Computational Narrative

Mubbasir Kapadia[*] Rutgers University
Steven Poulakos[†] Disney Research Zurich
Markus Gross[‡] Disney Research Zurich
Robert W. Sumner[§] Disney Research Zurich

Abstract

Despite the maturity in solutions for animating expressive virtual characters, innovations realizing the creative intent of story writers have yet to make the same strides. This problem is further exacerbated for interactive narrative content, such as games. The key challenge is to provide an accessible, yet expressive interface for story authoring that enables the rapid prototyping, iteration, and deployment of narrative concepts, while facilitate free-form interaction.

In this short course, we present the potential of computational intelligence to empower authors and content creators in creating their own interactive animated stories. There are 4 key contributions towards realizing this goal. First, we introduce a novel event-centric representation of narrative atoms which serve as the building blocks of any story. Second, we present a graphical platform for story architects to craft their own unique story worlds. Third, we present `CANVAS`, a computer-assisted visual authoring tool for synthesizing multi-character animations from sparsely- specified narrative events. In a process akin to storyboarding, authors lay out the key plot points in a story, and our system automatically fills in the missing details to synthesize a 3D animation that meets author constraints. Fourth, we present extensions to our logical formalisms to enable the transformation of a passive narrative into an interactive story, and how computational intelligence may be leveraged to identify and automatically resolve conflicts in the story. We analyse the authoring complexity of different story formalisms to present the benefits and tradeoffs of each. This course targets both Basic and Intermediate level attendees, with a preliminary background knowledge of Computer Animation and Artificial Intelligence recommended.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Digital Storytelling, Computational Narrative

1 Course Overview

Interactive narratives strive to offer immersive digital experiences in which users create or influence a dramatic storyline through their actions in interactive virtual worlds. The far-reaching goal is to immerse users in a virtual world where they become an integral part of an unfolding narrative and can significantly alter the story's outcome through their actions.

[*]mubbasir.kapadia@rutgers.edu
[†]steven.poulakos@disneyresearch.com
[‡]gross@disneyresearch.com
[§]bob.sumner@disneyresearch.com

Traditional linear narratives provide little user agency to influence the outcome of the story. Computer games often use linear plots interspersed with isolated interactive segments, with all users experiencing the same plot during successive sessions. Branching narratives [Gordon et al. 2004], where the narrative outcome depends on the user's decisions provide a discrete set of choices that influence the story. The authoring complexity of these approaches grows exponentially with the number of story arcs, the number of interaction possibilities, and the granularity of interaction. Story arcs are tightly coupled and new interactions require monolithic changes where the authoring complexity is kept tractable only by severely limiting user agency to discrete choices at key points in the story. Hence, traditional interactive narrative applications such as games either provide strong narrative experiences with limited user agency or provide compelling interactive experiences with simple narrative structure.

Creating an authoring platform that enables content creators to author free-form interactive narratives with multiple story arcs where the players can influence the narrative outcome, while using automation to facilitate the authoring process and not hinder it, is the main outcome of this course. There are two main issues which we will address: (1) An appropriate language for authoring interactive narratives that scales with story complexity, and freedom of interaction. (2) Integrated automation solutions to facilitate the story authoring process without sacrificing author control.

In this short course, we present the potential of computational intelligence to empower authors and content creators in creating their own interactive animated stories. There are 4 key contributions towards realizing this goal. First, we introduce a novel event-centric representation of narrative atoms which serve as the building blocks of any story. Second, we present a graphical platform for story architects to craft their own unique story worlds. Third, we present `CANVAS`, a computer-assisted visual authoring tool for synthesizing multi-character animations from sparsely- specified narrative events. In a process akin to storyboarding, authors lay out the key plot points in a story, and our system automatically fills in the missing details to synthesize a 3D animation that meets author constraints. Fourth, we present extensions to our logical formalisms to enable the transformation of a passive narrative into an interactive story, and how computational intelligence may be leveraged to identify and automatically resolve conflicts in the story. We analyse the authoring complexity of different story formalisms to present the benefits and tradeoffs of each. This course targets both Basic and Intermediate level attendees, with a preliminary background knowledge of Computer Animation and Artificial Intelligence recommended.

1.1 Topics Overview.

The course material is designed for both basic and intermediate level attendees, and is organized in four parts.

- **Part I: Knowledge Representation and Reasoning of Animated Stories.** Section 3 introduces the logical formalisms to represent the building blocks of a story world, referred to as domain knowledge. In particular, we introduce an event-centric representation for representing the atoms of a narrative in a modular, and extensible way. The relevant publications are [Shoulson et al. 2011; Shoulson et al. 2013].

- **Part II: Graphical Authoring of Story Worlds.** Section 4 describes a graphical platform to enable end users (not just story writers and architects) to author their own story worlds. The relevant publications are [Poulakos et al. 2015; Poulakos et al. 2016].
- **Part III: Computer-Assisted Narrative Animation Synthesis.** Section 5 introduces a visual storyboard metaphor for authoring animated stories, which is accessible to novice users and provides the right level of abstraction for creating story-driven animated content. We additionally introduce automation techniques to further help users by automatically detecting and resolving inconsistencies in their stories. The relevant publications are [Kapadia et al. 2016b; Kapadia et al. 2016c].
- **Part IV:Computer-Assisted Authoring of Interactive Narratives.** Section 6 introduces extensions to the logical formalisms to efficiently represent interactive animated stories. In particular, we focus on freeform user interaction where the player has the ability to interact with characters at any point in the narrative, and is not limited to a discrete set of story choices. We introduce algorithms that automatically detect potential user interactions which would conflict with the story specification and provide suggestions for automatically resolving conflicts through narrative mediation and intervention techniques. The relevant publication is [Kapadia et al. 2015b].
- **Part V: Authoring Complexity of Interactive Narratives.** Section 7 analyses the authoring complexity of different story representations, presenting the benefits and trade-offs of existing representations, with comparisons to the proposed formalisms. The relevant publication is [Kapadia et al. 2015c].

1.2 Course outline.

The 1.5 hour course will be divided into the following parts:

- Introduction (5 minutes, Mubbasir Kapadia)
- Building Blocks of Animated Stories. (15 minutes, Steven Poulakos)
- Graphical Authoring of Story Worlds (15 minutes, Steven Poulakos)
- Computer-Assisted Narrative Animation Synthesis (15 minutes, Mubbasir Kapadia)
- Computer-Assisted Authoring of Interactive Narratives (15 minutes, Mubbasir Kapadia)
- Authoring Complexity of Interactive Narratives (15 minutes, Mubbasir Kapadia)
- Q & A (5 minutes, Mubbasir Kapadia and Steven Poulakos)

1.3 Presenters

Mubbasir Kapadia is an Assistant Professor in the Computer Science Department at Rutgers University. Previously, he was an Associate Research Scientist at Disney Research Zurich. He was a postdoctoral researcher and Assistant Director at the Center for Human Modeling and Simulation at University of Pennsylvania. He received his PhD in Computer Science at University of California, Los Angeles. Mubbasir's research seeks to develop computational tools to assist end users to create and experience compelling, interactive, digital stories. To this end, we have revisited standard representations of interactive narratives and proposed new formalisms that scale independent of story complexity and user interaction. Our computational tools help mitigate the complexity of creating digital stories without sacrificing any authorial precision.

Steven Poulakos is an Associate Research Scientist at Disney Research Zurich. He received his PhD in Computer Science from ETH Zurich under the supervision of Prof. Markus Gross in 2014. His research interests include interactive digital storytelling and user perception.

1.4 Supplementary Material.

We have included as part of the supplementary material: (1) relevant publications, (2) sample presentation slides, (3) supplementary Videos, and (4) links to open-source software solutions. At the time of course presentation, a webpage with all this material will be released which will collate all this material and make it accessible to the audience.

2 Prior Work

The maturity of research in the simulation and animation of virtual characters has expanded the possibilities for authoring complex multi-character animations [Kovar et al. 2002; Lee 2010]. These methods represent tradeoffs between user-driven specification and automatic behavior generation.

The work of Kwon *et al.* [2008], and Kim *et al.* [2009; 2014] synthesizes synchronized multi-character motions and crowd animations by editing and stitching motion capture clips. "Motion patches" [Lee et al. 2006] annotate motion semantics in environments and can be concatenated [Kim et al. 2012] or precomputed [Shum et al. 2008] to synthesize complex multi-character interactions. Recent work [Won et al. 2014] provides sophisticated tools for generating and ranking complex interactions (e.g., fighting motions) between a small number of characters from a text-based specification of the scene. The focus of these approaches is to produce rich, complex, and populated scenes [Jordao et al. 2014] where the consistency of the interactions between characters toward an overarching narrative is less relevant.

In contrast, behavior-centric approaches use logical constructs [Vilhjálmsson et al. 2007; Stocker et al. 2010] and complex models [Yu and Terzopoulos 2007] to represent knowledge and action selection in agents. Parameterized Behavior Trees (PBT's) [Shoulson et al. 2011] are hierarchical, modular descriptions of coordinated activities between multiple actors. These constructs offer suitable abstractions to serve as the building blocks for defining story worlds and can be harnessed by future interfaces for story authoring.

Scripting. Scripted approaches [Loyall 1997; Mateas 2002] describe behaviors as pre-defined sequences of actions. However, small changes in scripting systems often require extensive modifications of monolithic specifications. Systems such as Improv [Perlin and Goldberg 1996], LIVE [Menou 2001], and "Massive" use rules to define an actor's response to external stimuli. These systems require domain expertise, making them inaccessible to end-users, and are not designed for authoring complicated multi-actor interactions over the course of a lengthy narrative.

Visual Authoring. Domain-specific visual languages have been successfully used in many applications [Whitley and Blackwell 1997] and storyboards are a natural metaphor for specifying narratives [Kurlander et al. 1996]. Game-design systems [Kelleher et al. 2007; Rosini 2014] facilitate the authoring of game logic by providing visual analogies to programming constructs. Skorupski and Mateas [2010] propose a storyboarding tool for novice users to author interactive comics. LongBoard [Jhala et al. 2008] provides a hybrid sketch and scripting interface for rendering an animated 3D scene. Physics Storyboards [Ha et al. 2013] focus on key events to accelerate the process of tuning interactive, procedural animations. Visual authoring tools offer accessible interfaces that don't require domain expertise, but the challenge is to provide an accessible metaphor that is intuitive, yet expressive. Hence, current visual authoring systems are limited to simple applications such as 2D comics.

Automation. Total-order planners [Fikes and Nilsson 1971; Hart et al. 1972] are promising for automated behavior generation [Funge et al. 1999; Kapadia et al. 2011; Shoulson et al. 2013; Riedl and Bulitko 2013]. These approaches require the specification of domain knowledge, and sacrifice some authoring precision, but they permit the automatic generation of a strict sequence of actions to satisfy a desired narrative outcome. Planning in the action space of all participating actors scales combinatorially, and these approaches are restricted to simplified problem domains [Jhala et al. 2008] with small numbers of agents. Partial-order planners [Sacerdoti 1975] relax the strict ordering constraints during planning to potentially handle more complex domains, and have been applied to accommodate player agency in interactive narratives [Kapadia et al. 2015a; Cavazza et al. 2002]. The work in [Porteous et al. 2011; Hoffmann et al. 2004] introduces the concept of "landmarks" to allow authors to specify additional narrative constraints for automated narrative synthesis.

Course Material. Existing authoring systems [Kapadia et al. 2011; Riedl and Bulitko 2013] offer automation at the cost of creative control, and allow users to specify only the narrative outcome, with little or no control over the intermediate plot points. Our motivations are different; to use automation to facilitate, not replace, human intervention in the story authoring process. To this end, we describe the following main learning outcomes of this proposal:

1. Knowledge Representation and Reasoning for Animated Stories, Symbolic Representations of Smart Objects, Affordances, and Events using Parameterized Behavior Trees [Shoulson et al. 2011; Shoulson et al. 2014].

2. Graphical Abstractions for Story World Specification [Poulakos et al. 2016].

3. Graphical Abstractions of Story Arcs as Visual Storyboards [Kapadia et al. 2016c].

4. Algorithms for Story Inconsistency Detection and Resolution [Kapadia et al. 2016b].

5. Interactive Parameterized Behavior Trees for Representating Interactive Narratives [Kapadia et al. 2015a].

6. Algorithms for Detecting and Resolving Conflicting User Actions [Kapadia et al. 2015a].

7. Cyclomatic Complexity Measures for Analysing Story Complexity [Kapadia et al. 2015c].

3 Knowledge Representation and Reasoning of Animated Stories

An underlying representation of the story world, referred to as domain knowledge, is a prerequisite to computer-assisted authoring. This includes annotated semantics that characterize the different ways in which objects and characters interact (affordances), and how these affordances are utilized to create interactions of narrative significance (events), the atoms of a story. CANVAS is no different from other intelligent systems [Riedl and Bulitko 2013] in this regard. However, the cost of specifying domain knowledge is

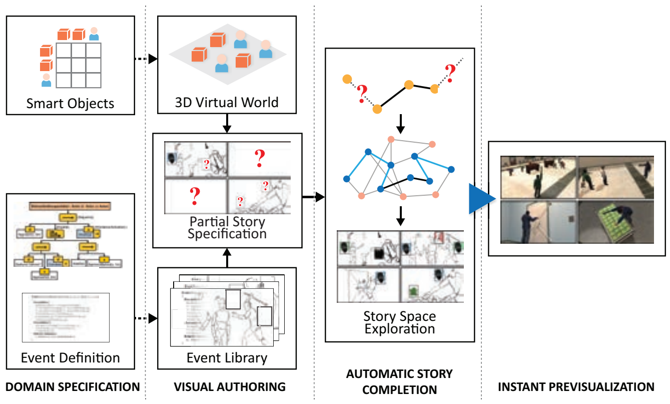


Figure 1: *CANVAS Framework Overview.*

greatly mitigated by the ability to author a variety of compelling narratives in a fashion that is accessible to story writers, artists, and casual users, enabling authors to focus only on key plot points while relying on automation to facilitate the creative process. Additionally, domain knowledge can be generalized and transferred across story worlds. We describe our representation of domain knowledge which balances ease of specification and efficiency of automation, by mitigating the combinatorial complexity of authoring individual characters in complex multi-character interactions. The one-time cost of story world building, is minimized using graphical interfaces [Poulakos et al. 2015].

Smart Objects and Actors. The virtual world $\mathcal{W}$ consists of smart objects [Kallmann and Thalmann 1999] with embedded information about how an actor can use the object. We define a smart object $w \in \mathcal{W}$ as $w = \langle \mathbf{F}, s \rangle$ with a set of advertised affordances $\mathbf{F}$ and a state s. Smart actors inherit all the properties of smart objects and can invoke affordances on other smart objects or actors in the world.

Affordances. An affordance $f(w_o, w_u) \in \mathbf{F}$ is an advertised capability offered by a smart object that manipulates the states of the owner of an affordance w_o and another smart object user (usually a smart actor) w_u. Reflexive affordances $f(w_o, w_o)$ can be invoked by the smart object owner. For example, a chair can advertise a "Sit" affordance that controls an actor to sit on the chair.

State. The state $s = \langle \theta, R \rangle$ of a smart object w comprises a set of attribute mappings θ, and a collection of pairwise relationships R with all other smart objects in $\mathcal{W}$. An attribute $\theta(i, j)$ is a bit that denotes the value of the j^{th} attribute for w_i. Attributes are used to identify immutable properties of a smart object such as its role (e.g., a chair or an actor) which never changes, or dynamic properties (e.g., `IsLocked`, `IsIncapacitated`) which may change during the story. A specific relationship $R_a(\cdot, \cdot)$ is a sparse matrix of $|\mathcal{W}| \times |\mathcal{W}|$, where $R_a(i, j)$ is a bit that denotes the current value of the a^{th} relationship between w_i and w_j. For example, an `IsFriendOf` relationship indicates that w_i is a friend of w_j. Note that relationships may not be symmetric, $\exists\,(i, j) \in |\mathcal{W}| \times |\mathcal{W}| : R_a(i, j) \neq R_a(j, i)$.

Events. Events are pre-defined context-specific interactions between any number of participating smart objects whose outcome is dictated by the current state of its participants. Events serve as the building blocks for authoring complex narratives. An event is formally defined as $e = \langle t, \mathbf{r}, \Phi, \Omega \rangle$ where t is a Parameterized Behavior Tree (PBT) [Shoulson et al. 2011] definition, and is an effective model for representing coordinated behaviors between multiple actors. t takes any number of participating smart objects as parameters where $\mathbf{r} = \{r_i\}$ define the desired roles for each participant. r_i is a logical formula specifying the desired value of the immutable attributes $\theta(\cdot, j)$ for w_j to be considered as a valid candidate for that particular role in the event. A precondition $\Phi : \mathbf{s_w} \leftarrow \{\texttt{TRUE}, \texttt{FALSE}\}$ is a logical expression on the compound state $\mathbf{s_w}$ of a particular set of smart objects $\mathbf{w} : \{w_1, w_2, \ldots w_{|\mathbf{r}|}\}$ that checks the validity of the states of each smart object. Φ is represented as a conjunction of clauses $\phi \in \Phi$ where each clause ϕ is a literal that specifies the desired attributes of smart objects, relationships, as well as rules between pairs of participants. A precondition is fulfilled by $\mathbf{w} \subseteq \mathcal{W}$ if $\Phi_e(\mathbf{w}) = \texttt{TRUE}$. The event postcondition $\Omega : \mathbf{s} \rightarrow \mathbf{s}'$ transforms the current state of all event participants $\mathbf{s}$ to $\mathbf{s}'$ by executing the effects of the event. When an event fails, $\mathbf{s}' = \mathbf{s}$. An event instance $I = \langle e, \mathbf{w} \rangle$ is an event e populated with an ordered list of smart object participants $\mathbf{w}$. $\Phi_e(\mathbf{w}) = \texttt{TRUE}$. The event postcondition $\Omega : \mathbf{s} \rightarrow \mathbf{s}'$ transforms the current state of all event participants $\mathbf{s}$ to $\mathbf{s}'$ by executing the effects of the event. When an event fails, $\mathbf{s}' = \mathbf{s}$. An event instance $I = \langle e, \mathbf{w} \rangle$ is an event e populated with an ordered list of smart object participants $\mathbf{w}$.

Ambient Activity. Individual smart objects can be grouped together to create smart groups: $w_g = \langle \mathbf{F}, s, \mathbf{w}, \mathcal{E}_g \rangle$ which contains a mutable set of smart objects $\mathbf{w} \subset \mathcal{W}$ ($w_g \notin \mathbf{w}$). To generate ambient activity that does not conflict with the overall narrative, a smart group has an ambient event scheduler that schedules events from a lexicon $\mathcal{E}_g$ for the members of $\mathbf{w}$ while satisfying a user-specified event distribution. We enforce that $\Omega_e(\mathbf{s}) = \mathbf{s}, \ \forall \ e \in \mathcal{E}_g$, implying that ambient activity will not change the narrative state of its participating actors. This allows for the seamless transition of smart actors from members of an anonymous crowd to protagonists in a story.

4 Graphical Authoring of Story Worlds

Our Story World Builder (SWB) is designed to build up components of a full story world with required semantics to enable computer-assisted narrative generation. The graphical platform is built within the Unity3D game engine. Our demonstration system conforms to the event-centric representations of Shoulson et al. [2013] and integrates with the CANVAS story authoring system of Kapadia et al. [2016a]. CANVAS provides a storyboard-based metaphor for visual story authoring of event and event participants, and it utilizes partial-order planning to enable computer-assisted generation of narratives. The underlying representation of the story world is general and can be easily used within other computer assisted narrative generation systems, such as PDDL.

Story world building begins with the creation of a new story world project and the specification of a scene. The scene specification can involve inclusion of static scene elements, environment lighting and navigation paths in the environment. Building a story world continues with three system components, which are described below: (1) Smart Object Editor: defining a set of smart objects and characters, and instantiating them in the scene. (2) State Editor: defining states, roles and relationships. (3) Affordance and Event Editors: defining capabilities for smart objects and the events which use them. Figure 2 illustrates bi-directional relationship between these components and illustrates an example story that was generated within the Haunted Castle scenario.

Smart Object Editor. Smart objects represent all of the characters and props that influence the story world. In the context of our overall formalism, smart objects have state and offer capabilities to interact with other smart objects and influence the state of the story world. Smart object characters are identified as "smart characters". We differentiate smart characters because they require additional components to, for example, support inverse kinematics to do complex physical actions, such as pressing a button and grasping objects. All humanoid smart characters can have the same base set of capabilities.

Figure 3 (a) shows the interface for creating a smart character called "Ghost", which is embodied by a ghost model. Once the smart character is created, the user may then edit properties of the smart object, accessed via the Edit tab. The Instantiate tab, visualized in Figure 3 (b), is then used to create one or more instances of the Ghost smart object within the scene. Each instance has a unique name, for example "MyGhost". Instantiated smart objects are listed at the bottom of the tab. Additional components are provided to specify other properties, including a representative icon for use in the authoring system.

State, Role and Relation Editor. The State Editor enables the creation of state attributes, roles and relationships available for use with smart objects in the world. States include high level descriptions of objects. For example, the state `IsInhabiting` is true when a ghost is inhabiting another object. Roles may be created to specify that all ghost smart objects have the role `IsGhost`. Relationships existing in the world may also be defined. For example, ghosts may have an `IsAlliedWith` or `IsInhabitedBy` relation. Each smart object provides a component for editing its states, roles and relations.

Affordance and Event Editors. Following the event-centric representation [Shoulson et al. 2013], events are defined as Parameterized Behavior Trees (PBT) [Shoulson et al. 2011], which provide a graphical, hierarchical representation for specifying complex multi-actor interactions in a modular, extensible fashion. Event creation involves specification of Affordance PBT, Event PBT, and event-based planning.

Affordance PBT. Affordances specify the capabilities of smart objects. An affordance owner offers a capability to the affordance user. Figure 4 (a) demonstrates an example "InhabitAffordance", which is owned by a smart object and used by a smart character. Figure 4 (b) represents an example behavior tree to achieve the affordance. A sequence control node is used to specify that a smart character user (a ghost) will jump into a smart object (the painting) and become invisible. The material property of the smart object will change to reflect that it is now inhabited. Affordances can be much more complex. An advantage of specifying affordances is that they may be reused in many events.

Event PBT. Events utilize the affordances and conditional logic to compose more complex forms of interaction. Figure 4 (c) and (d) present the creation of the InhabitEvent. This simple example uses a sequential control node to specify that a smart character (a ghost) will first approach a smart object (the painting) before inhabiting it. Events may have multiple smart object and character participants, however they must be consistently specified throughout the event behavior tree nodes. We use names of characters from an example scenarios to maintain consistency. Additional components are provided to specify a representative icon for use in the authoring system. Both events and affordance PBTs may be cloned and modified to support reuse.

Event-based planning. Additional semantics are specified at the event-level to enable reasoning about the logical connectivity of events. Preconditions and postconditions are defined as conjunctive normal form (CNF) expressions on the state and relations of the PBT participants. The event behavior tree in Figure 4 (d) includes a *Set Node*, which specifies a postcondition associated with the event. In our example, the `IsInhabiting` state of the ghost is set to true. In the context of our example, we may required that the smart object has role `IsInhabitable` or we may set an

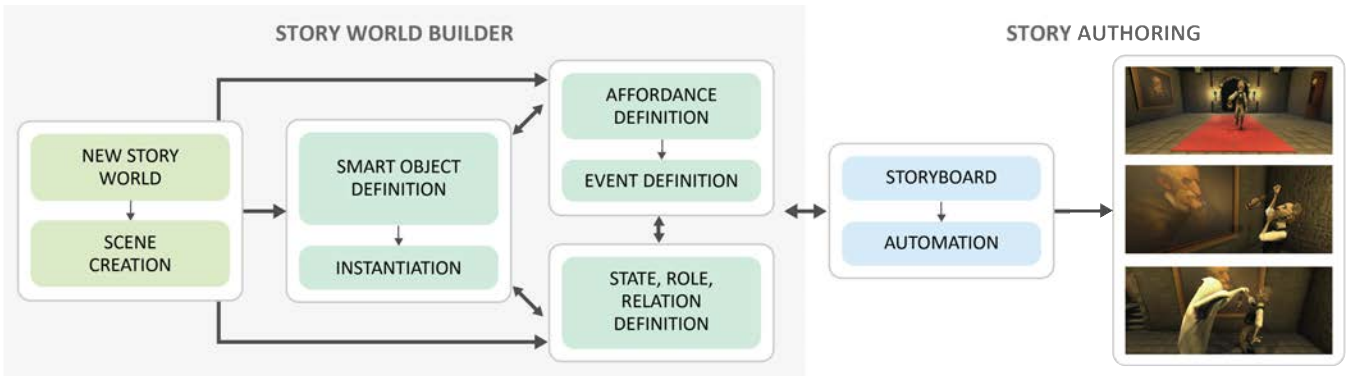
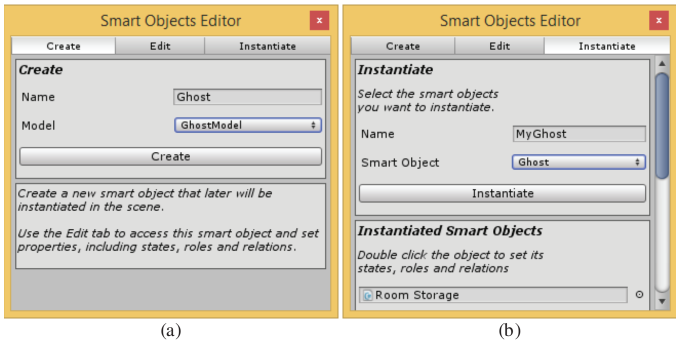


Figure 2: *Overview for building a story world and authoring stories. A scientist enters the haunted castle, investigates a painting and is then spooked by a ghost.*



Create

Name: Ghost

Model: GhostModel

Create

Create a new smart object that later will be instantiated in the scene.

Use the Edit tab to access this smart object and set properties, including states, roles and relations.

Instantiate

Select the smart objects you want to instantiate.

Name: MyGhost

Smart Object: Ghost

Instantiate

Instantiated Smart Objects

Double click the object to set its states, roles and relations

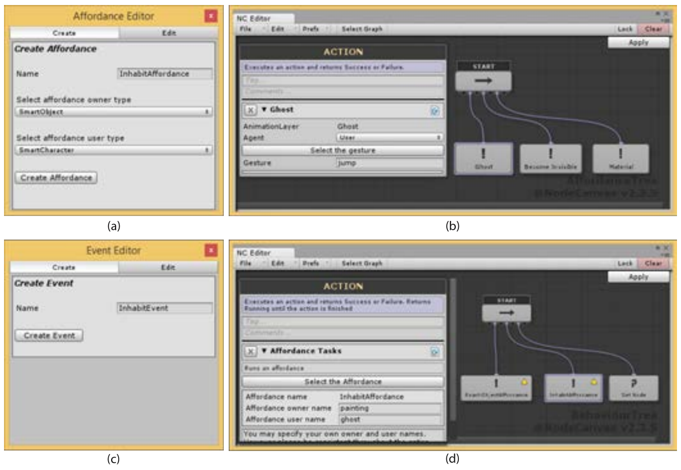
Room Storage

Figure 3: ***Smart Objects Editor.*** *(a) Create and (b) Instantiate smart objects.*

`IsInhabitedBy` relation between the smart object and character. The author may specify pre- and post-conditions within both Affordance and Event PBTs. The SWB will determine which conditions to use to support event-based planning and is compatible with the requirements of our example story authoring system [Kapadia et al. 2016a].

Coupled Story World Building and Story Authoring. Traditional systems decouple the act of building story worlds and defining narratives, which are often executed by different users (story world builders and story writers). We present a system that takes steps towards making story world building accessible to non-experts. However, in a traditional uni-directional workflow, a story world, once finalized, cannot be modified while authoring stories. This introduces certain limitations where users need to forecast all the foundational blocks (smart objects and events) that need to exist in a story world.

To mitigate this, we introduce a bi-directional workflow that couples story world building and story authoring. Our system naturally extends to facilitate seamlessly transitioning between these two acts. This workflow affords several benefits allowing traditional story authors to easily edit and modify existing story world definitions to accommodate new features, and introduce new smart objects or events, that may be necessary to realize their narrative.



Create Affordance

Name: InhabitAffordance

Select affordance owner type: SmartObject

Select affordance user type: SmartCharacter

Create Affordance

Executes an action and returns Success or Failure.

▼ Ghost

AnimationLayer: Ghost

Agent: User

Select the gesture

Gesture: jump

Create Event

Name: InhabitEvent

Create Event

Executes an action and returns Success or Failure. Returns Running until the action is finished

▼ Affordance Tasks

Runs an affordance

Select the Affordance

Affordance name: InhabitAffordance

Affordance owner name: painting

Affordance user name: ghost

You may specify your own owner and user names.

Figure 4: ***Affordance and Event Editors****. (a) The Inhabit affordance is created, which is owned by a smart object and used by a smart character to inhabit the smart object. (b) The affordance behavior tree represents the series of actions. (c) An Inhabit event is created. (d) An event behavior tree specifies a sequence of affordances and condition nodes.*

5 Computer-Assisted Narrative Animation Synthesis

Despite the maturity in solutions for animating expressive virtual characters, innovations realizing the creative intent of story writers have yet to make the same strides. The key challenge is to provide an accessible, yet expressive interface for story authoring that enables the rapid prototyping, iteration, and deployment of narrative concepts. We present `CANVAS`, a computer-assisted visual authoring tool for synthesizing multi-character animations from sparsely-specified narrative events. In a process akin to storyboarding, authors lay out the key plot points in a story, and our system automatically fills in the missing details to synthesize a 3D animation that meets author constraints. `CANVAS` can be used by artists and directors to pre-visualize storyboards in an iterative fashion, and casual users may provide arbitrarily sparse specifications and harness automation to rapidly generate diverse narratives. `CANVAS` provides an accessible interface for rapidly authoring and pre-visualizing complex narratives. Automation reduces the authoring effort further without undermining creative control or interfering with the storytelling process.

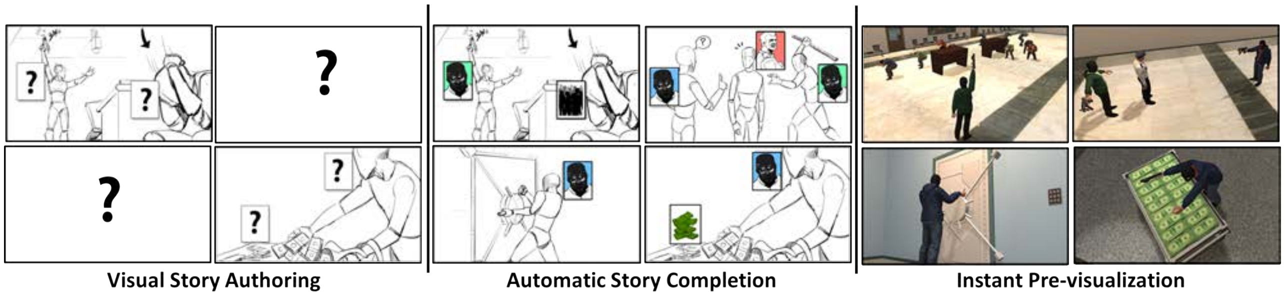


Figure 5: *`CANVAS` Overview. (a)* ***Visual Story Authoring****: Authors specify key plot points in the narrative as logical interactions between participating actors, which are represented as visual storyboards. (b)* ***Automatic Story Completion****: `CANVAS` automatically identifies and resolves incomplete stories by filling in missing participants and introducing new story elements to generate a* sound*,* consistent*, and* complete *narrative, while preserving the original intent of the author. (c)* ***Instant Pre-visualization****: A 3D animation of the narrative is instantaneously generated for rapid iteration.*

5.1 Graphical Authoring of Story Arcs

To make authoring narratives accessible to everyone, we abstract away the domain knowledge for end users, who experience the authoring of complex narratives as an ordering of key event instances between participating smart objects using a graphical storyboarding interface.

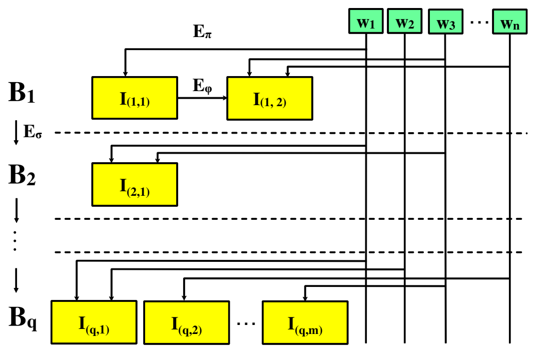


Figure 6: *Representation of a story arc as a story sequence diagram.*

Story Beats and Story Arcs. A story beat $\mathbf{B} = \{I_1, \ldots, I_n\}$ is a collection of event instances occurring simultaneously at a particular point in the story. The preconditions of a story beat $\mathbf{B} = \{I_1 \ldots I_n\}$ are a conjunction of the preconditions of all its event instances: $\Phi_{\mathbf{B}} = \Phi_{e1} \wedge \Phi_{e2}, \wedge \ldots \wedge \Phi_{e_n}$. Since all event instances within a beat execute in parallel, the same smart object is not allowed to participate in multiple instances of the same beat. A Story Arc $\mathbf{c} = (\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_m)$ is an ordered sequence of beats representing a story, where events can occur both sequentially and simultaneously throughout that story's execution.

Story Sequence Diagrams. Story Arcs are authored as Story Sequence Diagrams, which are exposed to users in the form of a graphical authoring interface. A Story Sequence Diagram is a directed acyclic graph $Q = \langle V, E \rangle$. The vertices $V = V_S \cup V_I$ consist of smart objects $V_S \subseteq \mathcal{W}$, and event instances $V_I \subseteq \mathcal{I}$. The edges $E = E_\pi \cup E_\sigma \cup E_\varphi$ indicate three relationship types. Participation edges $E_\pi \subseteq V_S \times V_I$ denote a "participates in" relationship between a smart object and an event instance (to populate a role). Sequence edges $E_\sigma \subseteq V_I \times V_I$ denote a "comes after" relationship between two event instances and is used to separate events in different story beats. Termination edges $E_\varphi \subseteq V_I \times V_I$ denote a termination dependency, where an edge between (I_i, I_j) indicates that I_j is terminated as soon as I_i finishes executing. Note that I_i and I_j must be in the same story beat. Sequence edges can be manually added by the author to define separate story beats, or are automatically inserted when a smart object participating in the instance is already involved in another event at the same time. Each horizontal row of event instances delineates a beat, and the ordered sequence of beats represents the resulting story arc. Fig. 6 illustrates a generic story arc represented as a story sequence diagram.

5.2 Graphical Authoring Interface

Starting with the default scene layout, the author introduces smart actors and objects into the scene to populate the world $\mathcal{W}$, and the event lexicon $\mathcal{E}$. Authoring a story arc entails the following simple steps. The author drags and drops event instances I, visualized as parameterized storyboard elements, into the story arc canvas. Participation edges between a smart object and an instance can be added by dragging a particular smart object portrait into the corresponding event parameter slot. Smart objects can be filtered by roles, and events by names to ease the selection process. When placing a smart object into an event parameter slot, `CANVAS` automatically checks for consistency of parameter specification and invalidates parameters which don't satisfy the role and precondition constraints of the event. Sequence edges denoting an ordering constraint are added by drawing a line between two instances, which places the second instance on the next story beat. New beats are created by simply dragging an event instance onto a new line in the story canvas. There is a separate panel for authoring ambient activity where authors select a collection of smart objects and specify a distribution of events which have no postconditions and are guaranteed to not conflict with the main story arc.

Using this simple and intuitive interface, `CANVAS` users can author compelling narratives by specifying the events that take place between the participating actors in the story. However, the user must author a complete story by specifying all the necessary events and its participants. In order to enable authors to focus only on the key events between the protagonists of the story, we introduce automation tools that complete partial story specifications. For a partially-authored story, pressing the "Complete" button automatically generates a complete story arc by filling in missing event parameters, inserting new events and story beats within the constraints of the original story definition. Pressing "Play" instantly creates a 3D pre-visualization of the scene with the animated characters acting out the story. The scene is animated and visualized using the ADAPT animation platform [Shoulson et al. 2014] in the Unity 3D engine. If the author is not satisfied with his creation, he or she can easily edit the original story definition and iteratively refine it. The instant pre-visualization and short validation and generation times are invaluable for rapid iteration. The supplementary video demonstrates the graphical authoring environment.

5.3 Automation

Problem Formulation. An author may specify a partial story $\alpha_p = (\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_m)$ using the CANVAS interface where event instances in the story beats $\mathbf{B} = \{I_1 \ldots I_n\}$ may contain open preconditions and unspecified parameters. The goal of automation is to take as input a partial story specification α_p and automatically generate a complete and consistent story α_c, subject to the following constraints: (1) **Author Constraints.** The original intent of the author, as specified in α_p must be preserved in α_c. In particular, all event instances in α_p must be present in α_c and the relative ordering of these instances must be preserved. The event participants that were specified in α_p must persist for the corresponding events in α_c. Also, the last story beat in both α_p and α_c must be identical. (2) **Story Completeness**. The event participants for all event instances in α_c must be completely specified and satisfy the role constraint. (3) **Story Consistency**. The preconditions of all event instances must be satisfied, and none of the ordering constraints between pairs of event instances in α_c may contradict each other. The equation below formalizes the resolution of a partial arc α_p into a complete arc α_c, subject to the above constraints:

$$\begin{aligned}
&\textbf{Resolve} : \alpha_p \rightarrow \alpha_c \\
&\quad \text{s.t.} \quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|}, \\
&\qquad\quad \exists\, I \in \alpha_c,\ \forall\, I \in \alpha_p, \\
&\qquad\quad (I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p\ \forall\, (I_i, I_j) \in \alpha_p, \\
&\qquad\quad \exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \texttt{TRUE}\ \forall\, r_i \in \mathbf{r}_e, \\
&\qquad\quad \Phi_e(\mathbf{w}) = \texttt{TRUE}\ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c, \\
&\qquad\quad \textbf{Consistent}(\alpha_c) = \texttt{TRUE}
\end{aligned} \tag{1}$$

Formulated as a discrete search in the space of all possible event instances, our problem domain has a very high branching factor that is combinatorial in the cardinality of the event lexicon $|\mathcal{E}|$, and the number of possible parameter bindings per event instance $|\mathcal{P}|$. The problem definition does not contain an explicit goal state formulation, where goals are implicitly specified as desired event preconditions that must be satisfied. Also, a partial story definition may contain many inconsistent or incomplete event instances with open preconditions that have conflicting solutions.

One approach is to identify all incomplete event instances in a story definition and generate solutions for each independently. However, this approach does not ensure completeness because solving one set of preconditions may invalidate existing solutions. Also, there may be cases where a coordinated resolution strategy is needed for multiple event instances across story beats, where the solution for one event instance may invalidate the possibility of *any* resolution for another instance. This problem of multiple contradicting goals is well documented in classical artificial intelligence and is a variation of the Sussman Anomaly [Sussman 1975].

Consider a simple scenario with a guard and robber. The robber has a weapon for coercion or incapacitation and the guard has the key to the room with a button to open the vault. We author an incomplete story where $\mathbf{B}_1$: the robber will unlock the door to the room (which requires the key) and $\mathbf{B}_2$: coerce the guard into pressing the vault button. By resolving the inconsistencies independently, we get a solution to $\mathbf{B}_1$ where the robber incapacitates the guard in order to take his key, thus allowing him to open the door. However, this prevents any possible solution for $\mathbf{B}_2$ since the guard is incapacitated and can no longer be coerced into pressing the button.

Partial Order Planning. An alternative approach is to search through the space of *partial plans* following the Principle of Least Commitment [Sacerdoti 1975], where event instances and ordering constraints are added to the plan only when strictly needed. This class of solutions is especially efficient for problems with high branching factor, no explicit goal formulation, multiple contradicting goal constraints, and many possible solutions that differ only in their ordering of execution. Partial-order planners avoid unnecessary choices early on in the search that may invalidate the solution and require expensive backtracking. In comparison, total-order planners [Fikes and Nilsson 1971; Hart et al. 1972] make strict commitments about the plan ordering at each step in the search, and require expensive backtracking due to wasted computations. For more details on a comparative analysis, please refer to Minton *et al.* [1992]. Section 5.3.1 introduces relevant terminology and Section 5.3.2 describes a provably sound, complete algorithm for automatically filling in partial story specifications using a plan-space approach.

5.3.1 Terminology

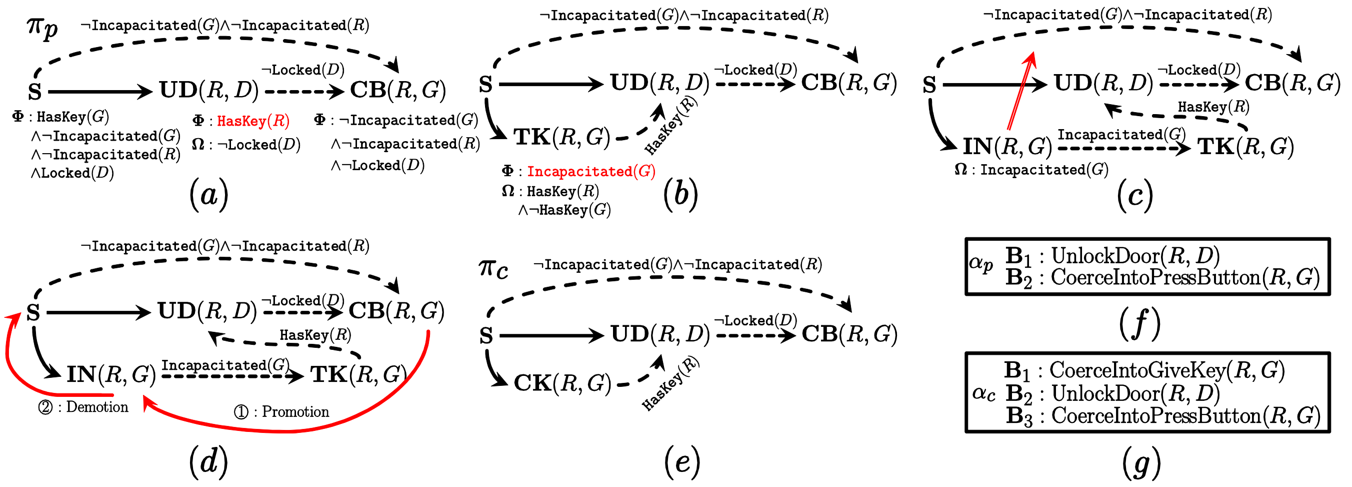
Parameter Bindings. The set of parameter bindings $\mathcal{P}(I)$ for an incomplete event instance $I = \langle e, \mathbf{w} \rangle$ where $\mathbf{w}$ is a partially-filled ordered set of n smart objects: $\{w_i \mid w_i \in \mathcal{W} \cup \{\varnothing\}\}$, comprises all possible unique permutations of smart object participants in $\mathcal{W}$ that satisfy event roles and preconditions.

Ordering Constraints. An ordering constraint $I_1 \prec I_2$ between two event instances implies that I_2 must execute some time after I_1 in the story $\mathbf{c}$, though not necessarily immediately after. Formally, $I_1 \prec I_2 \Rightarrow I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j,\ \exists\ \mathbf{B}_i, \mathbf{B}_j \in \mathbf{c}$ s.t. $i < j$. Ordering constraints are transitive in the set of event instances $\mathcal{I}$: $I_1 \prec I_2, I_2 \prec I_3 \Rightarrow I_1 \prec I_3\ \forall\ (I_1, I_2, I_3) \in \mathcal{I}$. Transitive relationships are henceforth represented as $I_1 \prec I_2 \prec I_3$. The notation $I_1 \sim I_2$ denotes when two events are in the same beat and said to execute simultaneously.

Causal Links. A causal link $\langle I_1, \phi, I_2 \rangle : I_1 \xrightarrow{\phi} I_2$ between two event instances I_1, I_2 indicates that executing the postconditions of I_1 satisfies a clause ϕ in the precondition of I_2. In other words, $\phi \in \Omega_{I_1} \wedge \phi \in \Phi_{I_2}$.

Threats. Causal links are used to detect and resolve threats. Threats are newly-introduced event instances which interfere with current events in the story by invalidating their preconditions. An event instance I_t threatens a causal link $I_1 \xrightarrow{\phi} I_2$ when the following two criteria are met: (1) $I_1 \prec I_t \prec I_2$. (2) $\neg\phi \in \Omega_{I_t}$. Threats can be resolved in two ways: (1) *Demotion*. Order I_t before the causal link by introducing an ordering constraint: $I_t \prec I_1$. (2) *Promotion*. Order I_t after the causal link by introducing an ordering constraint: $I_2 \prec I_t$. All threats must be resolved in order to generate a consistent story specification.

Partial Order Plan. A partial order plan π is a set of event instances together with a partial ordering between them. Formally, a partial order plan $\pi = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle$ where $\mathcal{I}$ is the set of event instances in π, $\mathcal{O}$ is a set of ordering constraints over $\mathcal{I}$, and $\mathcal{L}$ is a set of causal links. π is *consistent* if $\mathcal{I}$ and $\mathcal{O}$ are consistent. $\mathcal{I}$ is consistent if the parameter bindings $\mathbf{w}$ of all instances $I = \langle e, \mathbf{w} \rangle \in \mathcal{I}$ are completely filled, satisfy the event roles, don't contain duplicates, and don't violate the structure of the story arc. $\mathcal{O}$ is consistent if none of the ordering constraints contradict each other, and if there exists at least one total ordering of $\mathcal{I}$ that satisfies $\mathcal{O}$. π is *complete*



π_p
¬Incapacitated(G)∧¬Incapacitated(R)
S
UD(R, D)
¬Locked(D)
CB(R, G)
Φ : HasKey(G) ∧¬Incapacitated(G) ∧¬Incapacitated(R) ∧Locked(D)
Φ : HasKey(R)
Ω : ¬Locked(D)
Φ : ¬Incapacitated(G) ∧¬Incapacitated(R) ∧¬Locked(D)
(a)
TK(R, G)
HasKey(R)
Φ : Incapacitated(G)
Ω : HasKey(R) ∧¬HasKey(G)
(b)
IN(R, G)
Incapacitated(G)
Ω : Incapacitated(G)
(c)
② : Demotion
① : Promotion
(d)
π_c
CK(R, G)
(e)
α_p B1 : UnlockDoor(R, D) B2 : CoerceIntoPressButton(R, G)
(f)
α_c B1 : CoerceIntoGiveKey(R, G) B2 : UnlockDoor(R, D) B3 : CoerceIntoPressButton(R, G)
(g)

Figure 7: *Execution of Algorithm to complete a partial story specification α_p shown in (f). (a) Initial Partial plan π_p. Solid arrows are ordering constraints between event instances and dotted arrows are causal links. Open preconditions are highlighted in red. (b) Planning step 1. (c) Planning step 2. Newly introduced event instance threatens causal link, highlighted in red. (d) Both options to handle threat produce inconsistent orderings. (e) Complete partial plan π_c that satisfies all preconditions and produces a consistent ordering of event instances. (g) Linearization of π_c to produce a complete story arc α_c.* ***UD****: UnlockDoor,* ***CB****: CoerceIntoPressButton,* ***TK****: TakeKeyFromIncapacitated,* ***IN****: Incapacitate,* ***CK****: CoerceIntoGiveKey, R: Robber, G: Guard, D: Door.*

if every clause ϕ in the precondition of every event instance in $\mathcal{I}$ is satisfied; there exists an effect of an instance I_1 that comes before I_2 and satisfies ϕ, and no instance I_t comes between I_1 and I_2 that invalidates ϕ.

Plan Space. The plan space is an implicit directed graph whose nodes are partial plans and whose edges represent a transformation from one plan to another, obtained by adding new event instances and ordering constraints.

Linearization. The process of generating a total ordering of event instances from a partial order plan π, used to generate a complete, consistent story arc α_c is known as linearization.

5.3.2 Algorithm

Given an incomplete, inconsistent partial ordering of event instances π_p, of a partial story arc α_p, **Plan** (.) progressively completes partial instance specifications, and adds event instances to satisfy open preconditions. Causal links which are threatened by new event instances are protected by introducing additional ordering constraints. The resulting plan $\pi_c = \langle \mathcal{I}_c, \mathcal{O}_c, \mathcal{L}_c \rangle$ is linearized to produce a complete, consistent story arc α_c by generating a total ordering of the event instances $\mathcal{I}_c$ that satisfy the ordering constraints in $\mathcal{O}_c$. We describe a provably sound, complete algorithm for automatically completing partial story specifications while preserving author constraints. We provide an overview below.

Parameter Bindings. Consider an incomplete event instance $I = \langle e, \mathbf{w} \rangle \in \mathbf{c}$ where $\mathbf{w}$ is a partially-filled ordered set of n smart object elements. For every unspecified parameter $\{w_j \in \mathbf{w} | w_j = \varnothing\}$, we consider the domain of possible values $\mathbf{x}_i = \{x \mid x \in \mathcal{W}, r_i(x) = \texttt{TRUE}\}$ such that the corresponding role $r_j \in \mathbf{r}_e$ is satisfied, filtering smart objects that were already selected as participants by the author. Every permutation of possible participants for the unspecified parameters is used to create the power set of all possible parameter bindings $\mathcal{P}(I)$, while ensuring that each parameter combination has no duplicates. $\mathcal{P}$ is further filtered to remove all parameter combinations that don't satisfy the event preconditions Φ_e, and which have duplicate parameters with other event instances that are in the same beat to preserve the beat structure.

