash \Delta} \multimap \mathrm{L}$$

Thus, in order to prove the sequent Γ1, Γ2 ⊸ Γ3, Γ4 ⊢ Δ, we will have to prove both Γ1', Γ4' ⊢ Γ2 and Γ1'', Γ3, Γ4'' ⊢ Δ. Here Φ' is a subset of Φ and Φ'' = Φ - Φ', where Φ is Γ1 and Γ4 respectively. For example, if Γ1 = A, B (its subsets are {∅}, {A}, {B}, {A, B}), then either Γ1' = ∅ and Γ1'' = A, B; or Γ1' = A and Γ1'' = B; or Γ1' = B and Γ1'' = A; or Γ1' = A, B and Γ1'' = ∅. Although there are many solutions to consider in turn (depending on the combination between Γ1'/Γ1'' and Γ4'/Γ4''), if the original sequent is proved, then only one of them is successful (the others are unsuccessful). As a consequence, we will have to try each solution until either we find a successful one (the sequent is proved), or on the contrary, all of them are unsuccessful

(the sequent is not proved). For instance, let us consider the sequent A, A ⊗ B ⊸ D, B ⊢ D, we have Γ1 = A, Γ2 = A ⊗ B, Γ3 = D, Γ4 = B, Δ = D, so:

- either Γ1' = ∅ and Γ1'' = A; or Γ1' = A and Γ1'' = ∅;
- either Γ4' = ∅ and Γ4'' = B; or Γ4' = B and Γ4'' = ∅.

Thus, we will have to try the four following solutions:

- ∅, ∅ ⊢ A ⊗ B and A, D, B ⊢ D (means that Γ1' = ∅, Γ4' = ∅, Γ1'' = A, Γ4'' = B): It is unsuccessful;
- A, ∅ ⊢ A ⊗ B and ∅, D, B ⊢ D: It is unsuccessful;
- ∅, B ⊢ A ⊗ B and A, D, ∅ ⊢ D: It is unsuccessful;
- A, B ⊢ A ⊗ B and ∅, D, ∅ ⊢ D: It is successful.

As we have found one successful solution, the original sequent A, A ⊗ B ⊸ D, B ⊢ D is proved.

- Name = Linear Implication; PartType = 2; PriorityOrder = 5: This rule is applied to the connective ⊸ of a formula in the right part (PartType = 2) of the sequent. Although ⊸R does not make sense in our IS modeling approach, we still have to consider this rule because it may appear during the sequent proof process. More concretely, it is a consequence while executing some complex linear implication formulas in the left part (such as (A ⊸ B) ⊸ (C ⊸ D)). Therefore, this rule is verified after the rule ⊸L, so its priority order is 5.

$$\frac{\Delta 1, \Gamma \vdash \Delta 2}{\Gamma \vdash \Delta 1 \multimap \Delta 2} \multimap \mathrm{R}$$

- Name = Additive Disjunction; PartType = 2; PriorityOrder = 6: This rule is applied to the connective ⊕ of a formula in the right part of the sequent. As the right part expresses a list of outcomes that authors want to obtain, in the proof process we choose each outcome one after another, and verify whether or not the current branch is successful for that outcome. This rule is only considered after executing all applicable linear implication formulas in the left part, so its priority order is 6.

$$\frac{\Gamma \vdash \Delta 1}{\Gamma \vdash \Delta 1 \oplus \Delta 2} \oplus \mathrm{R} \qquad \frac{\Gamma \vdash \Delta 2}{\Gamma \vdash \Delta 1 \oplus \Delta 2} \oplus \mathrm{R}$$

- Name = Multiplicative Conjunction; PartType = 2; PriorityOrder = 7: This rule is applied to the connective ⊗ of a formula in the right part of the sequent. Its working is similar to the one of the rule ⊸L (see above) and it is only verified after dividing the right part into goals by the rule ⊕R, so its priority order is 7.

$$\frac{\Gamma' \vdash \Delta 1 \qquad \Gamma'' \vdash \Delta 2}{\Gamma \vdash \Delta 1 \otimes \Delta 2} \otimes R$$

To increase the processing speed of the algorithm, and make it simpler at the same time, we have proposed the following constraint: If in a formula, there are two types of connective: either &, ⊕ and ⊸ (if the formula is in the left part) or ⊕ and ⊗ (if the formula is in the right part), the formula will be transformed so that the connectives &, ⊕ (in the left part) and ⊕ (in the right part) are always the "principal one" of the formula. The usefulness here is that we can divide the sequent into "smaller" ones (by the rules &L, ⊕L, ⊕R) before applying the rules ⊸L, ⊗R. For example, consider the sequent A, A & B ⊸ C ⊢ C. For the formula A & B ⊸ C in the left part, its principal connective is ⊸, so we have to transform the formula into (A ⊸ C) & (B ⊸ C), its principal connective now is &. And hence, the sequent becomes A, (A ⊸ C) & (B ⊸ C) ⊢ C, therefore we can divide it into two sub-sequents A, A ⊸ C ⊢ C and A, B ⊸ C ⊢ C by the rule &L, then we continue to apply the rule ⊸L to prove these simpler sequents.