Initialization. An incomplete, inconsistent story arc α_p is first converted into a partial order plan π_p by computing the set of event instances $\mathcal{I}$, ordering constraints $\mathcal{O}$, and causal links $\mathcal{L}$ that are present in α_p. A dummy node I_0 corresponding to the starting state of all smart objects in the scene $\mathbf{s}_0$ is created and added to $\mathcal{I}$. All instances in $\mathcal{I}$ are ordered after I_0. The set of clauses ϕ_I in the preconditions of instances which are not yet satisfied are stored in $\mathcal{A}$. Additionally, the set of possible parameter bindings $\mathcal{P}(I)$ is also precomputed for all inconsistent event instances that have partially-specified event parameters. These computations are all incrementally performed while the user is authoring the story, to minimize the computational overhead of automation.

Termination Condition. The plan π_p is consistent and complete if the following conditions are met: (a) All open preconditions are resolved, (b) both $\mathcal{I}$ and $\mathcal{O}$ are consistent, and (c) no event instance $I \in \mathcal{I}$ threatens any of the causal links in $\mathcal{L}$.

Open Precondition Selection. An open precondition clause $\langle I_c, \phi_c \rangle$ is selected and removed from $\mathcal{A}$ for resolution. The order in which the precondition clauses in $\mathcal{A}$ are resolved have a major impact on the search, and can greatly reduce the the number of plan steps required to reach a solution. We use the following lexicographic ordering for $\mathcal{A} = \{\langle I, \phi \rangle\} : \langle k_1, k_2 \rangle \leq \langle k_1', k_2' \rangle \iff k_1 < k_1' \vee (k_1 = k_1' \wedge k_2 \leq k_2')$, where k_1 is a boolean indicating if ϕ can be resolved by an existing event instance in $\mathcal{I}$, and k_2 is the number of ordering constraints in $\mathcal{O}$ that involve I. This prioritizes the selection of open preconditions which can be resolved without searching, and event instances which are most constrained and have the smallest number of candidate solutions to minimize backtracking. Note that the order in which preconditions are resolved does not impact the soundness and completeness guarantees of the approach.

Parameter Binding Selection. If I_c contains partially-specified parameters, the clause ϕ_c cannot be determined. Hence, we non-deterministically select a complete and consistent parameter binding $\mathbf{w}_c$ from $\mathcal{P}(I_c)$ to generate a plausible set of smart object participants for the missing parameter slots. If no valid parameter combination is possible, failure is returned from that recursion level back to a previous choice of event selection, parameter binding, or constraint ordering.

Event Instance Selection. An event instance I_s is selected that satisfies ϕ_c, either directly from $\mathcal{I}$, or by non-deterministically picking a new instance and adding it to $\mathcal{I}$. A new causal link $I_s \xrightarrow{\phi_c} I_c$ is established to indicate that I_s satisfies ϕ_c for I_c. A new ordering constraint $I_s \prec I_c$ is added to ensure that I_c executes after I_s. If I_s is newly instantiated: (a) Additional constraints are introduced to ensure I_s is ordered after I_0, and before all instances in the last story beat $\mathcal{I}_{\text{end}}$. (b) $\mathcal{A}$ is updated to include all precondition clauses $\phi_s \in \Phi_s$ that are not satisfied. If no event instance exists that satisfies the precondition clause, we recursively roll back to the previous choice point.

Causal Link Protection. For every causal link $I_1 \xrightarrow{\phi_c} I_2 \in \mathcal{L}$ that is threatened by I_s, a new consistent ordering is established to guarantee that I_s does not interfere with the causal link, either by promoting or demoting I_s. If neither ordering is consistent, we recursively roll back to a previous choice point.

Recursive Invocation. This process is recursively repeated until the termination condition is met, or no solution is found. There are three non-deterministic choice points in our algorithm: (a) selection of parameter binding, (b) event instance selection, and (c) promotion or demotion of ordering constraints for threat resolution. Recursive back-tracking for these choice points is performed by first checking both ways to resolve a threat, choosing another valid selection of event parameters, then backtracking to another candidate event instance.

Linearization. A complete and consistent partial ordering of event instances π_c is totally ordered to generate the resulting story arc α_c. The last beat of the original story definition α_p is added to α_c, since the story ending is enforced not to change, and the corresponding event instances are removed from $\mathcal{I}$. For every remaining event instance I, the latest beat $\mathbf{B}$ in α_c is found such that I is constrained to be ordered before an instance in $\mathbf{B}$, and I is added to the previous beat. If it is the starting beat in the arc, a new beat is constructed with I and added to the beginning of α_c.

Let us revisit the example of the robber and guard, discussed earlier. Fig. 7(a) illustrates the initial partial plan π_p for the incomplete arc (f). The start node $\mathbf{S}$ is a dummy node which indicates the starting state of all smart objects. It is ordered before all other event instances. The door D is initially locked, and the guard G has the key. Solid arrows indicate ordering constraints between event instances, and dotted arrows are causal links. For example, the causal link $\langle \mathbf{UD}(R, D), \neg\texttt{Locked}(D), \mathbf{CB}(R, G)\rangle$ indicates that the door D must first be unlocked before the robber R can coerce the guard G into pressing the button. A causal link between two event instances implies an ordering constraint, which is not shown here for ease of explanation. The set of open preconditions, $\mathcal{A}$ contains only one element $\langle \mathbf{UD}(R, D), \texttt{HasKey}(R)\rangle$ which is selected for resolution. A candidate event $\mathbf{TK}(R, G)$ is introduced into π_p where the robber takes the key from the guard. This introduces another open clause $\texttt{Incapacitated}$(G) into $\mathcal{A}$, which is resolved by introducing an instance $\mathbf{IN}(R, G)$ where the robber incapacitates the guard before taking his key (Fig. 7(c)). This introduces a threat: $\mathbf{IN}(R, G)$ conflicts with the causal link, as indicated by the red arrow in Fig. 7(d). Both options for causal link protection (promotion and demotion of $\mathbf{IN}(R, G)$) produce inconsistent orderings, and no possible solution can be found. The planner backtracks to the previous choice point and chooses another event instance $\mathbf{CK}(R, G)$ where the robber first coerces the guard into giving the key, thus producing a complete plan π_c. The linearization of α_c produces a complete, consistent story arc α_c (Fig. 7(g)).

The constraint-satisfaction solver efficiently searches through the space of partial story arcs by building on top of classical partial-order principles. While staying within the conceptual framework of POP, each step is uniquely tailored to our particular problem domain. To handle the combinatorial complexity of searching through the space of all permutations of event participants for all possible sequences of events, our solver has 3 nondeterministic choice points (parameter binding selection, event instance selection, threat resolution) in comparison to 2 choice points in the classical POP solver. The lexicographic ordering used to prioritize constraint resolution is unique to our problem domain. The benefits of such an approach are crucial for providing a seamless authoring experience where the approach will always complete the story, if a possible solution exists, and will never violate any of the authors inputs.

5.4 Application

Given a story world which includes a library of smart objects and actors, and events which encode interactions between them, the end user authors digital stories by specifying the scene layout, and the narrative that ensues in it. The starting configuration of the story can be easily edited by simply dragging and dropping smart objects and actors in the scene, and adjusting the starting state of the characters. The scene configuration is completely independent of the authored domain knowledge. The starting state of the story world is automatically registered by CANVAS and used to filter the valid story participants and events that are possible.

The story authoring process involves the selection and ordering of events that take place between the characters and objects that are present in the scene, using the `CANVAS` storyboard interface. We apply `CANVAS` to author narratives in two scenarios: a bank and a marketplace. The domain knowledge (environment, actors, and events) for a bank robbery is briefly summarized in Section 5.4.1, and Section 5.4.2 describes some authored narratives. Please refer to the supplementary video for additional details.

5.4.1 Domain Knowledge for Bank Scenario

Smart Objects and Actors. The bank story world includes a variety of smart actors including robbers, guards, customers, bank managers and tellers. Additionally, there are a variety of smart objects for the actors to interact with, including drink dispensers, trash cans, bank documents, and weapons for firing warning shots and incapacitating other actors. Note that all actors have an identical set of affordances (e.g. unlock doors, use a baton to incapacitate someone), and specific roles such as guards and robbers are defined by simply modifying their visual appearance, and their starting state. For example, a guard may be equipped with a baton, and keycards, thus giving him access to locked portions of the bank. The domain knowledge for the bank scenario took 6 person-hours to author, which is a reasonable overhead considering the ease with which complex stories can be authored using different combinations of these elements.

Story Events. Table 1 outlines the definition of some representative events in our event lexicon. These include different kinds of conversations, characters cooperating to distract and incapacitate other characters, coercion to give up items and to surrender, crowds of characters fleeing etc. The event lexicon is modular and can be easily extended or modified by a domain expert, or reused across different story domains. We used a lexicon of 54 events for the narratives described in Section 5.4.2. This domain information (the event definitions and the state attributes and affordances for each smart object) was authored by two project volunteers. It took 6 per-

CoerceIntoUnlockDoor(Actor a_1, Actor a_2, Door d). Actor a_1 coerces a_2 into opening the door, d. In order for this event to be successful, a_1 must have a weapon, a_2 must have the keycard to open d, and must be able to access d.
IncapacitateStealthily(Actor a_1, Actor a_2). Actor a_1 sneaks up on a_2 and incapacitates him using his weapon. Actor a_1 must have a weapon and should be able to reach a_2 without being seen by him.
WarningShot(Actor a, Crowd: c). Actor a fires his weapon to warn the crowd c. The event precondition is that a must have a weapon.
TakeWeaponFromIncapacitated(Actor a_1, Actor a_2). Actor a_1 takes the weapon of a_2 who has been previously incapacitated. Actor a_2 must have a weapon and a_1 must be able to reach a_2.
DistractAndIncapacitate(Actor a_1, Actor a_2, Actor a_3). Actor a_1 distracts a_2 while a_3 sneaks up from behind to incapacitate a_2 using his weapon. For example, two robbers cooperate to distract and incapacitate a guard.
PressButton(Actor a, Button b). Actor a presses a button b which may have some effect elsewhere in the scene (e.g., unlocking the vault door). Actor a must have access to b in order to execute this event.
LockDoor(Actor a, Door d). Actor a locks the door d. In order to do this, he needs to have the keycard and should be able to access d. This event can be used to lock other characters in a room.
Flee(Crowd c). The members of c find the nearest exit and leave the bank. This event can be used to trigger the response of the crowd to the arrival of the robbers.

Table 1: *Descriptions of some events for the bank robbery scenario.*

son hours to author. The domain information supporting CANVAS is very easy to edit and iterate, whether by modifying existing definitions or by adding new attributes, affordances, and events.

5.4.2 Authoring Bank Robberies

Scene Specification. Given the domain knowledge, end users can easily author complex scenes using a combination of these smart objects. We author a default scene which contains 65 smart objects. These include 15 customers, 3 robbers, 2 tellers, a bank manager, and 3 guards. The vault has a locked door which can be opened by pressing the two buttons located in the manager's office and teller room. Guards are equipped with keycards to the locked doors in the scene, and may have weapons. Bank tellers serve the customers while the manager oversees bank operations. The customers can interact with the manager, tellers, and each other, while wandering the bank, purchasing and consuming beverages from the drink dispenser, recycling used cans, and filling in forms. Note that, depending on the author's intent, any character in the scene can be promoted to play more significant roles in the narrative. The stories described below used variations of this default scene.

Story Authoring. CANVAS empowers authors to create complex narratives with minimal effort. Within minutes, authors can previsualize their stories and iterate provides the complete specification of a few stories, where the highlighted parameters and events were automatically generated. We refer readers to the supplementary videos for how the stories were authored, and for the resulting animations.

The first narrative is an elaborate heist where three robbers resort to deception and violence to rob a bank. Within the main story arc, a subplot plays out in which the robbers double-cross each other until only one is remaining. A narrative of this length and complexity takes *minutes* to author, as a story arc of just 12 story beats and 20 story events. Authors may choose to focus only on the plot points of the story without specifying the participants, and CANVAS automatically selects a suitable combination of actors and smart objects (highlighted in bold) to complete the story and ensure its consistency. In another story, the author simply specifies a desired outcome, one in which the robber escapes with the money. The author is then free to iterate on the synthesized narrative. We illustrate a more complex use case for automation. The author focuses only on climactic plot points in the story arc, without specifying the events leading up to them. In our example, the author specifies that the robber open a locked door, and that the story end with a bank customer coercing the robber into surrendering. CANVAS works behind the scenes to ensure that the robber steals a key and that the customer is equipped with a gun before enacting the surrender. To satisfy the latter constraint, CANVAS generates a story in which the robber coerces the guard into dropping his weapon, and the customer later picks up the dropped weapon while the robber distracted. Note that none of the authored events are removed or modified during automation. The original intent of the author is thus always preserved. The authored stories are described in detail in [Kapadia et al. 2016b], which is included in the supplementary material.

5.4.3 Computational Performance

With 65 smart objects, 24 affordances per smart actor, and 54 events, the bank scene's effective branching factor is ~ 1000 by the standard of previous approaches [Kapadia et al. 2011]. For a single-threaded C# implementation, the computation times for automating stories were $0.1s$, $0.5s$, and $1.3s$, respectively, on an Intel(R) Core(TM) i7 3.2 Ghz CPU with 32 GB RAM. Additional performance optimizations can be expected using native code and parallel implementations. The synthesis of the 3D animation for previsualization is instantaneous and the video is a real-time recording of our system.

6 Computer-Assisted Authoring of Interactive Narratives

This part explores new authoring paradigms and computer-assisted authoring tools for free-form interactive narratives. We present a new design formalism, Interactive Behavior Trees (IBT's), which decouples the monitoring of user input, the narrative, and how the user may influence the story outcome. We introduce automation tools for IBT's, to help the author detect and automatically resolve inconsistencies in the authored narrative, or conflicting user interactions that may hinder story progression. We compare IBT's to traditional story graph representations and show that our formalism better scales with the number of story arcs, and the degree and granularity of user input. The authoring time is further reduced with the help of automation, and errors are completely avoided. Our approach enables content creators to easily author complex, branching narratives with multiple story arcs in a modular, extensible fashion while empowering players with the agency to freely interact with the characters in the story and the world they inhabit.

6.1 Authoring Interactive Narratives

An interactive narrative is traditionally represented as a branching story graph where the vertices correspond to story atoms during which the user has no outcome on the narrative, and the directed edges represent a discrete set of choices, which allow the user to influence the story outcome. To provide the user a dramatic storyline in which he can heavily influence the progression and outcome of the narrative, it is important to offer many decision points and a high branching factor. However, increasing the involvement of the user also heavily increases the combinatorial complexity of authoring such a story graph. We identify three main requirements towards authoring free-form interactive narrative experiences:

1. **Modular Story Definition.** Complex interactive narratives have many interconnected story arcs that are triggered based

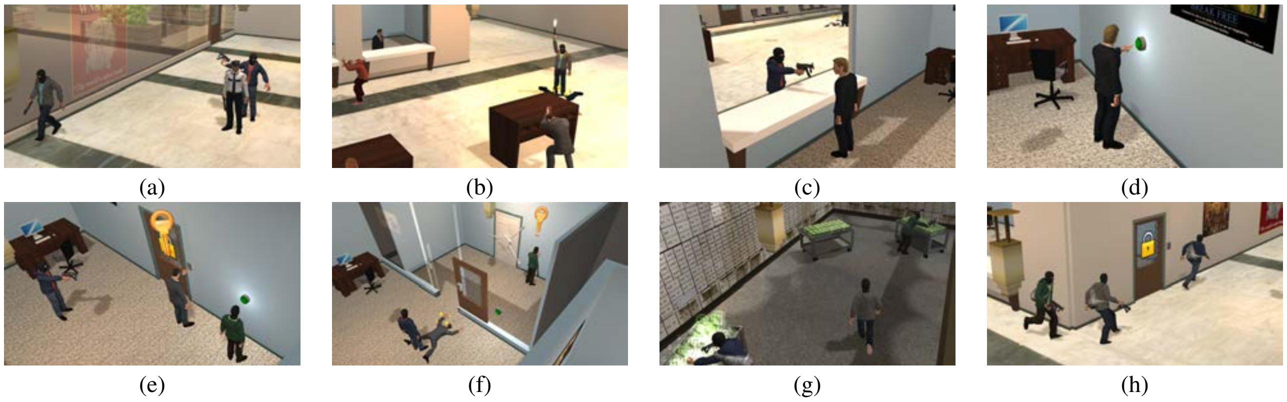


Figure 8: *A complex narrative authored using* `CANVAS`*. (a) Robbers enter the bank from the back door and begin incapacitating guards. (b) A robber fires a shot into the air to intimidate the crowd. (c) A second robber coerces the teller to (d) press a button behind his desk to unlock the vault. (e) The robbers enter the manager's office and coerce the manager to unlock the door leading to the vault, while also pressing the second button needed to unlock the vault door. (f) The robbers incapacitate the manager and open the vault door. (g) The three robbers steal the money from the vault and (h) they escape by running out the back entrance.*

on user input leading to widely divergent outcomes. The complexity of authoring narratives must scale linearly with the number of story arcs, which can be defined in a modular and independent fashion.

2. **User Interactions.** User interaction should be free-form, and not limited to discrete choices at key stages of the story, with far-reaching ramifications on the outcome of the narrative. Monitoring user input and story logic should be decoupled to facilitate the modification of user interactions without requiring far-reaching changes to the story definition.

3. **Persistent Stories.** The actions and interactions between the user and characters over the entire course of the narrative must persist and influence story progression.

6.2 Interactive Behavior Trees

Behavior Trees (BT's) provide a graphical paradigm for authoring complex narratives in a modular and extensible fashion. Story arcs can be independently authored as subtrees and then connected together using BT control nodes to author branching narratives. Recent extensions facilitate the authoring of complex multi-actor interactions in a parametrizable fashion, enabling the reuse of modular plot elements, and ensures that the complexity of the narrative scales independently of the number of characters. These properties of BT's make them ideally suited for authoring complex, branching narratives (Requirement 1). However, BT's cannot easily handle free-form interactions (Requirement 2) and don't have any means of explicitly storing the past state of characters involved in the narrative (Requirement 3). These challenges are described in the supplementary document in detail.

To meet these requirements, we introduce a new BT design formalism that facilitates free-form user interaction and state persistence. Interactive Behavior Trees (IBT's), as illustrated in Fig. 9(a) are divided into 3 independent sub-trees that are connected using a Parallel control node. An IBT $\mathbf{t}_{\mathrm{IBT}} = \langle \mathbf{t}_{\mathrm{ui}}, \mathbf{t}_{\mathrm{state}}, \mathbf{t}_{\mathrm{narr}} = \{\mathbf{t}^{\mathrm{arc}}_i | \mathbf{t}^{\mathrm{arc}}_1 \ldots \mathbf{t}^{\mathrm{arc}}_m\}, \beta \rangle$ where: (1) $\mathbf{t}_{\mathrm{narr}}$ is the narrative definition with modular story arcs $\{a_i\}$, each with their own independent subtree $\{\mathbf{t}^{\mathrm{arc}}_i\}$. (2) $\mathbf{t}_{\mathrm{ui}}$ processes the user interactions. Fig. 9(b) illustrates the story subtree. (3) $\mathbf{t}_{\mathrm{state}}$ monitors the state of the story to determine if the current story arc needs to be changed. Fig. 9(b) illustrates the story subtree. (4) The blackboard β stores the state of the story and its characters. (5) A fourth subtree $\mathbf{t}_{\mathrm{cr}}$ is added for conflict resolution, and will be described below.

Story Definition. $\mathbf{t}_{\mathrm{narr}}$ is responsible for handling the narrative progression and is further subdivided into subtrees that represent a separate story arc. Fig. 9(b) provides an example of $\mathbf{t}_{\mathrm{narr}}$ while Fig. 9(c) illustrates each arc definition $\mathbf{t}^{\mathrm{arc}}$, which is encapsulated as a separate subtree. This introduces an assertion node, which is checked at every frame whether the current arc is still active before proceeding with its execution. This minor extension to the story arc definition allows the story to instantaneously switch arcs at any moment in response to the user's interactions.

Monitoring User Input. $\mathbf{t}_{\mathrm{ui}}$ monitors the different interactions that are available to the user and can be easily changed depending on the application or device. Once an input is detected, it sets the corresponding state in the blackboard β, which is queried by $\mathbf{t}_{\mathrm{state}}$ to determine the current state of the story, and the active story arc. Since $\mathbf{t}_{\mathrm{ui}}$ is executed in parallel with the other subtrees, we are able to immediately respond and register the interactions of the user and use it to influence the narrative outcome. Fig. 9(d) illustrates an example.

Monitoring Story State. $\mathbf{t}_{\mathrm{state}}$ contains separate subtrees for each story arc, which checks if the precondition for the particular arc is satisfied. If so, β is updated to reflect the newly activated story arc, which is used to switch the active story in $\mathbf{t}_{\mathrm{narr}}$. Fig. 9(e,f) illustrates $\mathbf{t}_{\mathrm{state}}$ and a subtree used for checking the preconditions for an example story arc. It may be possible for the preconditions of multiple story arcs to be satisfied at any instance, in which case the story arcs are activated in order of priority (the order in which they appear in $\mathbf{t}_{\mathrm{narr}}$). It is also possible for multiple story arcs to be active simultaneously if they are operating on mutually exclusive characters and objects.

Message Passing and State Persistence. The overall design of the IBT results in three subtrees that execute independently in parallel with one another. The blackboard β stores internal state variables (e.g., the current active story arc) to facilitate communication between the subtrees, and maintains state persistence. $\mathbf{t}_{\mathrm{ui}}$ updates β when any input signal is detected. Tree $\mathbf{t}_{\mathrm{state}}$ monitors β to check if the preconditions of a particular story arc are satisfied, and updates the current arc. Finally, each arc subtree in $\mathbf{t}_{\mathrm{narr}}$ checks if it is the current active arc before continuing. Also, the user input and the narrative execution can update the story and character state to

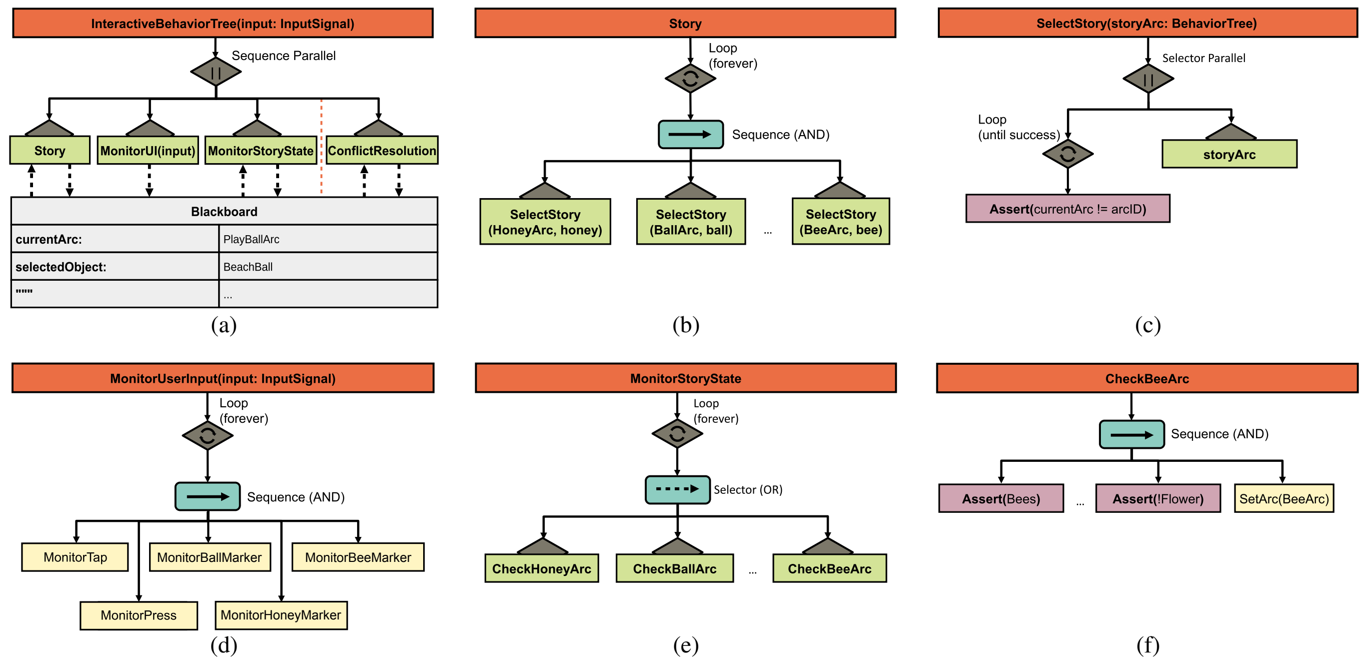


Figure 9: *(a) Design formalism of Interactive Behavior Trees (IBT's) with decoupled specification of user input, narrative definition, and the impact of user input on story state. (b) Narrative subtree with modular story arcs. (c) Each story arc definition is encapsulated in its own independent subtree, which first checks if this is the current active arc before proceeding with the narrative execution. (d) Subtree to monitor user input. (e) Subtree that changes story state based on user input, which triggers branches in story arc. (f) An example subtree from (e) which checks if all the preconditions for a particular story arc are satisfied before setting it as the current active arc.*

influence the progression of the narrative at a later stage.

Independent of the specification language (story graphs, behavior trees or any other authoring paradigm), interactive narratives require the author to consider the ramifications of all possible user interactions at all points in the story, which is prohibitive for complex stories with many different interaction modalities. To address this challenge, we introduce a suite of automation tools that exploit domain knowledge to automatically identify and resolve invalid story specifications (Section 6.4), potential user actions that may invalidate story arcs (Section 6.5,Section 6.6), and even automatically synthesize complete stories (Section 6.7).

User Interactions. We define a set of user interactions $u \in \mathbf{U}$, which define the different ways in which a user can interact with smart objects in the world $\mathbf{W}$. User interactions are treated as special kinds of affordances where the user is one of the affordance participants. This allows any underlying planning framework to accommodate user interactions during the planning process.

6.3 Problem Definition

Given the terminology defined above, we define a general problem description $\mathbf{P} = \langle \mathbf{s}_0, \mathbf{\Phi}_g, \mathbf{A} \rangle$ consisting of an initial state $\mathbf{s}_0$, a set of preconditions to satisfy the goal state $\mathbf{\Phi}_g$, and the set of affordance instances $\mathbf{A} = \{\mathbf{h}_i\}$, which may include instances of user interactions. The problem instance $\mathbf{P}$ will be defined in a variety of ways to resolve inconsistencies in the story, or potential conflicts, described in the remainder of this section.

Causal Links. We introduce the concept of a causal link to symbolize a connection between two affordance instances such that executing the postconditions of one affordance satisfies a clause in the preconditions of the other. Causal links are represented as $\mathbf{l} = \langle \mathbf{h}_1, \phi_2^i, \mathbf{h}_2 \rangle$. ϕ_2^i defines the i^{th} clause in $\mathbf{\Phi}_2$ such that $\phi_2^i(\Omega_1(\mathbf{s})) = \mathtt{TRUE}$.

Partial Order Planning. We use a partial order planner (POP) [Sacerdoti 1975] to compute a plan $\mathbf{\Pi}(,)c = \mathbf{Plan}(\mathbf{P})$ that generates an ordering of affordance instances from $\mathbf{s}_0$ which satisfies the preconditions $\mathbf{\Phi}_g$. While POP requires more computational power for processing a single node, it has been shown to outperform total-order planning (TOP) approaches [Pearl 1984] when dealing with goals that contain subgoals, allowing the planner to efficiently operate in the search space of partial plans. POP employs the Principle of Least Commitment where affordance execution is ordered only when strictly needed to ensure a consistent plan. This ensures efficiency when dealing with problems where there may exist multiple possible solutions that differ only in their ordering of affordance execution. In contrast, TOP strictly sequences actions when finding a solution. POP is also able to efficiently work for problem definitions where the goal state is partially specified – containing only the desired preconditions that must be satisfied. We provide a brief overview of the algorithm below.

At each iteration, POP selects a clause $\phi_{open} = \langle \mathbf{h}_i, \phi_i \rangle$ from the set of open preconditions $\mathbf{\Phi}_{open}$ and chooses an affordance instance $\mathbf{h}_j \in \mathbf{A}$ that satisfies ϕ_i. If $\mathbf{h}_j$ is not already present, it is inserted into the partial plan $\mathbf{\Pi}(,)p$. $\mathbf{h}_j$ must execute before $\mathbf{h}_i$, which is specified by adding a causal link $\mathbf{l} = \langle \mathbf{h}_j, \phi_i, \mathbf{h}_i \rangle$. Any instance $\mathbf{h} \in \mathbf{H}$ that contradicts ϕ_i must happen either before $\mathbf{h}_j$ or after $\mathbf{h}_i$, and is resolved by introducing additional causal links, as defined by the method **Protect**(). If $\mathbf{h}_j$ is added for the first time, its preconditions are added to $\mathbf{\Phi}_{open}$, and the process continues until all preconditions are satisfied: $\mathbf{\Phi}_{open} = \emptyset$.

Integrating Plan into IBT. The plan $\mathbf{\Pi}(,)c$ generated by POP represents an ordering $\mathbf{O}$ of affordance instances $\mathbf{H}$, which can be easily integrated into an existing behavior tree definition by choosing appropriate control nodes that constrain the order in which affordances in the plan can be executed. Fig. 10 illustrates an example of how a plan is converted into its corresponding BT definition.

6.4 Narrative Consistency

A consistent narrative does not violate the preconditions of any affordance instances that are executed during the narrative. A story author may not consider all possible preconditions when defining a story, leading to the definition of an inconsistent story. We define an inconsistent node $\mathbf{t}$ in a story arc as an affordance instance $\mathbf{h}_t$ associated with $\mathbf{t}$ such that $\mathbf{\Phi}_t(\mathbf{s_t}) = \texttt{FALSE}$ where $\mathbf{s_t}$ is the compound state of smart objects in the scene obtained by executing all nodes in the IBT leading to $\mathbf{t}$.

Belief States. Interactive narratives authored using IBT's can branch in many directions, depending on user interaction and the execution trace of nodes in the IBT. Hence, there may be many possible states that the smart objects are in at a current node $\mathbf{t}$. Therefore, we introduce the notion of a belief state $\mathbf{b_t} = \{\mathbf{s}^1_\mathbf{t}, \mathbf{s}^2_\mathbf{t} \cdots \mathbf{s}^n_\mathbf{t}\}$, which represents a set of partially specified states of all smart objects that may be reached at $\mathbf{t}$ due to different possible execution traces of the IBT. A partially specified state may contain attributes whose values cannot be determined. By considering the belief state of all possible executions of the narrative that led to $\mathbf{t}$, we can determine whether the preconditions of an affordance instance might be violated by any possible execution of the story arc. The supplementary document details the static analysis of different types of nodes in an IBT definition to compute the belief state at each node.

Inconsistency Detection and Resolution. When an inconsistent node $\mathbf{t}$ is detected in the story definition $\mathbf{t}_{\mathrm{narr}}$ of an authored IBT $\mathbf{t}_{\mathrm{IBT}}$, we compute the belief state $\mathbf{b}$ of all possible states that could arise from different execution traces of $\mathbf{t}_{\mathrm{IBT}}$ up to $\mathbf{t}$. For each of these states $\mathbf{s} \in \mathbf{b}$, we define a problem instance $\mathbf{P} = \langle \mathbf{s}, \mathbf{\Phi_t}, \mathbf{A} \rangle$ and generate a plan π to add additional nodes in the tree such that $\mathbf{\Phi_t}$ is satisfied.

6.5 Conflicts

The players actions may invalidate the successful execution of consistent narratives, and the author must consider the ramifications of all possible interactions at all possible points in the narrative definition. In order to make this problem tractable, we present automation tools that automatically detect potential user interactions that may invalidate affordance preconditions at any stage in the narrative, and provide resolution strategies to accommodate user interference, while still ensuring that the narrative is able to proceed down the intended path.

Conflicts. This allows us to formally define a conflict $\mathbf{c}$ as a pair $\langle u, \mathbf{l} \rangle$ where $\mathbf{l} = \langle \mathbf{h}_i, \phi^i_j, \mathbf{h}_j \rangle$ is an active causal link, such that if the user performs a particular interaction $u \in \mathbf{U}$ during the execution of $\mathbf{h}_i$, ϕ^i_j may be violated. Conflicts are detected at a particular node $\mathbf{t}$ if any active causal links at $\mathbf{t}$ are violated and can be resolved by generating a plan that satisfies the conditions of all needed links. Conflicts can be handled in two ways: (1) **Accommodation**. We allow the user to interfere with the narrative by successfully executing u such that $\mathbf{h}$ fails. In this case, we need to generate a conflict resolution strategy that is able to accomplish the same result, as executing $\mathbf{h}$. (2) **Interference**. The affordance instance $\mathbf{h}$ is unsuccessfully executed and u fails. No plan is needed in this case. It is up to the author to decide whether to accommodate or interfere for a particular conflict. For conflicts where no plan is possible, we are limited to interference where the user interaction is perceived to be unsuccessful.

Conflict Resolution Subtree. We add a new subtree into the IBT formalism $\mathbf{t}_{\mathrm{cr}}$ that is automatically populated and contains the conflict resolution strategies (plans) for all potential conflicts. During narrative execution, whenever a conflict occurs, control is transferred to the corresponding subtree in $\mathbf{t}_{\mathrm{cr}}$ that contains the plan for resolving that particular conflict.

Conflict Detection and Resolution. We check if any interaction violates the active links at that node. For a potential conflict $\mathbf{c} = \langle u, \mathbf{l} \rangle$, we consider the belief state $\mathbf{b}$ up to the execution of the current node $\mathbf{t}$ in the IBT. For each state $\mathbf{s} \in \mathbf{b}$, we define a problem instance $\mathbf{P} = \langle \mathbf{s}_0 = \Omega_u(\mathbf{s}), \mathbf{\Phi}_g = \mathbf{\Phi}_{\mathrm{needed}} \rangle$, where $\mathbf{\Phi}_{\mathrm{needed}}$ are the combined conditions of all needed links. A plan π is generated for $\mathbf{P}$ and inserted into the conflict resolution subtree $\mathbf{t}_{\mathrm{cr}}$ to accommodate u. If no plan is found, then we choose to interfere where u is said to fail. The appropriate conflict resolution strategy is added into $\mathbf{t}_{\mathrm{cr}}$.

Dynamic Conflict Detection and Resolution. Static analysis of the IBT is not able to detect *all possible* conflicts that may occur during execution of the narrative. In particular, we cannot detect conflicts (1) that occur while executing nodes in the conflict resolution subtree, (2) due to user actions in one story arc that violate the preconditions of nodes in another story arc. These unforeseen conflicts can be handled during the execution of the narrative to dynamically detect and resolve conflicts. This works well in practice as only a small number of conflicts remain undetected during static analysis and the algorithm for conflict resolution is very efficient and able to instantly generate plans for reasonably complex problem domains.

6.6 User Inaction

The user may choose not to execute actions that are required to progress the narrative further. For example, a narrative may require the user to throw a ball into the scene for two bears to play catch. To account for potential user inaction, our automation framework generates contingency plans where the characters in the story may adopt alternate means to accomplish the same effect of the user interaction. For each node $\mathbf{t}$ corresponding to an interaction u, we define a problem instance $\mathbf{P} = \langle \mathbf{s}, \Omega_u(\mathbf{s}), \mathbf{A} - \mathbf{U} \rangle$ where the action space $\mathbf{A} - \mathbf{U}$ only considers affordance instances with smart objects and discounts user interactions. This is used to generate a plan that achieves the same effect as $\Omega_u(\mathbf{s})$ and is integrated into the original IBT definition, as shown in Fig. 11. During narrative execution, if the user does not perform the desired interaction within a reasonable time threshold, it is said to fail and the contingency plan is executed.

6.7 Automated Narrative Synthesis

Authors may harness the power of automation to automatically synthesize narratives which can be integrated into the IBT and edited to meet author requirements. At a given node $\mathbf{t}$ in the IBT, the author simply specifies a desired set of preconditions $\mathbf{\Phi}_g$. This translates into multiple problem instances $\mathbf{P} = \langle \mathbf{s}, \mathbf{\Phi}_g, \mathbf{A} \rangle$ for each state $\mathbf{s}$ in the belief state $\mathbf{b}$, obtained as a result of executing the IBT up to $\mathbf{t}$. A plan π is generated for each problem instance $\mathbf{P}$ and inserted into the IBT, to provide a narrative that accommodates author-specified preconditions $\mathbf{\Phi}_g$.

We developed an interactive narrative authored using the tools described above and deployed it as an Augmented Reality application on mobile devices. AR applications on mobile devices with multiple sensors benefit from versatile input mechanisms and provide a strong use case for free-form interactive narratives with a host of interaction possibilities. The game application was implemented

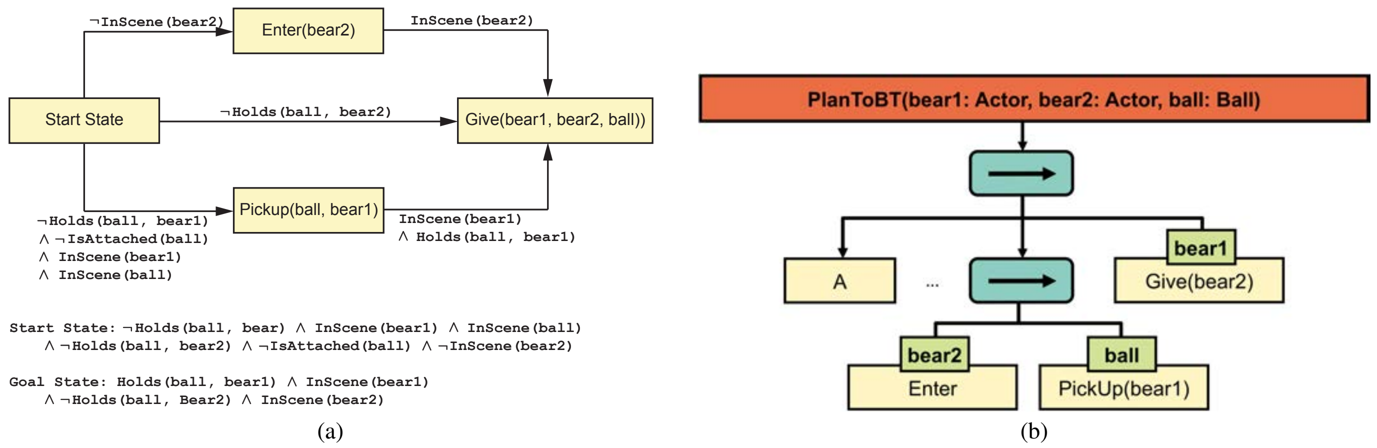

¬InScene(bear2)
Enter(bear2)
InScene(bear2)
Start State
¬Holds(ball, bear2)
Give(bear1, bear2, ball))
Pickup(ball, bear1)
¬Holds(ball, bear1) ∧ ¬IsAttached(ball) ∧ InScene(bear1) ∧ InScene(ball)
InScene(bear1) ∧ Holds(ball, bear1)
Start State: ¬Holds(ball, bear) ∧ InScene(bear1) ∧ InScene(ball) ∧ ¬Holds(ball, bear2) ∧ ¬IsAttached(ball) ∧ ¬InScene(bear2)
Goal State: Holds(ball, bear1) ∧ InScene(bear1) ∧ ¬Holds(ball, Bear2) ∧ InScene(bear2)
(a)
PlanToBT(bear1: Actor, bear2: Actor, ball: Ball)
A
...
bear1
Give(bear2)
bear2
Enter
ball
PickUp(bear1)
(b)

Figure 10: *(a) Illustrates a sample plan constructed by POP. The edges represent the causal links between the different affordances. (b) A concrete mapping of the plan to a BT.*

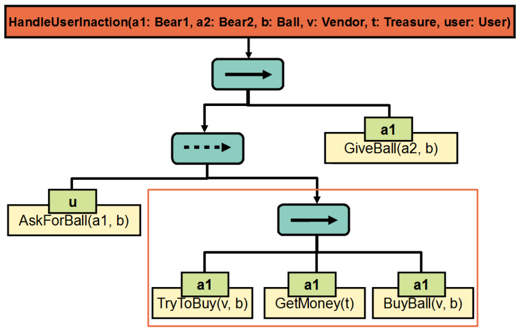

HandleUserInaction(a1: Bear1, a2: Bear2, b: Ball, v: Vendor, t: Treasure, user: User)
a1
GiveBall(a2, b)
u
AskForBall(a1, b)
a1
TryToBuy(v, b)
a1
GetMoney(t)
a1
BuyBall(v, b)

Figure 11: *Plan generation to accommodate user inaction. Our system automatically generates an alternate strategy (highlighted nodes) to accommodate potential user inactions that may hinder narrative progression.*

using the Unity3D game engine with a data-driven character animation system. The animation functionality is exposed to the author as affordances (e.g., `LookAt`(obj), `Reach`(target)) which can be invoked as leaf nodes in BT's. Smaller marker images were used as an additional interaction modality to trigger state changes and branch the story in different directions. Please refer to the supplementary document for additional implementation details.

6.8 Scenario and Story Definition

We author a narrative with two male bears B_1, B_2, and a female bear B_3. A shopkeeper B_4 is also present who is able to sell toys to the other bears. The characters have generic state attributes such as `InScene`, `IsHappy`, `IsPanicked`, `IsPlaying`, and relationships with other characters and smart objects such as `Knows`, `Holds`, and `Loves`. Other smart objects include a soccer ball, a beach ball (can be picked up by the bears and used to play catch), a honeypot (can be consumed to make the bears happy), bees (scare the bears away), and flowers (distract the bees). The smart objects are equipped with a variety of affordances including `Converse`, `Argue`, `ThrowBall`, and `EatHoney` which may include the user as a participant. For example, the ball has a `PickUp` affordance which the user can trigger to pick the ball from the scene. The representative affordances defined for the smart objects used to author the interactive narrative are outlined in the supplementary document for reference.

Baseline Story. The first bear B_1, enters the scene and looks up at the player with curiosity. The player can freely interact with B_1 using a host of interaction possibilities or introduce the second bear B_2 into the scene. B_2 asks B_1 for a beach ball so they can play catch. B_1 is unable to find a ball and turns to the player for help. The player may choose to give a soccer ball to the bears but they only want to play with the beach ball. Depending on where the player throws the beach ball, B_1 or B_2 may pick it up which influences future branches in the story. For example, B_1 chooses to involve the player in the game of catch if the player gave him the ball.

Additional Interactions. At any point, the player may use a honeypot image marker to trigger a honeypot in the world. The bears leave aside whatever they are doing, and make a beeline towards the honeypot, which represents one possible conclusion of the story. The player may also choose to trigger bees into the world, which chase the bears and disrupt the current arc (e.g., having a conversation, playing catch). Flowers may then be used to distract the bees and save the bears. Fig. 5 illustrates an example execution of the authored narrative.

State Persistence. The player's choices have ramifications later on in the story. For example, B_1 remembers if the player interacted him with at the beginning of the narrative, or helped him by giving him the ball, and includes him in the game of catch by periodically throwing him the ball. If the player adopts an antagonistic approach (e.g., by triggering bees), the bears are less friendly towards him.

Freedom of Interaction. Note that each of these interactions are possible at *any* point, and are not limited to discrete events at key stages in the narrative. For example, the player may choose to trigger the bees or the honeypot at the very beginning of the story, or while the bears are playing ball, and the story will naturally proceed as per the author's intentions. These different story arcs are authored as modular, independent units in the IBT and can be triggered at any stage without the need for complex connections and state checks in the story definition. The above narrative represents a very simple baseline to demonstrate the potential to author free-form interactive narrative experiences. IBT's empower the player with complete freedom of interaction where he may choose to play ball with the bears, give them honey, or simply wreak havoc by releasing a swarm bees at any point of time. The interactions elicit instantaneous and plausible interactions from the characters while staying true to the narrative intent.

Benefits of Automation. The automation tools described here facilitate the authoring process in a variety of ways.

Inconsistencies. The author specifies a story where B_1 gives the ball to B_2 without having the ball in his possession. The invalid preconditions are automatically detected and the appropriate nodes in the authored story arc are highlighted. Our system can automatically resolve inconsistencies by generating additional nodes in the story definition to satisfy the invalid preconditions. In this example, B_1 requests the player to throw the ball before proceeding to hand it to B_2.

Conflicts. Our system automatically detects potential user actions (or inactions) that may cause conflicts in the story. For example, the player may steal the ball from the bears during their game of catch thus invalidating the `PlayBall` arc, and our system automatically generates a strategy for the bears to request the player to return the ball. If player does not perform the expected interaction (e.g., does not give the bears the ball), an alternate strategy is generated where the bears find an alternative means to finding the ball. In this scenario, the bear purchases the ball from a vendor who in turn requires him to get money.

Automatic Story Synthesis. The author can simply specify desired preconditions and our system can generate story arcs that lead to the desired outcome. For example, the author may specify that the female bear B_3 must fall in love with B_2. A plan is generated whereby B_2 acquires the beach ball, which is desired by B_3 in order to win her heart.

7 Authoring Complexity of Interactive Narratives

In the context of software engineering, cyclomatic complexity quantifies the number of linearly independent paths through the program by measuring the number of branches in the code, and serves as a standard criteria for code evaluation without the need for dynamic code analysis. By analogy, interactive narratives that account for the different ways a user may influence the story outcome require many decision points, and are complex to author. For this reason, we use cyclomatic complexity to provide a reasonable theoretical estimate of the authoring complexity for interactive narratives. Cyclomatic complexity, $c(\mathbf{g})$ is computed by first converting the program into its equivalent control flow graph (CFG) representation $\mathbf{g}$, where the nodes correspond to atomic commands, and the directed edges connect commands that execute in sequence. c can be calculated as $c(\mathbf{g}) = p(\mathbf{g}) + 1$, where $p(\mathbf{g})$ is the number of decision points in the program. However, it assumes that there is a single termination point in the program which is not the case with BT's where every node may terminate with success or failure. Additionally, there is a third "runnning" state that is returned at each frame while the node is still executing. To generalize the measure of $c(\mathbf{g})$ for multiple termination points, we use a modified equation shown below where $s(\mathbf{g})$ denotes the number of exit points in $\mathbf{g}$.

$$c(\mathbf{g}) = p(\mathbf{g}) - s(\mathbf{g}) + 2 \tag{2}$$

7.1 Computing Cyclomatic Complexity for Behavior Trees

Fig. 12 illustrates the equivalent control flow representations for the different control nodes used for defining behavior trees, which are used to compute its cyclomatic complexity. All subsequent calculations of $c(.)$ assuming that the behavior trees are converted to their equivalent control flow graphs.

Leaf Node. A leaf node $\mathbf{t}_{\text{leaf}}$ represents an atomic command in a BT which returns either success or failure. If it is in the "running" state, it continues executing itself until it succeeds or fails. Fig. 12(a) shows the CFG for a leaf node. Depending on the number of return states in the particular leaf node implementation, we have $p(\mathbf{t}_{\text{leaf}}) = \{0, 1, 2\}$ and $s(\mathbf{t}_{\text{leaf}}) = \{1, 2\}$. If the leaf node immediately returns success or failure without using the running state, we have $c(\mathbf{t}_{\text{leaf}}) = 1$. If the leaf node can enter the running state, the complexity is $c(\mathbf{t}_{\text{leaf}}) = 2$.

Sequence Node. A sequence node $\mathbf{t}_{\text{seq}}$ returns failure if any one of its child nodes fails, else it returns success. If a child returns "running", it simply continues executing this child until it has reached failure or success. Fig. 12(b) illustrates the CFG for $\mathbf{t}_{\text{seq}}$. To calculate $c(\mathbf{t}_{\text{seq}})$ of $\mathbf{t}_{\text{seq}}$ with a set of m child nodes $\{\mathbf{t}_i | \mathbf{t}_1, \mathbf{t}_2 \ldots \mathbf{t}_m\}$, we need to consider that each child node may be its own subtree with multiple decision points.

$$p(\mathbf{t}_{\text{seq}}) = \sum_{i=1}^{m} p(\mathbf{t}_i),\ s(\mathbf{t}_{\text{seq}}) = 2,$$
$$\therefore\ c(\mathbf{t}_{\text{seq}}) = \sum_{i=1}^{m} p(\mathbf{t}_i) \tag{3}$$

Selector Node. The selector node $\mathbf{t}_{\text{sel}}$ returns success as soon as its first child node returns success, and produces a similar control flow graph as compared to $\mathbf{t}_{\text{seq}}$ (Fig. 12(c)). Hence, $c(\mathbf{t}_{\text{sel}}) = c(\mathbf{t}_{\text{seq}})$.

Loop Node. The loop node $\mathbf{t}_{\text{loop}}$ is used to repeatedly execute its child node $\mathbf{t}_{\text{c}}$ until a certain condition is met. A loop node may only have a single decorator or leaf node as its child and does not define how to traverse through multiple children. The following termination conditions may be used: (1) loop until success, (2) loop until failure, (3) loop N times, (4) loop forever. Fig. 12(d) illustrates the loop node which terminates when its child node returns success. It has only one termination node, and an additional decision point is introduced for looping.

$$p(\mathbf{t}_{\text{loop}}) = 1 + p(\mathbf{t}_{\text{c}}),\ s(\mathbf{t}_{\text{loop}}) = 1,$$
$$\therefore\ c(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}_{\text{c}}) + 2 \tag{4}$$

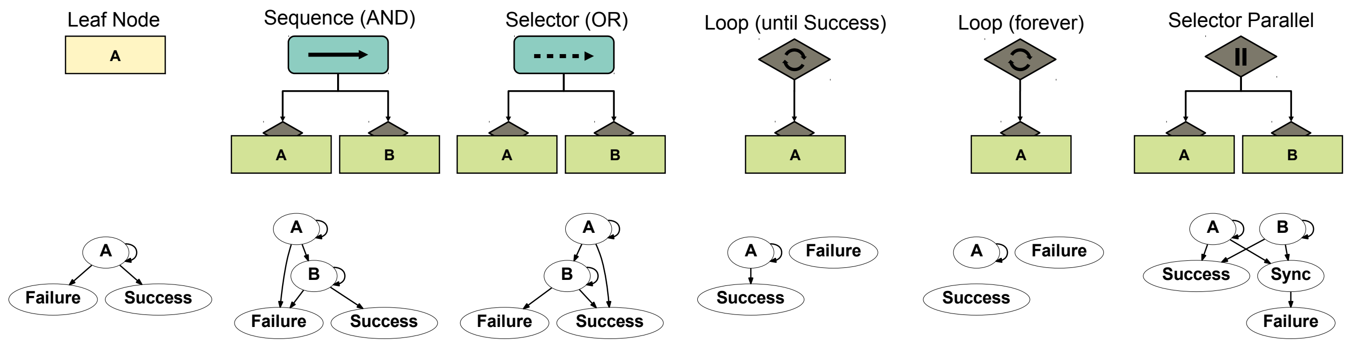
The same calculations apply for a loop node that repeats until failure. For $\mathbf{t}_{\text{loop}}$ which repeats a fixed number of times.

$$p(\mathbf{t}_{\text{loop}}) = 1 + p(\mathbf{t}),\ s(\mathbf{t}_{\text{loop}}) = 2,$$
$$\therefore\ c(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}) + 1 \tag{5}$$

Fig. 12(e) illustrates a loop node that never returns. Since the node never terminates and has neither a decision point nor an exit point, we only need to consider the child of the loop node.

$$p(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}),\ s(\mathbf{t}_{\text{loop}}) = 0,$$
$$\therefore\ c(\mathbf{t}_{\text{loop}}) = c(\mathbf{t}) \tag{6}$$

Parallel Node. The parallel node $\mathbf{t}_{\text{par}}$ executes its child nodes in parallel and has two types depending on the termination condition: (1) Selector Parallel: It executes until any child node returns success or all of them return failure. (2) Sequence Parallel: It executes until any child node returns failure or all of them succeed. Fig. 12(f) illustrates the control flow graph of a selector parallel node. An additional node "Sync" is used to symbolize the synchronization barrier between the child nodes and termination. The sequence parallel



(a) Leaf (b) Sequence (c) Selector (d) Loop until success (e) Loop Forever (f) Selector Parallel

Figure 12: *Control flow graphs for the different control nodes used in Behavior Trees.*

node exhibits similar behavior and both their complexity measures can be calculated as shown below.

$$p(\mathbf{t}_{\text{par}}) = 2 \cdot \sum_{i=1}^{m} p(\mathbf{t}_i),\; s(\mathbf{t}_{\text{par}}) = 2,$$
$$\therefore c(\mathbf{t}_{\text{par}}) = 2 \cdot \sum_{i=1}^{m} p(\mathbf{t}_i) \tag{7}$$

7.2 Authoring Complexity of Interactive Narratives

Using the complexity measure described above, we compare the authoring complexity of traditional story graphs, naive BT definitions for interactive narratives, and IBT's.

Story Graphs. For story graphs $\mathbf{g}_{\text{s}}$, a linear narrative represents a lower bound on $c(\mathbf{g}_{\text{s}}) = 1$ with no decision points. For interactive narratives however, an upper bound on the complexity represents a branching story graph where all possible user interactions are possible at each stage in the story, which is supported by IBT's. For a story graph $\mathbf{g}_{\text{s}}$ with m story arcs $\{a_i | a_1 \ldots a_m\}$, each with $|a_i|$ number of nodes, and d possible user interactions, there will be a maximum of $d+1$ outgoing edges per node in $\mathbf{g}_{\text{s}}$. Therefore, each node represents d decision points. Additionally, the number of exit points depends on how many story arcs can end the story $1 \leq s(\mathbf{g}_{\text{s}}) \leq m$. We calculate $c(\mathbf{g}_{\text{s}})$ as follows:

$$c(\mathbf{g}_{\text{s}}) = d \cdot \left(\sum_{i=1}^{m} |a_i| \right) - s(\mathbf{g}_{\text{s}}) + 2 \tag{8}$$

Interactive Behavior Trees. The benefit of an IBT $\mathbf{t}_{\text{IBT}}$ is that the subtrees $\mathbf{t}_{\text{ui}}, \mathbf{t}_{\text{state}}, \mathbf{t}_{\text{narr}}$ are all independent to each other and run in parallel using a Parallel Sequence node. Additionally, each subtree has a Loop forever node at its top, Hence, we can consider each subtree as an independent program and the total complexity $c(\mathbf{t}_{\text{IBT}})$ can be computed by adding the complexities of each independent subtree. We briefly describe the calculation of $c(\mathbf{t}_{\text{IBT}})$ below.

$\mathbf{t}_{\text{ui}}$ simply checks the different interactions in a sequence without any branching. As a result, $c(\mathbf{t}_{\text{ui}}) = 1$. $\mathbf{t}_{\text{state}}$ checks if all the preconditions for each arc are satisfied and sets the current story arc if needed. For l_i nodes per story arc a_i, $(l_i - 1)$ of those nodes are assertion nodes that represent a decision point and the last one sets the current story arc. Hence, $c(\mathbf{t}_{\text{state}}) = \sum_{i=1}^{m}(l_i - 1)$. However, $\mathbf{t}_{\text{ui}}$ and $\mathbf{t}_{\text{state}}$ can be automatically generated , thus mitigating their authoring complexity.

The story subtree $\mathbf{t}_{\text{narr}} = \{\mathbf{t}_i^{\text{arc}} | \mathbf{t}_1^{\text{arc}} \cdot \mathbf{t}_m^{\text{arc}}\}$ consists of a subtree for each story arc a_i that additionally checks if it is the current story arc before proceeding to execute the narrative (Fig. 9(d)). This ensures that story arcs can be switched seamlessly at any point in the narrative. This introduces 2 more decision points per arc, in addition to the arc complexity. The number of decision points $p(\mathbf{t}_i^{\text{arc}})$ for the i^{th} story arc is $p(\mathbf{t}_i^{\text{arc}}) = 2 \cdot (2 + p(a_i))$. The resulting complexity is $c(\mathbf{t}_{\text{narr}}) = 4 \cdot m + \sum_{i=1}^{m} 2 \cdot p(a_i)$. By automatically generating the structure of each story arc, $c\;(\mathbf{t}_{\text{narr}})$ is reduced where the author is only concerned with authoring the actual narrative. The conflict resolution subtree $\mathbf{t}_{\text{cr}}$ is automatically generated and does not need to be authored. Hence, The overall complexity $c(\mathbf{t}_{\text{IBT}})$ is calculated as follows:

$$\begin{aligned} c(\mathbf{t}_{\text{IBT}}) &= c(\mathbf{t}_{\text{ui}}) + c(\mathbf{t}_{\text{state}}) + c(\mathbf{t}_{\text{narr}}) \\ &= 1 + 4 \cdot m + \sum_{i=1}^{m} \left(2 \cdot p(a_i) + (l_i - 1)\right) \\ &\approx \sum_{i=1}^{m} 2 \cdot p(a_i) \quad \text{(with automation)} \end{aligned} \tag{9}$$

Conclusion. Eq. 8 shows that the number of user interactions and the number of nodes in the story arcs have a multiplicative effect on the authoring complexity for story graphs. This limits content creators to choose between freedom of interaction and the complexity of the narrative to prevent the authoring complexity from becoming prohibitive. In contrast, the number of decision points in the story arc have a linear effect on $c(\mathbf{t}_{\text{IBT}})$, while the number of ways a user can interact has no impact on the authoring complexity (Eq. 9). This complexity is further reduced due to the benefits of computer-assisted authoring, thus allowing content creators to create compelling interactive narrative experiences with free-form user interaction, without being burdened by limitations in the specification language and having to consider the ramifications of all possible user interactions at all points in the narrative.

8 Final Notes

This short course systematically presents an end-to-end system for the computer-assisted authoring of interactive animated stories. This course introduces logical representations for story worlds, graphical abstractions for defining story worlds and authoring stories, and algorithms that automatically detect and resolve inconsistencies in stories. A formal analysis shows that the proposed formalisms help mitigate the authoring complexity of interactive narratives, in comparison to existing story representations.

Like any intelligent system, our automation tool is only as good as its domain knowledge, which is currently specified by experts. Enabling end users to not only author their own stories, but also build their own unique story worlds, is an extremely relevant and challenging question. Additionally, a promising direction for future work is to automatically create knowledge bases from existing or crowd-sourced data [Li and Riedl 2015] which can be used for narrative synthesis.

References

CAVAZZA, M., CHARLES, F., AND MEAD, S. J. 2002. Character-based interactive storytelling. *IEEE Intelligent Systems 17*, 4 (July), 17–24.

FIKES, R. E., AND NILSSON, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *ACM SIGGRAPH*, 29–38.

GORDON, A., VAN LENT, M., VELSEN, M. V., CARPENTER, P., AND JHALA, A. 2004. Branching Storylines in Virtual Reality Environments for Leadership Development. In *AAAI*, 844–851.

HA, S., MCCANN, J., LIU, C. K., AND POPOVI, J. 2013. Physics storyboards. *Computer Graphics Forum 32*, 2pt2, 133–142.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1972. Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Bull.*, 37, 28–29.

HOFFMANN, J., PORTEOUS, J., AND SEBASTIA, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR) (JAIR) 22*, 215–278.

JHALA, A., RAWLS, C., MUNILLA, S., AND YOUNG, R. M. 2008. Longboard: A sketch based intelligent storyboarding tool for creating machinima. In *FLAIRS Conference*, AAAI Press, 386–390.

JORDAO, K., PETTRÉ, J., CHRISTIE, M., AND CANI, M.-P. 2014. Crowd Sculpting: A space-time sculpting method for populating virtual environments. *Computer Graphics Forum 33*, 2 (Apr.).

KALLMANN, M., AND THALMANN, D. 1999. Direct 3d interaction with smart objects. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM, New York, NY, USA, VRST '99, 124–130.

KAPADIA, M., SINGH, S., REINMAN, G., AND FALOUTSOS, P. 2011. A behavior-authoring framework for multiactor simulations. *Computer Graphics and Applications, IEEE 31*, 6, 45 –55.

KAPADIA, M., FALK, J., ZÜND, F., MARTI, M., SUMNER, R. W., AND GROSS, M. 2015. Computer-assisted authoring of interactive narratives. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, i3D '15, 85–92.

KAPADIA, M., FALK, J., ZUND, F., MARTI, M., SUMNER, R. W., AND GROSS, M. H. 2015. Computer-assisted authoring of interactive narratives. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, San Francisco, CA, USA, February 27 - March 01, 2015*, ACM, J. Keyser, P. V. Sander, K. Subr, and L.-Y. Wei, Eds., ACM, 85–92.

KAPADIA, M., ZUND, F., FALK, J., MARTI, M., AND SUMNER, R. W. 2015. Evaluating the authoring complexity of interactive narratives for augmented reality applications. In *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22-25, 2015*, Society for the Advancement of the Science of Digital Games, J. P. Zagal, E. MacCallum-Stewart, and J. Togelius, Eds., Society for the Advancement of the Science of Digital Games.

KAPADIA, M., FREY, S., SHOULSON, A., SUMNER, R. W., AND GROSS, M. 2016. CANVAS: Computer-assisted Narrative Animation Synthesis. In *ACM SIGGRAPH/EG SCA*, Eurographics, SCA '16.

KAPADIA, M., FREY, S., SHOULSON, A., SUMNER, R. W., AND GROSS, M. H. 2016. CANVAS: computer-assisted narrative animation synthesis. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Zurich, Switzerland, July 11-13, 2016*, Eurographics Association / ACM, B. Solenthaler, M. Teschner, L. Kavan, and C. Wojtan, Eds., Eurographics Association / ACM, 199–209.

KAPADIA, M., SHOULSON, A., STEIMER, C., OBERHOLZER, S., SUMNER, R. W., AND GROSS, M. 2016. An event-centric approach to authoring stories in crowds. In *Proceedings of the 9th International Conference on Motion in Games*, ACM, New York, NY, USA, MIG '16, 15–24.

KELLEHER, C., PAUSCH, R., AND KIESLER, S. 2007. Storytelling alice motivates middle school girls to learn computer programming. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, 1455–1464.

KIM, M., HYUN, K., KIM, J., AND LEE, J. 2009. Synchronized multi-character motion editing. In *ACM SIGGRAPH*.

KIM, M., HWANG, Y., HYUN, K., AND LEE, J. 2012. Tiling motion patches. In *ACM SIGGRAPH / Eurographics SCA*, 117–126.

KIM, J., SEOL, Y., KWON, T., AND LEE, J. 2014. Interactive manipulation of large-scale crowd animation. *ACM Transactions on Graphics (SIGGRAPH 2014, To Appear) 33*.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '02, 473–482.

KURLANDER, D., SKELLY, T., AND SALESIN, D. 1996. Comic chat. In *ACM SIGGRAPH*, 225–236.

KWON, T., LEE, K. H., LEE, J., AND TAKAHASHI, S. 2008. Group motion editing. In *ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, SIGGRAPH '08, 80:1–80:8.

LEE, K. H., CHOI, M. G., AND LEE, J. 2006. Motion patches: building blocks for virtual environments annotated with motion data. In *ACM SIGGRAPH*, 898–906.

LEE, J. 2010. Introduction to data-driven animation: Programming with motion capture. In *ACM SIGGRAPH ASIA 2010 Courses*, ACM, New York, NY, USA, SA '10, 4:1–4:50.

LI, B., AND RIEDL, M. O. 2015. Scheherazade: Crowd-powered interactive narrative generation. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.

LOYALL, A. B. 1997. *Believable agents: building interactive personalities*. PhD thesis, Pittsburgh, PA, USA.

MATEAS, M. 2002. *Interactive drama, art and artificial intelligence*. PhD thesis, Pittsburgh, PA, USA.

MENOU, E. 2001. Real-time character animation using multi-layered scripts and spacetime optimization. In *ICVS*, 135–144.

MINTON, S., DRUMMOND, M., BRESINA, J. L., AND PHILIPS, A. B. 1992. Total order vs. partial order planning: Factors influencing performance. In *KR*, Morgan Kaufmann, 83–92.

PEARL, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *ACM SIGGRAPH*, 205–216.

PORTEOUS, J., TEUTENBERG, J., PIZZI, D., AND CAVAZZA, M. 2011. Visual programming of plan dynamics using constraints and landmarks. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*.

POULAKOS, S., KAPADIA, M., SCHUPFER, A., ZUND, F., SUMNER, R., AND GROSS, M., 2015. Towards an accessible interface for story world building.

POULAKOS, S., KAPADIA, M., MAIGA, G. M., ZÜND, F., GROSS, M., AND SUMNER, R. W. 2016. Evaluating accessible graphical interfaces for building story worlds. In *Interactive Storytelling: 9th International Conference on Interactive Digital Storytelling, ICIDS 2016, Los Angeles, CA, USA, November 15–18, 2016, Proceedings 9*, Springer, 184–196.

RIEDL, M. O., AND BULITKO, V. 2013. Interactive narrative: An intelligent systems approach. *AI Magazine 34*, 1, 67–77.

ROSINI, R., 2014. Storybricks. Namaste Entertainment Inc.

SACERDOTI, E. D. 1975. The nonlinear nature of plans. In *IJCAI*, 206–214.

SHOULSON, A., GARCIA, F. M., JONES, M., MEAD, R., AND BADLER, N. I. 2011. Parameterizing behavior trees. In *Motion in Games*, 144–155.

SHOULSON, A., GILBERT, M. L., KAPADIA, M., AND BADLER, N. I. 2013. An event-centric planning approach for dynamic real-time narrative. In *Proceedings of Motion on Games*, ACM, New York, NY, USA, MIG '13, 99:121–99:130.

SHOULSON, A., KAPADIA, M., MARSHAK, N., AND BADLER, N. I. 2014. Adapt: The agent development and prototyping testbed. *IEEE Transactions on Visualization and Computer Graphics 99*, 1.

SHUM, H. P. H., KOMURA, T., SHIRAISHI, M., AND YAMAZAKI, S. 2008. Interaction patches for multi-character animation. In *ACM SIGGRAPH Asia*, 114:1–114:8.

SKORUPSKI, J., AND MATEAS, M. 2010. Novice-friendly authoring of plan-based interactive storyboards. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*.

STOCKER, C., SUN, L., HUANG, P., QIN, W., ALLBECK, J. M., AND BADLER, N. I. 2010. Smart events and primed agents. In *IVA*, vol. 6356, 15–27.

SUSSMAN, G. J. 1975. *A computer model of skill acquisition*. Artificial intelligence series. American Elsevier Pub. Co., New York.

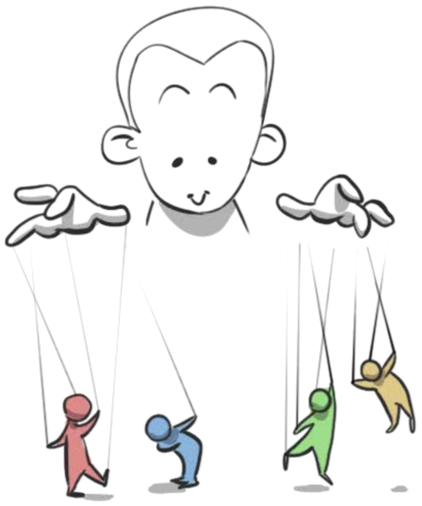
VILHJÁLMSSON, H., CANTELMO, N., CASSELL, J., E. CHAFAI, N., KIPP, M., KOPP, S., MANCINI, M., MARSELLA, S., MARSHALL, A. N., PELACHAUD, C., RUTTKAY, Z., THÓRISSON, K. R., WELBERGEN, H., AND WERF, R. J. 2007. The behavior markup language: Recent developments and challenges. In *Intelligent Virtual Agents*, 99–111.

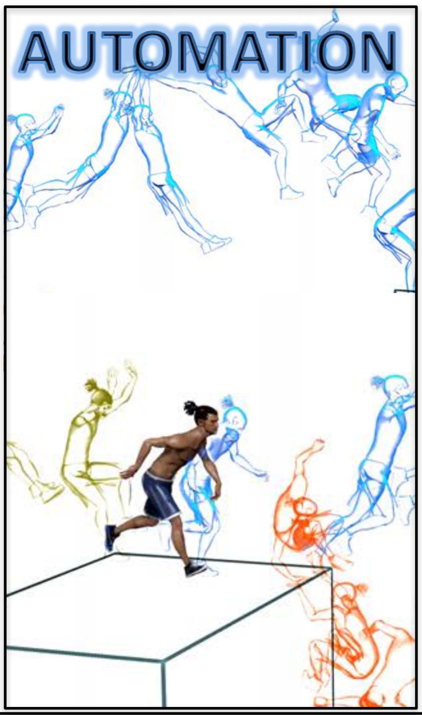
WHITLEY, K. N., AND BLACKWELL, A. F. 1997. Visual programming: The outlook from academia and industry. In *Seventh Workshop on Empirical Studies of Programmers*, ACM, ESP '97, 180–208.

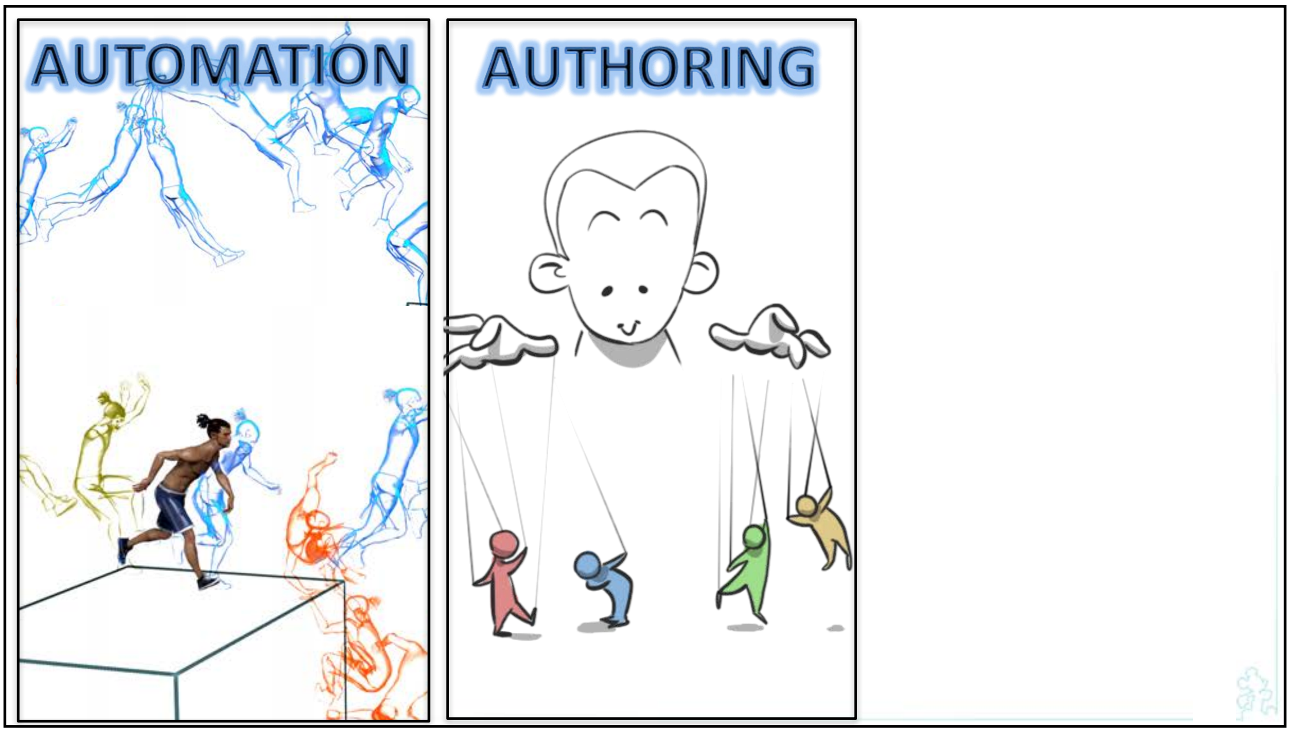
WON, J., LEE, K., HODGINS, J., O'SULLIVAN, C., AND LEE, J. 2014. Generating and ranking diverse multi-character interactions. In *ACM SIGGRAPH Asia*.

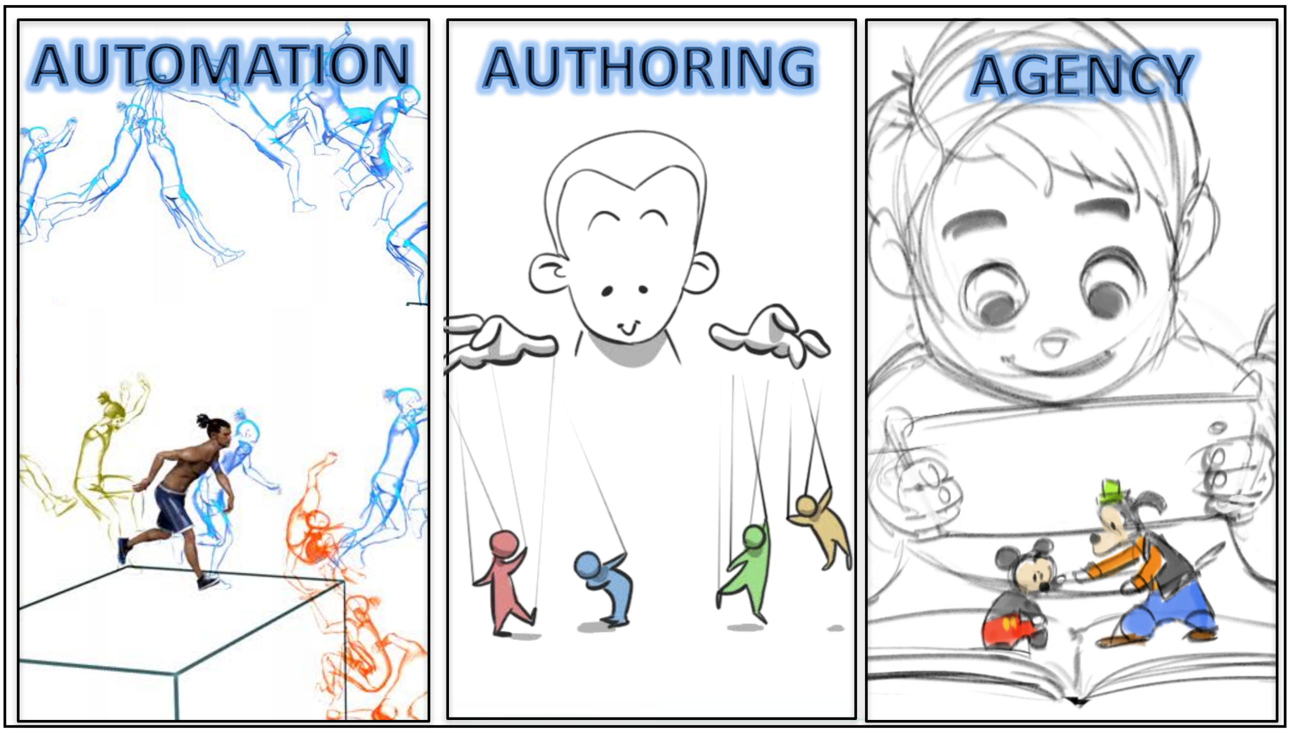
YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. In *ACM SIGGRAPH/Eurographics SCA*, 119–128.

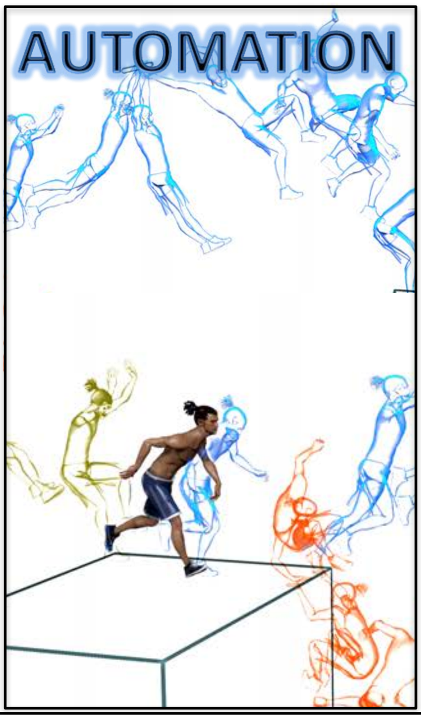
Computational Narrative






AUTOMATION
AUTHORING


AUTOMATION
AUTHORING
AGENCY



Motivation

- Want to
  - Build story worlds
  - Utilize computational intelligence
  - Automatically generate narratives
- Requires platform to
  - Specify domain knowledge of the story world
  - Provide a graphical interface to support world creation

Design principles

- Usability → Accessible GUI
- Reuse → Cost reduction
- Bi-directional workflow → Iterative Design
- Automation → Complexity reduction

Related Work

- Manual authoring
- Automated authoring
- Story World Building

Manual Authoring

- Scripted Approaches [Loyall 1997; Mateas 2002]
- Rule-based Systems [Perlin and Goldberg 1996; Menou 2001]
- Façade [Mateas and Stern 2003, 2004; Dow *et al.* 2006]
- Story Graphs [Gordon *et al.* 2004]
- Behavior Trees [Hecker *et al.* 2007; Isla 2008; Shoulson *et al.* 2011; Millington and Funge 2008]

Automated Authoring

- Domain-Independent Planners [Fikes and Nilsson 1971; Sacerdoti 1975]
- Narrative generation systems [Riedl et al. 2003, Riedl and Young 2006]
- Virtual Directors & Drama Managers [Magerko et al. 2004, Riedl et al. 2008, Shoulson et al. 2013]

Domain Knowledge

- Smart Objects
- State
- Affordances
- Events

Domain Knowledge

- **Smart Objects**
- State
- Affordances
- Events

$$w = \langle \mathbf{F}, s \rangle$$

Domain Knowledge

- Smart Objects
- **State**
- Affordances
- Events

$$s = \langle \theta, R \rangle$$

Domain Knowledge

- Smart Objects
- State
- **Affordances**
- Events

$$f(w_o, w_u) \in \mathbf{F}$$

Domain Knowledge

- Smart Objects
- State
- Affordances
- **Events**

$$e = \langle t, \mathbf{r}, \Phi, \Omega \rangle$$

Domain Knowledge

- Smart Objects
- State
- Affordances
- **Events**

$$e = \langle t \boxed{, \mathbf{r},} \Phi, \Omega \rangle$$

$$\mathbf{r} = \{r_i\}$$

Domain Knowledge

- Smart Objects
- State
- Affordances
- **Events**

$$e = \langle t, \mathbf{r}, \boxed{\Phi}, \Omega \rangle$$

$$\Phi : \mathbf{s_w} \leftarrow \{\texttt{TRUE}, \texttt{FALSE}\}$$

Domain Knowledge

- Smart Objects
- State
- Affordances
- **Events**

$$e = \langle t, \mathbf{r}, \Phi, \boxed{\Omega} \rangle$$

$$\Omega : \mathbf{s} \rightarrow \mathbf{s}'$$

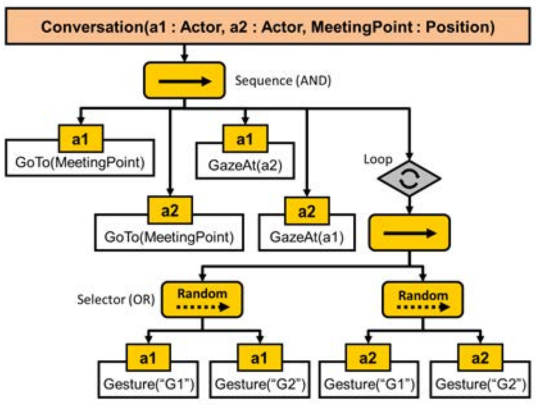
Domain Knowledge

- Smart Objects
- State
- Affordances
- **Events**

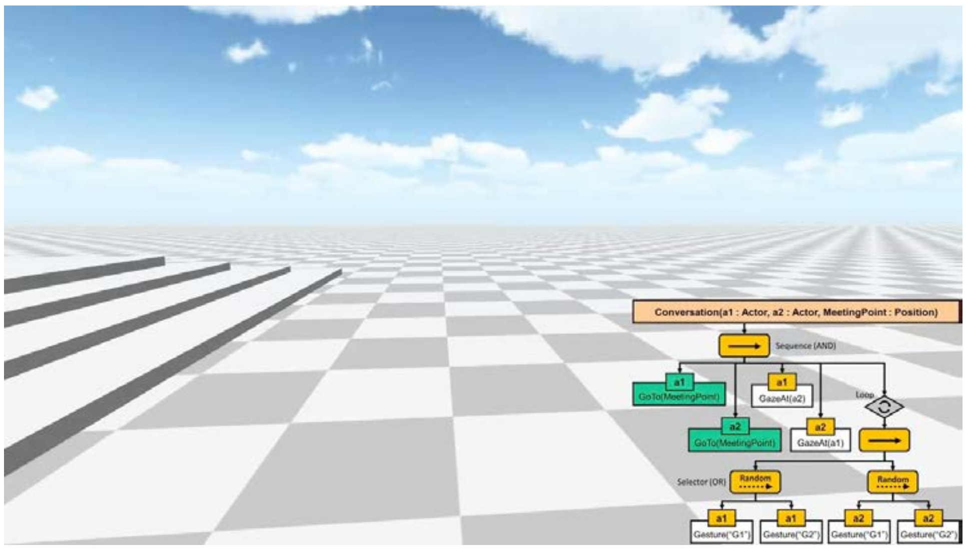
$$e = \langle \boxed{t,} \mathbf{r}, \Phi, \Omega \rangle$$

Parameterized Behavior Trees

Parameterized Behavior Trees

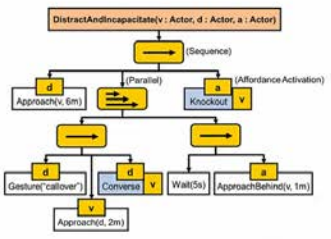


ADAPT: Agent Development and Prototyping Testbed. Alexander Shoulson, Nathan Marhsak, Mubbasir Kapadia, Norman I. Badler. *ACM SIGGRAPH Interactive 3D Graphics and Games (I3D)*, 2013



Event Definition

Parameterized Behavior Tree Definition



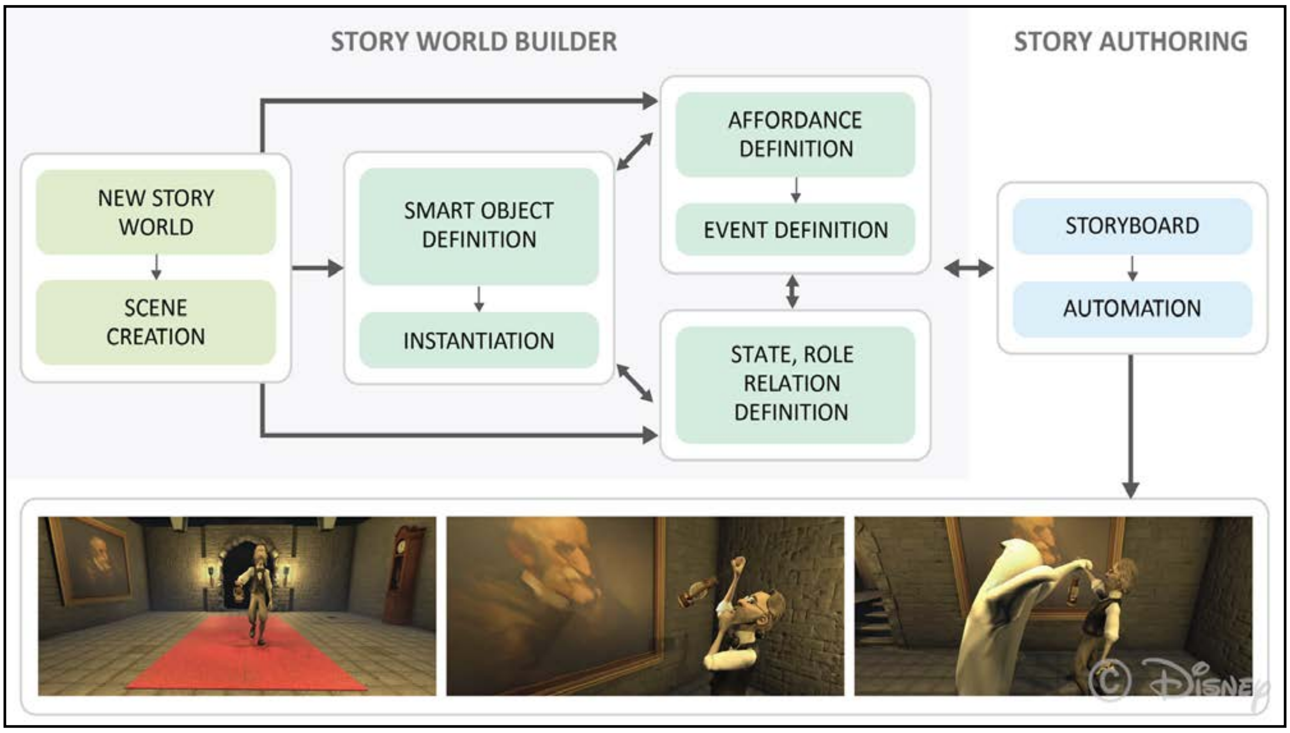
Event Pre- and Post-conditions

Event *DistractAndIncapacitate* (Actor: a_1, a_2, a_3, [*Zone*: z_1, z_2, z_3]) {
Precondition Φ:
$\mathtt{HasWeapon}(a_3) \wedge \neg\mathtt{Incapacitated}(a_1)$
$\wedge\neg\mathtt{Incapacitated}(a_2) \wedge \neg\mathtt{AlliedWith}(a_1, a_2)$
$\wedge\neg\mathtt{Incapacitated}(a_3) \wedge \neg\mathtt{AlliedWith}(a_3, a_2)$
$\wedge\mathtt{InZone}(a_1, z_1) \wedge \mathtt{InZone}(a_2, z_2)$
$\wedge\mathtt{InZone}(a_3, z_3) \wedge \mathtt{CanAccess}(a_1, a_2)$
$\wedge\mathtt{CanAccess}(a_3, a_2)$;
Postcondition Ω:
$\mathtt{Incapacitated}(a_2) \wedge \neg\mathtt{InZone}(a_1, z_1)$
$\wedge\mathtt{InZone}(a_1, z_2) \wedge \neg\mathtt{InZone}(a_3, z_3)$
$\wedge\mathtt{InZone}(a_3, z_2)$;
Behavior Summmary:
a_1 distracts a_2, so a_3 can incapacitate a_2
}

Varied Event Execution

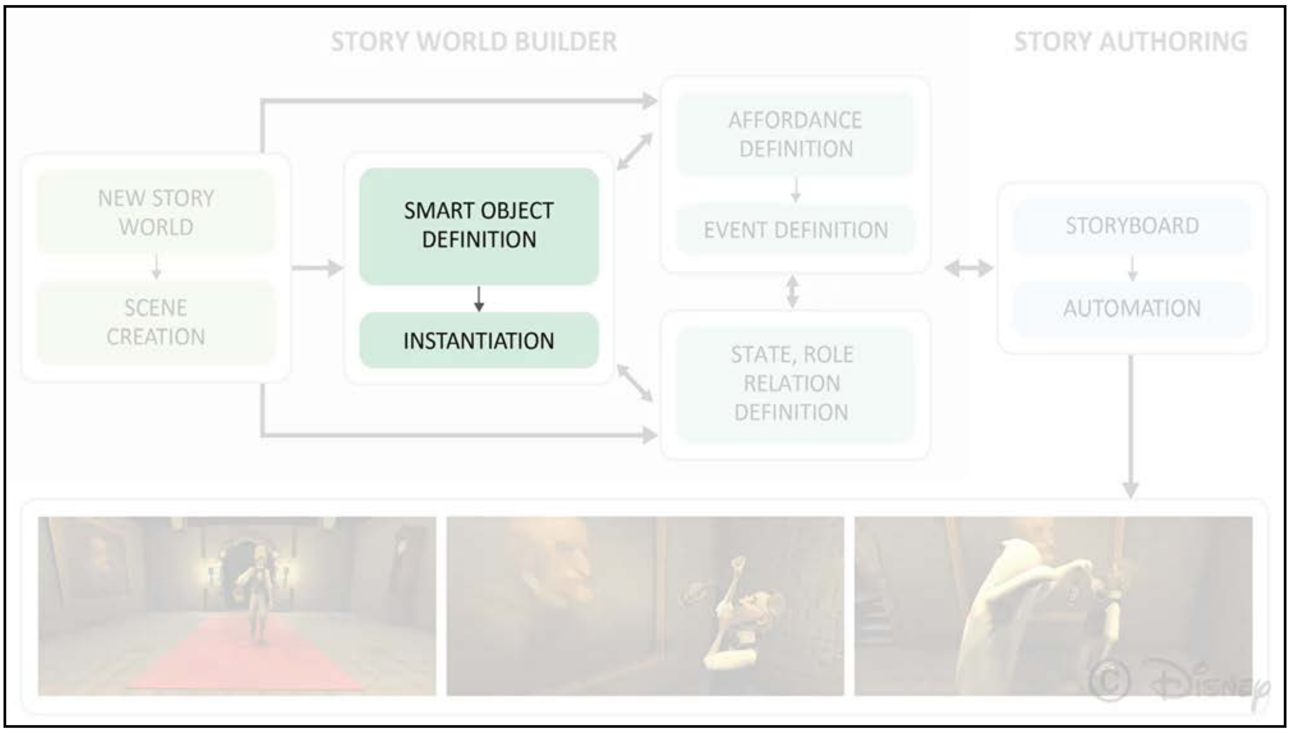
Story World Building

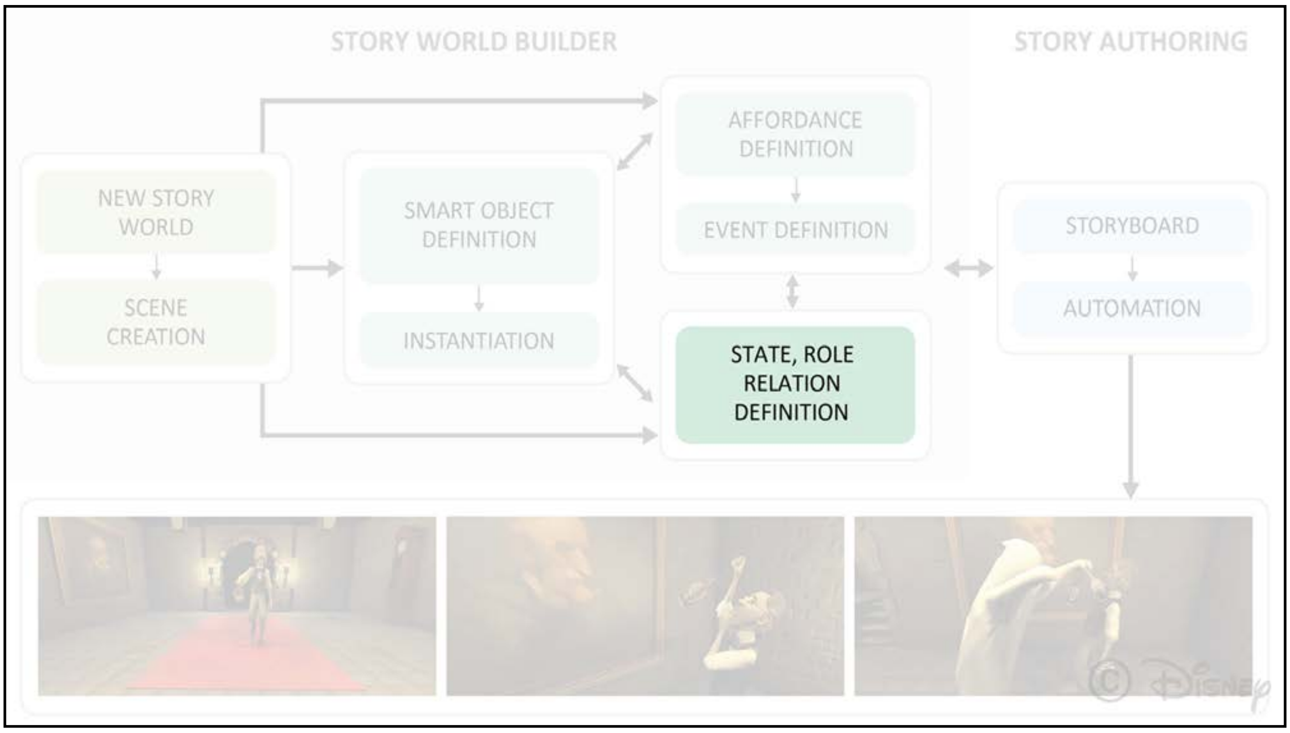
- Wide ruled: A friendly interface to author-goal based story generation [Skorupski et al. 2007]
- Towards an accessible interface for story world building. [Poulakos et al. 2015, presented at INT8]
  - Preliminary work
  - Lacking Affordance creation, bi-directionality, and interface improvements

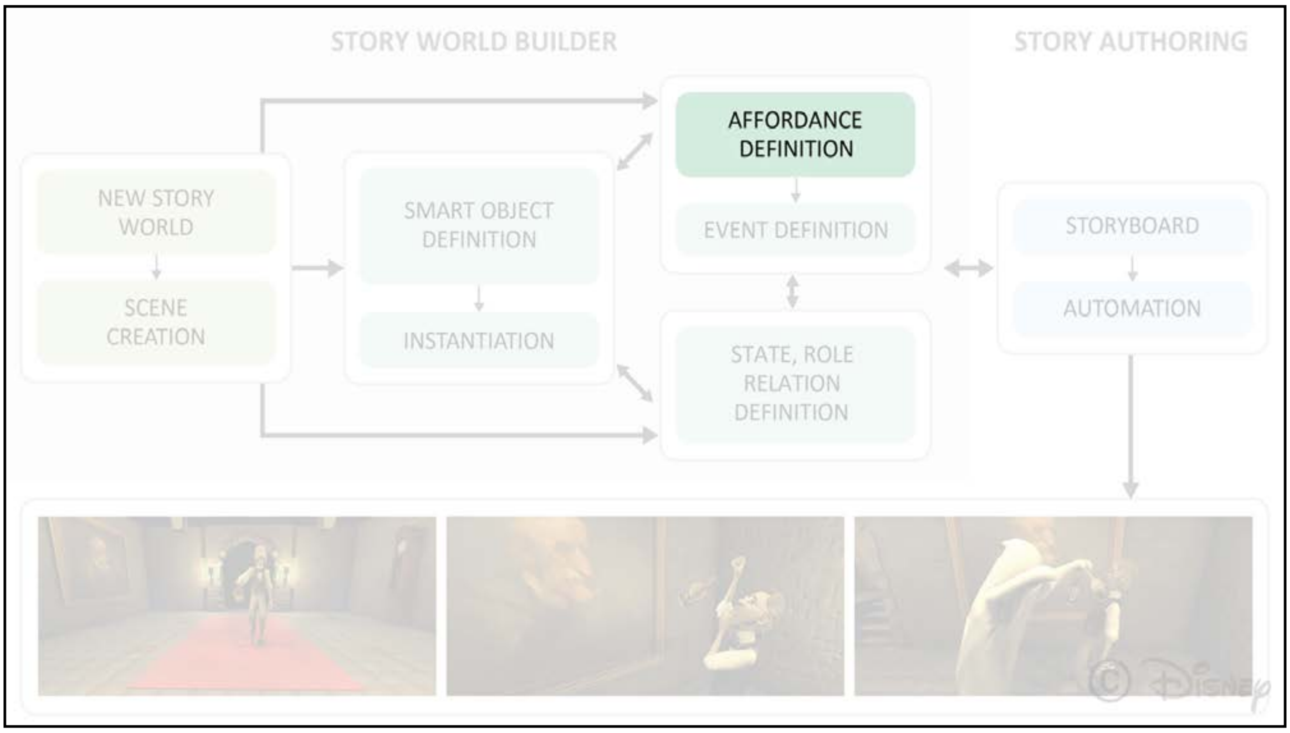


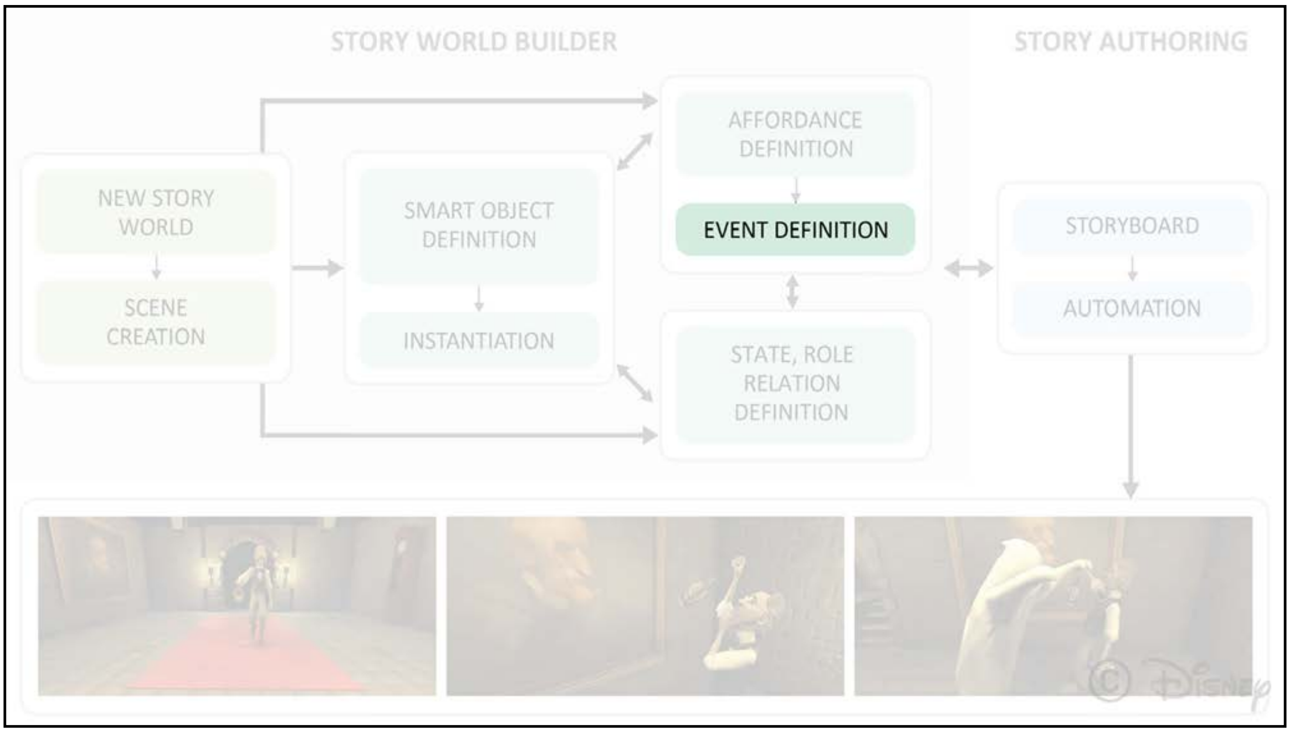
New Story World

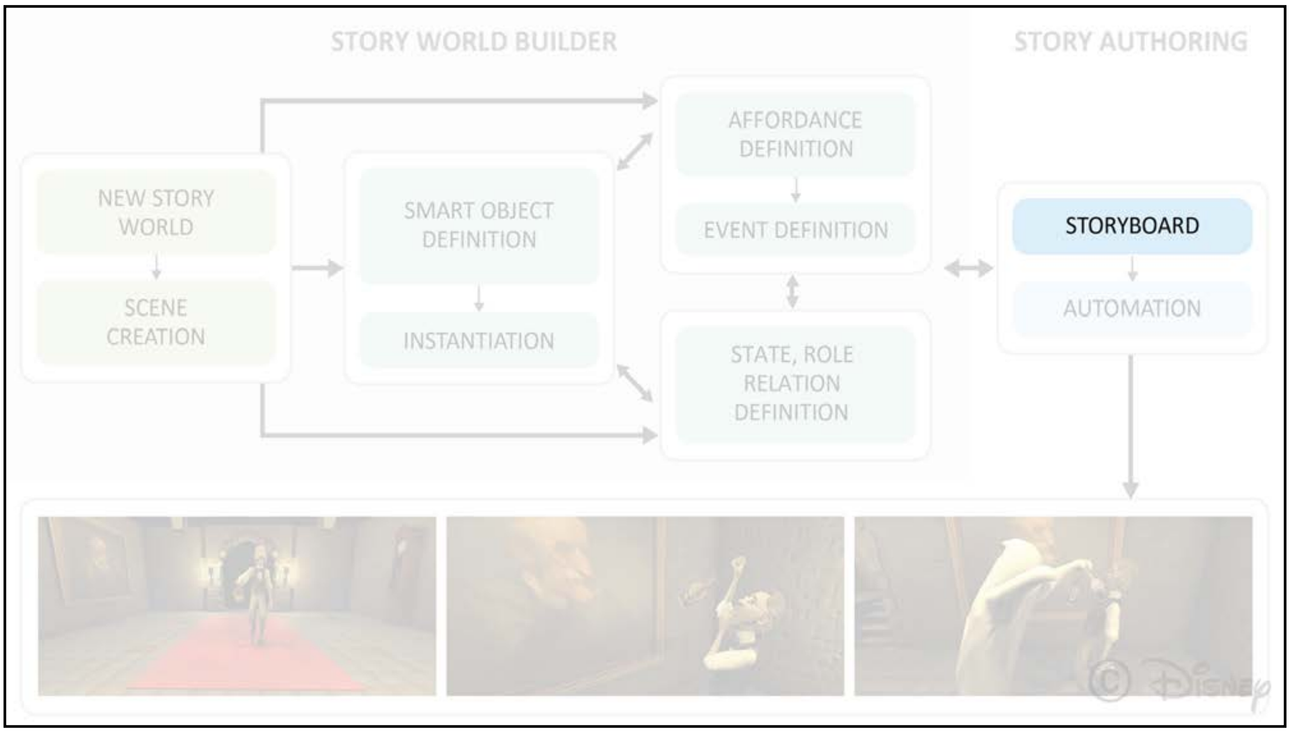
- Unity 3D Project
- Include CANVAS and SWB
- Create a Scene