- A *sequent* is composed of two *parts* (separated by ⊢): Left part (Type = 1) and Right part (Type = 2) but each may be empty. A part includes *atoms* and/or *formulas*. The atoms' name represents their sense. The formulas are distinguished by the *Order* attribute (= 1, 2,…) which shows the formulas' order in the left part or in the right part of the sequent. Each formula is composed of at least three *elements* (atom, connective, atom). The elements are distinguished by their order number (= 1, 2,…) which expresses their position in the formulas. There are seven types of element:
  - Type = "Open Parenthesis", Name = "(", EventActionName = "", PriorityOrder = "0", ParenthesisLevel is the level of the parenthesis;
  - Type = "Close Parenthesis", Name = ")", EventActionName = "", PriorityOrder = "0", ParenthesisLevel is the level of the parenthesis;
  - Type = "Additive Conjunction", Name = "with", EventActionName = "", ParenthesisLevel = "0", PriorityOrder = "0";
  - Type = "Additive Disjunction", Name = "plus", EventActionName = "", ParenthesisLevel = "0", PriorityOrder = "0";
  - Type = "Multiplicative Conjunction", Name = "times", EventActionName = "", ParenthesisLevel = "0", PriorityOrder = "0";
  - Type = "Linear Implication", Name = "imply", ParenthesisLevel = "0", its *EventActionName* and *PriorityOrder* attributes store the name and the priority order of the corresponding event/action in the game;
  - Type = "Atom", EventActionName = "", ParenthesisLevel = "0", PriorityOrder = "0", the element's name represents its sense.

As every formula always has one principal connective, it always has one and only one applicable rule corresponding with this connective. Thus, a sequent may have at the same time a lot of applicable rules depending on its formula number. We have to choose only one rule among them by considering the priority order of the rules (see above). If there are some rules with the same priority order, the rule corresponding with the "most left" formula will

be selected (in the case that there are several applicable rules ⊸L, the rule corresponding with the event/action with the highest priority in the story will be selected). For example, let us analyze the sequent A, B, C ⊕ D, (A ⊸ D) & ((A ⊸ E) ⊕ (B ⊸ F)) ⊢ F. The left part (Type = 1) A, B, C ⊕ D, (A ⊸ D) & ((A ⊸ E) ⊕ (B ⊸ F)) has two available atoms A and B, two formulas C ⊕ D (Order = 1) and (A ⊸ D) & ((A ⊸ E) ⊕ (B ⊸ F)) (Order = 2). The first formula C ⊕ D owns three elements, the second one (A ⊸ D) & ((A ⊸ E) ⊕ (B ⊸ F)) owns nineteen elements. In the second formula, there are four open parentheses and four close parentheses. The level of the open parentheses is 1, 1, 2, 2; the level of the close parentheses is 1, 2, 2, 1 respectively. The right part (Type = 2) only has one atom F. As there are two formulas C ⊕ D and (A ⊸ D) & ((A ⊸ E) ⊕ (B ⊸ F)) in the left part of the sequent, we can apply two rules ⊕L and &L. However, the priority of the rule &L is higher (PriorityOrder = 2), and hence it is the unique rule used for this sequent at the first step of the algorithm.

- *AppliedRule:* This component is employed to represent the rule applied to the sequent at each iteration step in the proof process. We use the following attributes to describe it:
  - FormulaOrder: It is the order of the formula to which the rule is applied.
  - PartType = 1 (or = 2) if the formula is in the left (right) part of the sequent.
  - ElementOrder: It is the order of the connective corresponding with the rule in the formula.
  - Name: It is the name of the rule corresponding with the type of the connective (Multiplicative Conjunction, Addititive Conjunction, Addititive Disjunction, Linear Implication).
  - UsedSolutionOrder: This attribute shows the order of the solution used in the result after executing the applied rule to the sequent, its objective is to keep the trace while proving the sequent. As presented above, depending on each rule, the received result may be composed of: either one solution including one sequent (⊗L, ⊸R), so UsedSolutionOrder is always 1; or two solutions that each include one sequent (&L, ⊕L, ⊕R) so UsedSolutionOrder is either 1 or 2; or many more solutions (⊸L, ⊗R) that each include two sequents, so UsedSolutionOrder = 1, 2, 3, 4,…

For instance, let us review the sequent A, B, C ⊕ D, (A ⊸ D) & ((A ⊸ E) ⊕ (B ⊸ F)) ⊢ F. The rule applied to it is &L. This rule, used for the second formula in the left part, corresponds to the sixth element (the connective &) of the formula. The result received after applying the rule is composed of two solutions that each include one sequent:

- Solution 1: A, B, C ⊕ D, A ⊸ D ⊢ F. If we use this solution as the received result, the applied rule will be represented by the following XML code segment:

```
<Sequent>
   <Part Type="1"> … </Part>
   <Part Type="2"> … </Part>
   <AppliedRule Name="Additive Conjunction" FormulaOrder="2"
   ElementOrder="6" PartType="1" UsedSolutionOrder="1"/>
</Sequent>
```

- Solution 2: A, B, C ⊕ D, (A ⊸ E) ⊕ (B ⊸ F) ⊢ F. If we use this solution as the received result, the applied rule will be represented by the above XML code segment with one difference (**`UsedSolutionOrder="2"`**).

- A proof graph is composed of one or many *branches* depending on the possibilities of choice of the player (using &L) and/or of the IS controller (using ⊕L). The branches are distinguished by their order number (= 1, 2,…). Each branch may have one or many *iteration* steps which are distinguished by their order number (= 0, 1,…). Each iteration step may have one or many *sequents* which are distinguished by their order number (= 1, 2,…). The first iteration step (Order = 0) in the first branch (Order = 1) will contain the original sequent (Order = 1). For each iteration step, in the beginning, we always consider its first sequent, if it is an initial one, then we continue to consider the second one and so on. The order of the considered sequent is placed in *ConsideredSequentOrder* (its default value is 1). If all sequents in the current iteration step are initial ones, then the branch corresponding with this iteration step is successful. If the considered sequent is not an initial one then we have to look for a rule applicable to it (paying attention to the priority between the rules ⊗L, &L, ⊕L, ⊸L, ⊸R, ⊕R and ⊗R).
    - Find a rule applicable to this sequent, we continue as follows:
        - Add this rule to the sequent to keep the trace of the proof process.
        - Record the received result after applying the rule to the sequent in *ResultAfterApplyingRule*:
            - ✓ BranchOrder is the order of the current branch.
            - ✓ IterationOrder is the order of the current iteration step.
            - ✓ One result may own one, two or many more *solution(s)* which are distinguished by their order number (= 1, 2,…). Each solution may include either one or two sequent(s), so their order number is 1, 2 respectively.
        - Add a new iteration step to the current branch whose order is the order of the current iteration step plus 1. The new iteration step will contain the first solution in ResultAfterApplyingRule. If in the current iteration step, there is (are) one (some) sequent(s) following the considered sequent, we have to add it (them) to the new iteration step (behind the sequent(s) of the first solution).