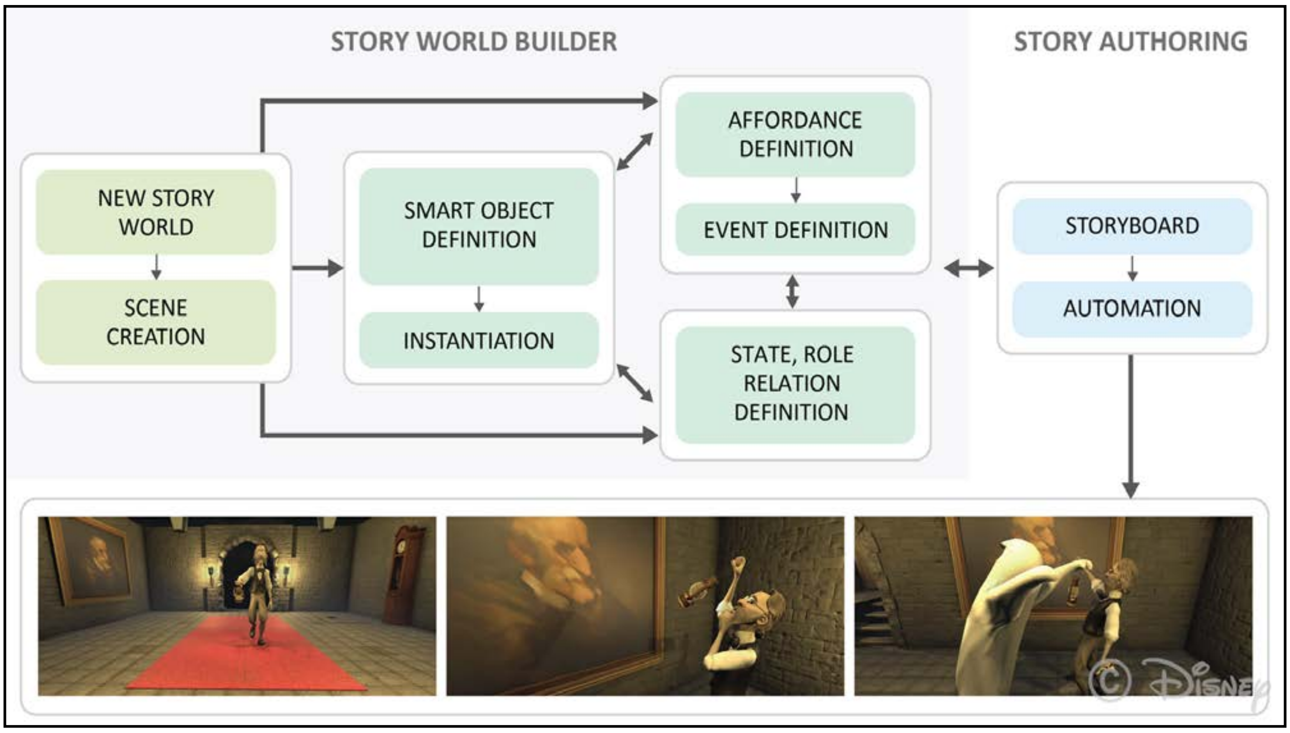


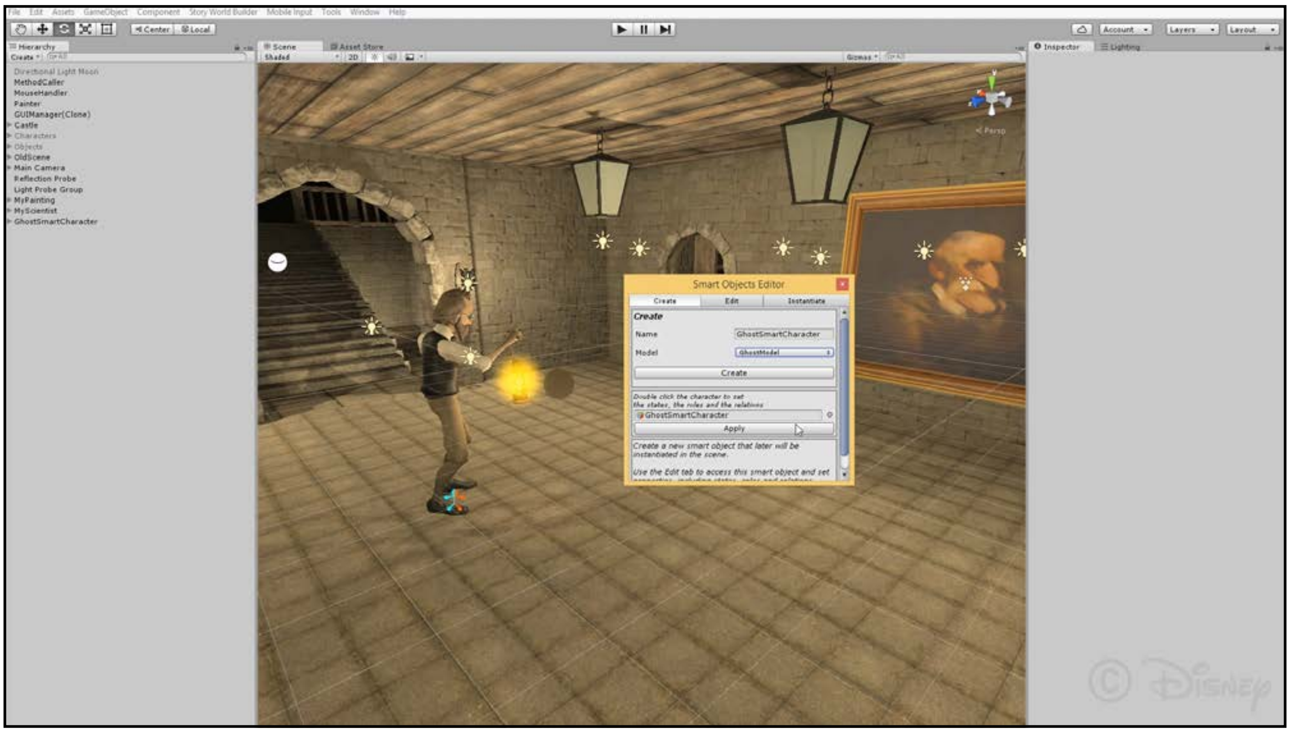

STORY WORLD BUILDER
STORY AUTHORING
NEW STORY WORLD
SCENE CREATION
SMART OBJECT DEFINITION
INSTANTIATION
AFFORDANCE DEFINITION
EVENT DEFINITION
STATE, ROLE RELATION DEFINITION
STORYBOARD
AUTOMATION
© Disney


STORY WORLD BUILDER
STORY AUTHORING
NEW STORY WORLD
SCENE CREATION
SMART OBJECT DEFINITION
INSTANTIATION
AFFORDANCE DEFINITION
EVENT DEFINITION
STATE, ROLE RELATION DEFINITION
STORYBOARD
AUTOMATION
© Disney


STORY WORLD BUILDER
STORY AUTHORING
NEW STORY WORLD
SCENE CREATION
SMART OBJECT DEFINITION
INSTANTIATION
AFFORDANCE DEFINITION
EVENT DEFINITION
STATE, ROLE RELATION DEFINITION
STORYBOARD
AUTOMATION
© Disney


STORY WORLD BUILDER
STORY AUTHORING
NEW STORY WORLD
SCENE CREATION
SMART OBJECT DEFINITION
INSTANTIATION
AFFORDANCE DEFINITION
EVENT DEFINITION
STATE, ROLE RELATION DEFINITION
STORYBOARD
AUTOMATION
© Disney





Create

Name: GhostSmartCharacter

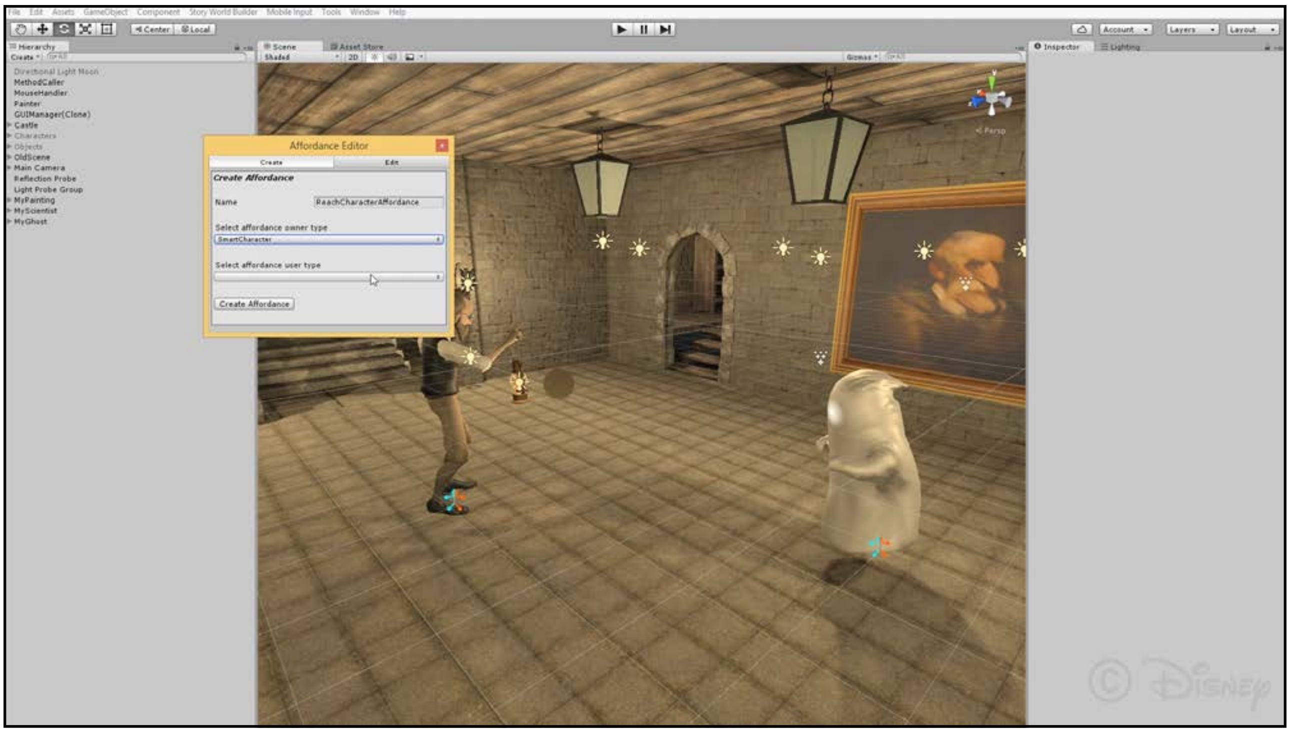
Model: GhostModel

Double click the character to set the states, the roles and the relations

GhostSmartCharacter

Create a new smart object that later will be instantiated in the scene.

Use the Edit tab to access this smart object and set

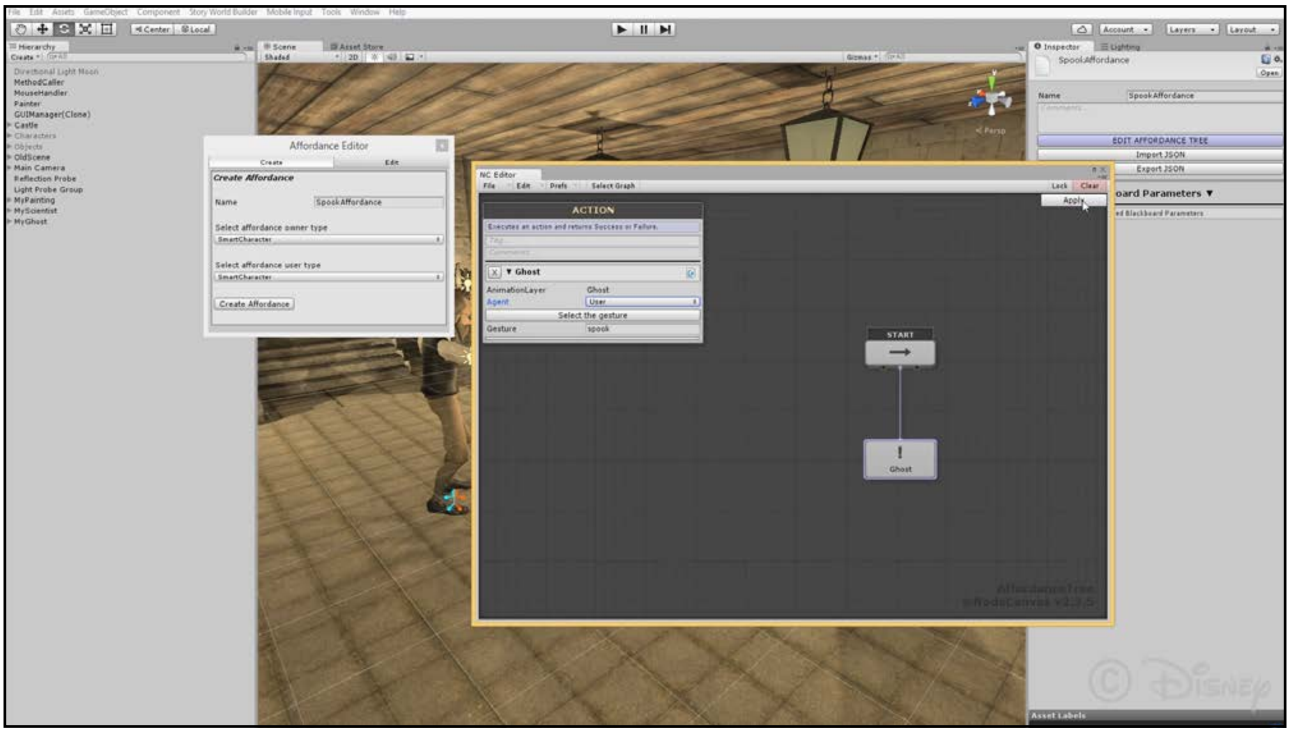


Create Affordance

Name: ReachCharacterAffordance

Select affordance owner type: SmartCharacter

Select affordance user type:



Create Affordance

Name: SpookAffordance

Select affordance owner type: SmartCharacter

Select affordance user type: SmartCharacter

ACTION

Executes an action and returns Success or Failure.

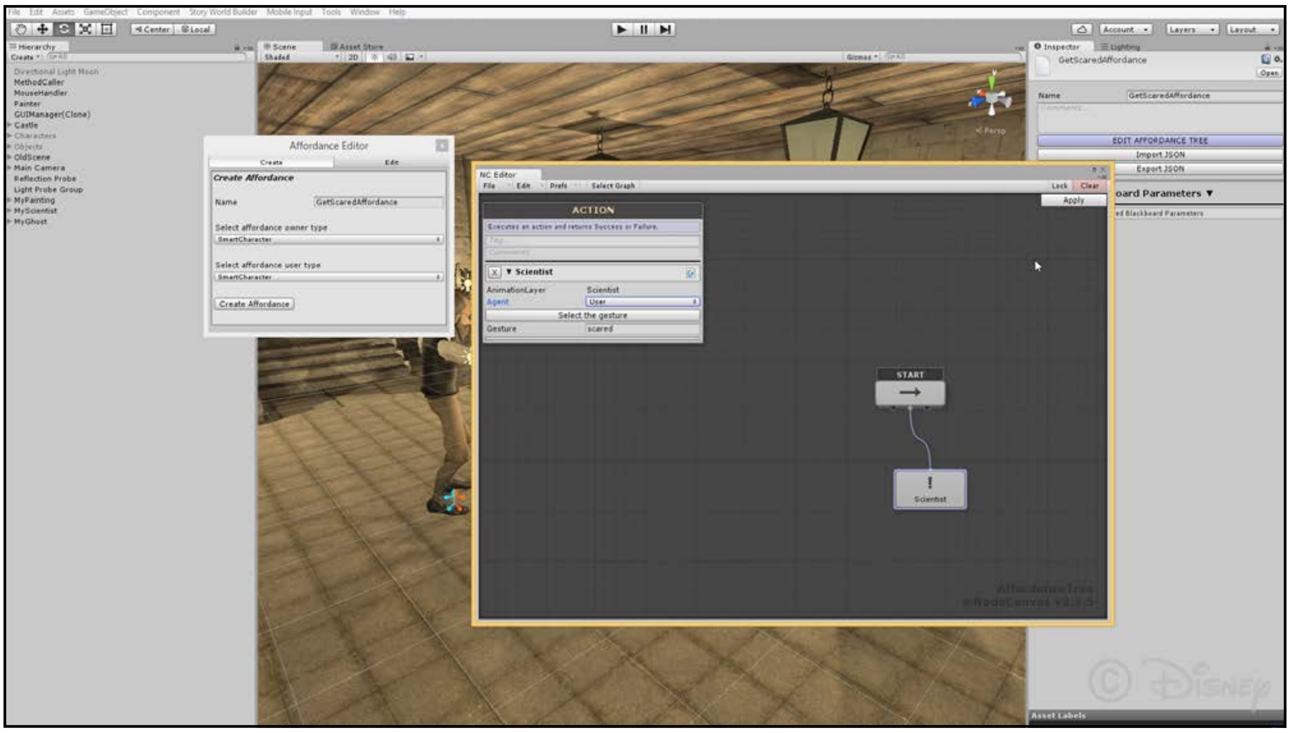
▼ Ghost

AnimationLayer: Ghost

Agent: User

Select the gesture

Gesture: spook



Create Affordance

Name GetScaredAffordance

Select affordance owner type: SmartCharacter

Select affordance user type: SmartCharacter

Create Affordance

ACTION

Executes an action and returns Success or Failure.

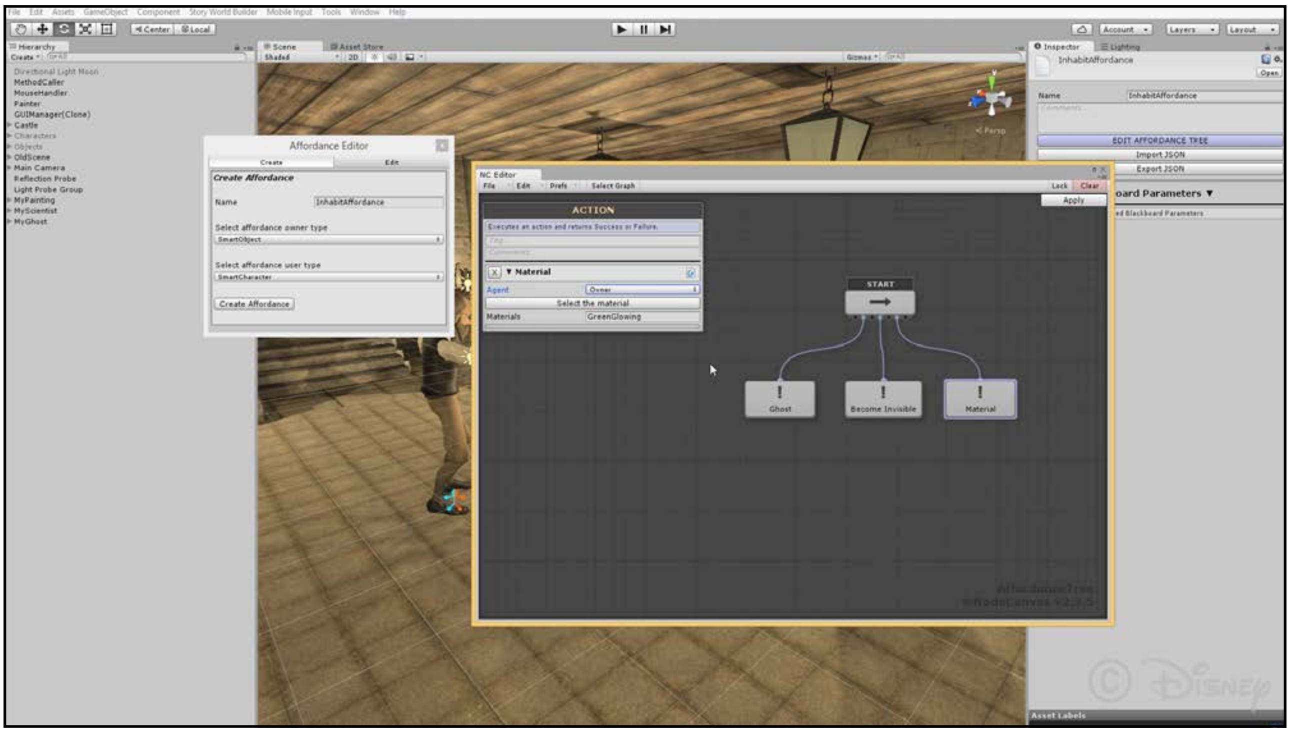
▼ Scientist

AnimationLayer Scientist

Agent User

Select the gesture

Gesture scared



Create Affordance

Name InhabitAffordance

Select affordance owner type: SmartObject

Select affordance user type: SmartCharacter

Create Affordance

ACTION

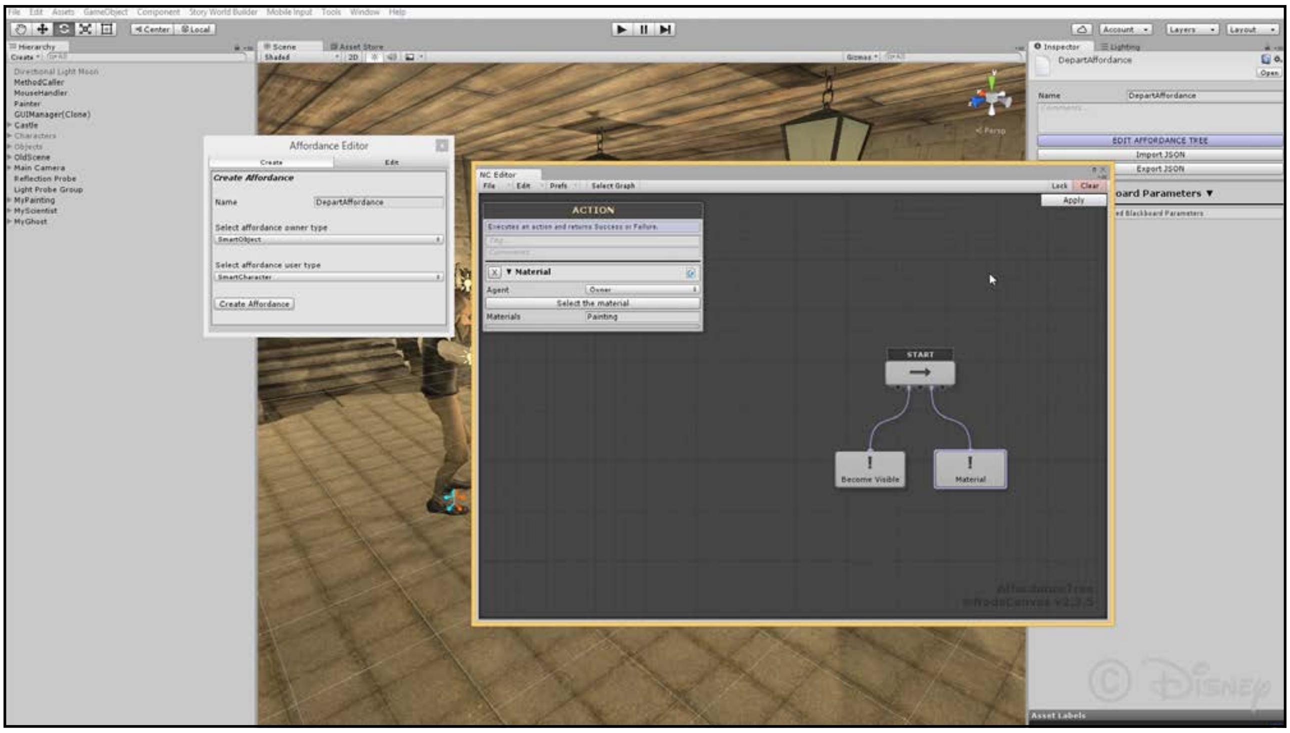
Executes an action and returns Success or Failure.

▼ Material

Agent Owner

Select the material

Materials GreenGlowing



Affordance Editor

Create Affordance

Name: DepartAffordance

Select affordance owner type: SmartObject

Select affordance user type: SmartCharacter

Create Affordance

ACTION

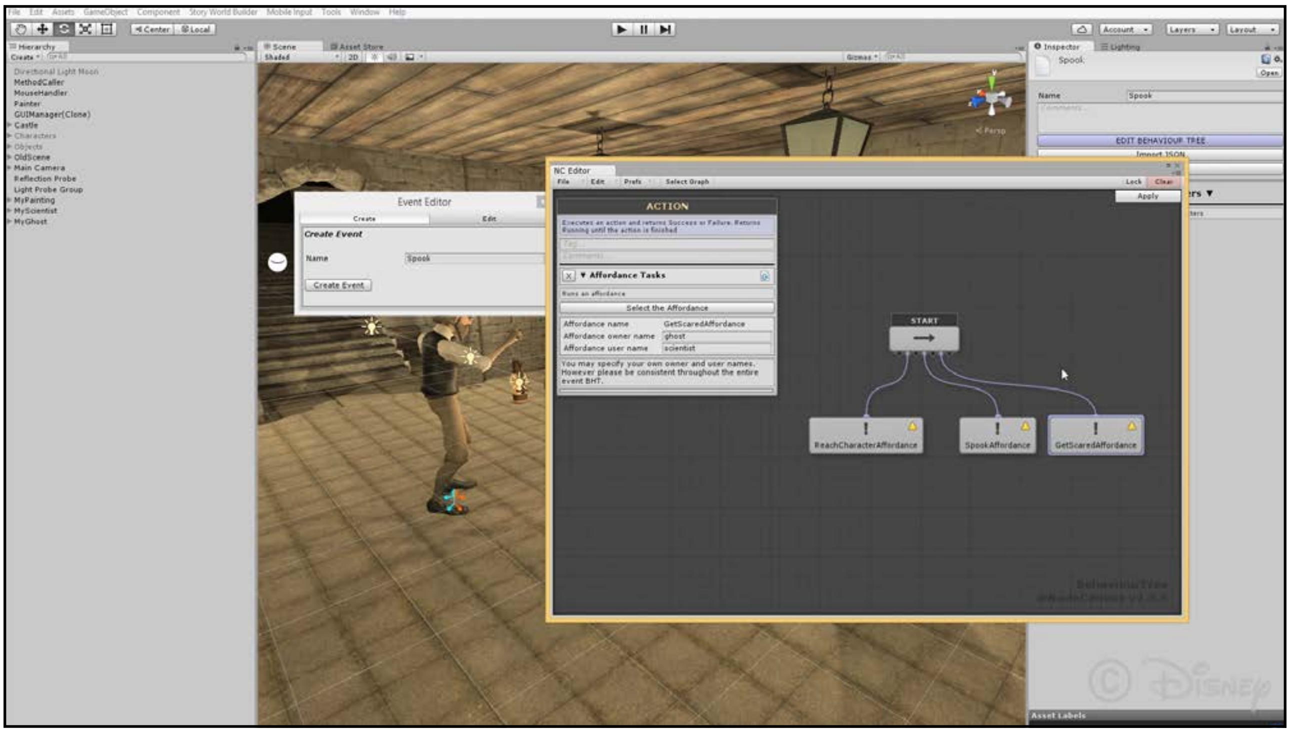
Executes an action and returns Success or Failure.

▼ Material

Agent: Owner

Select the material

Materials: Painting



Event Editor

Create Event

Name: Spook

Create Event

ACTION

Executes an action and returns Success or Failure. Returns Running until the action is finished

▼ Affordance Tasks

Runs an affordance

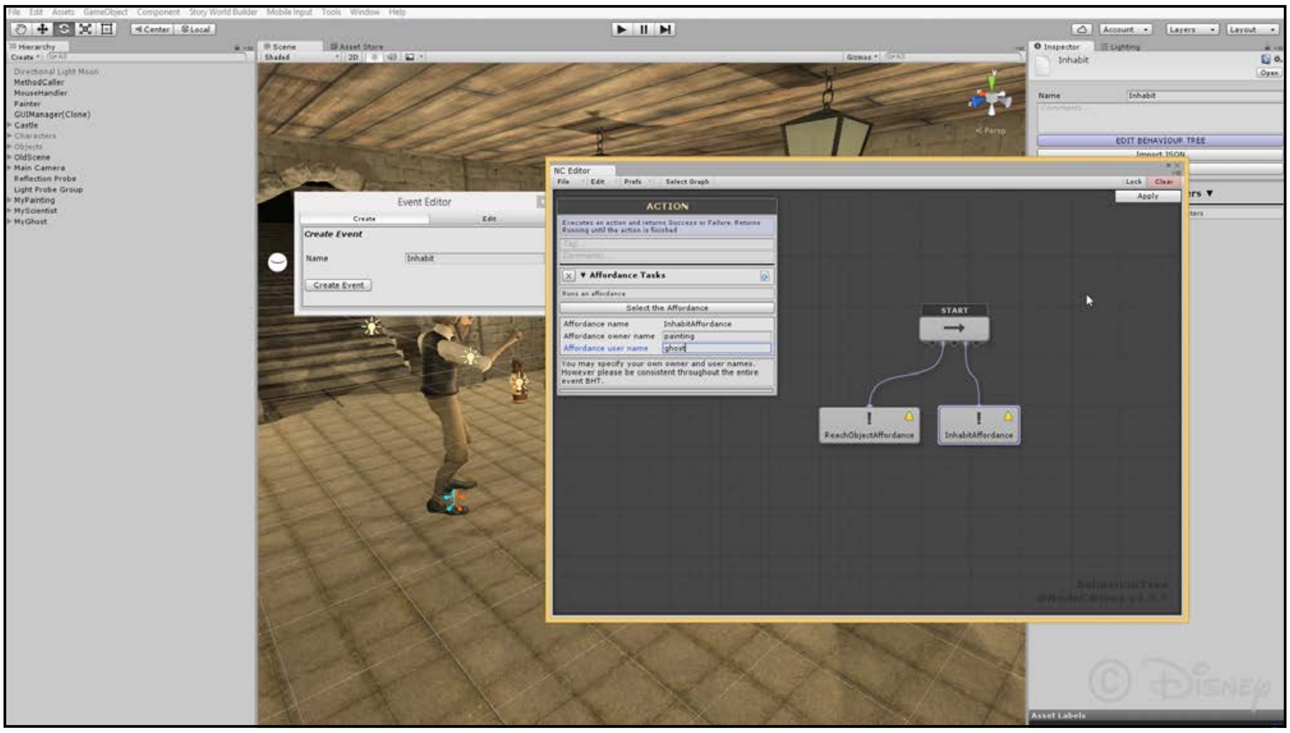
Select the Affordance

Affordance name: GetScaredAffordance

Affordance owner name: ghost

Affordance user name: scientist

You may specify your own owner and user names. However please be consistent throughout the entire event BHT.



Event Editor

Create Event

Name: Inhabit

Create Event

ACTION

Executes an action and returns Success or Failure. Returns Running until the action is finished

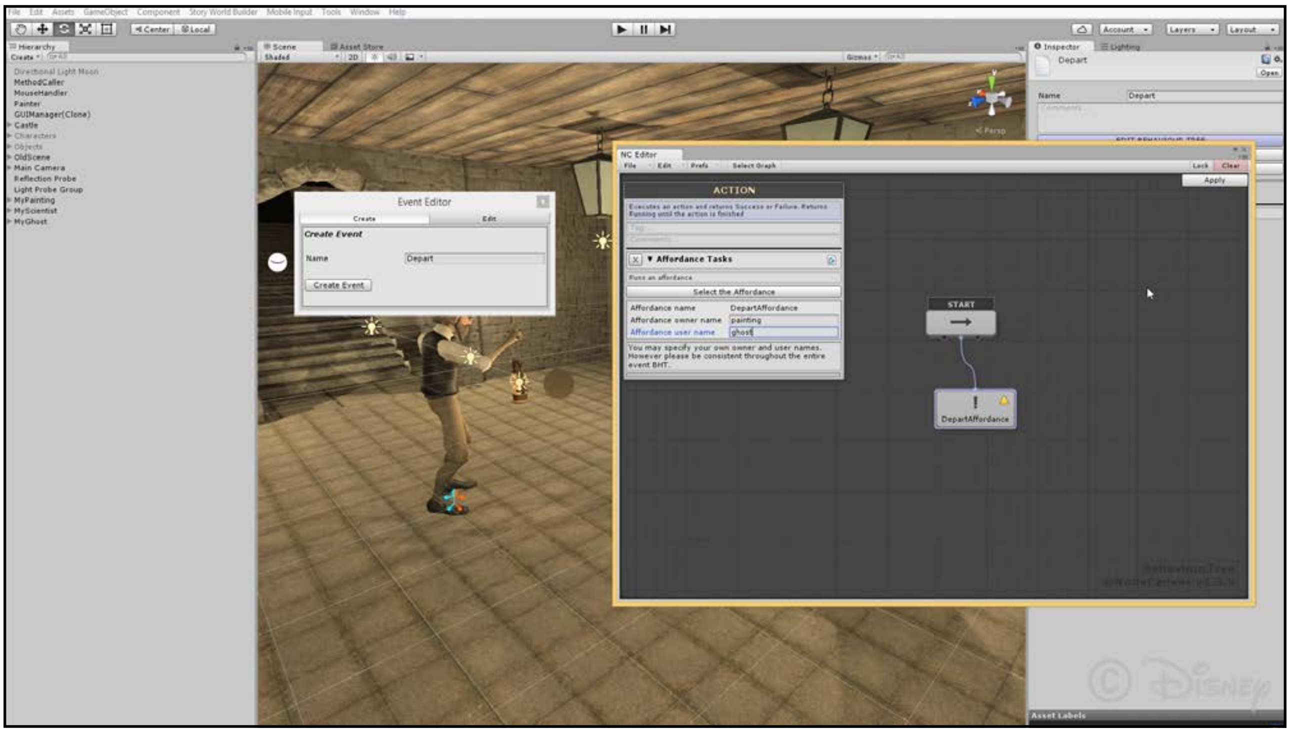
Affordance Tasks

Runs an affordance

Select the Affordance

Affordance name: InhabitAffordance
Affordance owner name: painting
Affordance user name: ghost

You may specify your own owner and user names. However please be consistent throughout the entire event BHT.



Event Editor

Create Event

Name: Depart

Create Event

ACTION

Executes an action and returns Success or Failure. Returns Running until the action is finished

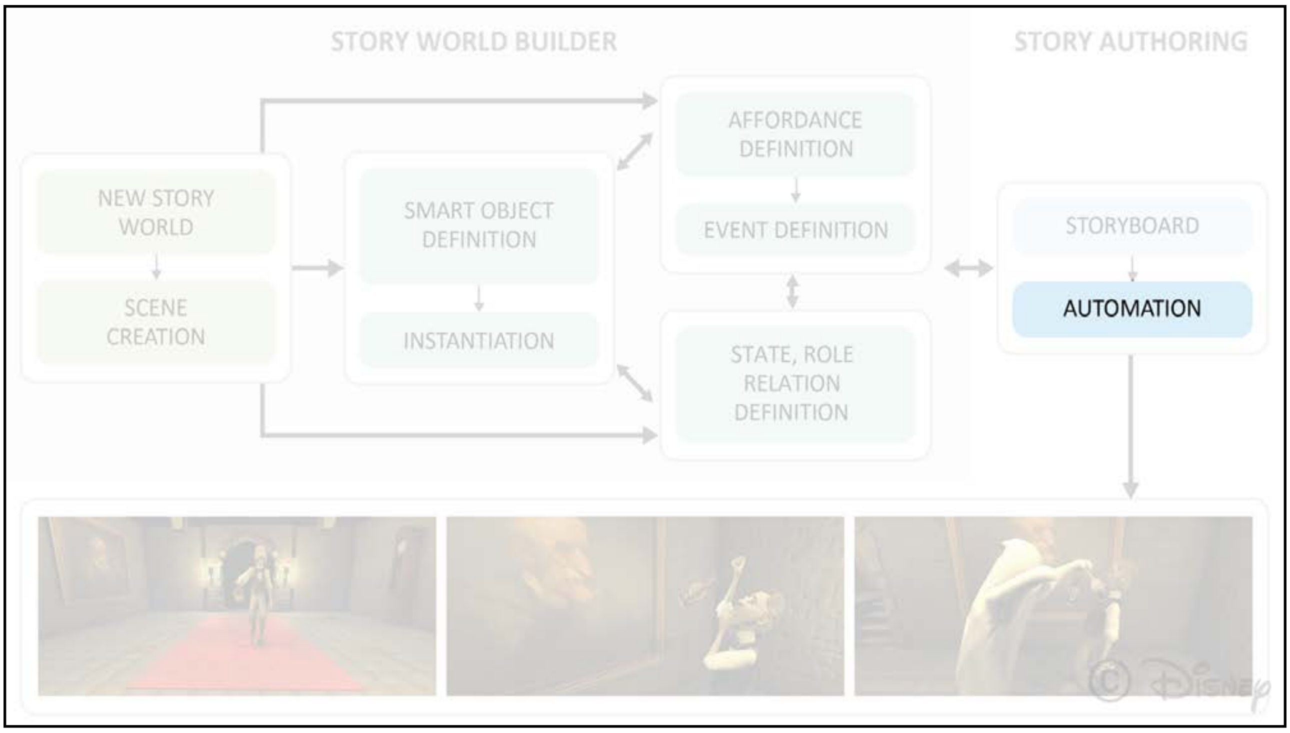
Affordance Tasks

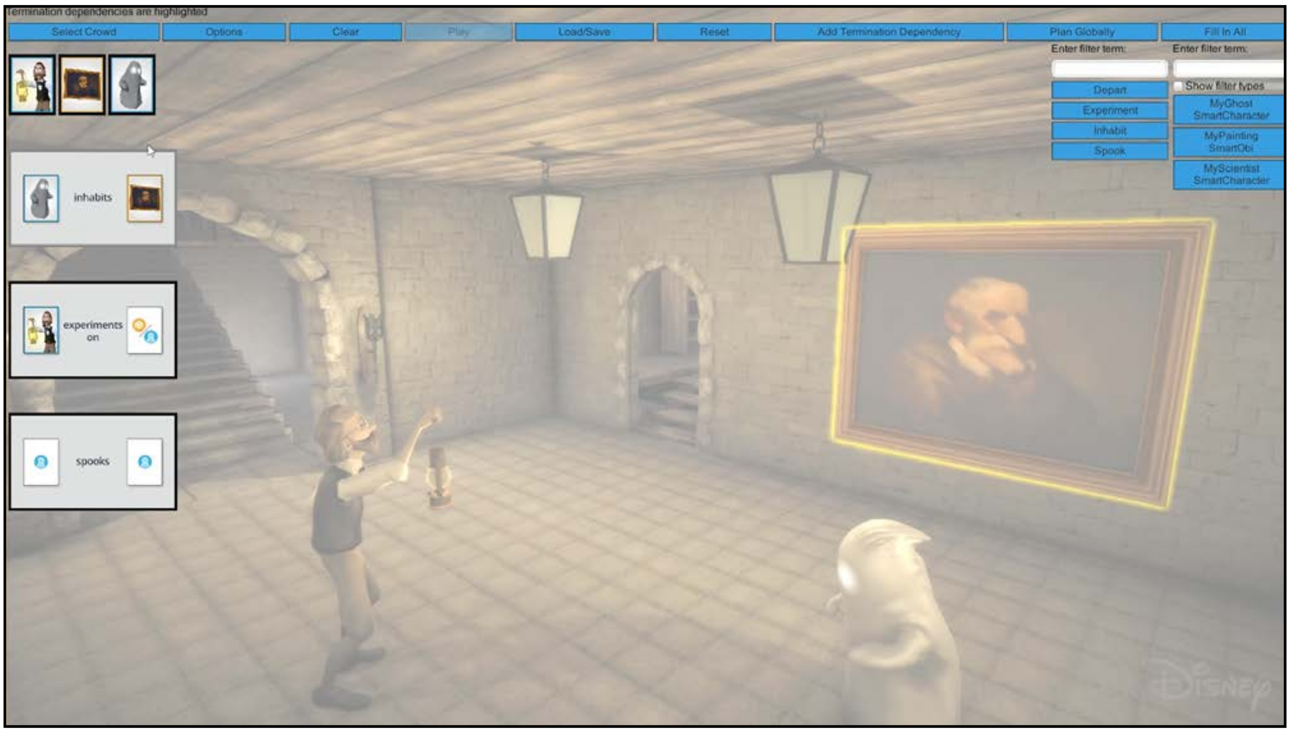
Runs an affordance

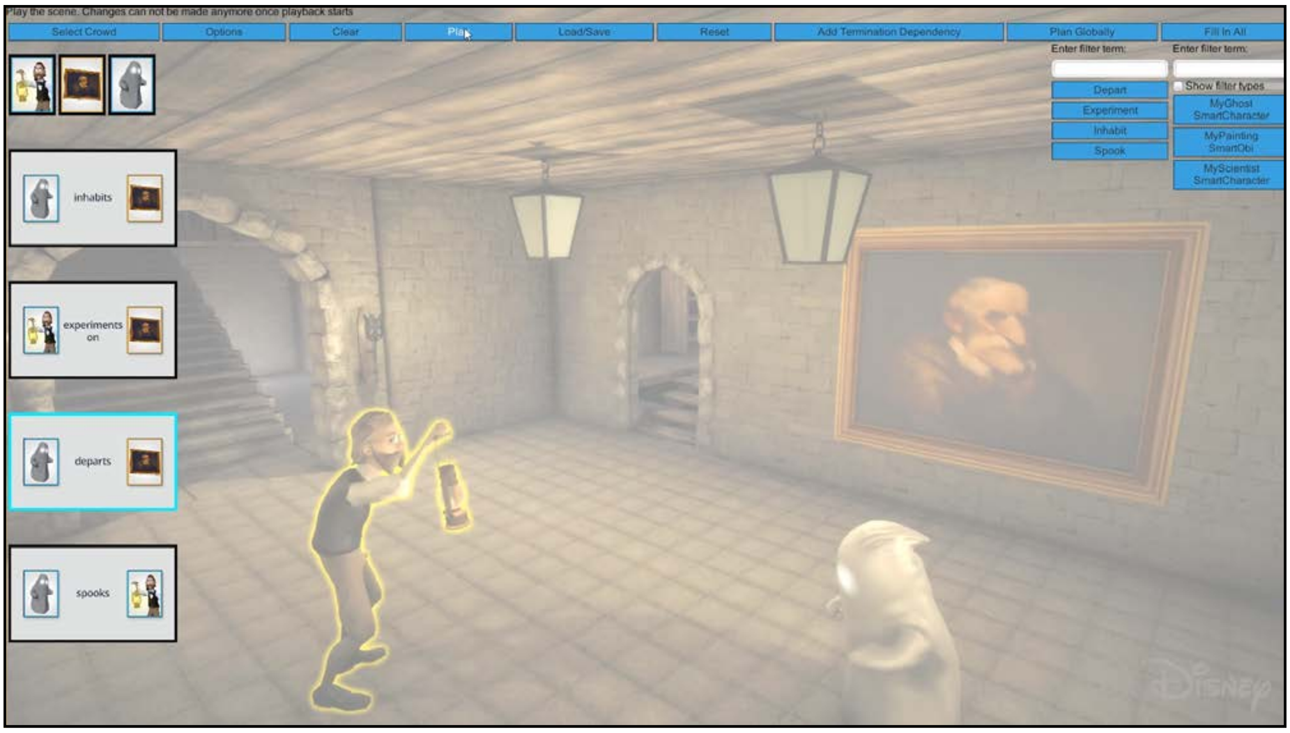
Select the Affordance

Affordance name: DepartAffordance
Affordance owner name: painting
Affordance user name: ghost

You may specify your own owner and user names. However please be consistent throughout the entire event BHT.




Termination dependencies are highlighted


Play the scene. Changes can not be made anymore once playback starts

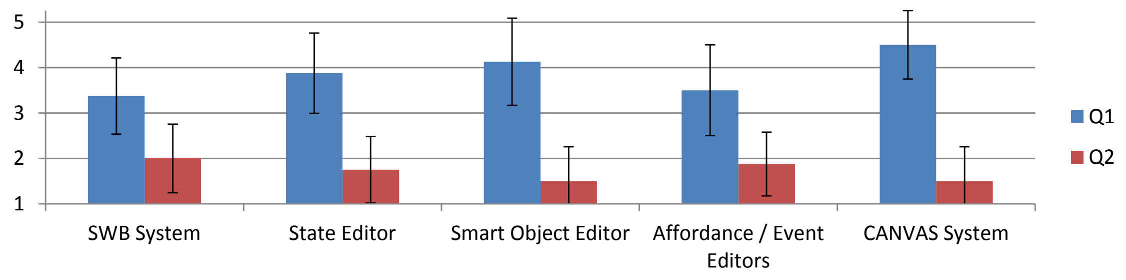
Evaluation (Method)

- Part 1: Introduction Video
- Part 2: Extend story world and author story
- Part 3: Extend story world to utilize planning
- Part 4: Usability Questionnaire

Evaluation Results

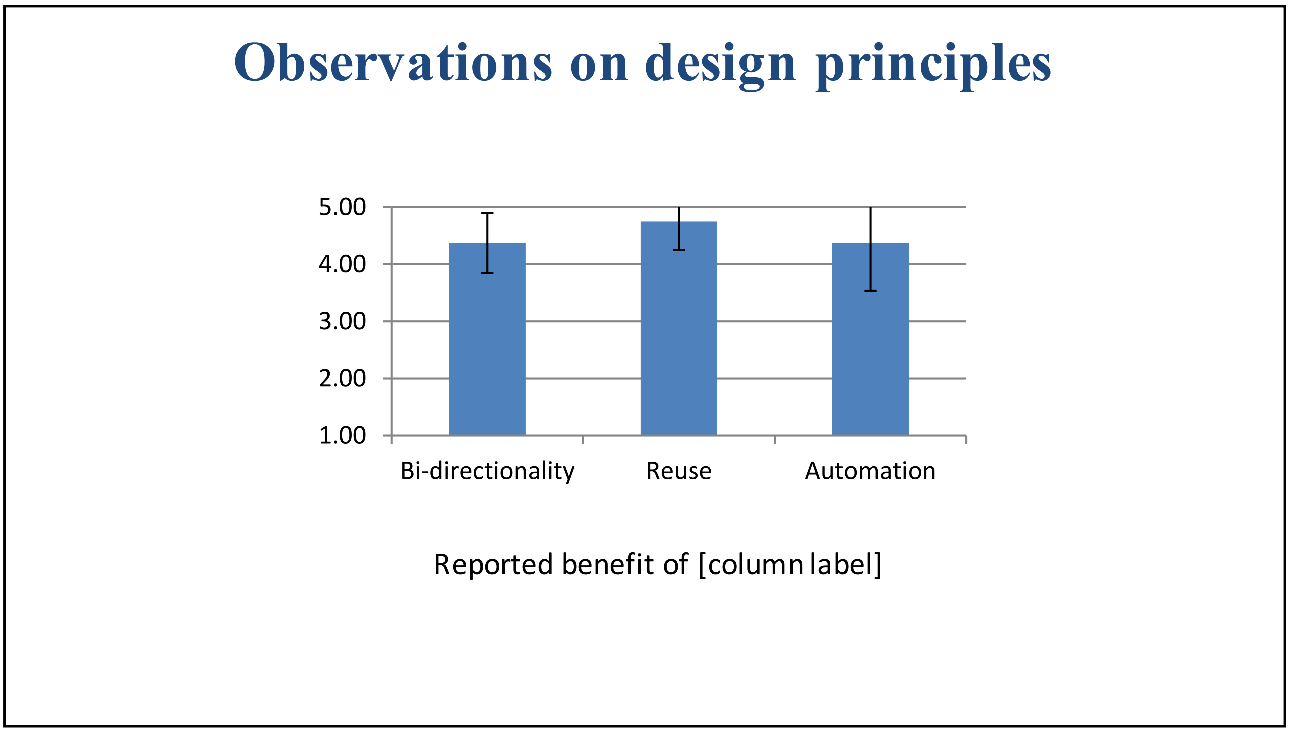
- All subject successfully
  - Created Haunted Castle Story World
  - Authored Stories
- System Usability Survey (SUS)
  - Demonstrated usable system (score: 66.88)

Evaluation of Subcomponents

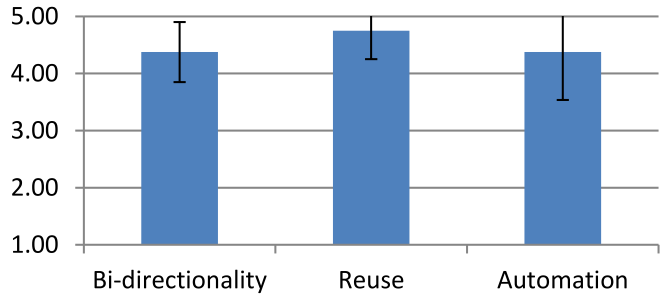


Q1: I thought the [column label] was easy to use.

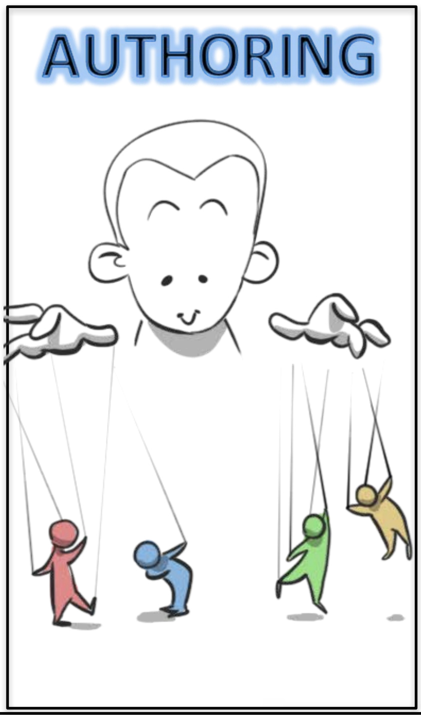
Q2: I think that I would need the support of a technical person to be able to use this [column label]



Observations on design principles



Reported benefit of [column label]

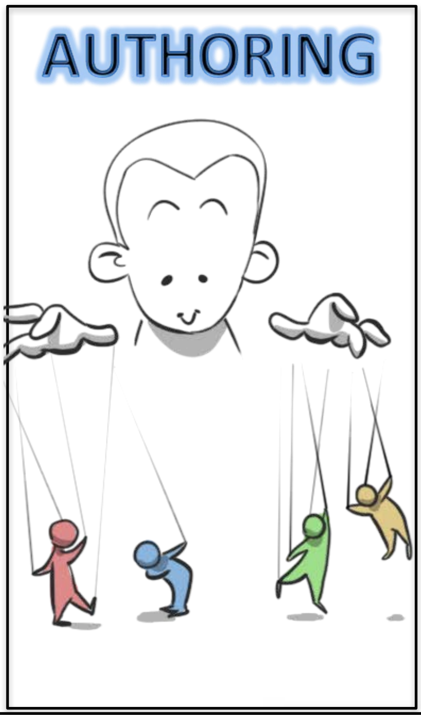


Requirements

Accessibility

Expressivity

Collaboration

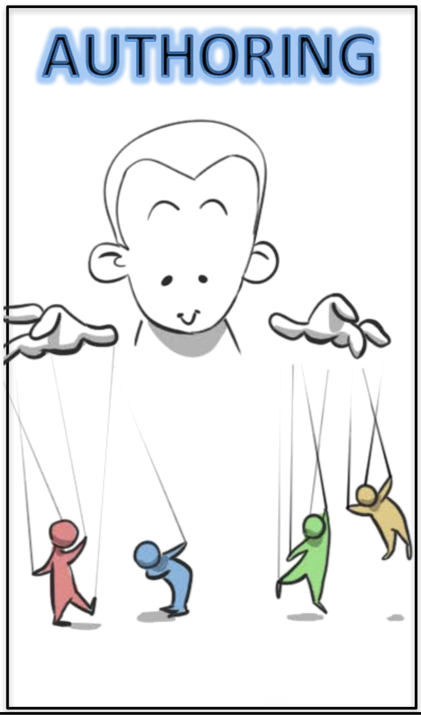


Requirements

Visual Storyboards

Expressivity

Collaboration

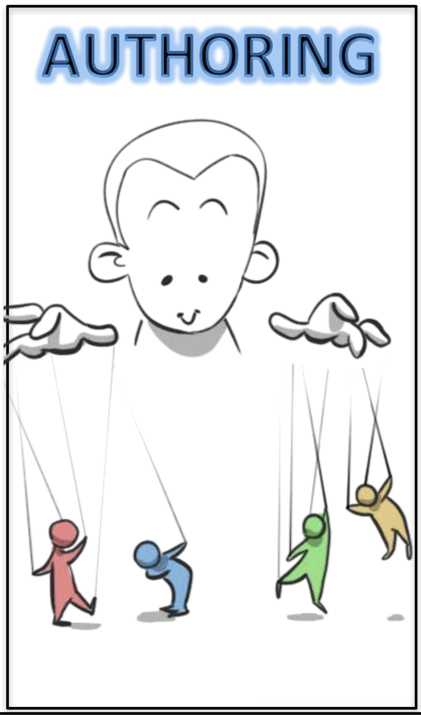


Requirements

Visual Storyboards

Computer-Assisted Story Authoring

Collaboration

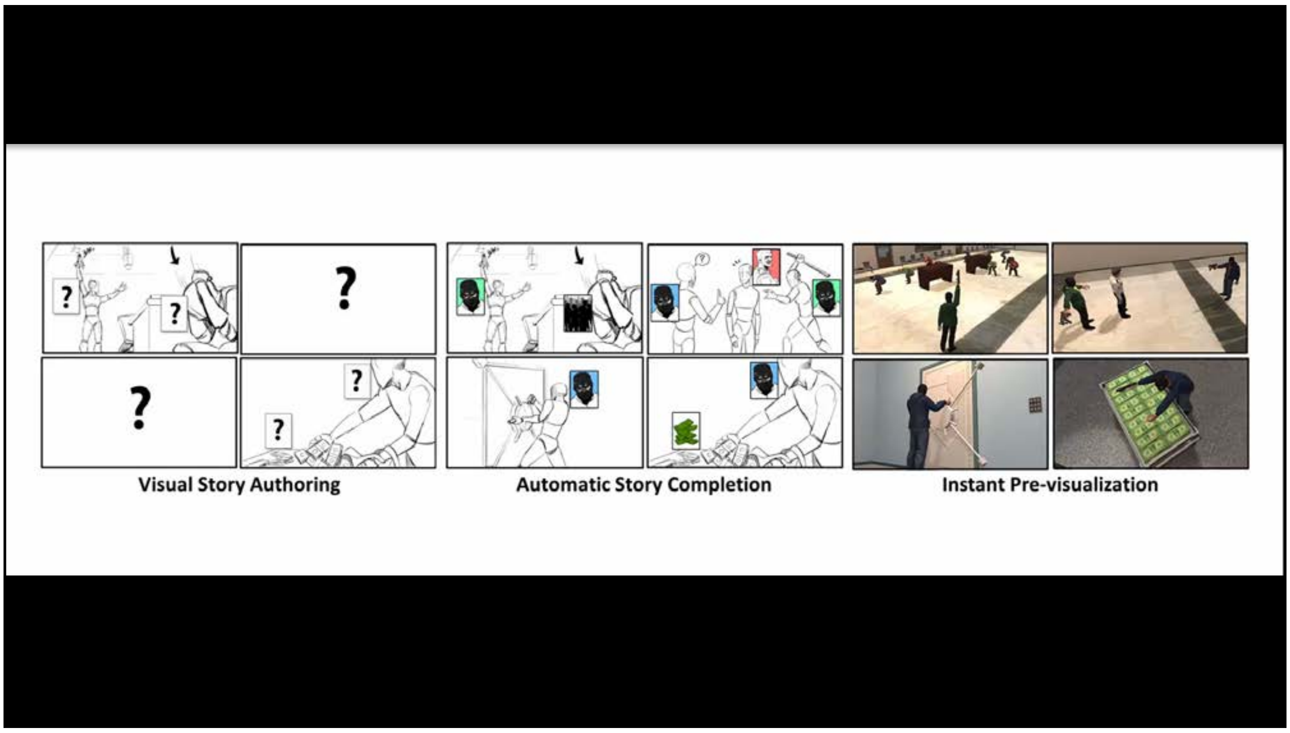


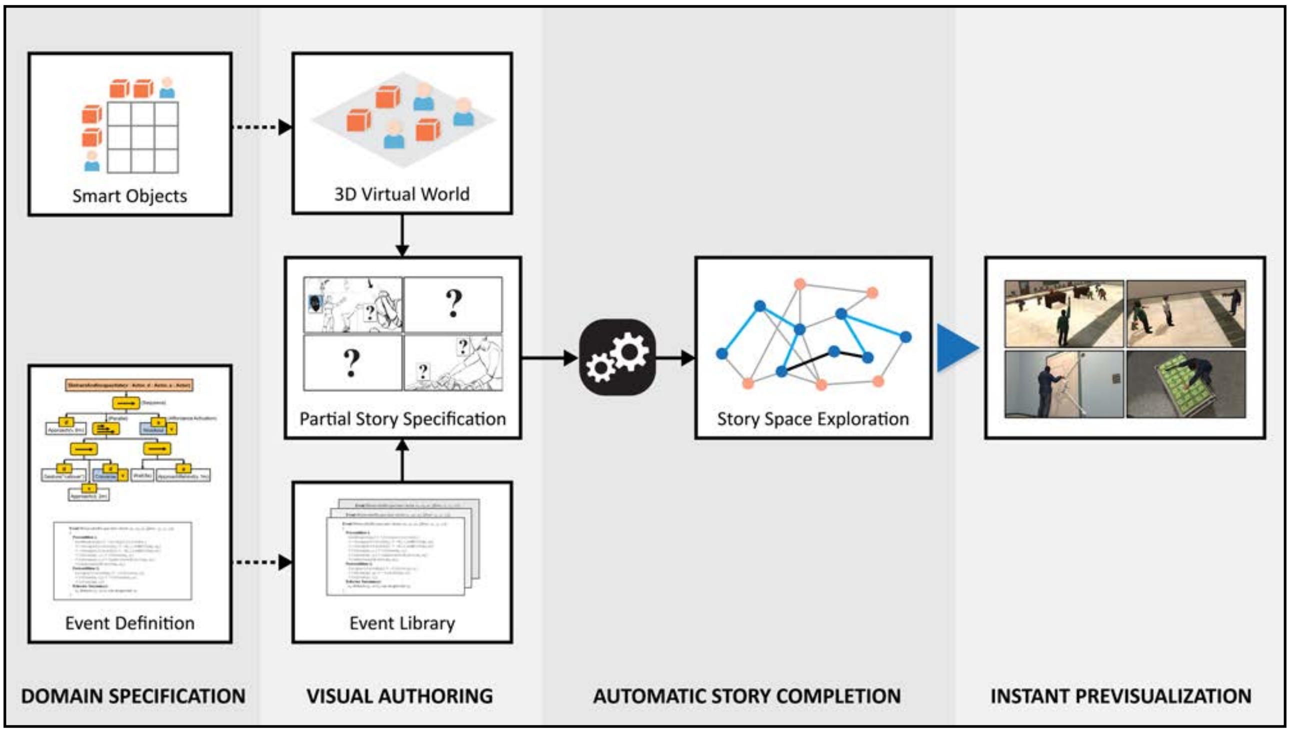
Solutions

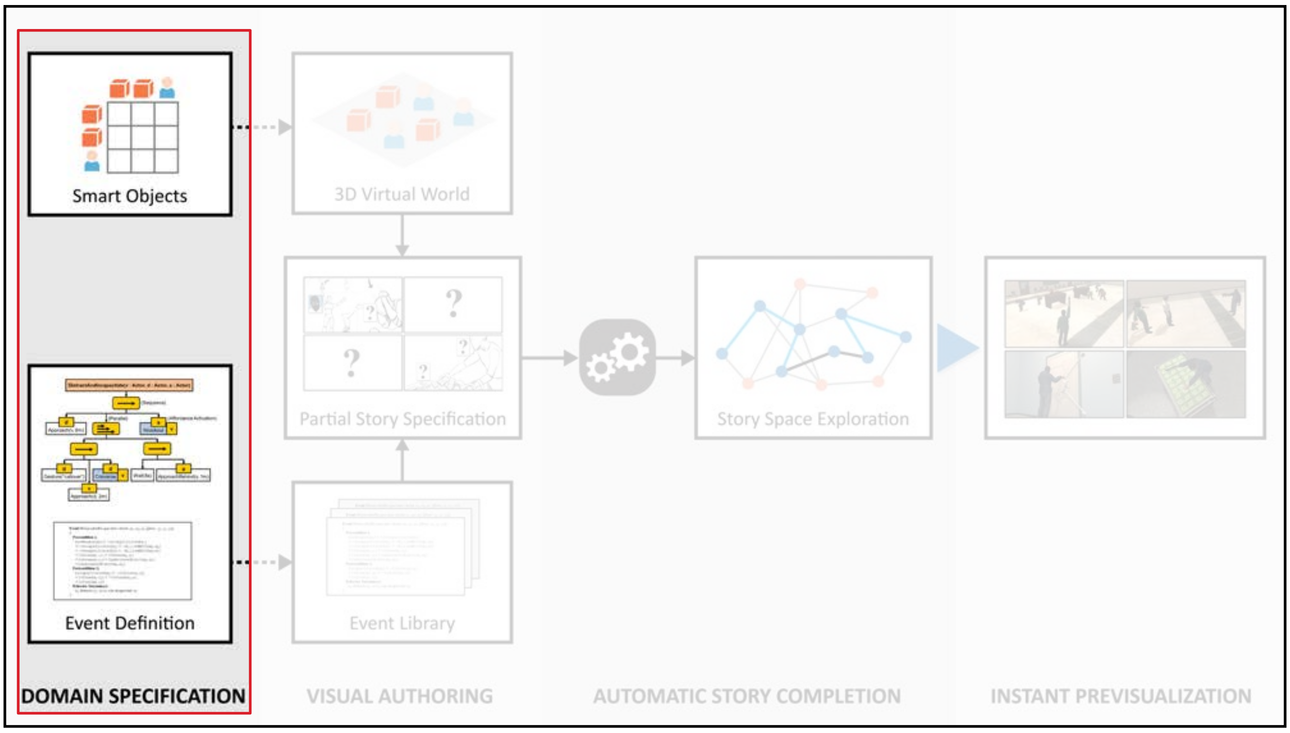
CANVAS

An Event Centric Approach to Authoring Stories in Crowds. Mubbasir Kapadia, Alexander Shoulson, Cyril Steimer, Samuel Oberholzer, Robert W. Sumner, Markus Gross. *ACM SIGGRAPH Motion in Games 2016.*

CANVAS: Computer-Assisted Narrative Animation Synthesis. Mubbasir Kapadia, Seth Frey, Alexander Shoulson, Robert W. Sumner, Markus Gross. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2016.*

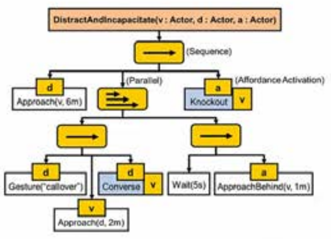

Visual Story Authoring
Automatic Story Completion
Instant Pre-visualization


Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION



Event Definition

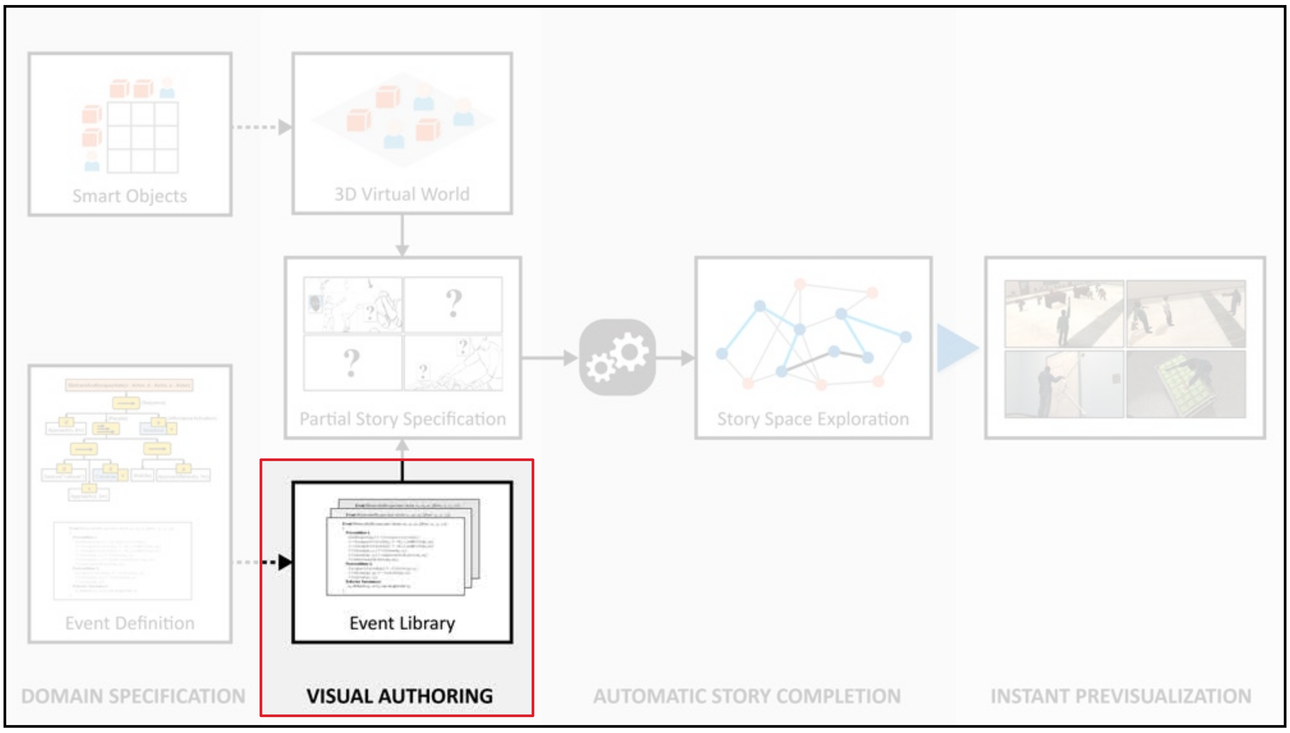
Parameterized Behavior Tree Definition

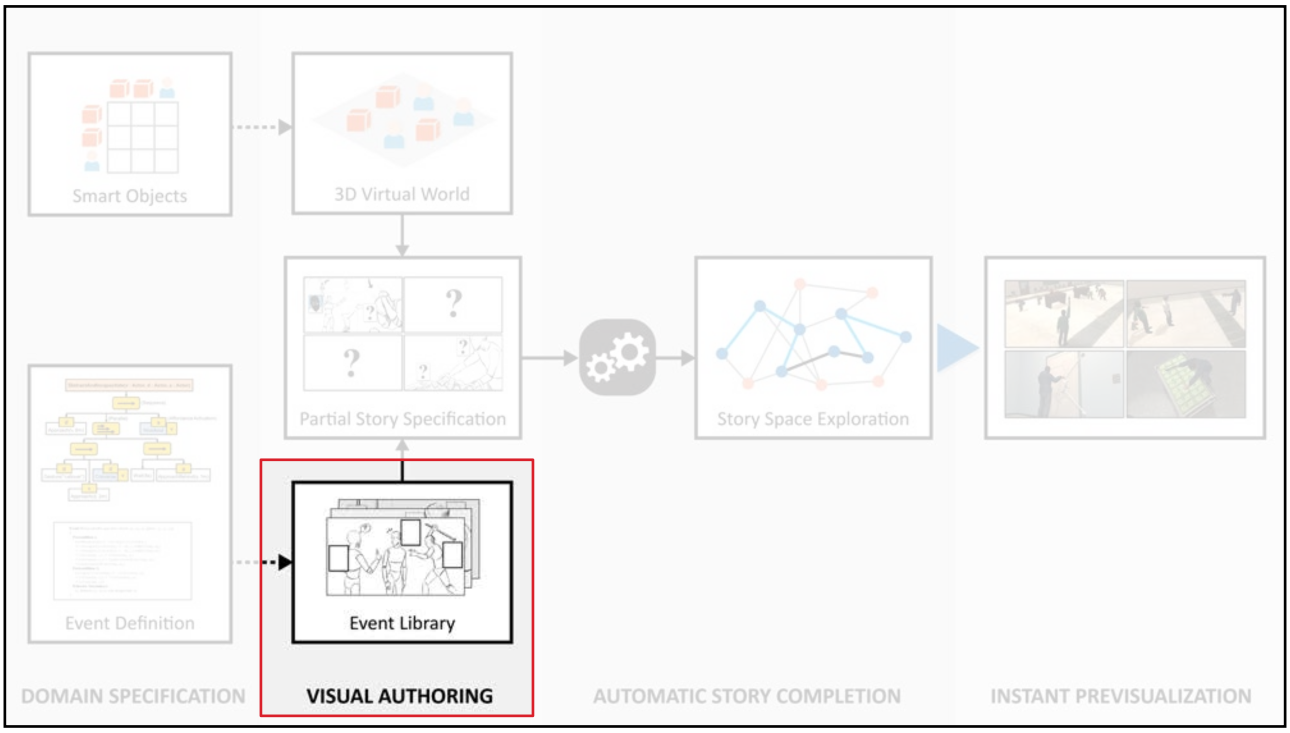


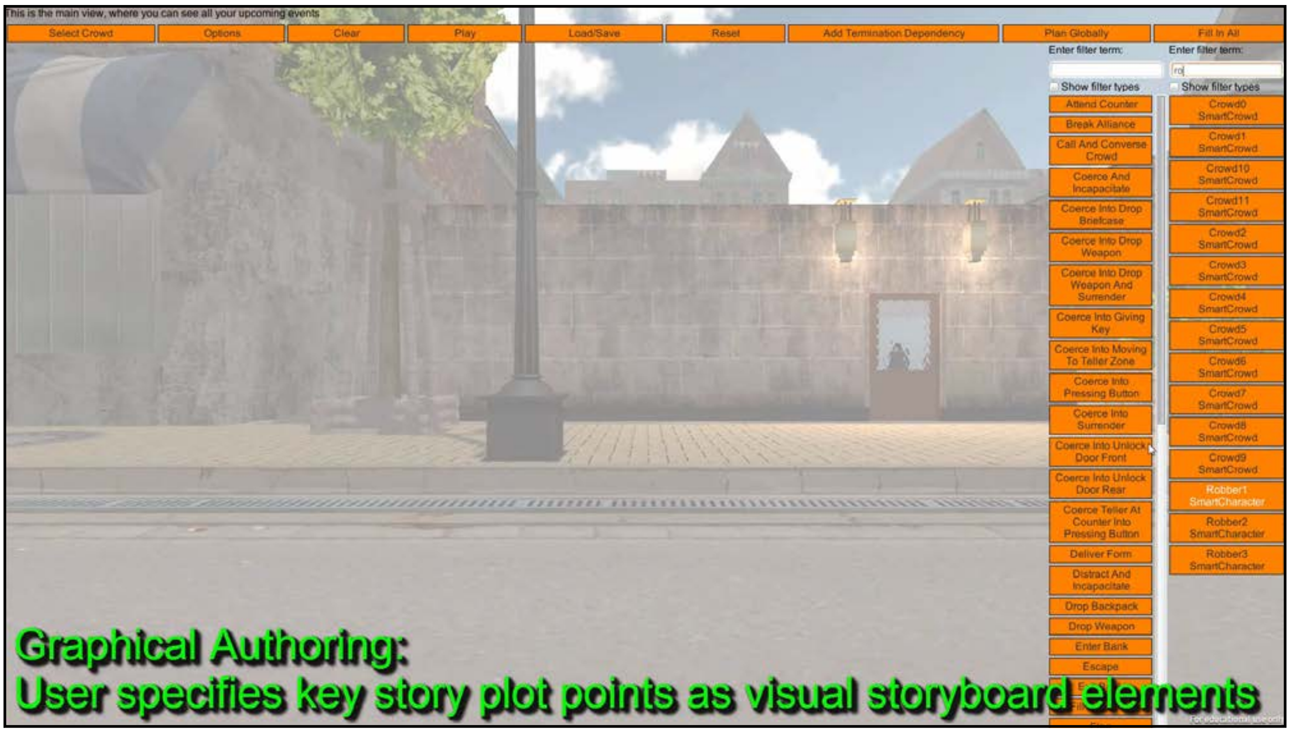
Event Pre- and Post-conditions

Event *DistractAndIncapacitate* (Actor: a_1, a_2, a_3, [*Zone*: z_1, z_2, z_3]) {
  Precondition Φ:
    $\mathtt{HasWeapon}(a_3) \land \neg\mathtt{Incapacitated}(a_1)$
    $\land\neg\mathtt{Incapacitated}(a_2) \land \neg\mathtt{AlliedWith}(a_1, a_2)$
    $\land\neg\mathtt{Incapacitated}(a_3) \land \neg\mathtt{AlliedWith}(a_3, a_2)$
    $\land\mathtt{InZone}(a_1, z_1) \land \mathtt{InZone}(a_2, z_2)$
    $\land\mathtt{InZone}(a_3, z_3) \land \mathtt{CanAccess}(a_1, a_2)$
    $\land\mathtt{CanAccess}(a_3, a_2)$;
  Postcondition Ω:
    $\mathtt{Incapacitated}(a_2) \land \neg\mathtt{InZone}(a_1, z_1)$
    $\land\mathtt{InZone}(a_1, z_2) \land \neg\mathtt{InZone}(a_3, z_3)$
    $\land\mathtt{InZone}(a_3, z_2)$;
  Behavior Summmary:
    a_1 distracts a_2, so a_3 can incapacitate a_2
}

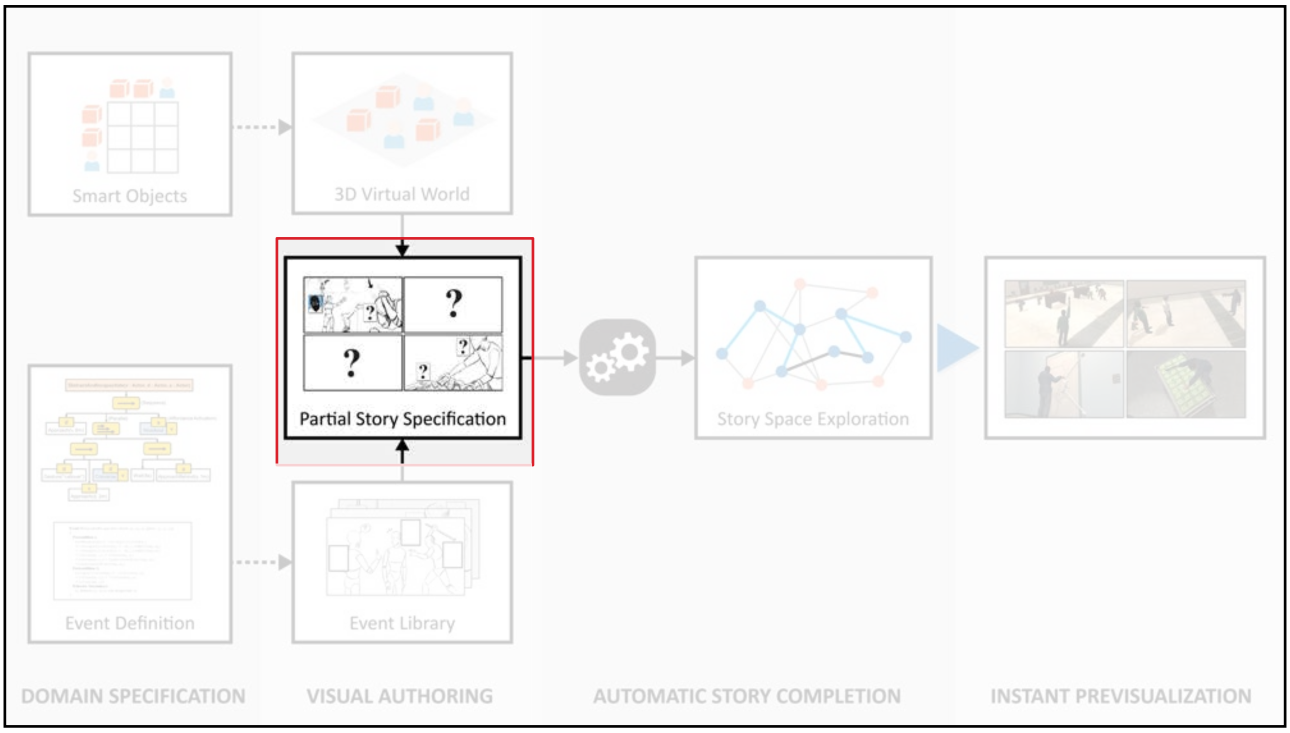
Varied Event Execution

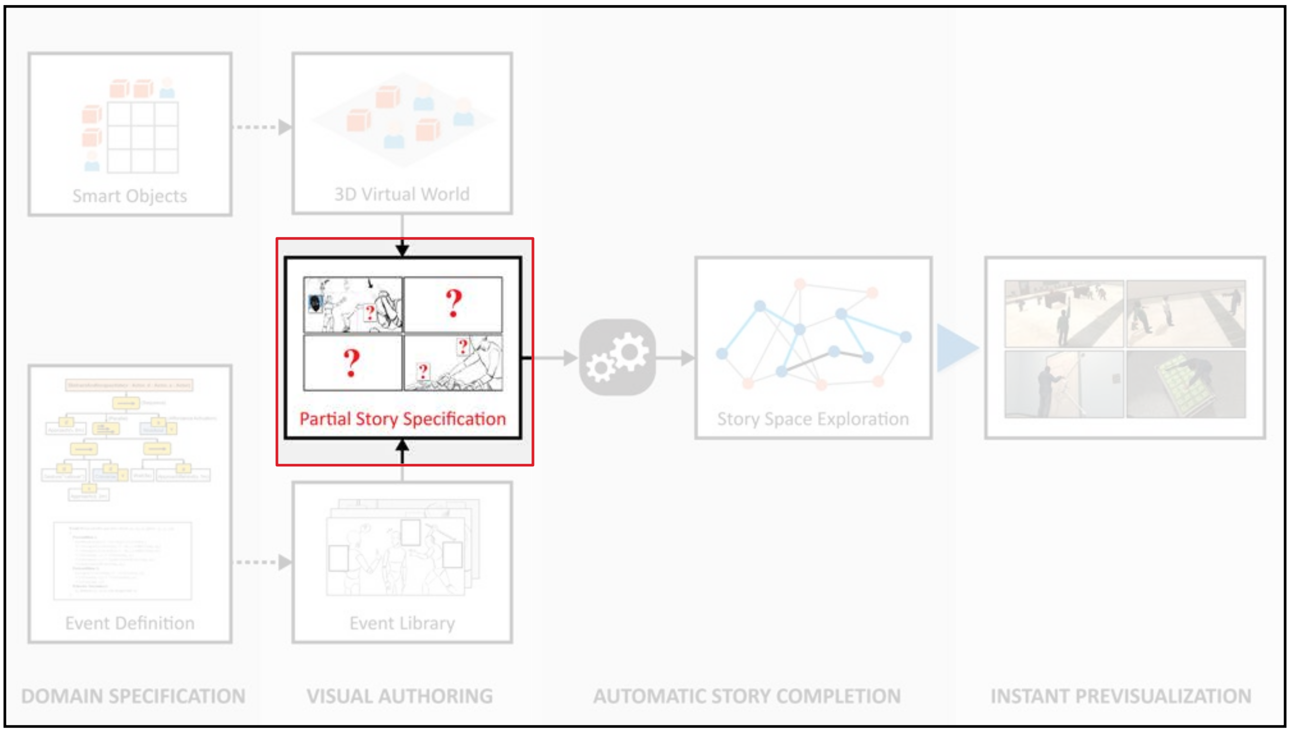

Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION

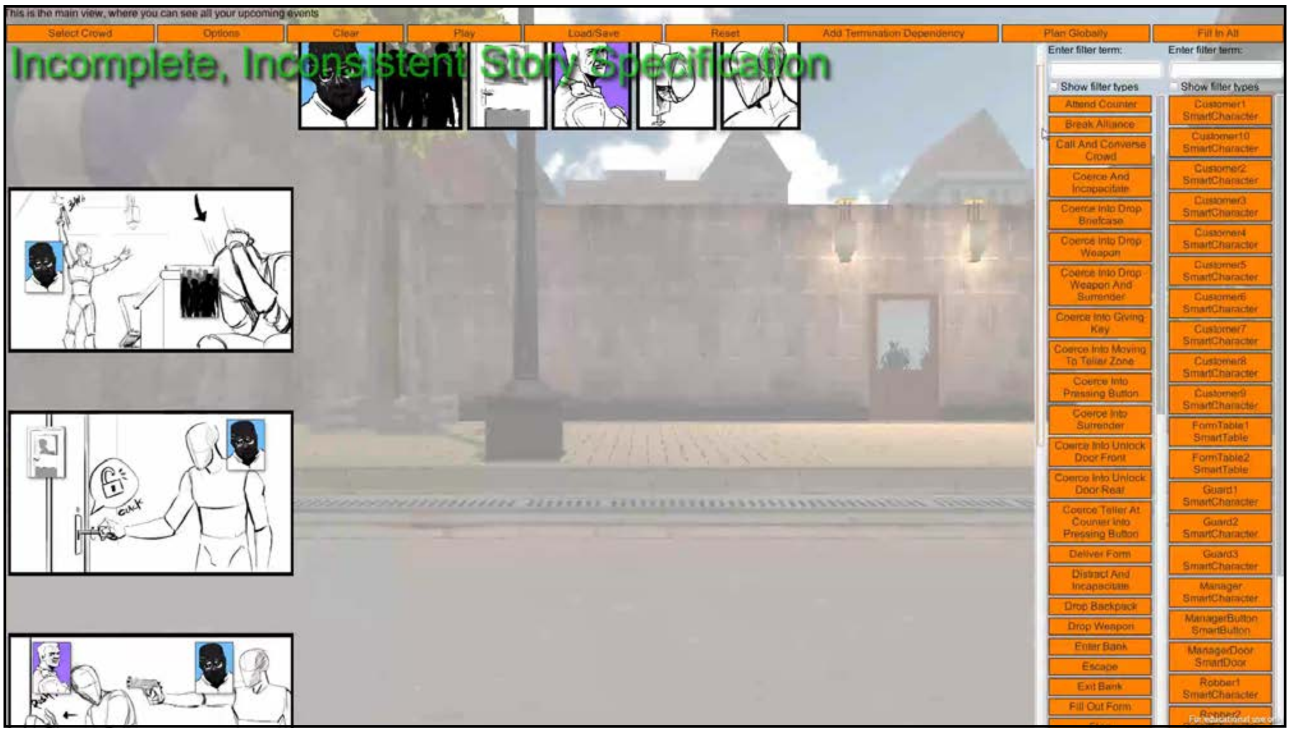

Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION

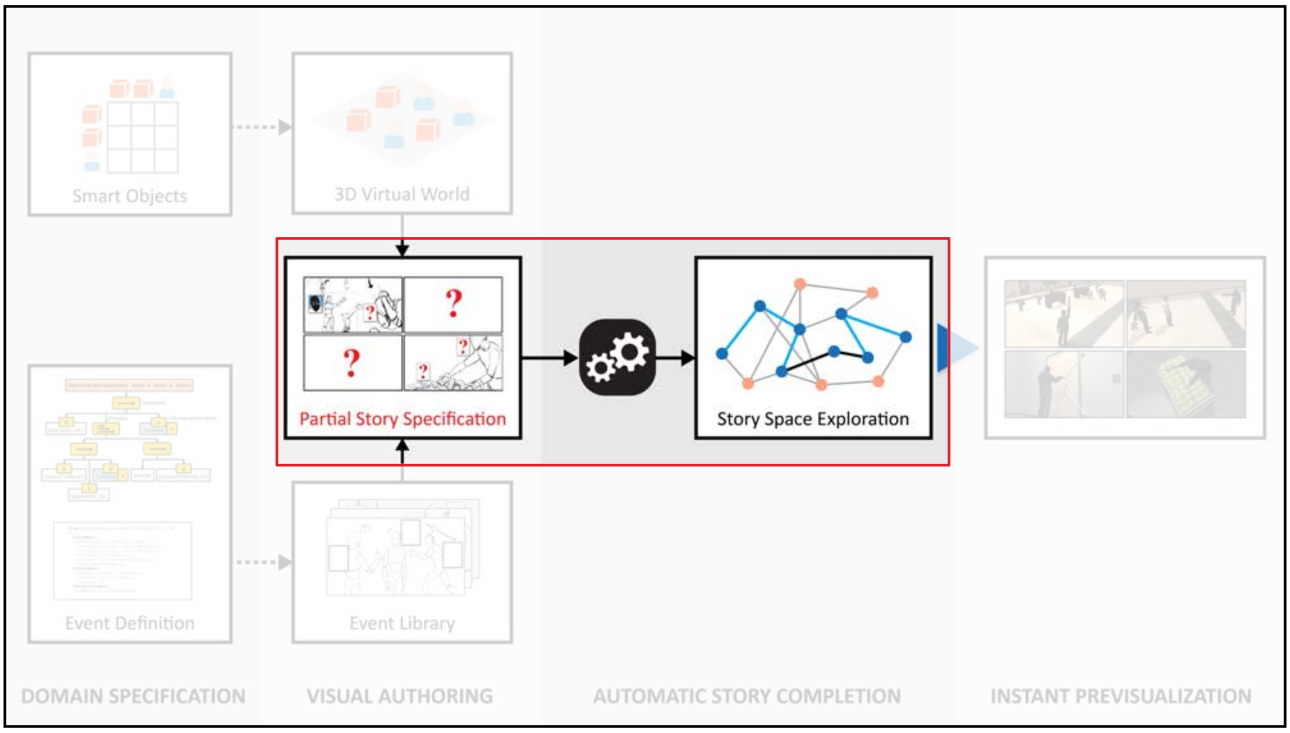


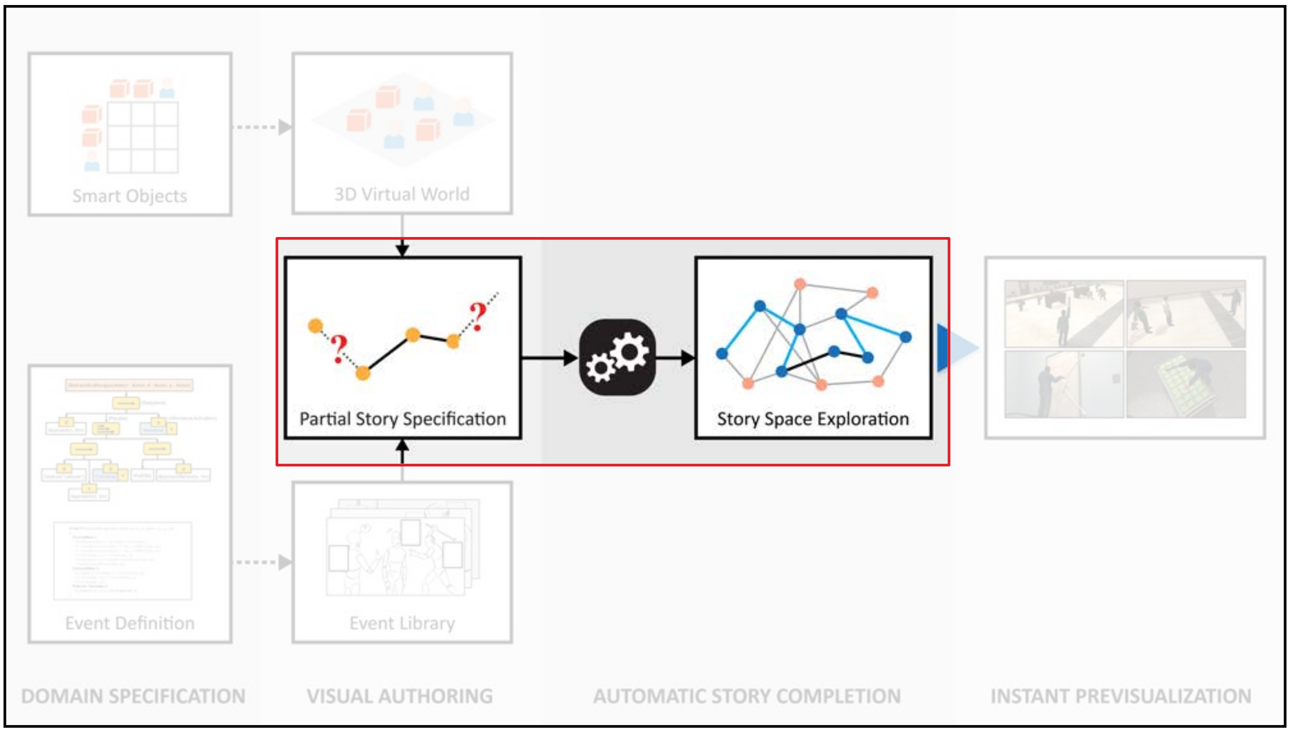
Graphical Authoring:
User specifies key story plot points as visual storyboard elements

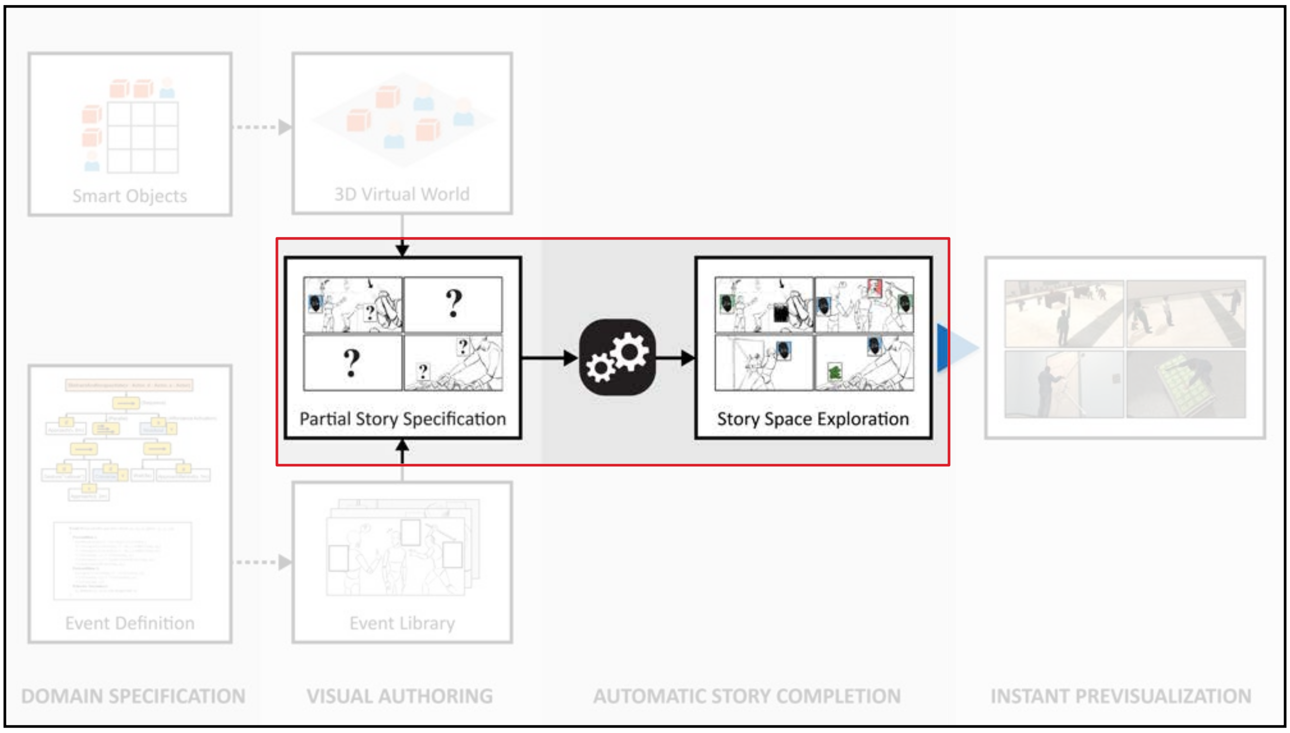

Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION

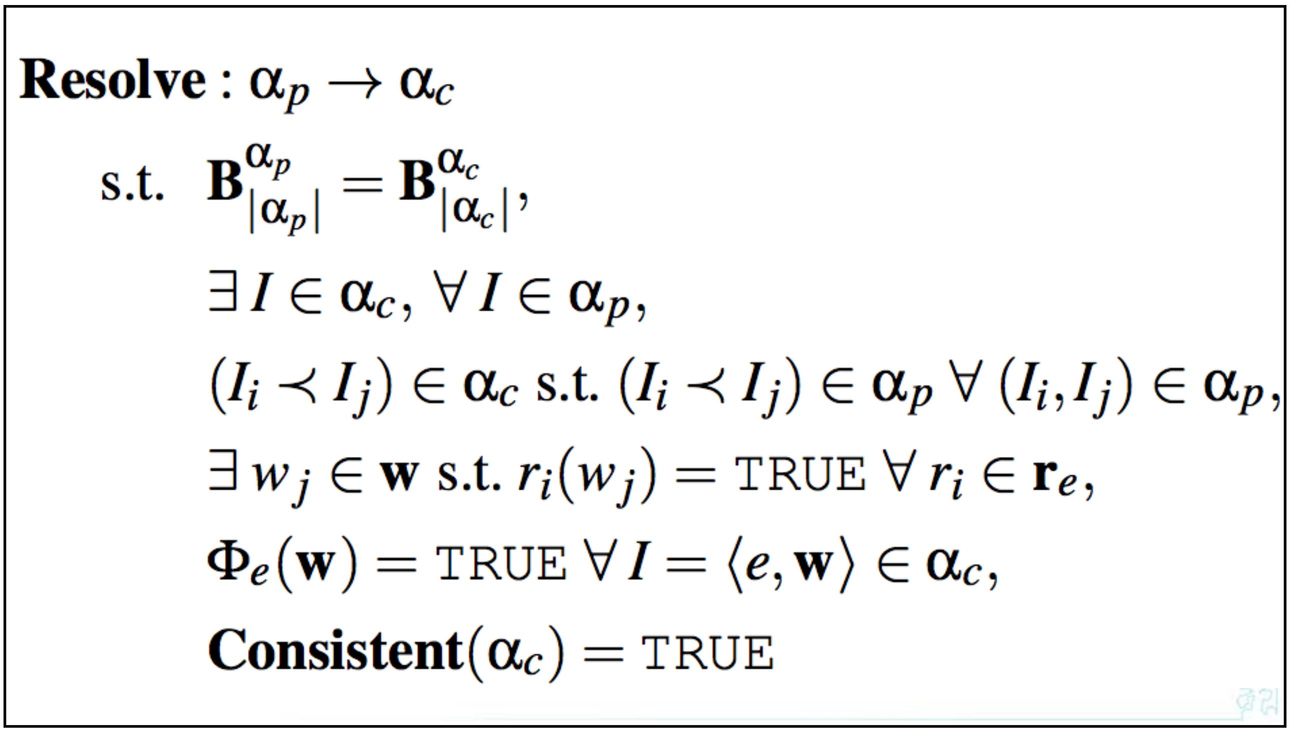

Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION


his is the main view, where you can see all your upcoming events
Select Crowd
Options
Clear
Play
Load/Save
Reset
Add Termination Dependency
Plan Globally
Fill In All
Incomplete, Inconsistent Story Specification
Enter filter term:
Enter filter term:
Show filter types
Show filter types
Attend Counter
Break Alliance
Call And Converse Crowd
Coerce And Incapacitate
Coerce Into Drop Briefcase
Coerce Into Drop Weapon
Coerce Into Drop Weapon And Surrender
Coerce Into Giving Key
Coerce Into Moving To Teller Zone
Coerce Into Pressing Button
Coerce Into Surrender
Coerce Into Unlock Door Front
Coerce Into Unlock Door Rear
Coerce Teller At Counter Into Pressing Button
Deliver Form
Distract And Incapacitate
Drop Backpack
Drop Weapon
Enter Bank
Escape
Exit Bank
Fill Out Form
Customer1 SmartCharacter
Customer10 SmartCharacter
Customer2 SmartCharacter
Customer3 SmartCharacter
Customer4 SmartCharacter
Customer5 SmartCharacter
Customer6 SmartCharacter
Customer7 SmartCharacter
Customer8 SmartCharacter
Customer9 SmartCharacter
FormTable1 SmartTable
FormTable2 SmartTable
Guard1 SmartCharacter
Guard2 SmartCharacter
Guard3 SmartCharacter
Manager SmartCharacter
ManagerButton SmartButton
ManagerDoor SmartDoor
Robber1 SmartCharacter
Robber2
For educational use o