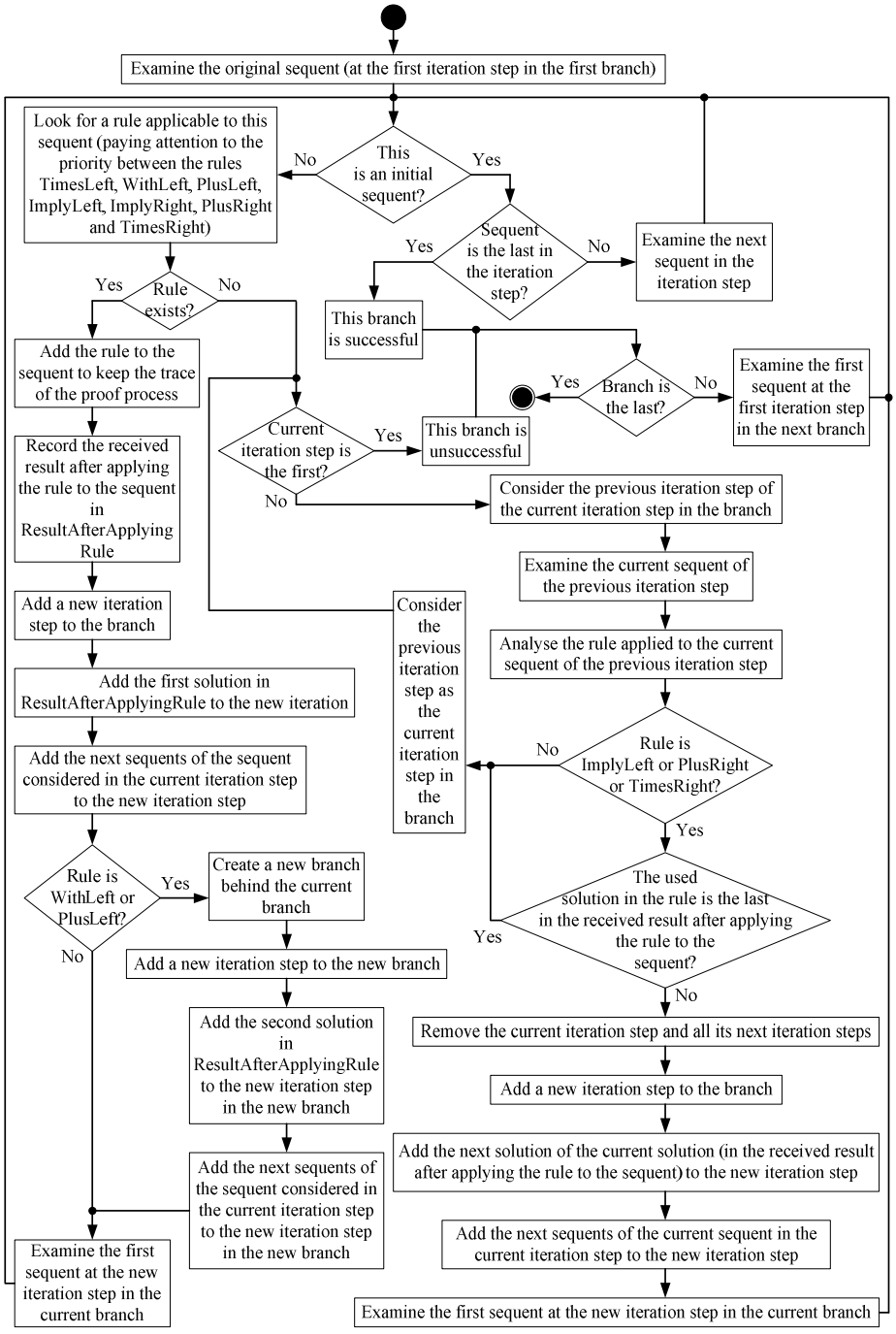

Fig. 7. Detailed operation process of the algorithm.

        - If the applied rule is either &L or ⊕L, then we will add a new branch whose order is the order of the current branch plus 1, the order of its parent branch (ParentBranchOrder) is the order of the current branch, the order of its parent iteration step (ParentIterationOrder) is the order of the current iteration step (for the first branch, ParentBranchOrder = 0 and ParentIterationOrder = 0). After that, we add a new iteration step (Order = 0, ConsideredSequentOrder = 1) to the new branch which contains the second solution in ResultAfterApplyingRule. If in the current iteration step, there is (are) one (some) sequent(s) following the considered sequent, we have to add it (them) to the new iteration step in the new branch (behind the sequent(s) of the second solution).
    - Do not find any rule applicable to this sequent, if the current iteration step is the first one in the current branch, then this branch is unsuccessful. On the contrary, we continue to consider the previous iteration step of the current iteration step. If the rule applied to the current sequent of the previous iteration step is ⊸L or ⊕R or ⊗R, we will use the next solution of the current solution in ResultAfterApplyingRule as the received result after applying this rule to the current sequent of the previous iteration step.

Fig. 7 describes the detailed operation process of the algorithm which is a kind of depth first search that pays attention to the priority between the rules ⊗L, &L, ⊕L, ⊸L, ⊸R, ⊕R and ⊗R. It settles directly and entirely our requirements as well as overcomes all the inconveniences of the existing solutions, in which the most important thing is that the new algorithm reduces considerably the combinatorial explosion while proving a Linear Logic sequent. Its output (that contains necessary information on the considered scenario) is described in the next section.

5.4 Output

In order to facilitate the execution of the algorithm for users, we have developed a tool which helps them create a Linear Logic sequent (that models a scenario of a story), build and examine automatically its proof graph as well as show some important statistical information on the corresponding scenario (number of discourses, number of successful/unsuccessful discourses and number of successful discourses for each outcome/goal).

For example, let us see the sequent A, (A ⊸ B) & (A ⊸ C) ⊢ C. Its proof graph (drawn by the tool) is given in Fig. 8, where:

- Si: List of sequents to prove at each iteration step
    - S1 (original sequent): A, (A ⊸ B) & (A ⊸ C) ⊢ C
    - S2: A, A ⊸ B ⊢ C
    - S3 (2 sequents): A ⊢ A and B ⊢ C
    - S4: A, A ⊸ C ⊢ C

- S5 (2 sequents): A ⊢ A and C ⊢ C

- The rule applied to the sequent which is examined at each iteration step is as follows:
  - L-Fi (R-Fi): The rule is applied to **F**ormula ***i*** in the **L**eft (**R**ight) part of the sequent
  - T, W, P, I: They are the rules corresponding with the connectives ⊗ (**T**imes), & (**W**ith), ⊕ (**P**lus), ⊸ (**I**mply) respectively.

For instance, the rule L-F1-W means that we apply the rule &L to the formula 1 in the left part of the considered sequent.