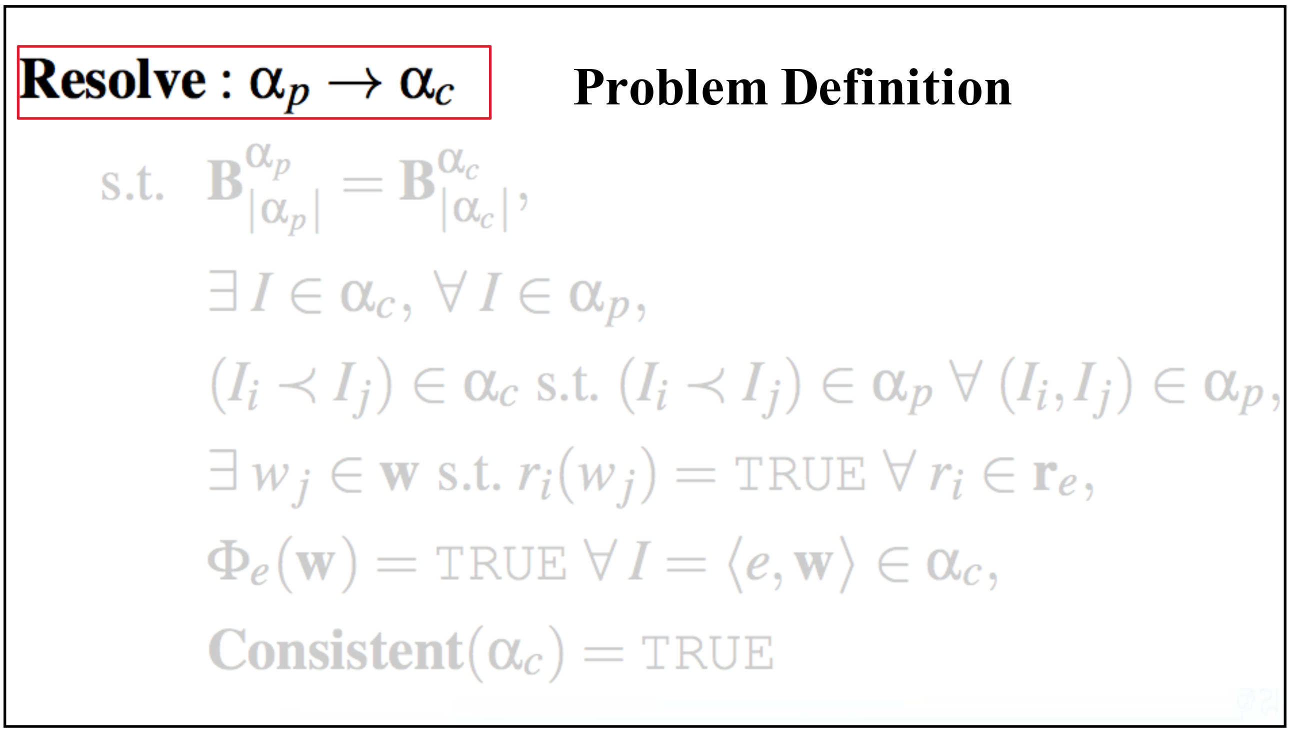

Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION


Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION


Smart Objects
3D Virtual World
Partial Story Specification
Story Space Exploration
Event Definition
Event Library
DOMAIN SPECIFICATION
VISUAL AUTHORING
AUTOMATIC STORY COMPLETION
INSTANT PREVISUALIZATION



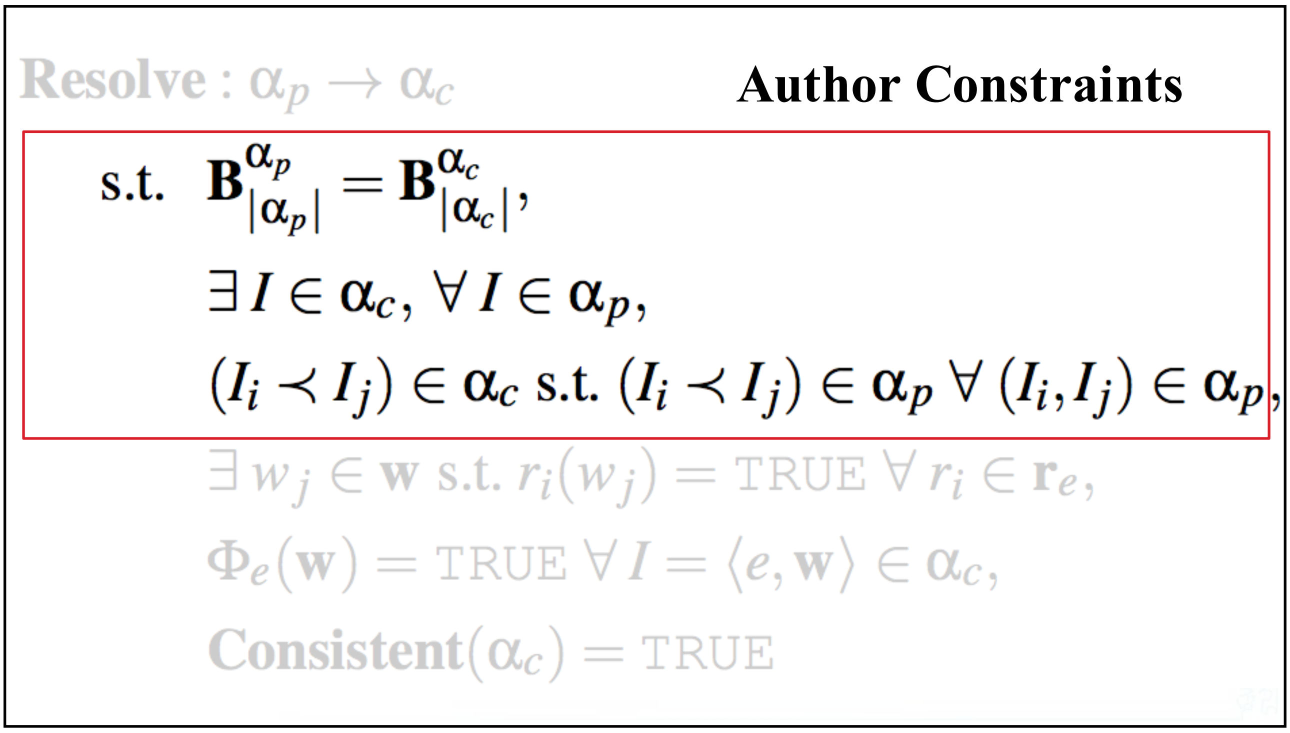
$$\begin{aligned}
&\textbf{Resolve}: \alpha_p \to \alpha_c \\
&\quad \text{s.t.} \quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|}, \\
&\qquad\quad \exists\, I \in \alpha_c,\ \forall\, I \in \alpha_p, \\
&\qquad\quad (I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p\ \forall\, (I_i, I_j) \in \alpha_p, \\
&\qquad\quad \exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \texttt{TRUE}\ \forall\, r_i \in \mathbf{r}_e, \\
&\qquad\quad \Phi_e(\mathbf{w}) = \texttt{TRUE}\ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c, \\
&\qquad\quad \textbf{Consistent}(\alpha_c) = \texttt{TRUE}
\end{aligned}$$



$$\boxed{\textbf{Resolve}: \alpha_p \to \alpha_c}$$

Problem Definition

$$\begin{aligned}
&\quad \text{s.t.} \quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|}, \\
&\qquad\quad \exists\, I \in \alpha_c,\ \forall\, I \in \alpha_p, \\
&\qquad\quad (I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p\ \forall\, (I_i, I_j) \in \alpha_p, \\
&\qquad\quad \exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \texttt{TRUE}\ \forall\, r_i \in \mathbf{r}_e, \\
&\qquad\quad \Phi_e(\mathbf{w}) = \texttt{TRUE}\ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c, \\
&\qquad\quad \textbf{Consistent}(\alpha_c) = \texttt{TRUE}
\end{aligned}$$



Author Constraints

$$\textbf{Resolve}: \alpha_p \rightarrow \alpha_c$$

$$\text{s.t.} \quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|},$$

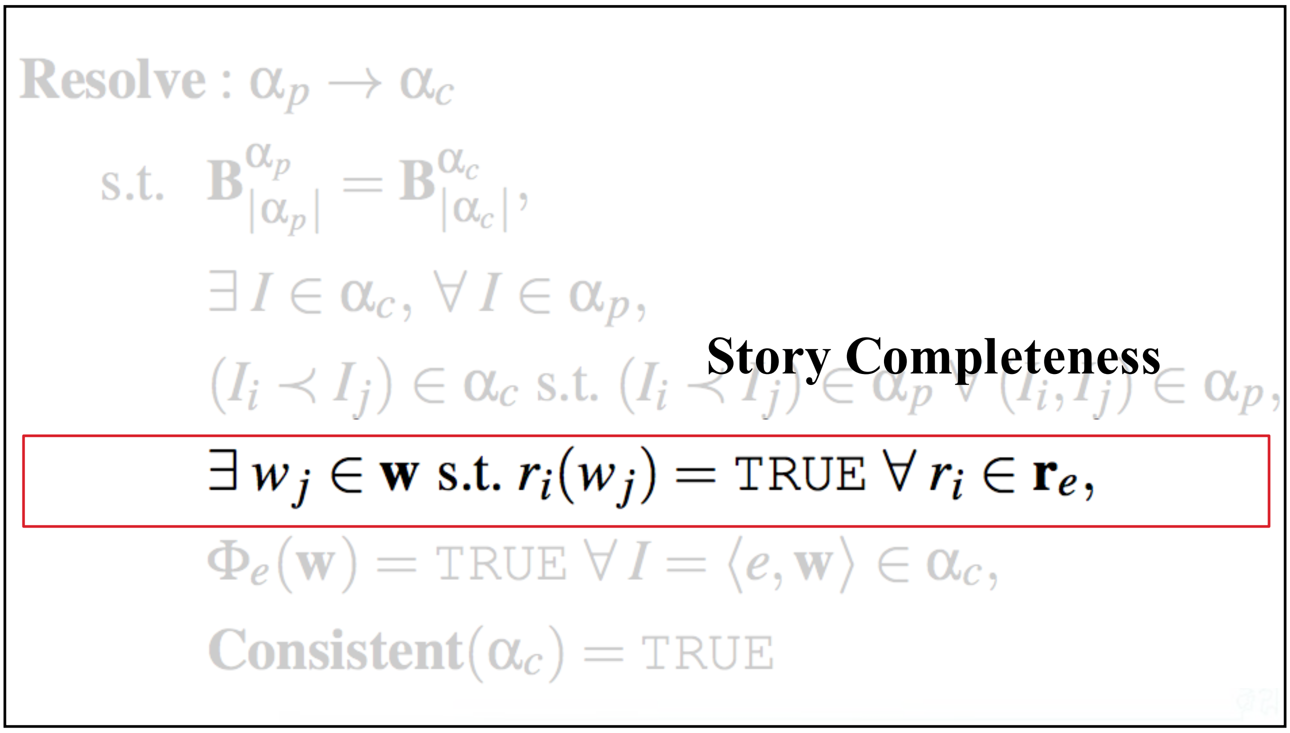
$$\exists\, I \in \alpha_c,\ \forall\, I \in \alpha_p,$$

$$(I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p\ \forall\, (I_i, I_j) \in \alpha_p,$$

$$\exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \text{TRUE}\ \forall\, r_i \in \mathbf{r}_e,$$

$$\Phi_e(\mathbf{w}) = \text{TRUE}\ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c,$$

$$\textbf{Consistent}(\alpha_c) = \text{TRUE}$$



Story Completeness

$$\textbf{Resolve}: \alpha_p \rightarrow \alpha_c$$

$$\text{s.t.} \quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|},$$

$$\exists\, I \in \alpha_c,\ \forall\, I \in \alpha_p,$$

$$(I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p\ \forall\, (I_i, I_j) \in \alpha_p,$$

$$\exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \text{TRUE}\ \forall\, r_i \in \mathbf{r}_e,$$

$$\Phi_e(\mathbf{w}) = \text{TRUE}\ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c,$$

$$\textbf{Consistent}(\alpha_c) = \text{TRUE}$$

$$\begin{aligned}
&\mathbf{Resolve}: \alpha_p \rightarrow \alpha_c\\
&\text{s.t.}\quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|},\\
&\quad \exists\, I \in \alpha_c,\ \forall\, I \in \alpha_p,\\
&\quad (I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p\ \forall\, (I_i, I_j) \in \alpha_p,\\
&\quad \exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \mathtt{TRUE}\ \forall\, r_i \in \mathbf{r}_e,
\end{aligned}$$

Story Consistency

$$\boxed{\begin{aligned}
&\Phi_e(\mathbf{w}) = \mathtt{TRUE}\ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c,\\
&\mathbf{Consistent}(\alpha_c) = \mathtt{TRUE}
\end{aligned}}$$

Terminology

- Parameter Bindings
- Ordering Constraints
- Causal Links
- Threats
- Partial Plan

Terminology

- **Parameter Bindings**
- Ordering Constraints
- Causal Links
- Threats
- Partial Plan

$$\mathcal{P}(I)$$

$$I = \langle e, \mathbf{w} \rangle$$

$$\{w_i \mid w_i \in \mathcal{W} \cup \{\varnothing\}\}$$

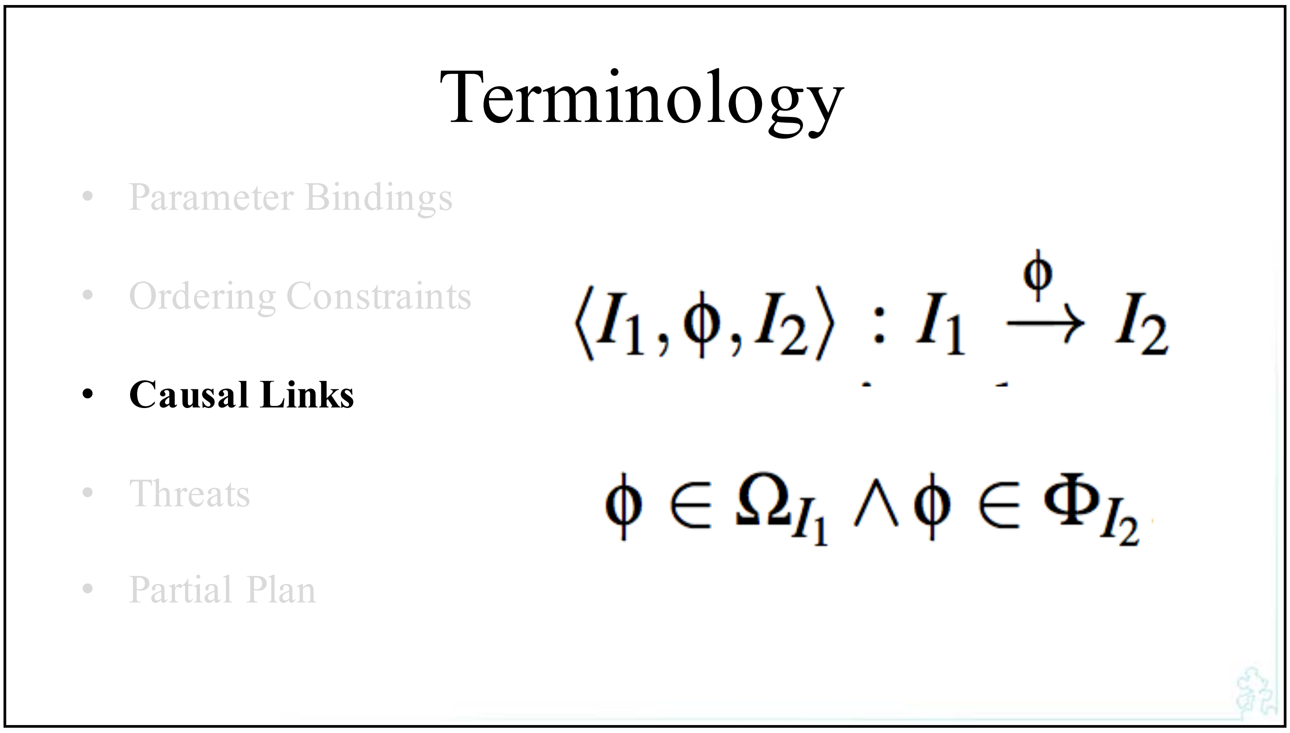
Terminology

- Parameter Bindings
- **Ordering Constraints**
- Causal Links
- Threats
- Partial Plan

$$I_1 \prec I_2$$

$$\Rightarrow I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j,$$

$$\exists\, \mathbf{B}_i, \mathbf{B}_j \in \alpha \text{ s.t. } i < j.$$

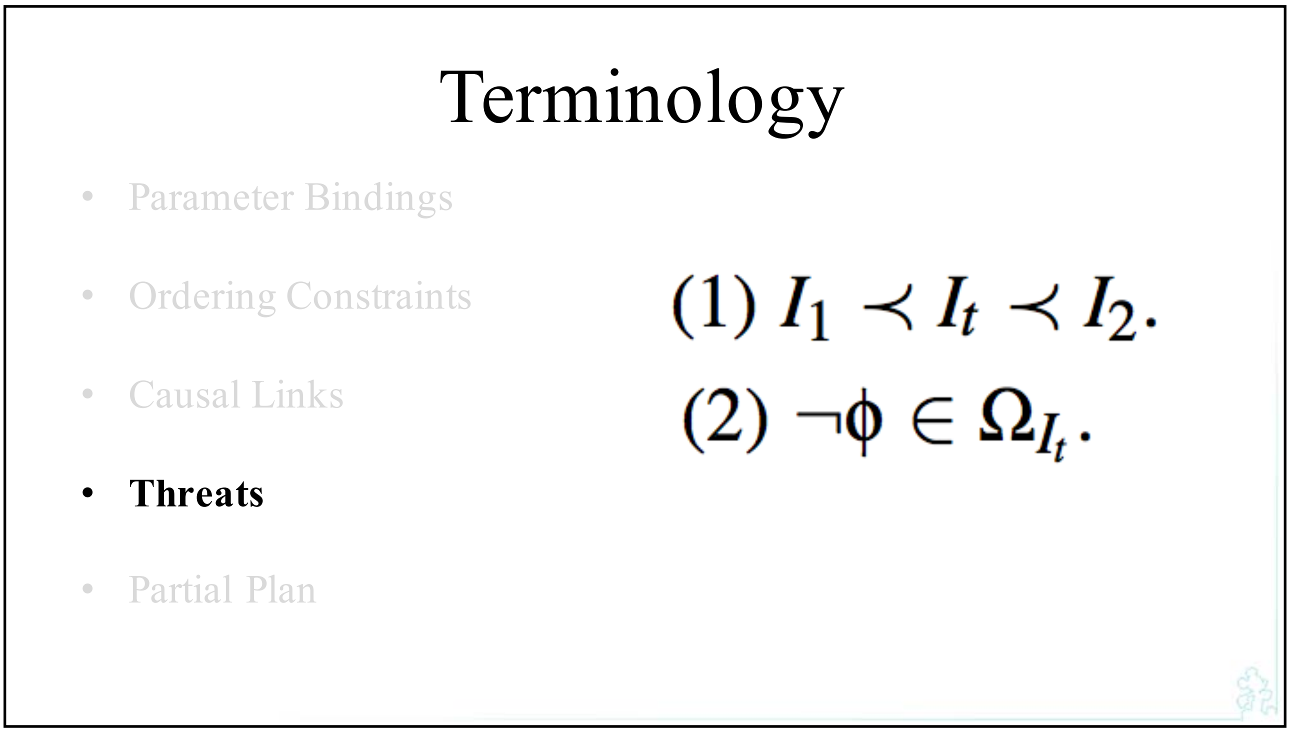


Terminology

- Parameter Bindings
- Ordering Constraints
- **Causal Links**
- Threats
- Partial Plan

$$\langle I_1, \phi, I_2 \rangle : I_1 \xrightarrow{\phi} I_2$$

$$\phi \in \Omega_{I_1} \wedge \phi \in \Phi_{I_2}$$

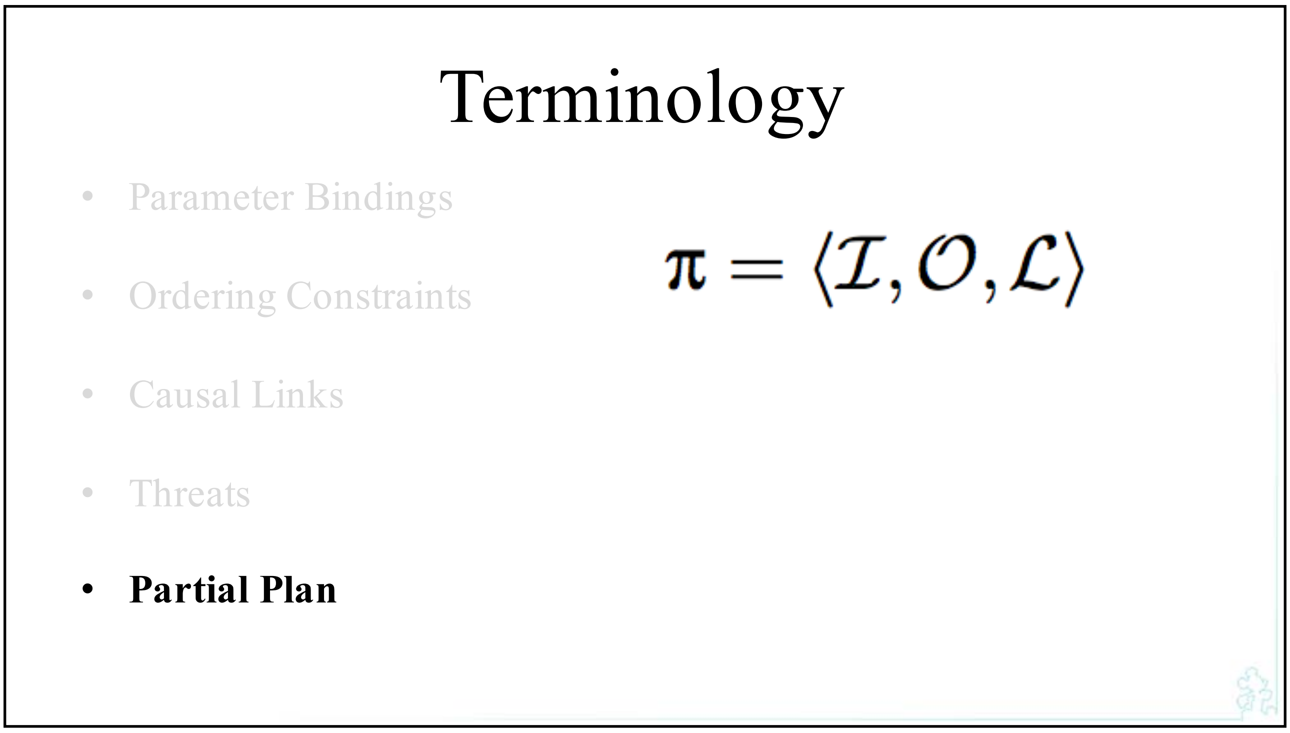


Terminology

- Parameter Bindings
- Ordering Constraints
- Causal Links
- **Threats**
- Partial Plan

$$(1)\ I_1 \prec I_t \prec I_2.$$

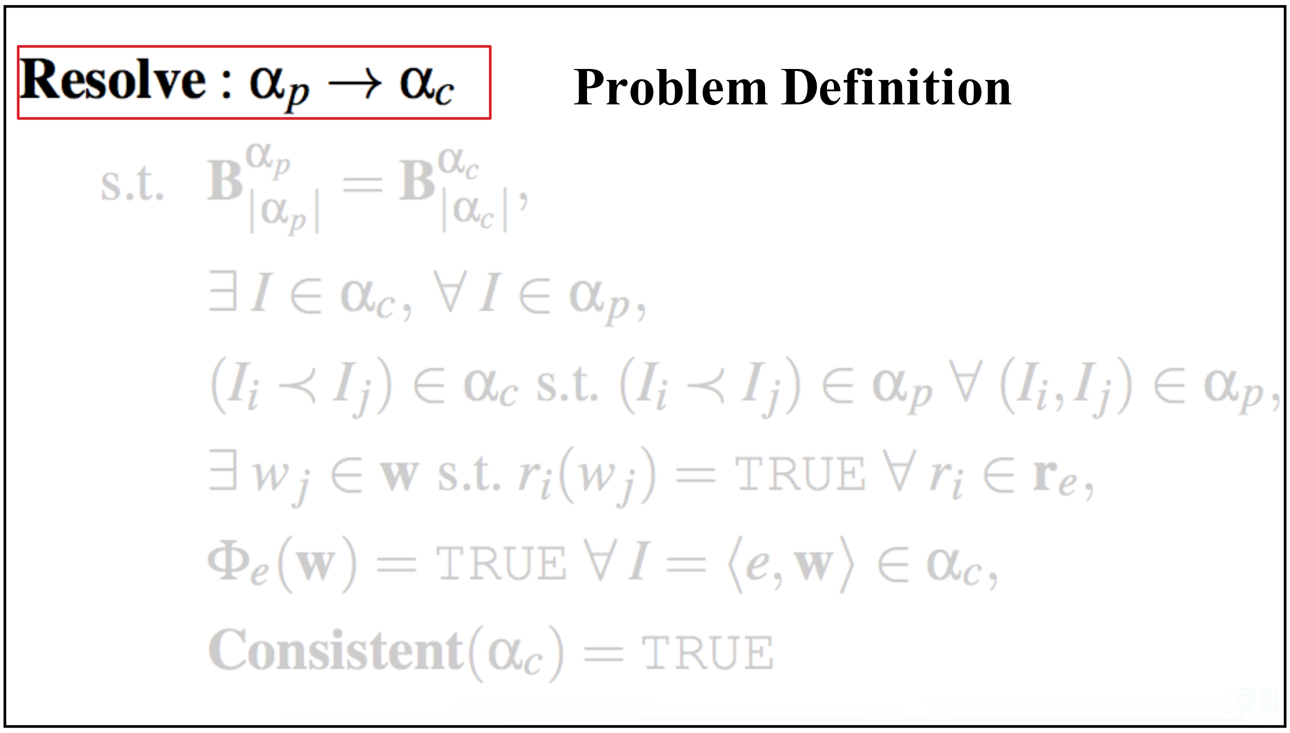
$$(2)\ \neg\phi \in \Omega_{I_t}.$$



Terminology

- Parameter Bindings
- Ordering Constraints
- Causal Links
- Threats
- **Partial Plan**

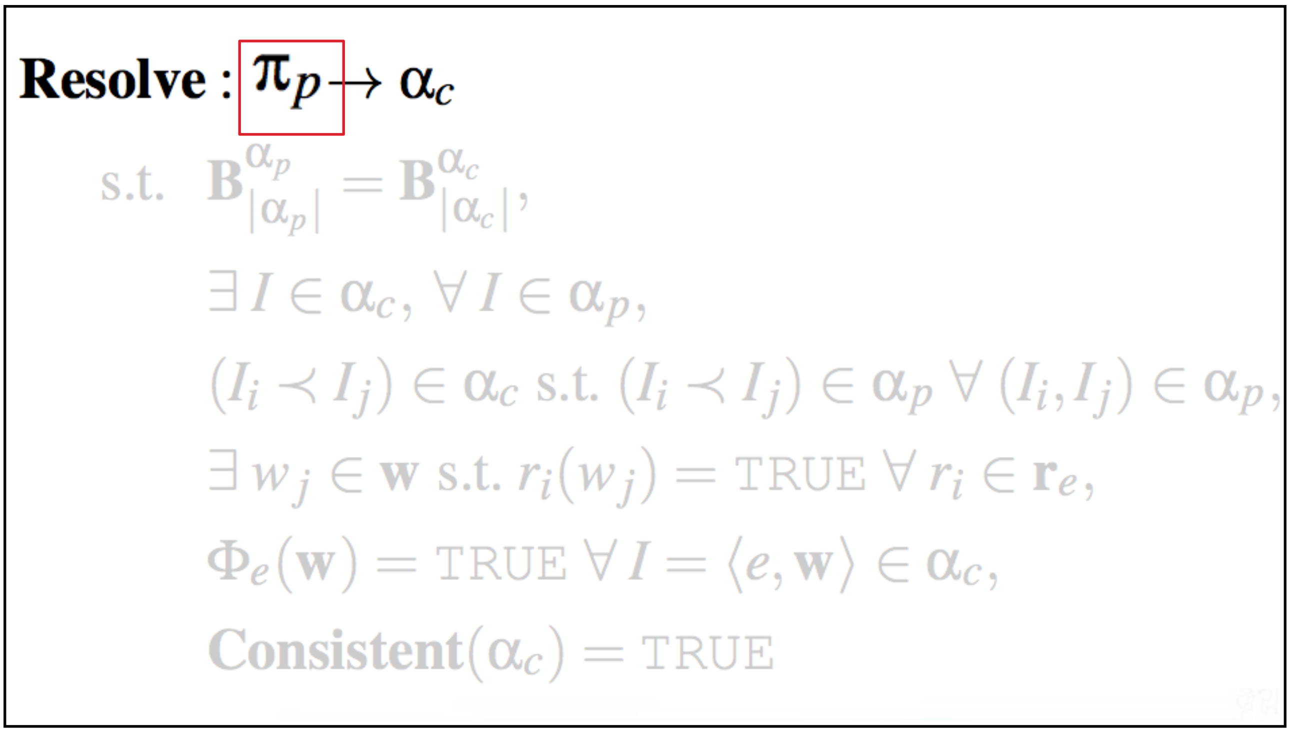
$$\pi = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle$$



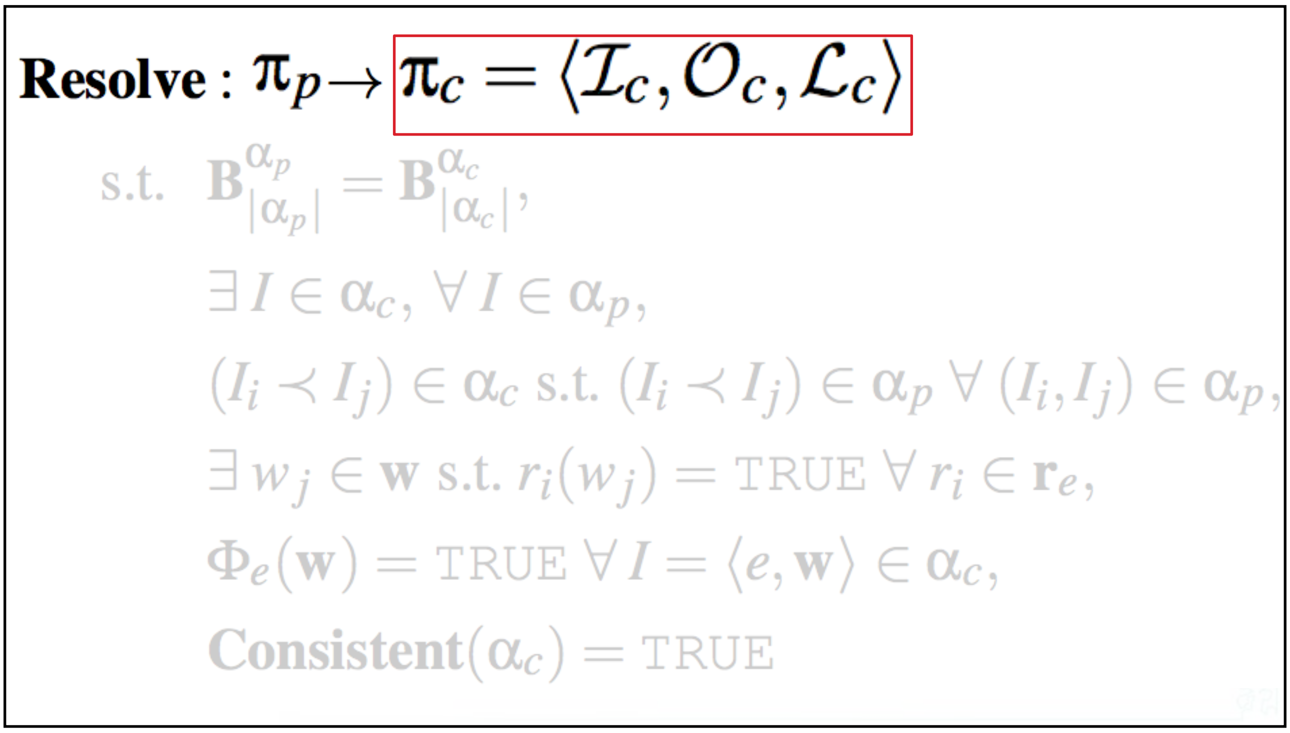
$$\boxed{\textbf{Resolve} : \alpha_p \rightarrow \alpha_c}$$

Problem Definition

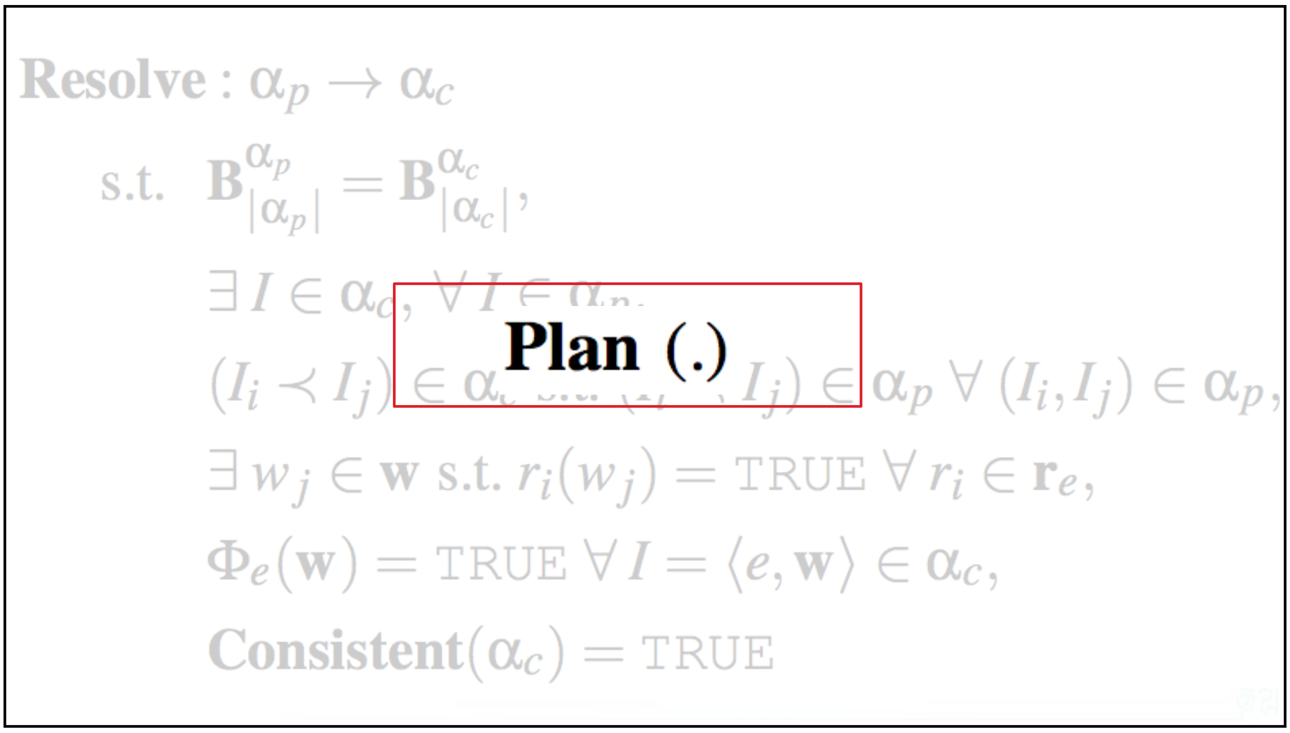
$$\begin{aligned}
\text{s.t.} \quad & \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|}, \\
& \exists\, I \in \alpha_c, \forall\, I \in \alpha_p, \\
& (I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p \,\forall\, (I_i, I_j) \in \alpha_p, \\
& \exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \text{TRUE} \,\forall\, r_i \in \mathbf{r}_e, \\
& \Phi_e(\mathbf{w}) = \text{TRUE} \,\forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c, \\
& \textbf{Consistent}(\alpha_c) = \text{TRUE}
\end{aligned}$$



$$\begin{aligned}
&\textbf{Resolve : } \boxed{\pi_p} \rightarrow \alpha_c \\
&\text{s.t.} \quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|}, \\
&\qquad \exists\, I \in \alpha_c, \forall\, I \in \alpha_p, \\
&\qquad (I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p \ \forall\, (I_i, I_j) \in \alpha_p, \\
&\qquad \exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \mathrm{TRUE} \ \forall\, r_i \in \mathbf{r}_e, \\
&\qquad \Phi_e(\mathbf{w}) = \mathrm{TRUE} \ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c, \\
&\qquad \textbf{Consistent}(\alpha_c) = \mathrm{TRUE}
\end{aligned}$$



$$\begin{aligned}
&\textbf{Resolve : } \pi_p \rightarrow \boxed{\pi_c = \langle \mathcal{I}_c, \mathcal{O}_c, \mathcal{L}_c \rangle} \\
&\text{s.t.} \quad \mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|}, \\
&\qquad \exists\, I \in \alpha_c, \forall\, I \in \alpha_p, \\
&\qquad (I_i \prec I_j) \in \alpha_c \text{ s.t. } (I_i \prec I_j) \in \alpha_p \ \forall\, (I_i, I_j) \in \alpha_p, \\
&\qquad \exists\, w_j \in \mathbf{w} \text{ s.t. } r_i(w_j) = \mathrm{TRUE} \ \forall\, r_i \in \mathbf{r}_e, \\
&\qquad \Phi_e(\mathbf{w}) = \mathrm{TRUE} \ \forall\, I = \langle e, \mathbf{w} \rangle \in \alpha_c, \\
&\qquad \textbf{Consistent}(\alpha_c) = \mathrm{TRUE}
\end{aligned}$$



Resolve : $\alpha_p \rightarrow \alpha_c$

s.t. $\mathbf{B}^{\alpha_p}_{|\alpha_p|} = \mathbf{B}^{\alpha_c}_{|\alpha_c|}$,

$\exists I \in \alpha_c, \forall I \in \alpha_p$

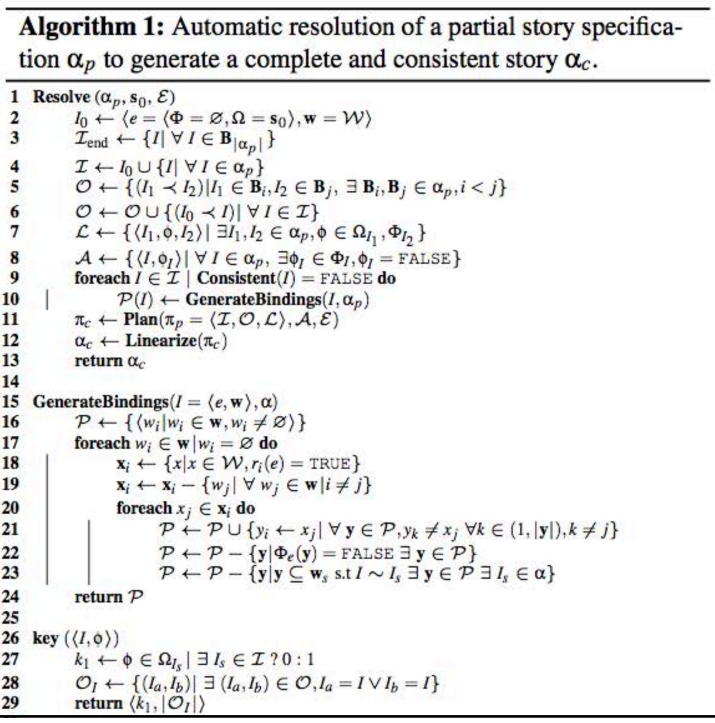
Plan (.)

$(I_i \prec I_j) \in \alpha_c$ $I_j) \in \alpha_p \; \forall \, (I_i, I_j) \in \alpha_p,$

$\exists \, w_j \in \mathbf{w}$ s.t. $r_i(w_j) = \mathrm{TRUE} \; \forall \, r_i \in \mathbf{r}_e,$

$\Phi_e(\mathbf{w}) = \mathrm{TRUE} \; \forall \, I = \langle e, \mathbf{w} \rangle \in \alpha_c,$

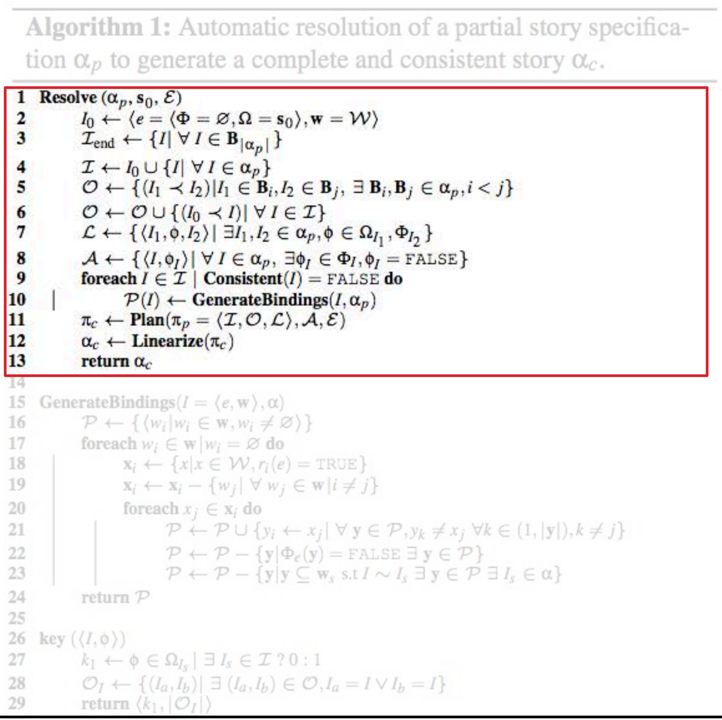
$\mathbf{Consistent}(\alpha_c) = \mathrm{TRUE}$



Algorithm 1: Automatic resolution of a partial story specification α_p to generate a complete and consistent story α_c.

```
Resolve ($\alpha_p, \mathbf{s}_0, \mathcal{E}$)
    $I_0 \leftarrow \langle e = \langle \Phi = \varnothing, \Omega = \mathbf{s}_0 \rangle, \mathbf{w} = \mathcal{W} \rangle$
    $\mathcal{I}_{\text{end}} \leftarrow \{I \mid \forall I \in \mathbf{B}_{|\alpha_p|}\}$
    $\mathcal{I} \leftarrow I_0 \cup \{I \mid \forall I \in \alpha_p\}$
    $\mathcal{O} \leftarrow \{(I_1 \prec I_2) \mid I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j, \exists\, \mathbf{B}_i, \mathbf{B}_j \in \alpha_p, i < j\}$
    $\mathcal{O} \leftarrow \mathcal{O} \cup \{(I_0 \prec I) \mid \forall I \in \mathcal{I}\}$
    $\mathcal{L} \leftarrow \{\langle I_1, \phi, I_2 \rangle \mid \exists I_1, I_2 \in \alpha_p, \phi \in \Omega_{I_1}, \Phi_{I_2}\}$
    $\mathcal{A} \leftarrow \{\langle I, \phi_I \rangle \mid \forall I \in \alpha_p, \exists \phi_I \in \Phi_I, \phi_I = \texttt{FALSE}\}$
    foreach $I \in \mathcal{I}$ | Consistent($I$) = FALSE do
        $\mathcal{P}(I) \leftarrow$ GenerateBindings($I, \alpha_p$)
    $\pi_c \leftarrow$ Plan($\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E}$)
    $\alpha_c \leftarrow$ Linearize($\pi_c$)
    return $\alpha_c$

GenerateBindings($I = \langle e, \mathbf{w} \rangle, \alpha$)
    $\mathcal{P} \leftarrow \{\langle w_i \mid w_i \in \mathbf{w}, w_i \neq \varnothing \rangle\}$
    foreach $w_i \in \mathbf{w} \mid w_i = \varnothing$ do
        $\mathbf{x}_i \leftarrow \{x \mid x \in \mathcal{W}, r_i(e) = \texttt{TRUE}\}$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i - \{w_j \mid \forall w_j \in \mathbf{w} \mid i \neq j\}$
        foreach $x_j \in \mathbf{x}_i$ do
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{y_i \leftarrow x_j \mid \forall \mathbf{y} \in \mathcal{P}, y_k \neq x_j\ \forall k \in (1, |\mathbf{y}|), k \neq j\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \Phi_e(\mathbf{y}) = \texttt{FALSE}\ \exists\, \mathbf{y} \in \mathcal{P}\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \mathbf{y} \subseteq \mathbf{w}_s \text{ s.t } I \sim I_s\ \exists\, \mathbf{y} \in \mathcal{P}\ \exists\, I_s \in \alpha\}$
    return $\mathcal{P}$

key ($\langle I, \phi \rangle$)
    $k_1 \leftarrow \phi \in \Omega_{I_s} \mid \exists\, I_s \in \mathcal{I}\ ?\ 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) \mid \exists\, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$
```

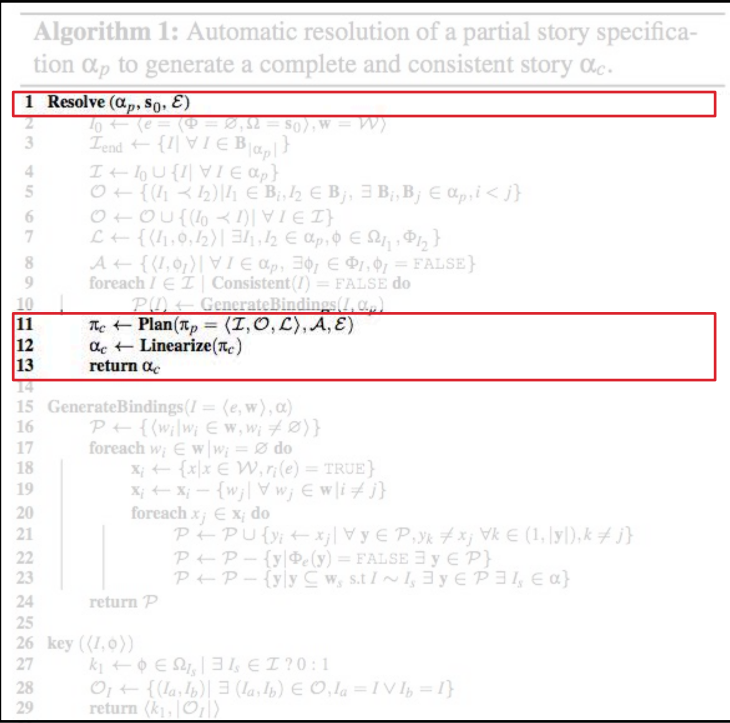
Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



Algorithm 1: Automatic resolution of a partial story specification α_p to generate a complete and consistent story α_c.

```
Resolve ($\alpha_p, \mathbf{s}_0, \mathcal{E}$)
    $I_0 \leftarrow \langle e = \langle \Phi = \varnothing, \Omega = \mathbf{s}_0 \rangle, \mathbf{w} = \mathcal{W} \rangle$
    $\mathcal{I}_{\text{end}} \leftarrow \{I \mid \forall I \in \mathbf{B}_{|\alpha_p|}\}$
    $\mathcal{I} \leftarrow I_0 \cup \{I \mid \forall I \in \alpha_p\}$
    $\mathcal{O} \leftarrow \{(I_1 \prec I_2) \mid I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j, \exists\, \mathbf{B}_i, \mathbf{B}_j \in \alpha_p, i < j\}$
    $\mathcal{O} \leftarrow \mathcal{O} \cup \{(I_0 \prec I) \mid \forall I \in \mathcal{I}\}$
    $\mathcal{L} \leftarrow \{\langle I_1, \phi, I_2 \rangle \mid \exists I_1, I_2 \in \alpha_p, \phi \in \Omega_{I_1}, \Phi_{I_2}\}$
    $\mathcal{A} \leftarrow \{\langle I, \phi_I \rangle \mid \forall I \in \alpha_p, \exists \phi_I \in \Phi_I, \phi_I = \texttt{FALSE}\}$
    foreach $I \in \mathcal{I}$ | Consistent($I$) = FALSE do
        $\mathcal{P}(I) \leftarrow$ GenerateBindings($I, \alpha_p$)
    $\pi_c \leftarrow$ Plan($\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E}$)
    $\alpha_c \leftarrow$ Linearize($\pi_c$)
    return $\alpha_c$

GenerateBindings($I = \langle e, \mathbf{w} \rangle, \alpha$)
    $\mathcal{P} \leftarrow \{\langle w_i \mid w_i \in \mathbf{w}, w_i \neq \varnothing \rangle\}$
    foreach $w_i \in \mathbf{w} \mid w_i = \varnothing$ do
        $\mathbf{x}_i \leftarrow \{x \mid x \in \mathcal{W}, r_i(e) = \texttt{TRUE}\}$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i - \{w_j \mid \forall w_j \in \mathbf{w} \mid i \neq j\}$
        foreach $x_j \in \mathbf{x}_i$ do
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{y_i \leftarrow x_j \mid \forall \mathbf{y} \in \mathcal{P}, y_k \neq x_j\ \forall k \in (1, |\mathbf{y}|), k \neq j\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \Phi_e(\mathbf{y}) = \texttt{FALSE}\ \exists\, \mathbf{y} \in \mathcal{P}\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \mathbf{y} \subseteq \mathbf{w}_s \text{ s.t } I \sim I_s\ \exists\, \mathbf{y} \in \mathcal{P}\ \exists\, I_s \in \alpha\}$
    return $\mathcal{P}$

key ($\langle I, \phi \rangle$)
    $k_1 \leftarrow \phi \in \Omega_{I_s} \mid \exists\, I_s \in \mathcal{I}\ ?\ 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) \mid \exists\, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$
```

Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



Algorithm 1: Automatic resolution of a partial story specification α_p to generate a complete and consistent story α_c.

```
Resolve ($\alpha_p, s_0, \mathcal{E}$)
    $I_0 \leftarrow \langle e = \langle \Phi = \varnothing, \Omega = s_0 \rangle, \mathbf{w} = \mathcal{W} \rangle$
    $\mathcal{I}_{\text{end}} \leftarrow \{I \mid \forall I \in \mathbf{B}_{|\alpha_p|}\}$
    $\mathcal{I} \leftarrow I_0 \cup \{I \mid \forall I \in \alpha_p\}$
    $\mathcal{O} \leftarrow \{(I_1 \prec I_2) \mid I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j, \exists\, \mathbf{B}_i, \mathbf{B}_j \in \alpha_p, i < j\}$
    $\mathcal{O} \leftarrow \mathcal{O} \cup \{(I_0 \prec I) \mid \forall I \in \mathcal{I}\}$
    $\mathcal{L} \leftarrow \{\langle I_1, \phi, I_2 \rangle \mid \exists I_1, I_2 \in \alpha_p, \phi \in \Omega_{I_1}, \Phi_{I_2}\}$
    $\mathcal{A} \leftarrow \{\langle I, \phi_I \rangle \mid \forall I \in \alpha_p,\ \exists \phi_I \in \Phi_I, \phi_I = \text{FALSE}\}$
    foreach $I \in \mathcal{I}$ | Consistent($I$) = FALSE do
        $\mathcal{P}(I) \leftarrow$ GenerateBindings($I, \alpha_p$)
    $\pi_c \leftarrow$ Plan($\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E}$)
    $\alpha_c \leftarrow$ Linearize($\pi_c$)
    return $\alpha_c$

GenerateBindings($I = \langle e, \mathbf{w} \rangle, \alpha$)
    $\mathcal{P} \leftarrow \{\langle w_i \mid w_i \in \mathbf{w}, w_i \neq \varnothing \rangle\}$
    foreach $w_i \in \mathbf{w} \mid w_i = \varnothing$ do
        $\mathbf{x}_i \leftarrow \{x \mid x \in \mathcal{W}, r_i(e) = \text{TRUE}\}$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i - \{w_j \mid \forall w_j \in \mathbf{w} \mid i \neq j\}$
        foreach $x_j \in \mathbf{x}_i$ do
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{y_i \leftarrow x_j \mid \forall \mathbf{y} \in \mathcal{P}, y_k \neq x_j\ \forall k \in (1, |\mathbf{y}|), k \neq j\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \Phi_e(\mathbf{y}) = \text{FALSE}\ \exists\, \mathbf{y} \in \mathcal{P}\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \mathbf{y} \subseteq \mathbf{w}_s$ s.t $I \sim I_s\ \exists\, \mathbf{y} \in \mathcal{P}\ \exists\, I_s \in \alpha\}$
    return $\mathcal{P}$

key ($\langle I, \phi \rangle$)
    $k_1 \leftarrow \phi \in \Omega_{I_s} \mid \exists\, I_s \in \mathcal{I}\ ?\ 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) \mid \exists\, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$
```

Overview

Initialization

Parameter Bindings

Termination Condition

Open Precondition Selection

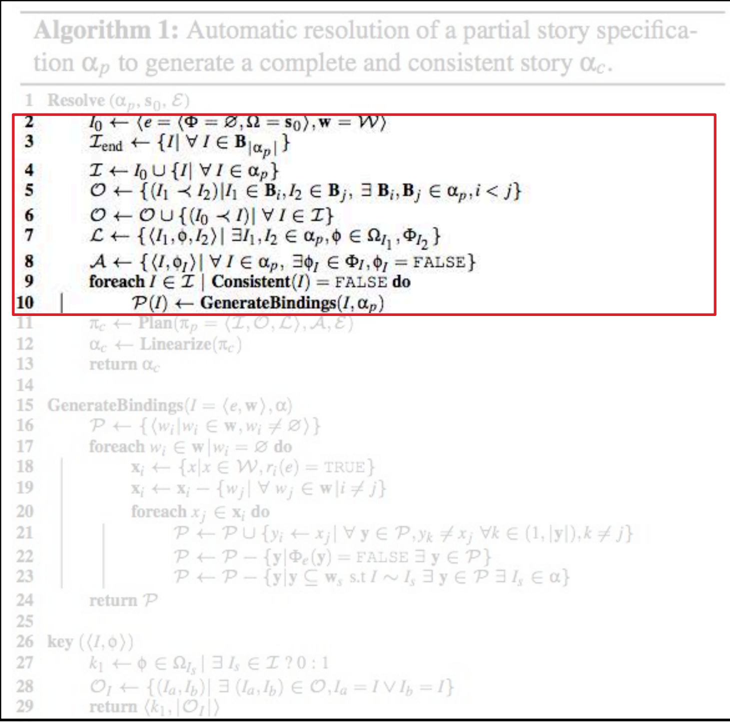
Parameter Binding Selection

Event Instance Selection

Causal Link Protection

Recursive Invocation

Linearization



Algorithm 1: Automatic resolution of a partial story specification α_p to generate a complete and consistent story α_c.

```
Resolve ($\alpha_p, s_0, \mathcal{E}$)
    $I_0 \leftarrow \langle e = \langle \Phi = \varnothing, \Omega = s_0 \rangle, \mathbf{w} = \mathcal{W} \rangle$
    $\mathcal{I}_{\text{end}} \leftarrow \{I \mid \forall I \in \mathbf{B}_{|\alpha_p|}\}$
    $\mathcal{I} \leftarrow I_0 \cup \{I \mid \forall I \in \alpha_p\}$
    $\mathcal{O} \leftarrow \{(I_1 \prec I_2) \mid I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j, \exists\, \mathbf{B}_i, \mathbf{B}_j \in \alpha_p, i < j\}$
    $\mathcal{O} \leftarrow \mathcal{O} \cup \{(I_0 \prec I) \mid \forall I \in \mathcal{I}\}$
    $\mathcal{L} \leftarrow \{\langle I_1, \phi, I_2 \rangle \mid \exists I_1, I_2 \in \alpha_p, \phi \in \Omega_{I_1}, \Phi_{I_2}\}$
    $\mathcal{A} \leftarrow \{\langle I, \phi_I \rangle \mid \forall I \in \alpha_p,\ \exists \phi_I \in \Phi_I, \phi_I = \text{FALSE}\}$
    foreach $I \in \mathcal{I}$ | Consistent($I$) = FALSE do
        $\mathcal{P}(I) \leftarrow$ GenerateBindings($I, \alpha_p$)
    $\pi_c \leftarrow$ Plan($\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E}$)
    $\alpha_c \leftarrow$ Linearize($\pi_c$)
    return $\alpha_c$

GenerateBindings($I = \langle e, \mathbf{w} \rangle, \alpha$)
    $\mathcal{P} \leftarrow \{\langle w_i \mid w_i \in \mathbf{w}, w_i \neq \varnothing \rangle\}$
    foreach $w_i \in \mathbf{w} \mid w_i = \varnothing$ do
        $\mathbf{x}_i \leftarrow \{x \mid x \in \mathcal{W}, r_i(e) = \text{TRUE}\}$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i - \{w_j \mid \forall w_j \in \mathbf{w} \mid i \neq j\}$
        foreach $x_j \in \mathbf{x}_i$ do
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{y_i \leftarrow x_j \mid \forall \mathbf{y} \in \mathcal{P}, y_k \neq x_j\ \forall k \in (1, |\mathbf{y}|), k \neq j\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \Phi_e(\mathbf{y}) = \text{FALSE}\ \exists\, \mathbf{y} \in \mathcal{P}\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \mathbf{y} \subseteq \mathbf{w}_s$ s.t $I \sim I_s\ \exists\, \mathbf{y} \in \mathcal{P}\ \exists\, I_s \in \alpha\}$
    return $\mathcal{P}$

key ($\langle I, \phi \rangle$)
    $k_1 \leftarrow \phi \in \Omega_{I_s} \mid \exists\, I_s \in \mathcal{I}\ ?\ 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) \mid \exists\, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$
```

Overview

Initialization

Parameter Bindings

Termination Condition

Open Precondition Selection

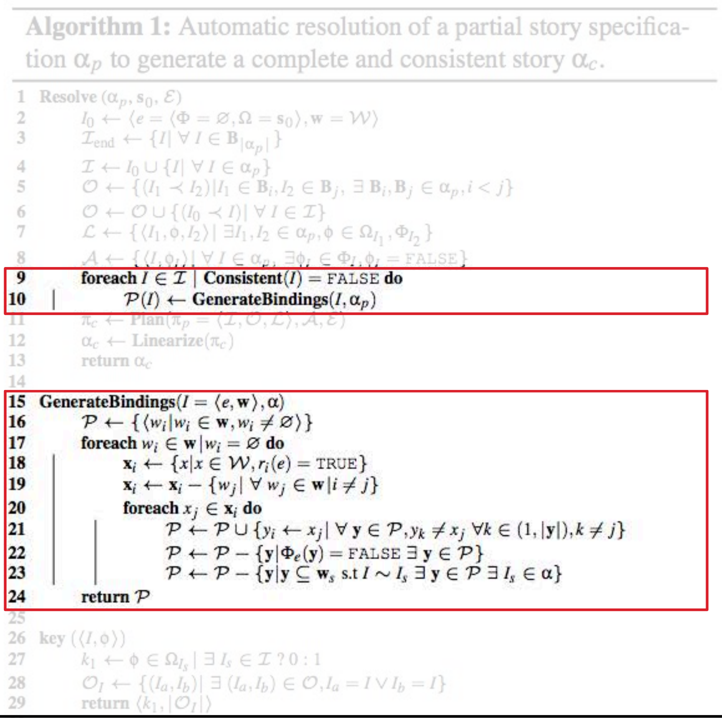
Parameter Binding Selection

Event Instance Selection

Causal Link Protection

Recursive Invocation

Linearization



Algorithm 1: Automatic resolution of a partial story specification α_p to generate a complete and consistent story α_c.

```
Resolve ($\alpha_p$, $\mathbf{s}_0$, $\mathcal{E}$)
    $I_0 \leftarrow \langle e = \langle \Phi = \varnothing, \Omega = \mathbf{s}_0 \rangle, \mathbf{w} = \mathcal{W} \rangle$
    $\mathcal{I}_{\text{end}} \leftarrow \{I \mid \forall I \in \mathbf{B}_{|\alpha_p|}\}$
    $\mathcal{I} \leftarrow I_0 \cup \{I \mid \forall I \in \alpha_p\}$
    $\mathcal{O} \leftarrow \{(I_1 \prec I_2) \mid I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j, \exists \mathbf{B}_i, \mathbf{B}_j \in \alpha_p, i < j\}$
    $\mathcal{O} \leftarrow \mathcal{O} \cup \{(I_0 \prec I) \mid \forall I \in \mathcal{I}\}$
    $\mathcal{L} \leftarrow \{\langle I_1, \phi, I_2 \rangle \mid \exists I_1, I_2 \in \alpha_p, \phi \in \Omega_{I_1}, \Phi_{I_2}\}$
    $\mathcal{A} \leftarrow \{\langle I, \phi_I \rangle \mid \forall I \in \alpha_p, \exists \phi_I \in \Phi_I, \phi_I = \texttt{FALSE}\}$
    foreach $I \in \mathcal{I} \mid$ Consistent$(I) = \texttt{FALSE}$ do
        $\mathcal{P}(I) \leftarrow$ GenerateBindings$(I, \alpha_p)$
    $\pi_c \leftarrow$ Plan$(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
    $\alpha_c \leftarrow$ Linearize$(\pi_c)$
    return $\alpha_c$

GenerateBindings$(I = \langle e, \mathbf{w} \rangle, \alpha)$
    $\mathcal{P} \leftarrow \{\langle w_i \mid w_i \in \mathbf{w}, w_i \neq \varnothing \rangle\}$
    foreach $w_i \in \mathbf{w} \mid w_i = \varnothing$ do
        $\mathbf{x}_i \leftarrow \{x \mid x \in \mathcal{W}, r_i(e) = \texttt{TRUE}\}$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i - \{w_j \mid \forall w_j \in \mathbf{w} \mid i \neq j\}$
        foreach $x_j \in \mathbf{x}_i$ do
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{y_i \leftarrow x_j \mid \forall \mathbf{y} \in \mathcal{P}, y_k \neq x_j \; \forall k \in (1, |\mathbf{y}|), k \neq j\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \Phi_e(\mathbf{y}) = \texttt{FALSE} \; \exists \mathbf{y} \in \mathcal{P}\}$
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \mathbf{y} \subseteq \mathbf{w}_s$ s.t $I \sim I_s \; \exists \mathbf{y} \in \mathcal{P} \; \exists I_s \in \alpha\}$
    return $\mathcal{P}$

key $(\langle I, \phi \rangle)$
    $k_1 \leftarrow \phi \in \Omega_{I_s} \mid \exists I_s \in \mathcal{I} \; ? \; 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) \mid \exists (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$
```

Overview

Initialization

Parameter Bindings

Termination Condition

Open Precondition Selection

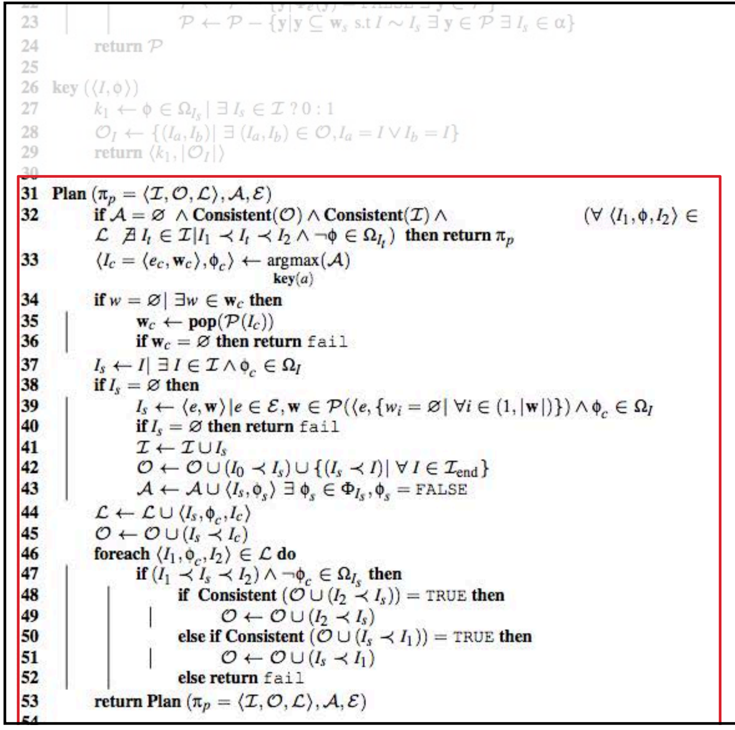
Parameter Binding Selection

Event Instance Selection

Causal Link Protection

Recursive Invocation

Linearization



```
            $\mathcal{P} \leftarrow \mathcal{P} - \{\mathbf{y} \mid \mathbf{y} \subseteq \mathbf{w}_s$ s.t $I \sim I_s \; \exists \mathbf{y} \in \mathcal{P} \; \exists I_s \in \alpha\}$
    return $\mathcal{P}$

key $(\langle I, \phi \rangle)$
    $k_1 \leftarrow \phi \in \Omega_{I_s} \mid \exists I_s \in \mathcal{I} \; ? \; 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) \mid \exists (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$

Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
    if $\mathcal{A} = \varnothing \wedge$ Consistent$(\mathcal{O}) \wedge$ Consistent$(\mathcal{I}) \wedge$ $(\forall \langle I_1, \phi, I_2 \rangle \in$
    $\mathcal{L} \;\; \nexists I_t \in \mathcal{I} \mid I_1 \prec I_t \prec I_2 \wedge \neg\phi \in \Omega_{I_t})$ then return $\pi_p$
    $\langle I_c = \langle e_c, \mathbf{w}_c \rangle, \phi_c \rangle \leftarrow \underset{\text{key}(a)}{\text{argmax}}(\mathcal{A})$
    if $w = \varnothing \mid \exists w \in \mathbf{w}_c$ then
        $\mathbf{w}_c \leftarrow$ pop$(\mathcal{P}(I_c))$
        if $\mathbf{w}_c = \varnothing$ then return fail
    $I_s \leftarrow I \mid \exists I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
    if $I_s = \varnothing$ then
        $I_s \leftarrow \langle e, \mathbf{w} \rangle \mid e \in \mathcal{E}, \mathbf{w} \in \mathcal{P}(\langle e, \{w_i = \varnothing \mid \forall i \in (1, |\mathbf{w}|)\}) \wedge \phi_c \in \Omega_I$
        if $I_s = \varnothing$ then return fail
        $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
        $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) \mid \forall I \in \mathcal{I}_{\text{end}}\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \; \exists \phi_s \in \Phi_{I_s}, \phi_s = \texttt{FALSE}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
    $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
    foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
        if $(I_1 \prec I_s \prec I_2) \wedge \neg\phi_c \in \Omega_{I_s}$ then
            if Consistent $(\mathcal{O} \cup (I_2 \prec I_s)) = \texttt{TRUE}$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
            else if Consistent $(\mathcal{O} \cup (I_s \prec I_1)) = \texttt{TRUE}$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
            else return fail
    return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
```

Overview

Initialization

Parameter Bindings

Termination Condition

Open Precondition Selection

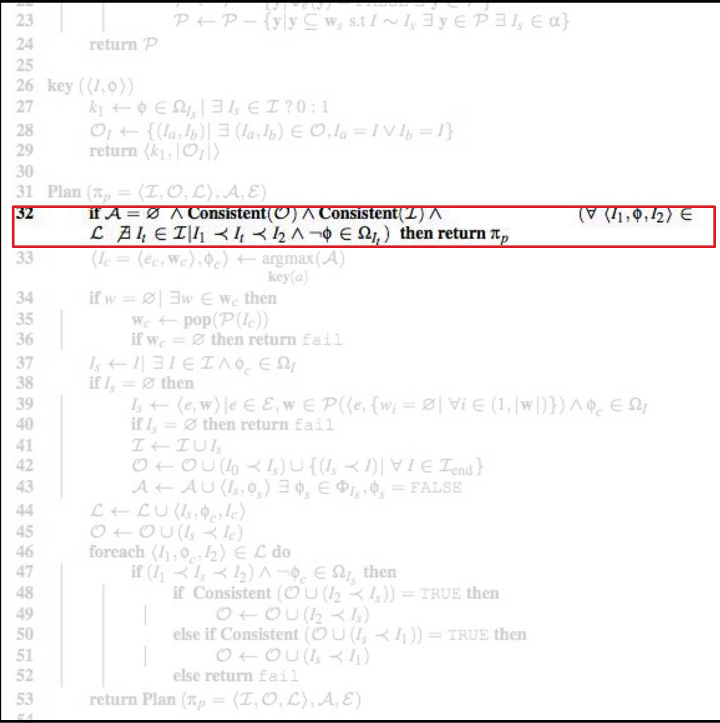
Parameter Binding Selection

Event Instance Selection

Causal Link Protection

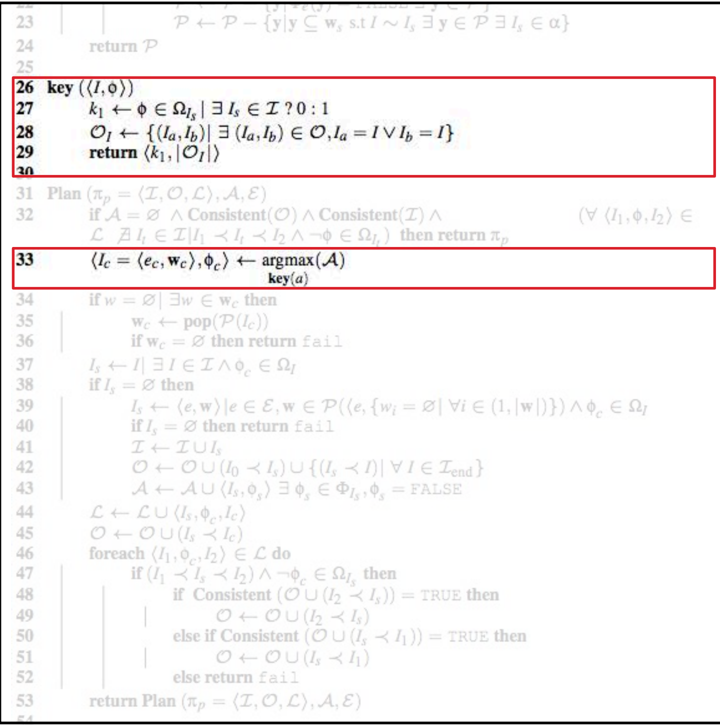
Recursive Invocation

Linearization



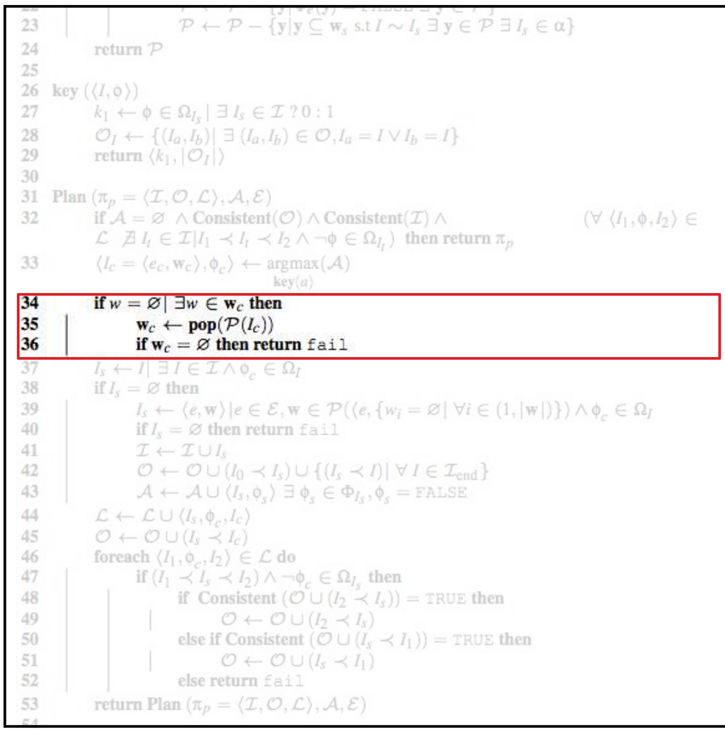
```
23            $\mathcal{P} \leftarrow \mathcal{P} - \{y | y \subseteq w_s \text{ s.t } I \sim I_s \; \exists \, y \in \mathcal{P} \; \exists \, I_s \in \alpha\}$
24      return $\mathcal{P}$
25
26  key $(\langle I, \phi \rangle)$
27      $k_1 \leftarrow \phi \in \Omega_{I_s} | \; \exists \, I_s \in \mathcal{I} \; ? \; 0 : 1$
28      $\mathcal{O}_I \leftarrow \{(I_a, I_b) | \; \exists \, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
29      return $\langle k_1, |\mathcal{O}_I| \rangle$
30
31  Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
32      if $\mathcal{A} = \varnothing \; \wedge$ Consistent$(\mathcal{O}) \wedge$ Consistent$(\mathcal{I}) \wedge$      $(\forall \langle I_1, \phi, I_2 \rangle \in$
        $\mathcal{L} \; \nexists \, I_t \in \mathcal{I} | I_1 \prec I_t \prec I_2 \wedge \neg \phi \in \Omega_{I_t})$ then return $\pi_p$
33      $\langle I_c = \langle e_c, w_c \rangle, \phi_c \rangle \leftarrow \underset{\text{key}(a)}{\text{argmax}}(\mathcal{A})$
34      if $w = \varnothing | \; \exists w \in w_c$ then
35          $w_c \leftarrow$ pop$(\mathcal{P}(I_c))$
36          if $w_c = \varnothing$ then return fail
37      $I_s \leftarrow I | \; \exists \, I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
38      if $I_s = \varnothing$ then
39          $I_s \leftarrow \langle e, w \rangle | e \in \mathcal{E}, w \in \mathcal{P}(\langle e, \{w_i = \varnothing | \; \forall i \in (1, |w|)\} \rangle) \wedge \phi_c \in \Omega_I$
40          if $I_s = \varnothing$ then return fail
41          $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
42          $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) | \; \forall \, I \in \mathcal{I}_{\text{end}}\}$
43          $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \; \exists \, \phi_s \in \Phi_{I_s}, \phi_s = \text{FALSE}$
44      $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
45      $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
46      foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
47          if $(I_1 \prec I_s \prec I_2) \wedge \neg \phi_c \in \Omega_{I_s}$ then
48              if Consistent $(\mathcal{O} \cup (I_2 \prec I_s)) = \text{TRUE}$ then
49                  $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
50              else if Consistent $(\mathcal{O} \cup (I_s \prec I_1)) = \text{TRUE}$ then
51                  $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
52              else return fail
53      return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
```

Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



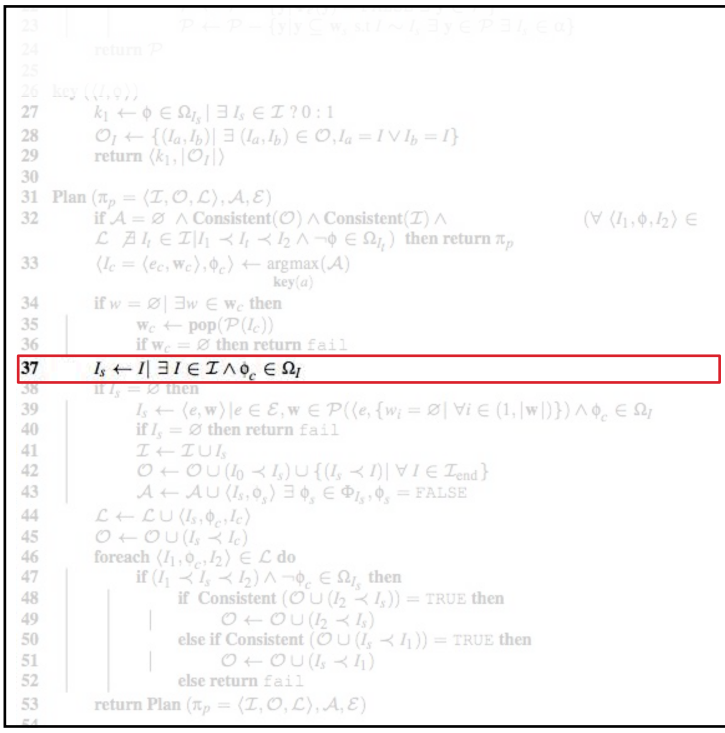
```
23            $\mathcal{P} \leftarrow \mathcal{P} - \{y | y \subseteq w_s \text{ s.t } I \sim I_s \; \exists \, y \in \mathcal{P} \; \exists \, I_s \in \alpha\}$
24      return $\mathcal{P}$
25
26  key $(\langle I, \phi \rangle)$
27      $k_1 \leftarrow \phi \in \Omega_{I_s} | \; \exists \, I_s \in \mathcal{I} \; ? \; 0 : 1$
28      $\mathcal{O}_I \leftarrow \{(I_a, I_b) | \; \exists \, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
29      return $\langle k_1, |\mathcal{O}_I| \rangle$
30
31  Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
32      if $\mathcal{A} = \varnothing \; \wedge$ Consistent$(\mathcal{O}) \wedge$ Consistent$(\mathcal{I}) \wedge$      $(\forall \langle I_1, \phi, I_2 \rangle \in$
        $\mathcal{L} \; \nexists \, I_t \in \mathcal{I} | I_1 \prec I_t \prec I_2 \wedge \neg \phi \in \Omega_{I_t})$ then return $\pi_p$
33      $\langle I_c = \langle e_c, w_c \rangle, \phi_c \rangle \leftarrow \underset{\text{key}(a)}{\text{argmax}}(\mathcal{A})$
34      if $w = \varnothing | \; \exists w \in w_c$ then
35          $w_c \leftarrow$ pop$(\mathcal{P}(I_c))$
36          if $w_c = \varnothing$ then return fail
37      $I_s \leftarrow I | \; \exists \, I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
38      if $I_s = \varnothing$ then
39          $I_s \leftarrow \langle e, w \rangle | e \in \mathcal{E}, w \in \mathcal{P}(\langle e, \{w_i = \varnothing | \; \forall i \in (1, |w|)\} \rangle) \wedge \phi_c \in \Omega_I$
40          if $I_s = \varnothing$ then return fail
41          $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
42          $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) | \; \forall \, I \in \mathcal{I}_{\text{end}}\}$
43          $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \; \exists \, \phi_s \in \Phi_{I_s}, \phi_s = \text{FALSE}$
44      $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
45      $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
46      foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
47          if $(I_1 \prec I_s \prec I_2) \wedge \neg \phi_c \in \Omega_{I_s}$ then
48              if Consistent $(\mathcal{O} \cup (I_2 \prec I_s)) = \text{TRUE}$ then
49                  $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
50              else if Consistent $(\mathcal{O} \cup (I_s \prec I_1)) = \text{TRUE}$ then
51                  $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
52              else return fail
53      return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
```

Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



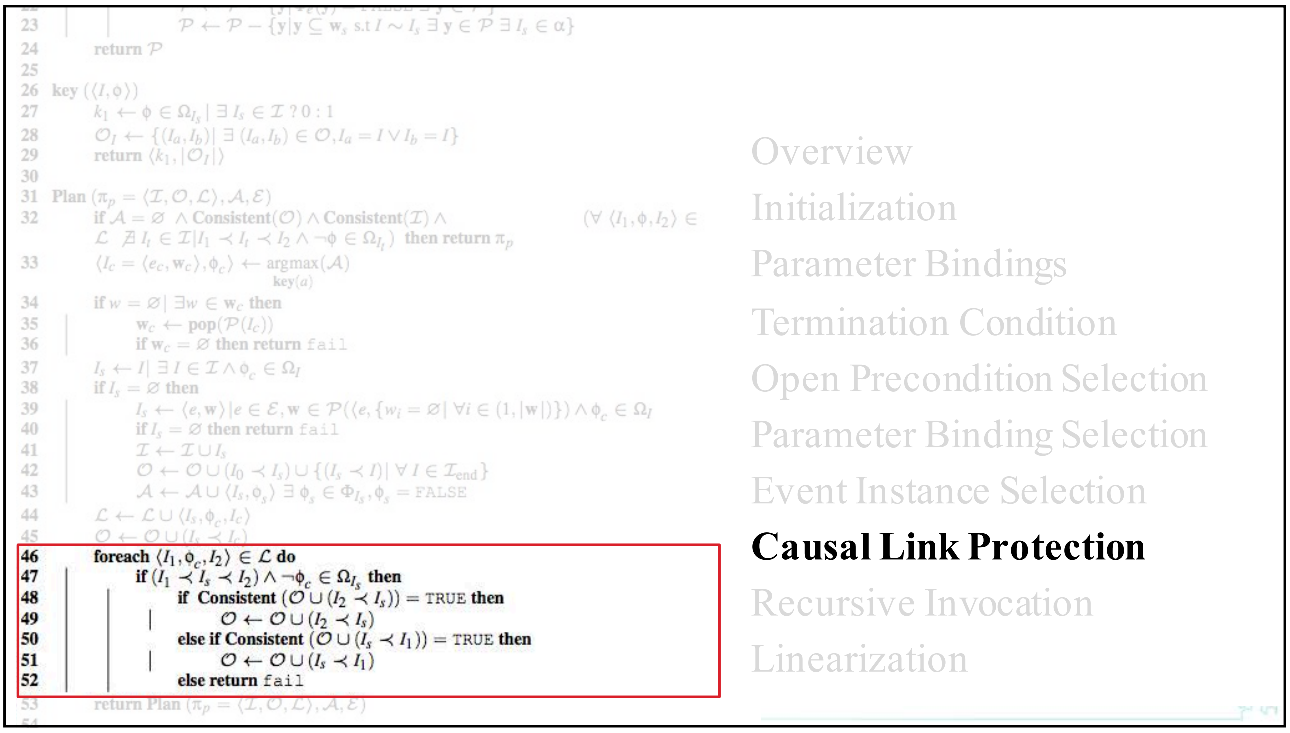
```
        $\mathcal{P} \leftarrow \mathcal{P} - \{y | y \subseteq w_s \text{ s.t } I \sim I_s \; \exists \, y \in \mathcal{P} \; \exists \, I_s \in \alpha\}$
    return $\mathcal{P}$

key $(\langle I, \phi \rangle)$
    $k_1 \leftarrow \phi \in \Omega_{I_s} \,|\, \exists \, I_s \in \mathcal{I} \, ? \, 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) | \; \exists \, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$

Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
    if $\mathcal{A} = \varnothing \; \wedge$ Consistent$(\mathcal{O}) \wedge$ Consistent$(\mathcal{I}) \wedge$ $(\forall \, \langle I_1, \phi, I_2 \rangle \in$
    $\mathcal{L} \;\; \nexists \, I_t \in \mathcal{I} | I_1 \prec I_t \prec I_2 \wedge \neg\phi \in \Omega_{I_t})$ then return $\pi_p$
    $\langle I_c = \langle e_c, w_c \rangle, \phi_c \rangle \leftarrow \underset{key(a)}{\text{argmax}}(\mathcal{A})$
    if $w = \varnothing | \; \exists w \in w_c$ then
        $w_c \leftarrow$ pop$(\mathcal{P}(I_c))$
        if $w_c = \varnothing$ then return fail
    $I_s \leftarrow I | \; \exists \, I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
    if $I_s = \varnothing$ then
        $I_s \leftarrow \langle e, w \rangle | e \in \mathcal{E}, w \in \mathcal{P}(\langle e, \{w_i = \varnothing | \; \forall i \in (1, |w|)\} \rangle) \wedge \phi_c \in \Omega_I$
        if $I_s = \varnothing$ then return fail
        $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
        $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) | \; \forall \, I \in \mathcal{I}_{end}\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \; \exists \, \phi_s \in \Phi_{I_s}, \phi_s =$ FALSE
    $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
    $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
    foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
        if $(I_1 \prec I_s \prec I_2) \wedge \neg\phi_c \in \Omega_{I_s}$ then
            if Consistent $(\mathcal{O} \cup (I_2 \prec I_s))$ = TRUE then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
            else if Consistent $(\mathcal{O} \cup (I_s \prec I_1))$ = TRUE then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
            else return fail
    return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
```

Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



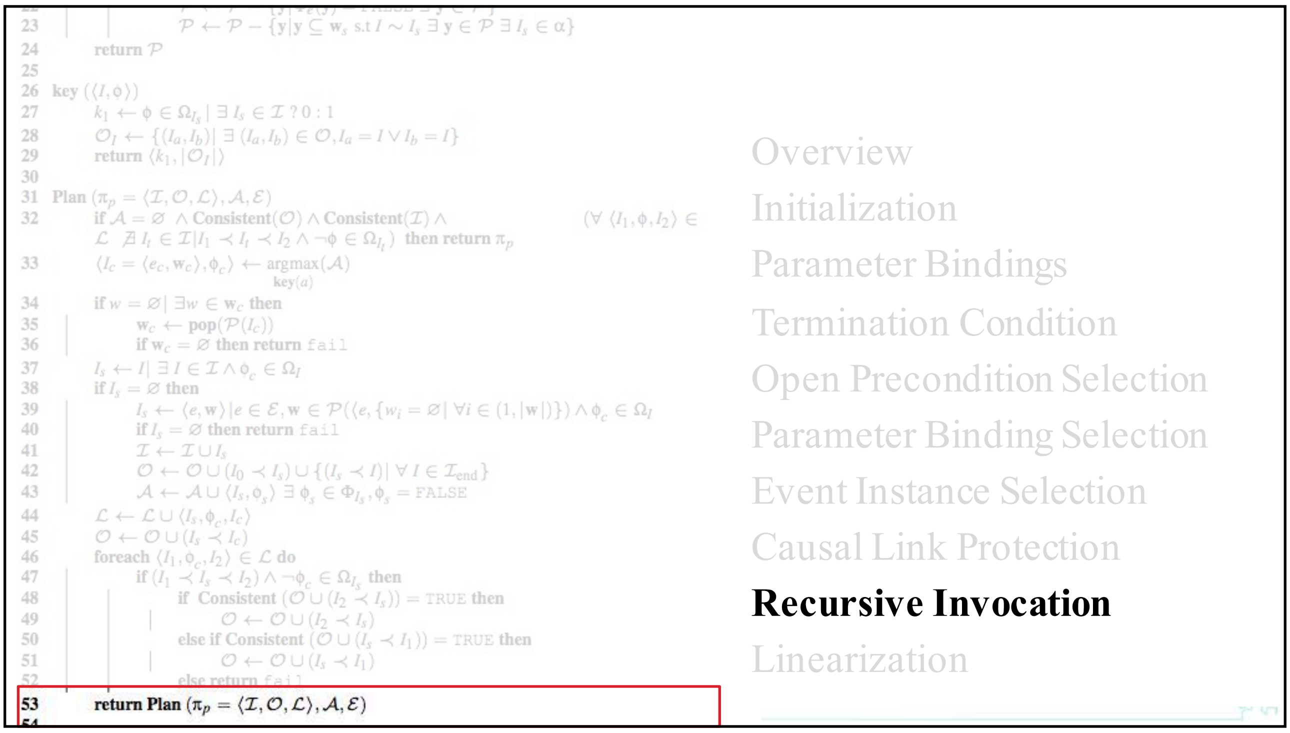
```
        $\mathcal{P} \leftarrow \mathcal{P} - \{y | y \subseteq w_s \text{ s.t } I \sim I_s \; \exists \, y \in \mathcal{P} \; \exists \, I_s \in \alpha\}$
    return $\mathcal{P}$

key $(\langle I, \phi \rangle)$
    $k_1 \leftarrow \phi \in \Omega_{I_s} \,|\, \exists \, I_s \in \mathcal{I} \, ? \, 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) | \; \exists \, (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$

Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
    if $\mathcal{A} = \varnothing \; \wedge$ Consistent$(\mathcal{O}) \wedge$ Consistent$(\mathcal{I}) \wedge$ $(\forall \, \langle I_1, \phi, I_2 \rangle \in$
    $\mathcal{L} \;\; \nexists \, I_t \in \mathcal{I} | I_1 \prec I_t \prec I_2 \wedge \neg\phi \in \Omega_{I_t})$ then return $\pi_p$
    $\langle I_c = \langle e_c, w_c \rangle, \phi_c \rangle \leftarrow \underset{key(a)}{\text{argmax}}(\mathcal{A})$
    if $w = \varnothing | \; \exists w \in w_c$ then
        $w_c \leftarrow$ pop$(\mathcal{P}(I_c))$
        if $w_c = \varnothing$ then return fail
    $I_s \leftarrow I | \; \exists \, I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
    if $I_s = \varnothing$ then
        $I_s \leftarrow \langle e, w \rangle | e \in \mathcal{E}, w \in \mathcal{P}(\langle e, \{w_i = \varnothing | \; \forall i \in (1, |w|)\} \rangle) \wedge \phi_c \in \Omega_I$
        if $I_s = \varnothing$ then return fail
        $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
        $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) | \; \forall \, I \in \mathcal{I}_{end}\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \; \exists \, \phi_s \in \Phi_{I_s}, \phi_s =$ FALSE
    $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
    $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
    foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
        if $(I_1 \prec I_s \prec I_2) \wedge \neg\phi_c \in \Omega_{I_s}$ then
            if Consistent $(\mathcal{O} \cup (I_2 \prec I_s))$ = TRUE then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
            else if Consistent $(\mathcal{O} \cup (I_s \prec I_1))$ = TRUE then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
            else return fail
    return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
```

Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



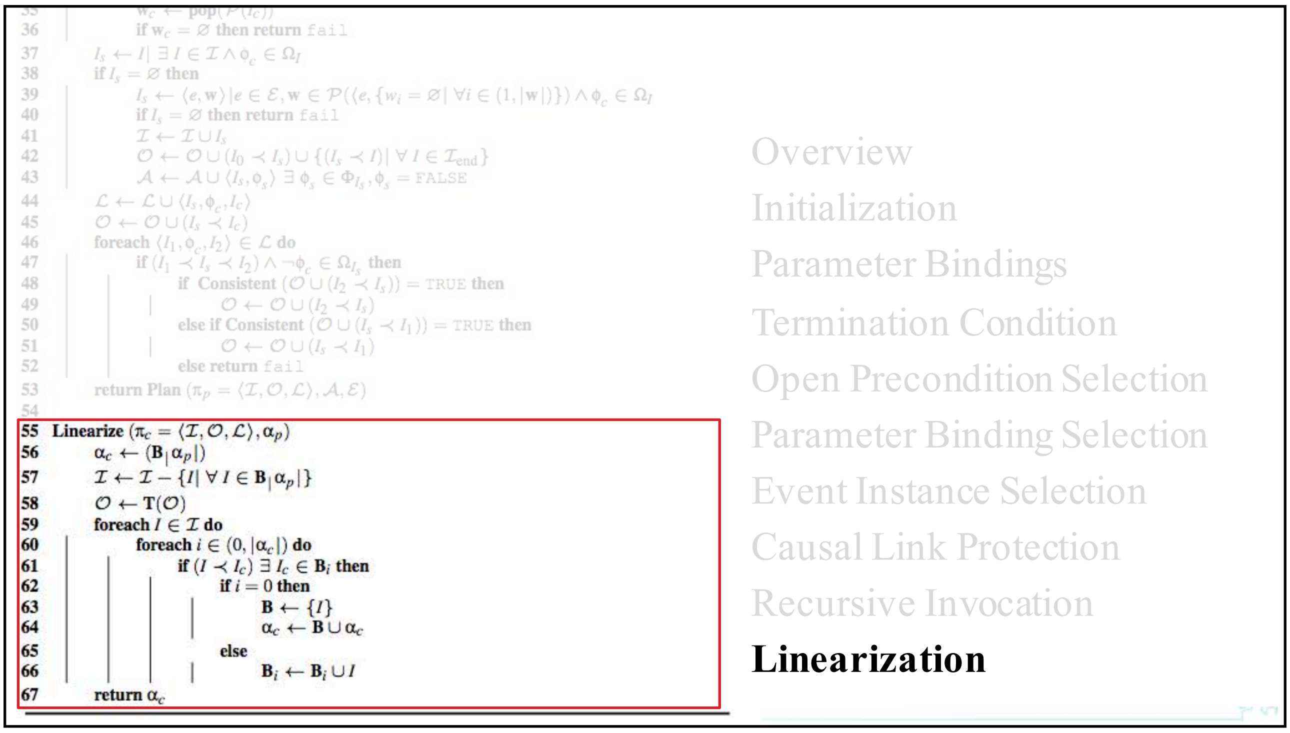
```
        $\mathcal{P} \leftarrow \mathcal{P} - \{y | y \subseteq w_s \text{ s.t } I \sim I_s \exists y \in \mathcal{P} \exists I_s \in \alpha\}$
    return $\mathcal{P}$

key $(\langle I, \phi \rangle)$
    $k_1 \leftarrow \phi \in \Omega_{I_s} | \exists I_s \in \mathcal{I} ? 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) | \exists (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$

Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
    if $\mathcal{A} = \varnothing \wedge$ Consistent$(\mathcal{O}) \wedge$ Consistent$(\mathcal{I}) \wedge$ $(\forall \langle I_1, \phi, I_2 \rangle \in \mathcal{L} \;\nexists I_t \in \mathcal{I} | I_1 \prec I_t \prec I_2 \wedge \neg\phi \in \Omega_{I_t})$ then return $\pi_p$
    $\langle I_c = \langle e_c, w_c \rangle, \phi_c \rangle \leftarrow \underset{key(a)}{\operatorname{argmax}}(\mathcal{A})$
    if $w = \varnothing | \exists w \in w_c$ then
        $w_c \leftarrow$ pop$(\mathcal{P}(I_c))$
        if $w_c = \varnothing$ then return fail
    $I_s \leftarrow I | \exists I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
    if $I_s = \varnothing$ then
        $I_s \leftarrow \langle e, w \rangle | e \in \mathcal{E}, w \in \mathcal{P}(\langle e, \{w_i = \varnothing | \forall i \in (1, |w|)\}) \wedge \phi_c \in \Omega_I$
        if $I_s = \varnothing$ then return fail
        $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
        $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) | \forall I \in \mathcal{I}_{end}\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \exists \phi_s \in \Phi_{I_s}, \phi_s = \text{FALSE}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
    $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
    foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
        if $(I_1 \prec I_s \prec I_2) \wedge \neg\phi_c \in \Omega_{I_s}$ then
            if Consistent $(\mathcal{O} \cup (I_2 \prec I_s)) = \text{TRUE}$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
            else if Consistent $(\mathcal{O} \cup (I_s \prec I_1)) = \text{TRUE}$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
            else return fail
    return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
```

Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



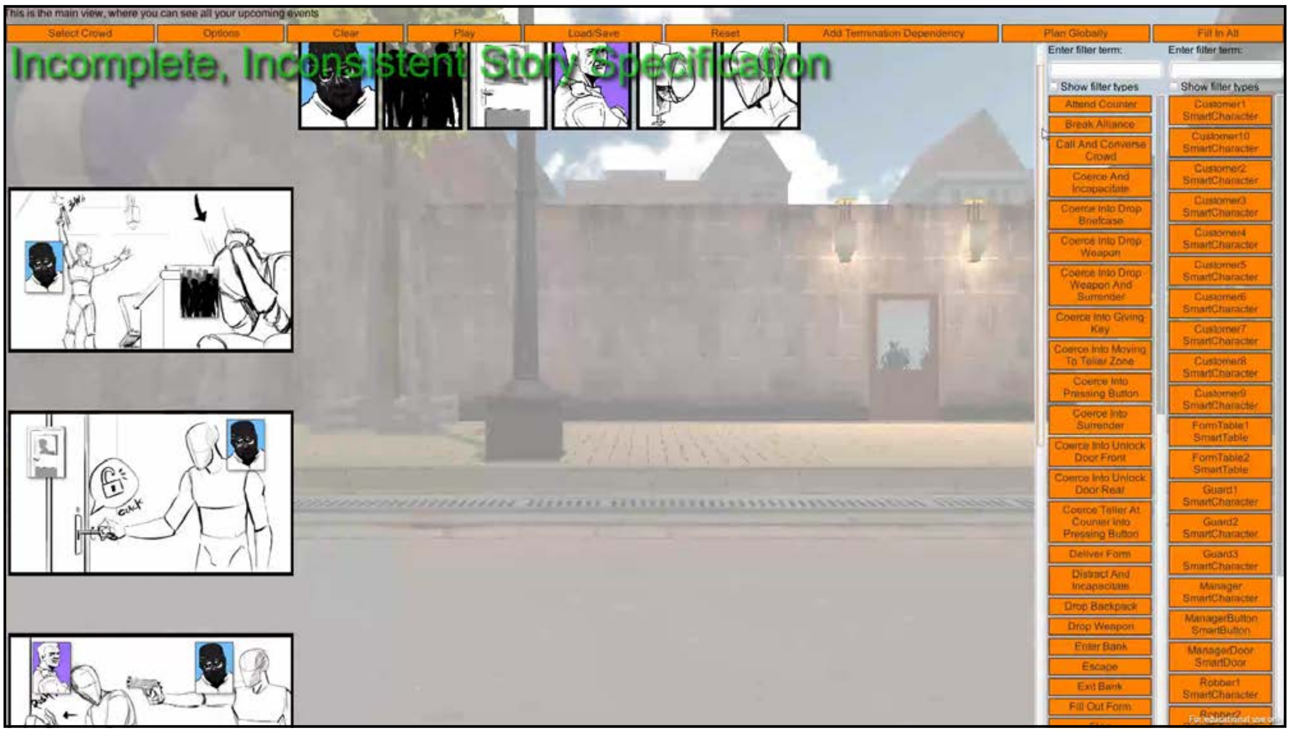
```
        $\mathcal{P} \leftarrow \mathcal{P} - \{y | y \subseteq w_s \text{ s.t } I \sim I_s \exists y \in \mathcal{P} \exists I_s \in \alpha\}$
    return $\mathcal{P}$

key $(\langle I, \phi \rangle)$
    $k_1 \leftarrow \phi \in \Omega_{I_s} | \exists I_s \in \mathcal{I} ? 0 : 1$
    $\mathcal{O}_I \leftarrow \{(I_a, I_b) | \exists (I_a, I_b) \in \mathcal{O}, I_a = I \vee I_b = I\}$
    return $\langle k_1, |\mathcal{O}_I| \rangle$

Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
    if $\mathcal{A} = \varnothing \wedge$ Consistent$(\mathcal{O}) \wedge$ Consistent$(\mathcal{I}) \wedge$ $(\forall \langle I_1, \phi, I_2 \rangle \in \mathcal{L} \;\nexists I_t \in \mathcal{I} | I_1 \prec I_t \prec I_2 \wedge \neg\phi \in \Omega_{I_t})$ then return $\pi_p$
    $\langle I_c = \langle e_c, w_c \rangle, \phi_c \rangle \leftarrow \underset{key(a)}{\operatorname{argmax}}(\mathcal{A})$
    if $w = \varnothing | \exists w \in w_c$ then
        $w_c \leftarrow$ pop$(\mathcal{P}(I_c))$
        if $w_c = \varnothing$ then return fail
    $I_s \leftarrow I | \exists I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
    if $I_s = \varnothing$ then
        $I_s \leftarrow \langle e, w \rangle | e \in \mathcal{E}, w \in \mathcal{P}(\langle e, \{w_i = \varnothing | \forall i \in (1, |w|)\}) \wedge \phi_c \in \Omega_I$
        if $I_s = \varnothing$ then return fail
        $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
        $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) | \forall I \in \mathcal{I}_{end}\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \exists \phi_s \in \Phi_{I_s}, \phi_s = \text{FALSE}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
    $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
    foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
        if $(I_1 \prec I_s \prec I_2) \wedge \neg\phi_c \in \Omega_{I_s}$ then
            if Consistent $(\mathcal{O} \cup (I_2 \prec I_s)) = \text{TRUE}$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
            else if Consistent $(\mathcal{O} \cup (I_s \prec I_1)) = \text{TRUE}$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
            else return fail
    return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$
```