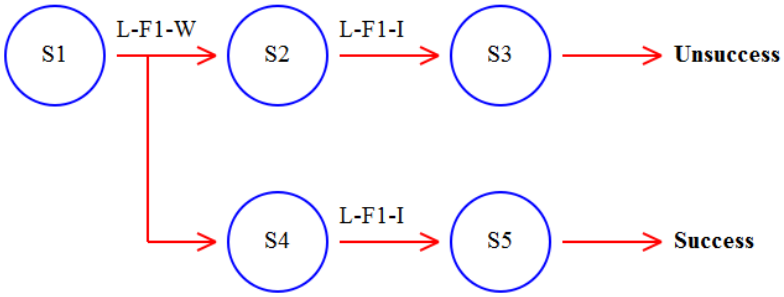

Fig. 8. Proof graph of the sequent A, (A ⊸ B) & (A ⊸ C) ⊢ C.

Thus, we conclude that the proof graph of the sequent includes two branches/discourses, in which the first one is unsuccessful but the second one is successful.

6 Example on Validation of Scenario of a Story

Let us consider an example that illustrates how to use the proof graph of a sequent to validate the scenario corresponding to this sequent, and at the same time not to restrict the player's freedom. It is an extract of an educational game which warns of domestic electrical accidents whose objective consists in causing an electric shock for the player [6]. At first, the game designer anticipates that the player, from her/his initial position, will go to the kitchen, where the IS controller will start the strategy of causing the electric shock for her/him, via appliances there such as a fridge, a microwave oven, an electric cooker,… However, what will happen if the player does not go into the kitchen, but has other choices, for instance, staying at the initial position to work or going to the bathroom? We model this game by means of Linear Logic as follows:

- *States of the game:* Gi - Being at the initial situation (this state is available); Gk - Starting the strategy of causing the electric shock for the player in the kitchen; Gr - Reaching the goal (the player has got the electric shock).

- *States of the player:* Pi - Being at the initial situation (this state is available); Pw - Working at the initial position; Pk - Being in the kitchen; Pb - Being in the bathroom; Pe - Getting the electric shock.
- *Inputs of the player (her/his action choices):* Iw - Deciding to work at the initial position; Ik - Deciding to go to the kitchen; Ib - Deciding to go to the bathroom.
- *Events/actions of the game:* The player decides to work at the initial position (EA01: Pi ⊗ Iw ⊸ Pw) by choosing Iw; the player decides to go from the initial position to the kitchen (EA02: Pi ⊗ Ik ⊸ Pk) by choosing Ik; the player decides to go from the initial position to the bathroom (EA03: Pi ⊗ Ib ⊸ Pb) by choosing Ib; the IS controller starts the strategy of causing the electric shock for the player in the kitchen (EA04: Pk ⊗ Gi ⊸ Pk ⊗ Gk); the player gets the electric shock (EA05: Pk ⊗ Gk ⊸ Pe ⊗ Gr).
- *Outcome of the game:* Pe ⊗ Gr - The player gets the electric shock.
- Finally, we have the following sequent, in which the events/actions are replaced by their label: Gi, Pi, Iw ⊗ EA01 & Ik ⊗ EA02 ⊗ EA04 ⊗ EA05 & Ib ⊗ EA03 ⊢ Pe ⊗ Gr. It gives all the possible discourses (scenario) of the game. Its proof graph is represented in Fig. 9.

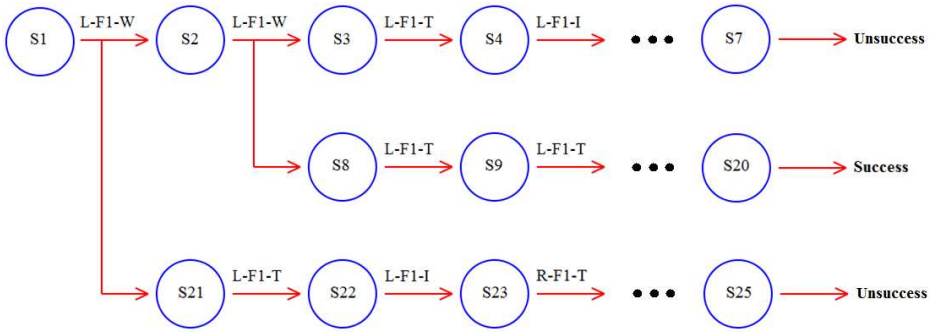

Fig. 9. Proof graph of the desirability sequent in an invalid scenario.

We can see that there are two branches which lead to unsatisfactory endings of the goal of the game (if the player decides to work at the initial position or to go from the initial position to the bathroom). Therefore we have two possibilities:

- either remove the actions of the player causing the unsatisfactory endings (EA01 - Working at the initial position and EA03 - Going to the bathroom), but that may restrict the player's freedom, so we do not choose this possibility;
- or enrich the contents of the plot:
  - if the player decides to work at the initial position, then the IS controller will ask him to go to the kitchen (for example, a non-player character asks him to take an apple in the fridge);
  - if the player decides to go to the bathroom, then the IS controller will start the strategy of causing the electric shock for him there (by tools such as a hair-dryer, a light bulb,…).

Thus we remodel the game as follows:

- *States of the game:* Gi - Being at the initial situation (this state is available); Ga - Asking the player (who is working at the initial position) to go to the kitchen; Gk - Starting the strategy of causing the electric shock for the player in the kitchen; Gb - Starting the strategy of causing the electric shock for the player in the bathroom; Gr - Reaching the goal (the player has got the electric shock).
- *States of the player:* Pi - Being at the initial situation (this state is available); Pw - Working at the initial position; Pk - Being in the kitchen; Pb - Being in the bathroom; Pe - Getting the electric shock.
- *Inputs of the player (her/his action choices):* Iw - Deciding to work at the initial position; Ik - Deciding to go to the kitchen; Ib - Deciding to go to the bathroom.
- *Events/actions of the game:* The player decides to work at the initial position (EA01: Pi ⊗ Iw ⊸ Pw) by choosing Iw; the player decides to go from the initial position to the kitchen (EA02: Pi ⊗ Ik ⊸ Pk) by choosing Ik; the player decides to go from the initial position to the bathroom (EA03: Pi ⊗ Ib ⊸ Pb) by choosing Ib; the IS controller asks the player (who is working at the initial position) to go to the kitchen (EA04: Pw ⊗ Gi ⊸ Pw ⊗ Ga); the player (who is working at the initial position) goes to the kitchen according to the request of the IS controller (EA05: Pw ⊗ Ga ⊸ Pk ⊗ Ga); after the player goes from the initial position to the kitchen by deciding to choose Ik, the IS controller starts the strategy of causing the electric shock for her/his via appliances there (EA06: Pk ⊗ Gi ⊸ Pk ⊗ Gk); after the player goes from the initial position to the kitchen according to the request of the IS controller, the IS controller starts the strategy of causing the electric shock for her/his via appliances there (EA07: Pk ⊗ Ga ⊸ Pk ⊗ Gk); the IS controller starts the strategy of causing the electric shock for the player in the bathroom (EA08: Pb ⊗ Gi ⊸ Pb ⊗ Gb); the player gets the electric shock in the kitchen (EA09: Pk ⊗ Gk ⊸ Pe ⊗ Gr); the player gets the electric shock in the bathroom (EA10: Pb ⊗ Gb ⊸ Pe ⊗ Gr).
- *Outcome of the game:* Pe ⊗ Gr - The player gets the electric shock.
- Lastly, we have the following sequent, in which the events/actions are replaced by their label: Gi, Pi, Iw ⊗ EA01 ⊗ EA04 ⊗ EA05 ⊗ EA07 ⊗ EA09 & Ik ⊗ EA02 ⊗ EA06 ⊗ EA09 & Ib ⊗ EA03 ⊗ EA08 ⊗ EA10 ⊢ Pe ⊗ Gr. It gives all the possible discourses (scenario) of the game. Its proof graph is represented in Fig. 10.

We can notice that all the branches lead to the satisfactory endings of the goal of the game (which means the player always gets the electric shock in any case), and at the same time her/his freedom is also guaranteed. Thus this example demonstrates the Linear Logic model is effective in determining whether a proof graph contains error branches. In other words, it is efficient in validating a scenario of a story.

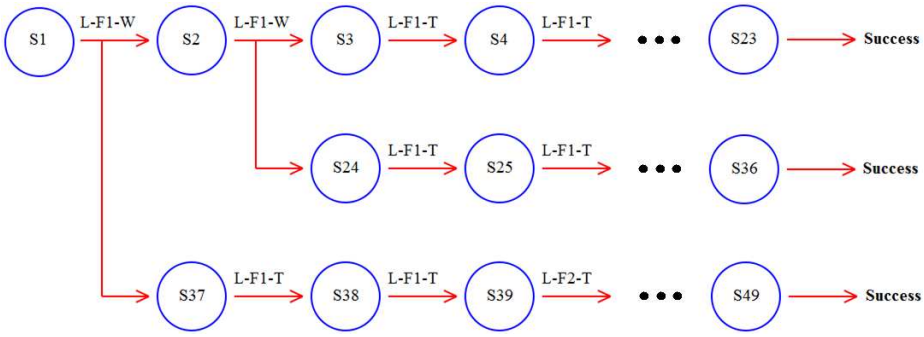

Fig. 10. Proof graph of the desirability sequent after manually modifying the scenario. Now valid, all branches lead to success.

7 Conclusion

In this paper, we have presented the methodology using Linear Logic, based on analyzing automatically the resource allocation mechanisms, that allows authors to derive a valid scenario of an IS. Indeed, this methodology, which is executed in the scenario building phase of a story, guarantees that all the decisions (that may be made during unfolding the story) lead to satisfactory endings of authors' goals. Besides, it also helps authors have quickly an overall view on the corresponding scenario, such as: number of discourses and number of discourses for each outcome/goal.

The main contributions of the paper are: (1) introduce the connectives & and ⊕ to model the choices of the player and of the IS controller; (2) assign a priority order to each event/action in the story; and the most important one, (3) propose a novel sequent proof algorithm that is based absolutely on the meaning of the Linear Logic connectives. Thus, we can enhance the modeling capacity and the authors' control on the unfolding of the story, as well as overcome the inconveniences that our previous works and other Linear Logic sequent proof solutions have met (especially the new algorithm eliminates the combinatorial explosion caused by the scheduling algorithm of events/actions (whose complexity is exponential), because it considers the priority order between the events/actions in the proof process, thereby reduces considerably the calculation time while proving a Linear Logic sequent). Besides, we have developed a graphical tool to facilitate the execution of the algorithm for users.

We also realize that, despite the promising results achieved with the Linear Logic approach for the development of formalism supporting IS modeling, a number of issues should be settled. Firstly, we have only used four connectives &, ⊗, ⊸, ⊕, although they can express almost anything necessary in the IS modeling, we will supplement several other features of Linear Logic such as the connective "!" and the constant "1" to enhance the modeling capacity. Secondly, we can actually validate a scenario but have not evaluated its quality yet. In other words, how to show an "interesting/ludic scenario" for a game? In [5], we have proposed a new class of properties (impartiality, complexity, concurrence) that allows estimating the relevance

of a scenario, and as a result, we will have to quantify theses properties for each game as well as test them by Linear Logic.

Finally, concerning future works to be done on the Linear Logic approach, in order to reduce the complexity in employing Linear Logic to model an IS for ordinary users, we must propose a solution so that Linear Logic is implicit for them. To this purpose, we will apply a model transformation method, in which users model the story by a graphical editor (built on a metamodel that does not contain the notions of Linear Logic), then the system generates automatically a Linear Logic sequent that models the corresponding scenario of the story.

Acknowledgments. This work has been funded (in part) by the European Commission under grant agreement IRIS (FP7-ICT-231824).

References

1. Abrusci, V. M., Ruet, P.: Non-commutative logic I: The multiplicative fragment. Annals of Pure and Applied Logic, 101(2000) 29--64 (1999)
2. Aylett, R.: Narrative in Virtual Environments – Towards Emergent Narrative. In: Proceedings of the AAAI Symposium on Narrative Intelligence, pp. 83–86. AAAI Press, Menlo Park (1999)
3. Aylett, R., Louchart, S., Dias, J., Paiva, A., Vala, M.: FearNot! - An Experiment in Emergent Narrative. In: Panayiotopoulos, T. et al. (eds.) IVA 2005. LNAI, vol. 3661, pp. 305--316. Springer, Heidelberg (2005)
4. Cavazza, M., Charles, F., Mead, S.J.: Character-based Interactive Storytelling. In: IEEE Intelligent Systems, special issue on AI in Interactive Entertainment, pp. 17–24 (2002)
5. Champagnat, R., Prigent, A., Estraillier, P.: Scenario building based on formal methods and adaptative execution. In: ISAGA 2005 - International Simulation and Gaming Association, Atlanta, USA (2005)
6. Dang, K. D., Champagnat, R., Augeraud, M.: Modeling of Interactive Storytelling and Validation of Scenario by Means of Linear Logic. In: Aylett, R. et al. (eds.) ICIDS 2010. LNCS, vol. 6432, pp. 153--164. Springer, Heidelberg (2010)
7. Dang, K. D., Champagnat, R., Augeraud, M.: Interactive Storytelling Control for Video Games: an Approach Based on a Linear Logic Model. Internal report L3i-2011-001, L3i laboratory, University of La Rochelle (may be offered if necessary) (2011)
8. Delmas, G., Champagnat, R., Augeraud, M.: A storytelling model for educational games: hero's interactive journey. International Journal of Technology Enhanced Learning, special issue on the workshops of EC-TEL (2008)
9. Delmas, G., Champagnat, R., Augeraud, M.: From Tabletop RPG to Interactive Storytelling: Definition of a Story Manager for Videogames. In: Iurgel, I.A., Zagalo, N., Petta, P. (eds.) ICIDS 2009. LNCS, vol. 5915, pp. 121–126. Springer, Heidelberg (2009)
10. Figueiredo, R., Dias, J., Aylett, R., Louchart, S., Paiva, A.: Shaping Emergent Narratives for a Pedagogical Application. Proceedings of the 4th International Conference on Narrative and Interactive Learning Environments (2006)
11. Girard, J.-Y.: Linear Logic. Theoretical Computer Science 50(1), 1–101 (1987)
12. Girault, F.: Using linear logic to formalize Petri nets (in French). PhD Thesis, University of Toulouse III, France (1997)
13. Göbel, S., Iurgel, I. A., Rössler, M., Hülsken, F., Eckes, C.: Design and Narrative Structure for the Virtual Human Scenarios. International Journal of Virtual Reality, 6(4):1-10 (2007)
14. Göbel, S., Malkewitz, R., Becker, F.: Story Pacing in Interactive Storytelling. In: Pan, Z. et al. (eds.) Edutainment 2006. LNCS, vol. 3942, pp. 419--428. Springer, Heidelberg (2006)

15. Habert, L., Notin, J.-M., Galmiche, D.: LINK: A Proof Environment Based on Proof Nets. In: Egly, U., Fermuller, C.G. (eds.) TABLEAUX 2002. LNAI 2381, pp. 330--334. Springer, Heidelberg (2002)
16. Hebert, L.: Tools for Text and Image Analysis: An Introduction to Applied Semiotics, Texto! (2006) (last accessed August 15, 2011), http://www.revue-texto.net/Parutions/Livres-E/Hebert_AS/Hebert_Tools.html
17. Indrzejczak, A.: Jaskowski and Gentzen approaches to natural deduction and related systems. The Lvov-Warsaw School and Contemporary Philosophy, Kluwer Academic Publishers, Printed in the Netherlands, 253--264 (1998)
18. Juul, J.: A Clash Between Game and Narrative. In: Digital Arts and Culture Conference. Bergen (1998)
19. Kanovich, M. I.: Linear logic as a logic of computations. Annals of Pure and Applied Logic, 67(1-3):183–212 (1994)
20. Kim, J., Blythe, J.: Supporting plan authoring and analysis. In: Proceedings of the 8th international conference on Intelligent user interfaces, pp. 109-116 (2003)
21. Küngas, P.: Using Linear Logic Planning to Make Knowledge Bases Reactive. In: Proceedings of Seventh Symposium on Programming Languages and Software Tools, pp. 135--148. Szeged, Hungary (2001)
22. Magerko, B.: Story Representation and Interactive Drama. In: 1st Artificial Intelligence and Interactive Digital Entertainment Conference, Los Angeles, California (2005)
23. Mateas, M.: Interactive Drama, Art, and Artificial Intelligence. PhD Thesis, School of Computer Science, Carnegie Mellon University (2002)
24. Pizzi, D., Cavazza, M.: From Debugging to Authoring: Adapting Productivity Tools to Narrative Content Description. In: Spierling, U., Szilas, N. (eds.) ICIDS 2008. LNCS, vol. 5334, pp. 285--296. Springer, Heidelberg (2008)
25. Porello, D., Endriss, U.: Modeling Multilateral Negotiation in Linear Logic. In: Proceedings of the 19th European Conference on Artificial Intelligence (2010)
26. Riedl, M.O.: Narrative Generation: Balancing Plot and Character. PhD Thesis, Department of Computer Science, North Carolina State University (2004)
27. Si, M., Marsella, S.C., Pynadath, D.V.: Directorial Control in a Decision-Theoretic Framework for Interactive Narrative. In: Iurgel, I.A., Zagalo, N., Petta, P. (eds.) ICIDS 2009. LNCS, vol. 5915, pp. 221–233. Springer, Heidelberg (2009)
28. Szilas, N.: IDtension: a Narrative Engine for Interactive Drama. TIDSE, LNCS 3105, pp. 183--203. Darmstadt, Germany (2003)
29. Vega, L., Natkin, S.: A petri net model for the analysis of the ordering of actions in computer games. In: Proceedings of 4th annual European GAME-ON Conference. London, United Kingdom (2003)
30. Wong, W. L., Shen, C., Nocera, L., Carriazo, E., Tang, F., Bugga, S., Narayanan, H., Wang, H., Ritterfeld, U.: Serious Video Game Effectiveness. ACE, Salzburg, Austria (2007)
31. Young, R.M., Riedl, M.O., Brandy, M., Martin, J., Saretto, C.J.: An Architecture for Integrating Plan-Based Behavior Generation with Interactive Game Environments. Journal of Game Development, 51–70 (2004)
32. A Linear Logic Prover (llprover) (last accessed August 15, 2011), http://bach.istc.kobe-u.ac.jp/llprover
33. Lolli: A Linear Logic Programming Language (last accessed August 15, 2011), http://www.lix.polytechnique.fr/~dale/lolli
34. Lygon: Logic Programming with Linear Logic (last accessed August 15, 2011), http://www.cs.rmit.edu.au/lygon
35. Object Management Group/Business Process Management Initiative (last accessed August 15, 2011), http://www.bpmn.org