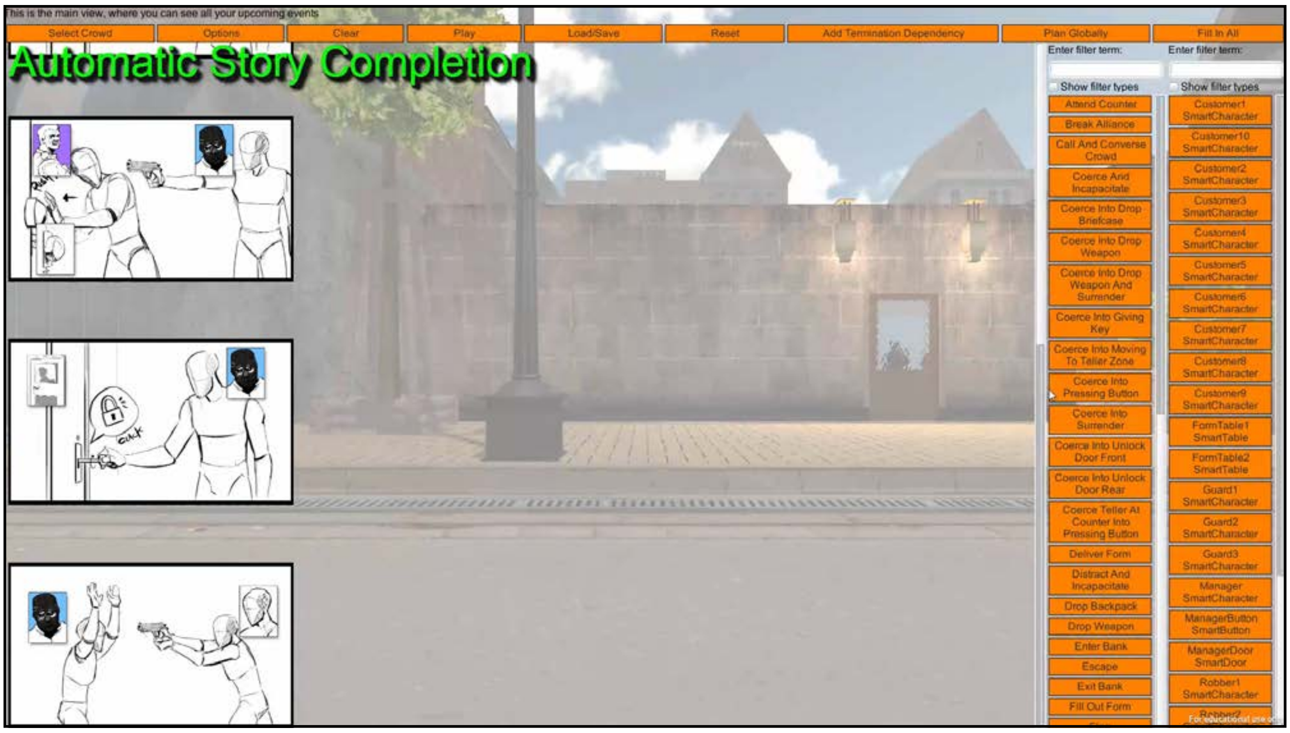
Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization



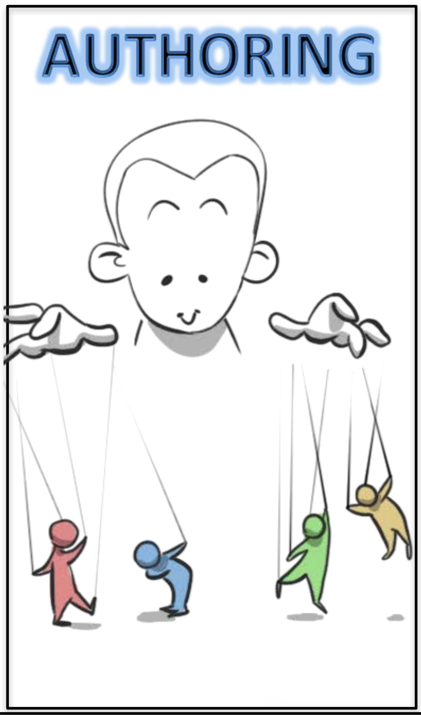
```
        $w_c \leftarrow pop(\mathcal{P}(I_c))$
        if $w_c = \varnothing$ then return fail
    $I_s \leftarrow I \mid \exists I \in \mathcal{I} \wedge \phi_c \in \Omega_I$
    if $I_s = \varnothing$ then
        $I_s \leftarrow \langle e, \mathbf{w} \rangle \mid e \in \mathcal{E}, \mathbf{w} \in \mathcal{P}(\langle e, \{w_i = \varnothing \mid \forall i \in (1, |\mathbf{w}|)\}) \wedge \phi_c \in \Omega_I$
        if $I_s = \varnothing$ then return fail
        $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
        $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) \mid \forall I \in \mathcal{I}_{end}\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle \exists \phi_s \in \Phi_{I_s}, \phi_s = FALSE$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
    $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
    foreach $\langle I_1, \phi_c, I_2 \rangle \in \mathcal{L}$ do
        if $(I_1 \prec I_s \prec I_2) \wedge \neg\phi_c \in \Omega_{I_s}$ then
            if Consistent $(\mathcal{O} \cup (I_2 \prec I_s)) = TRUE$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_2 \prec I_s)$
            else if Consistent $(\mathcal{O} \cup (I_s \prec I_1)) = TRUE$ then
                $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_1)$
            else return fail
    return Plan $(\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E})$

Linearize $(\pi_c = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \alpha_p)$
    $\alpha_c \leftarrow (\mathbf{B}_{|} \alpha_p |)$
    $\mathcal{I} \leftarrow \mathcal{I} - \{I \mid \forall I \in \mathbf{B}_{|} \alpha_p |\}$
    $\mathcal{O} \leftarrow \mathbf{T}(\mathcal{O})$
    foreach $I \in \mathcal{I}$ do
        foreach $i \in (0, |\alpha_c|)$ do
            if $(I \prec I_c) \exists I_c \in \mathbf{B}_i$ then
                if $i = 0$ then
                    $\mathbf{B} \leftarrow \{I\}$
                    $\alpha_c \leftarrow \mathbf{B} \cup \alpha_c$
                else
                    $\mathbf{B}_i \leftarrow \mathbf{B}_i \cup I$
    return $\alpha_c$
```

Overview
Initialization
Parameter Bindings
Termination Condition
Open Precondition Selection
Parameter Binding Selection
Event Instance Selection
Causal Link Protection
Recursive Invocation
Linearization





Automatic Story Completion

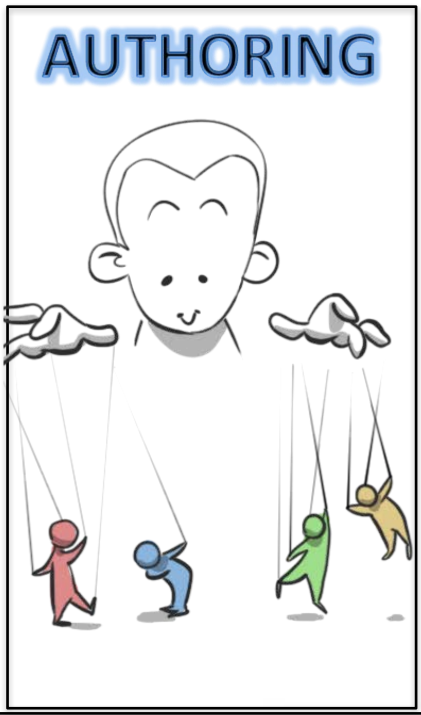


Requirements

Visual Storyboards

Computer-Assisted Story Authoring

Collaboration

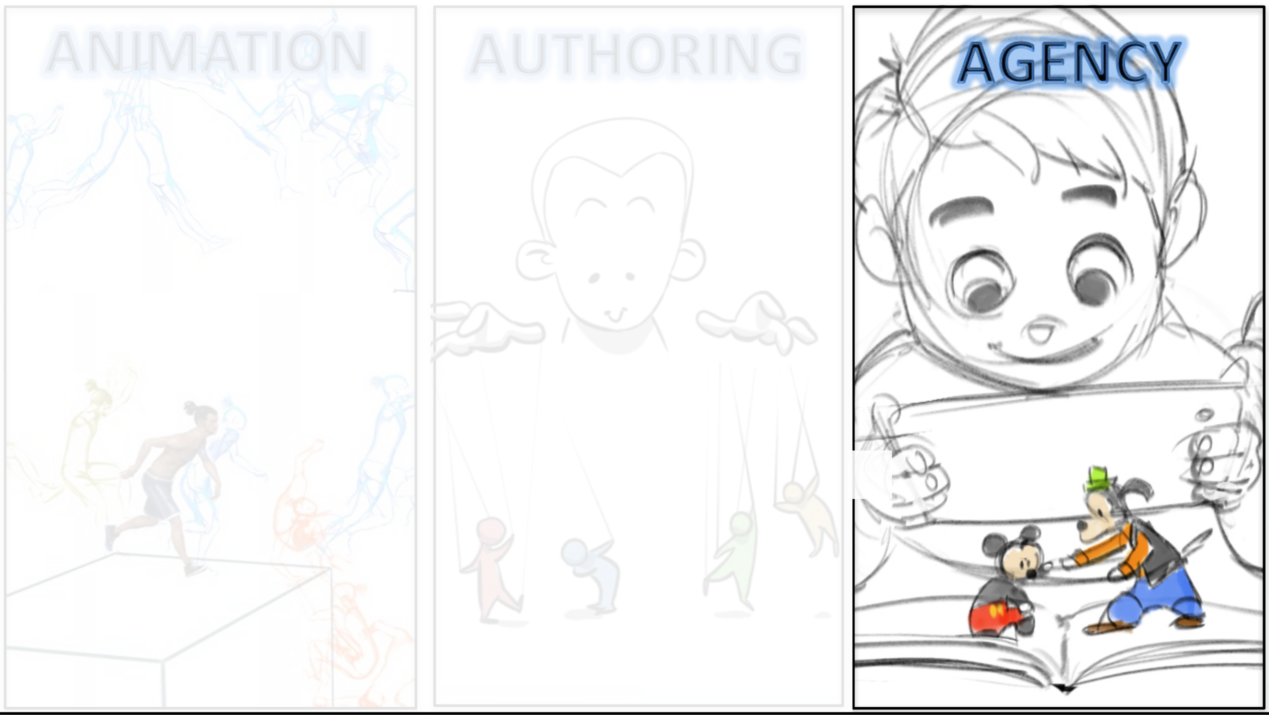


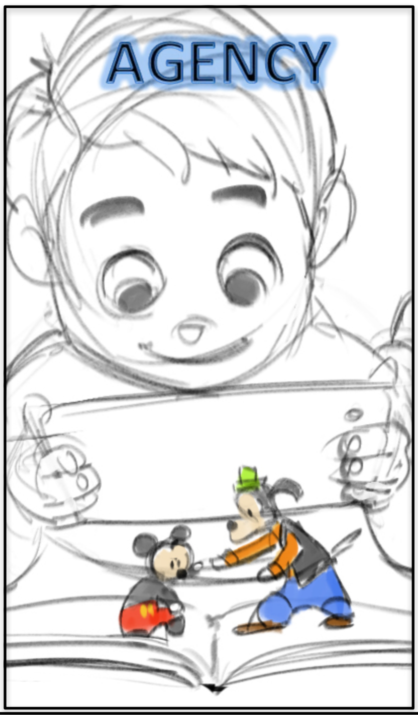
Requirements

Visual Storyboards

Computer-Assisted Story Authoring

Story Version Control

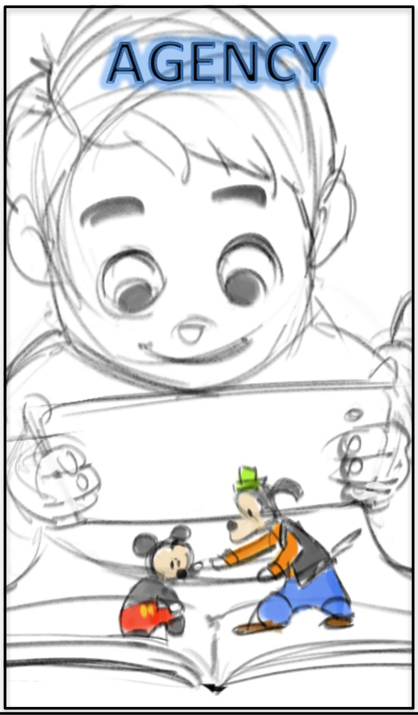




Requirements

Emergent Stories

Freeform Interactive Narratives



Solutions

Computer-Assisted Authoring of Interactive Narratives

Computer-Assisted Authoring of Interactive Narratives. Mubbasir Kapadia, Jessica Falk, Fabio Zund, Marcel Marti, Robert W. Sumner, Markus Gross. *ACM SIGGRAPH Interactive 3D Graphics and Games (I3D), 2015.*

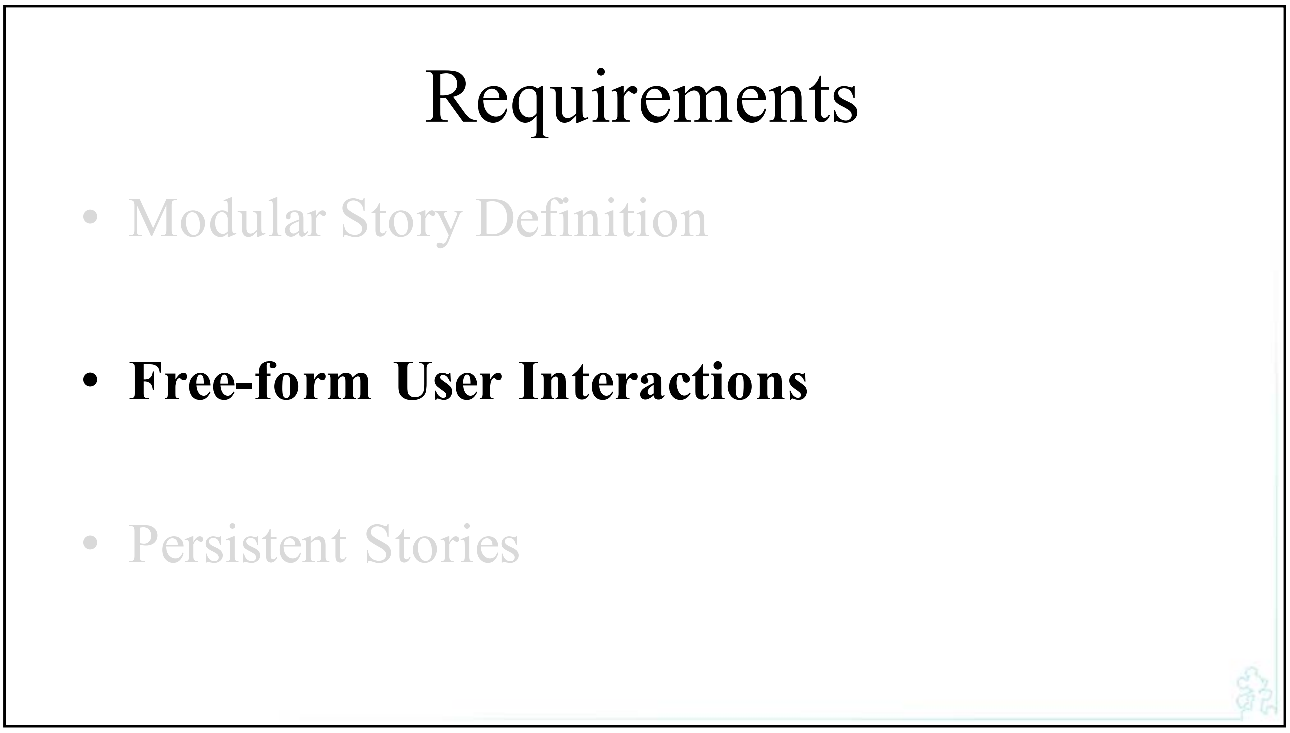
Evaluating the Authoring Complexity of Interactive Narratives with Interactive Behavior Trees. Mubbasir Kapadia, Jessica Falk, Fabio Zund, Marcel Marti, Robert W. Sumner, Markus Gross. *Foundations of Digital Games (FDG), 2015.*

Requirements

- Modular Story Definition
- Free-form User Interactions
- Persistent Stories

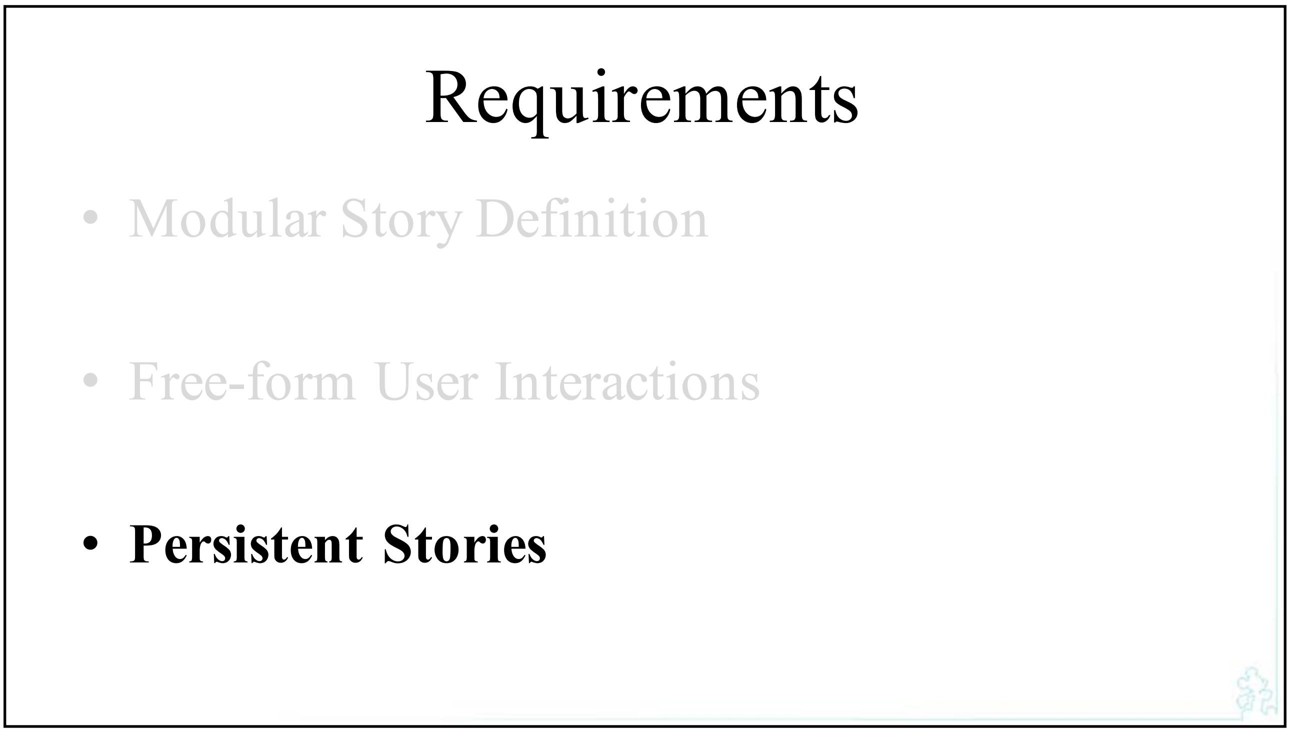
Requirements

- **Modular Story Definition**
- Free-form User Interactions
- Persistent Stories



Requirements

- Modular Story Definition
- **Free-form User Interactions**
- Persistent Stories



Requirements

- Modular Story Definition
- Free-form User Interactions
- **Persistent Stories**

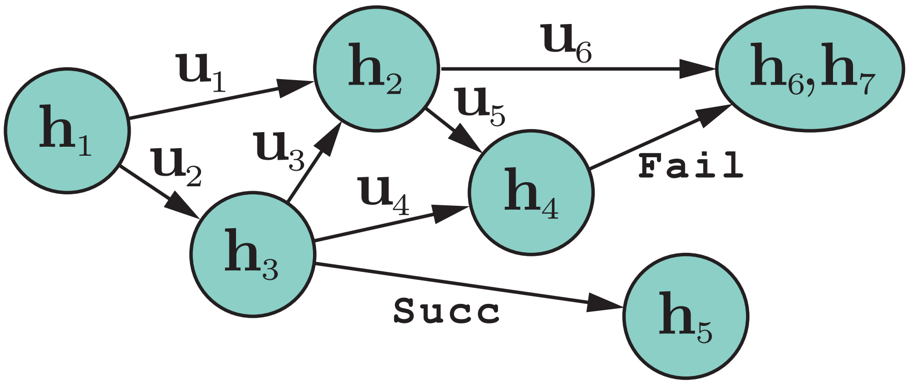
Manual Approaches

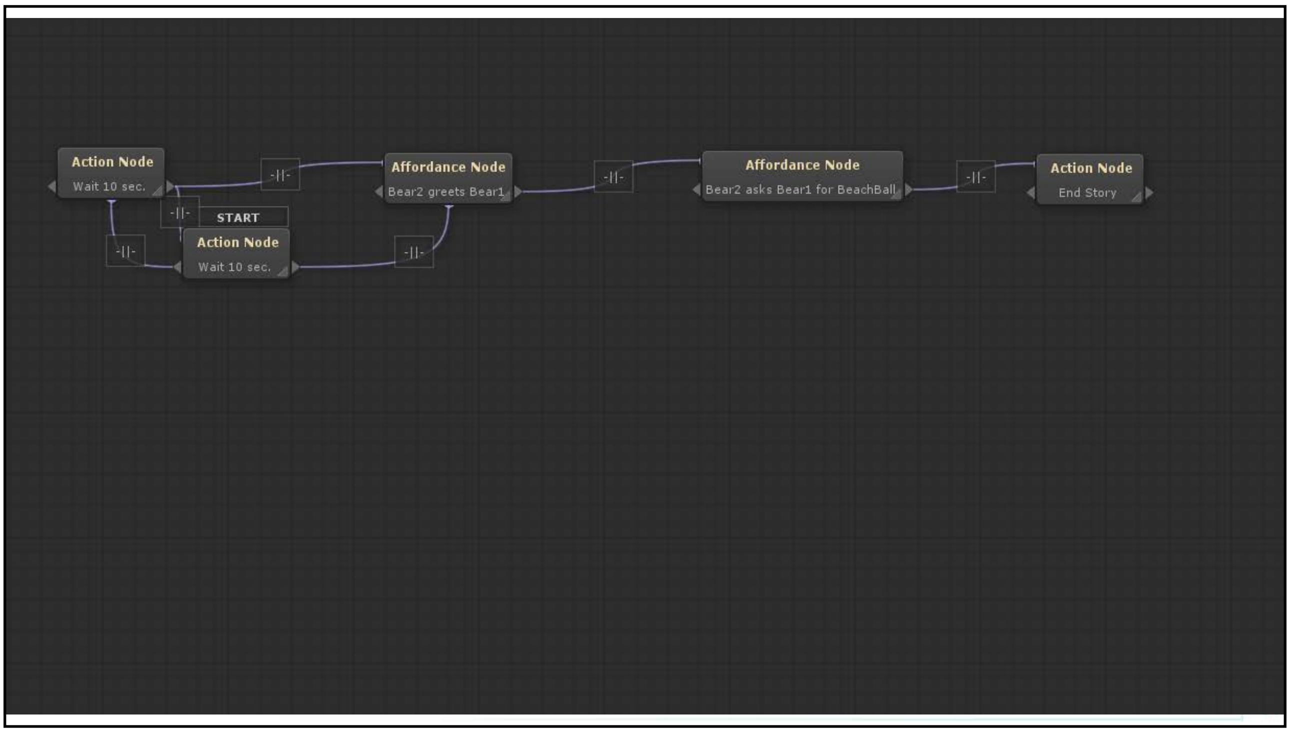
- Scripted Approaches [Loyall 1997; Mateas 2002]
- Rule-based Systems [Perlin and Goldberg 1996; Menou 2001]
- Façade [Mateas and Stern 2003, 2004; Dow et al. 2006]
- Dialogue Trees [Prakken and Sartor 1997]
- Story Graphs [Gordon et al. 2004]
- Behavior Trees [Hecker et al. 2007; Isla 2008; Shoulson et al. 2011; Millington and Funge 2008]

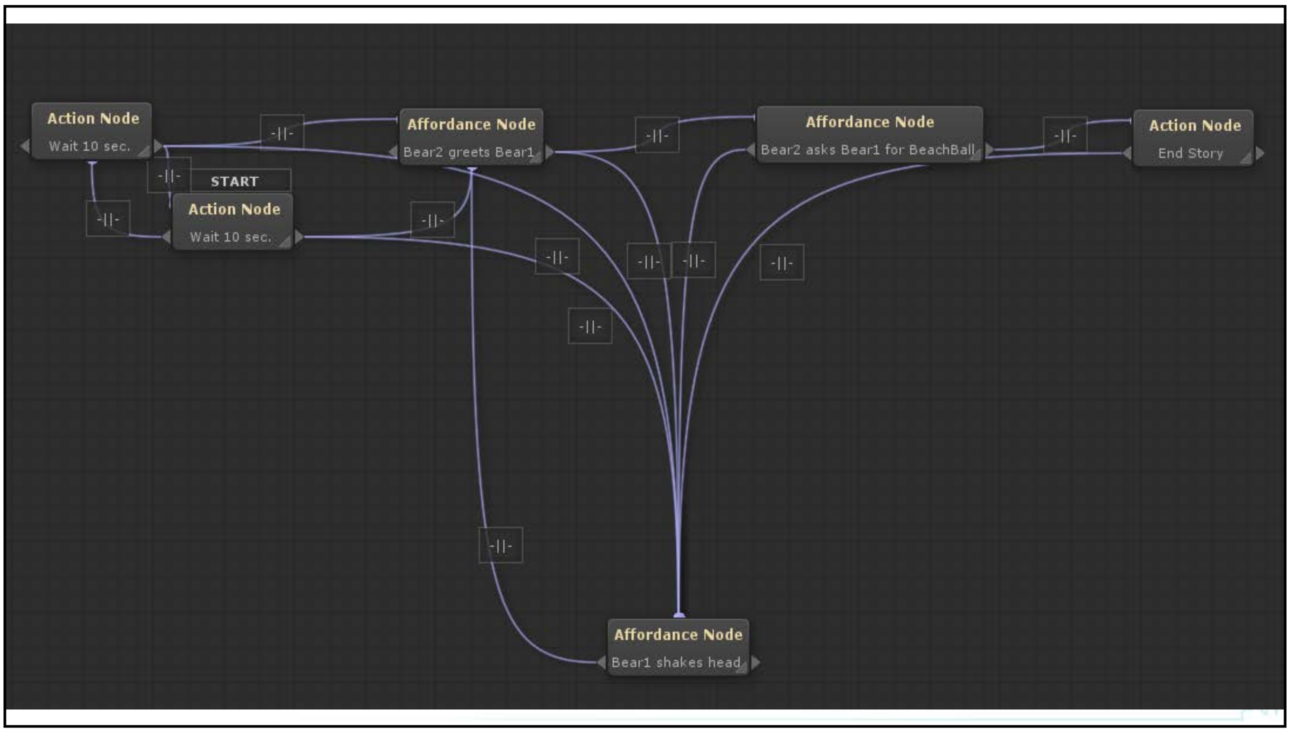
Automated Approaches

- Domain-Independent Planners [Fikes and Nilsson 1971; Sacerdoti 1975]
- Narrative generation systems [Riedl et al. 2003, Riedl and Young 2006]
- Virtual Directors & Drama Managers [Magerko et al. 2004, Riedl et al. 2008, Shoulson et al. 2013]

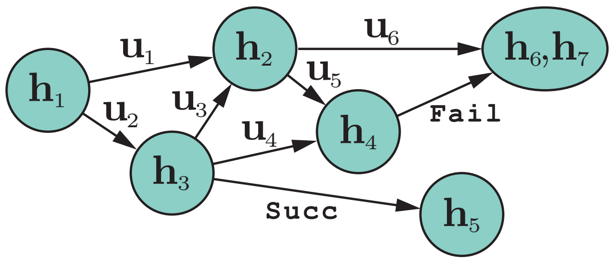
Story Graphs





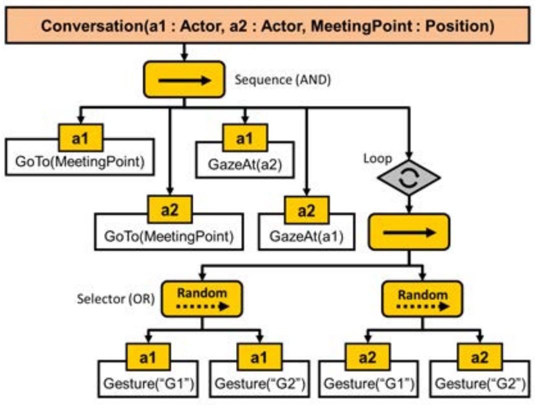


Story Graphs

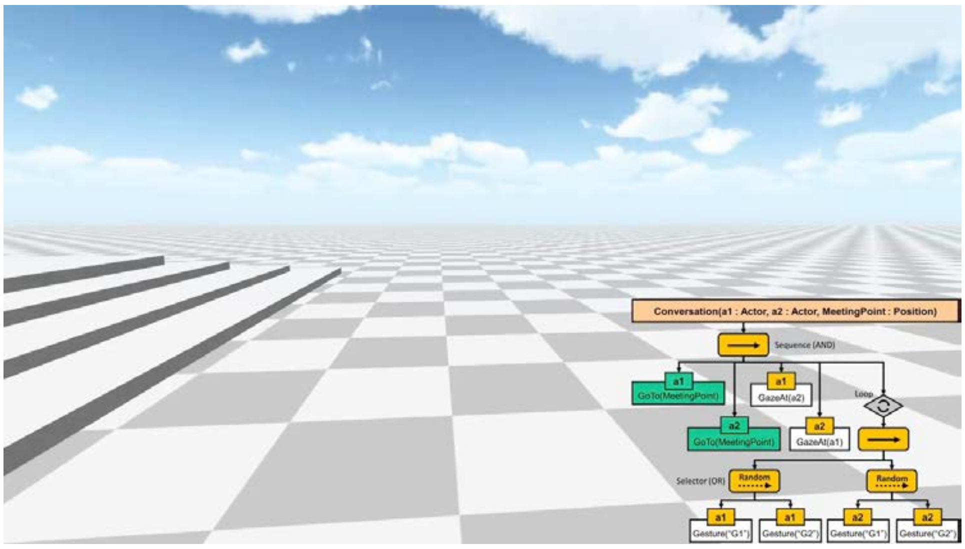


- **Modular Story Definition**
- **Free-form User Interactions**
- **Persistent Stories**

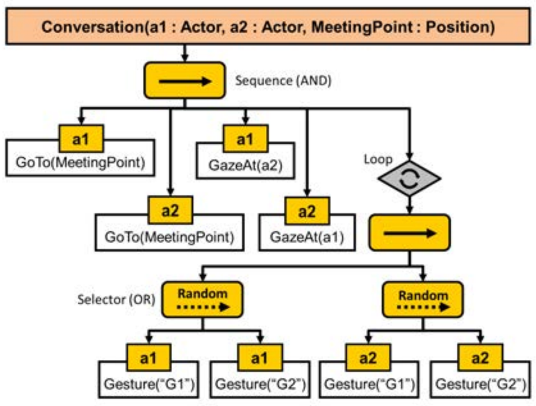
Behavior Trees



ADAPT: Agent Development and Prototyping Testbed. Alexander Shoulson, Nathan Marhsak, Mubbasir Kapadia, Norman I. Badler. *IEEE Transactions on Visualization and Computer Graphics (TVCG), 2014.*


Conversation(a1 : Actor, a2 : Actor, MeetingPoint : Position)
Sequence (AND)
Loop
Selector (OR)
Random

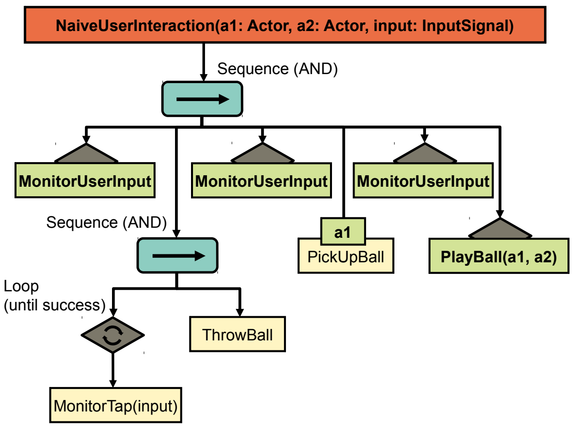
Behavior Trees



- **Modular Story Definition**
- Free-form User Interactions
- Persistent Stories

ADAPT: Agent Development and Prototyping Testbed. Alexander Shoulson, Nathan Marhsak, Mubbasir Kapadia, Norman I. Badler. *IEEE Transactions on Visualization and Computer Graphics (TVCG), 2014.*

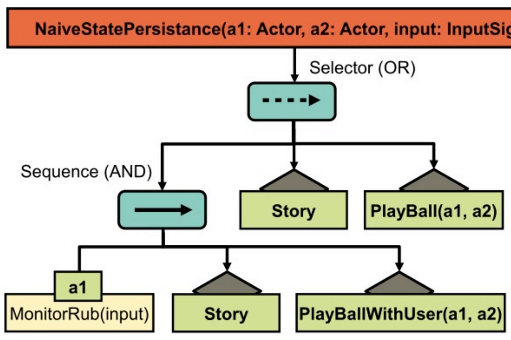
Behavior Trees



- **Modular Story Definition**
- **Free-form User Interactions**
- Persistent Stories

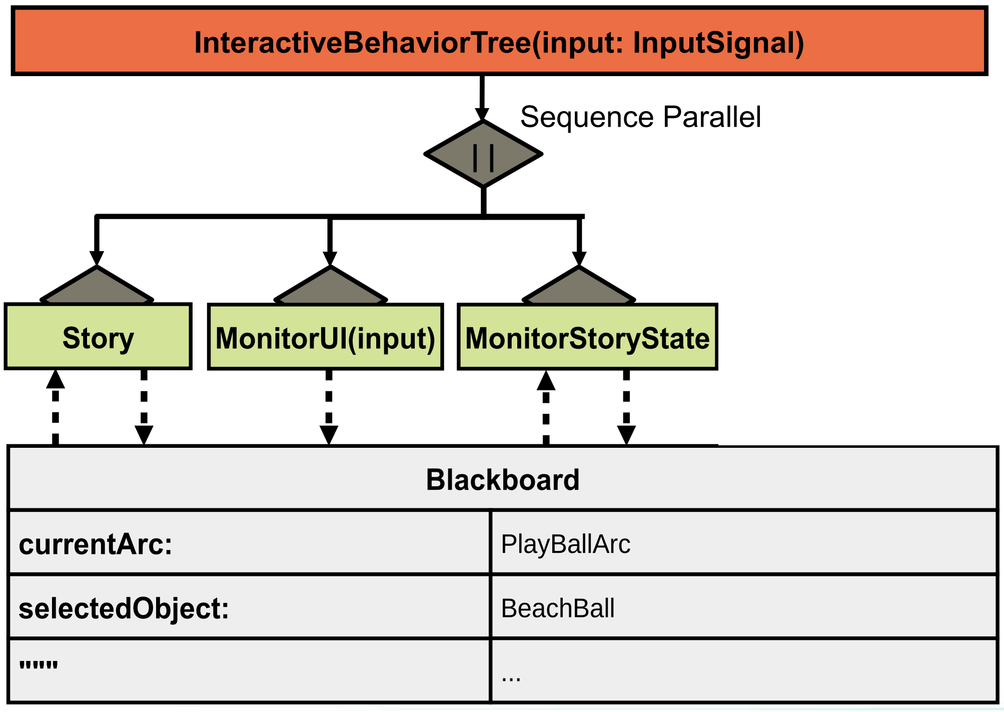
ADAPT: Agent Development and Prototyping Testbed. Alexander Shoulson, Nathan Marhsak, Mubbasir Kapadia, Norman I. Badler. *IEEE Transactions on Visualization and Computer Graphics (TVCG), 2014.*

Behavior Trees

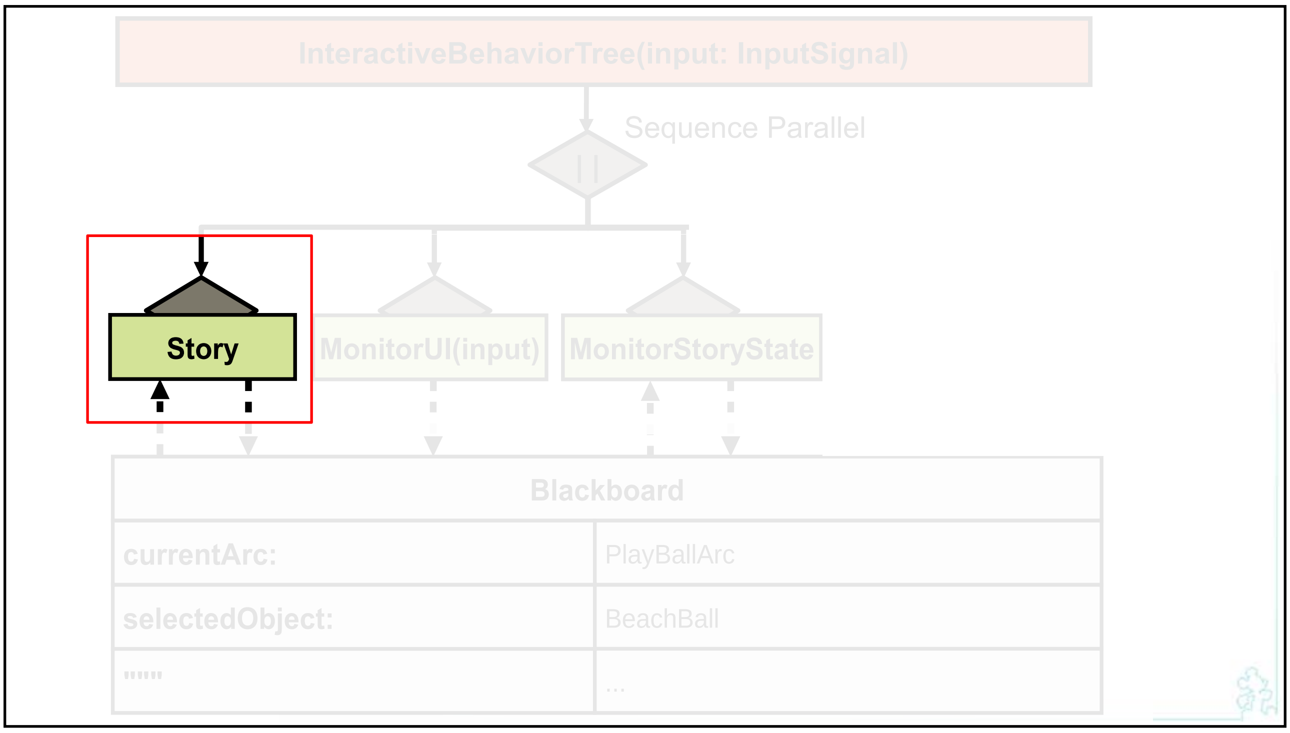


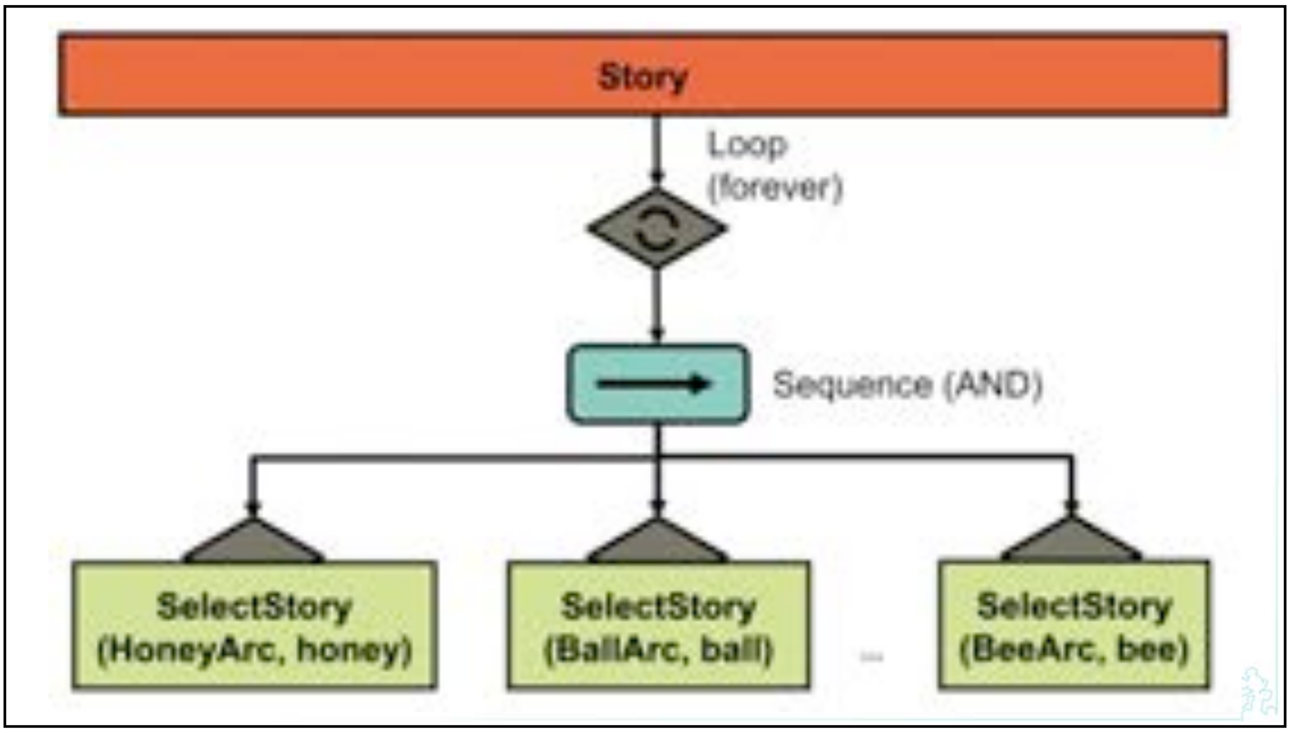
- **Modular Story Definition**
- **Free-form User Interactions**
- **Persistent Stories**

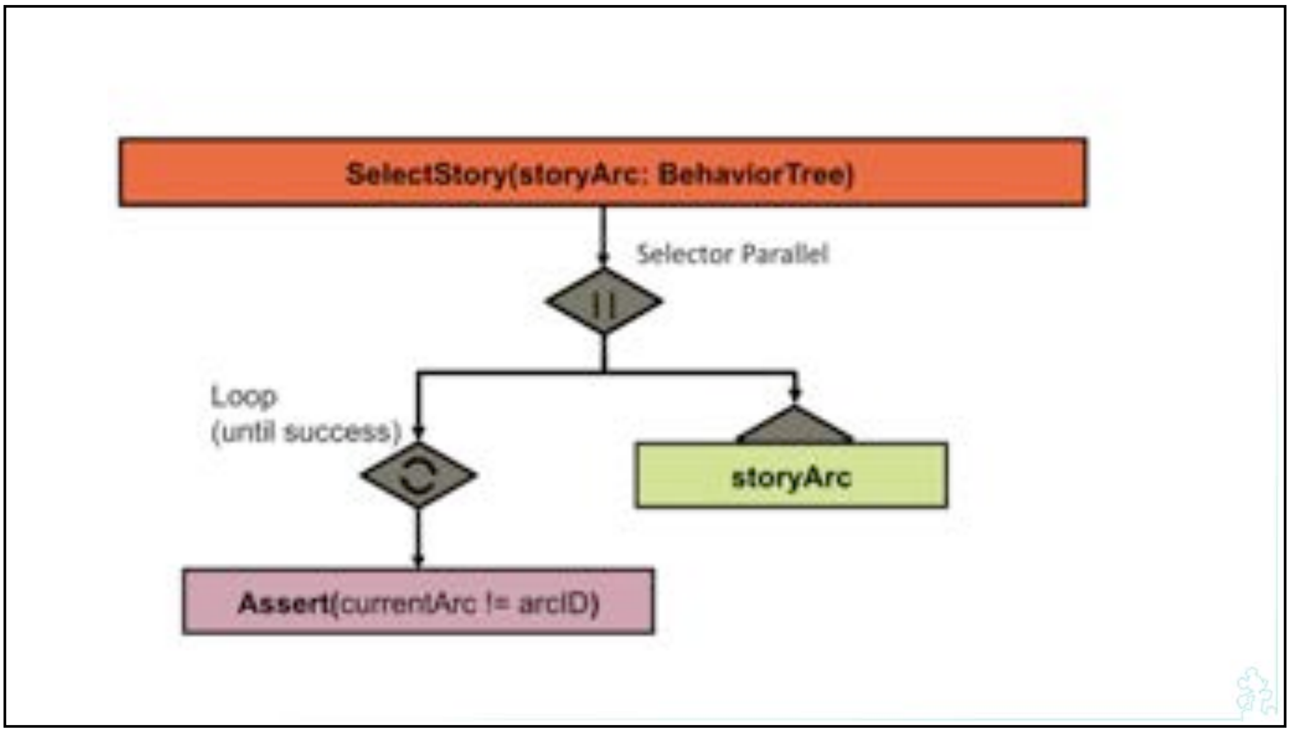
ADAPT: Agent Development and Prototyping Testbed. Alexander Shoulson, Nathan Marhsak, Mubbasir Kapadia, Norman I. Badler. *IEEE Transactions on Visualization and Computer Graphics (TVCG), 2014.*

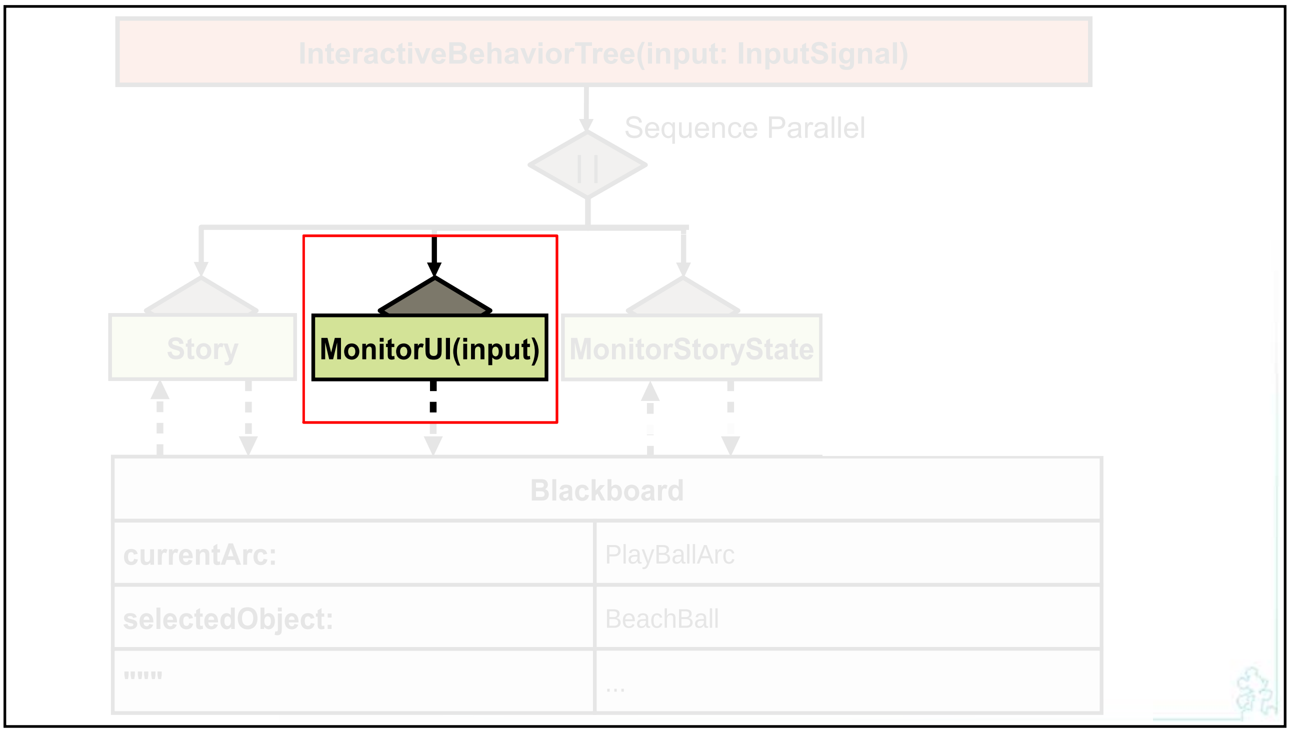


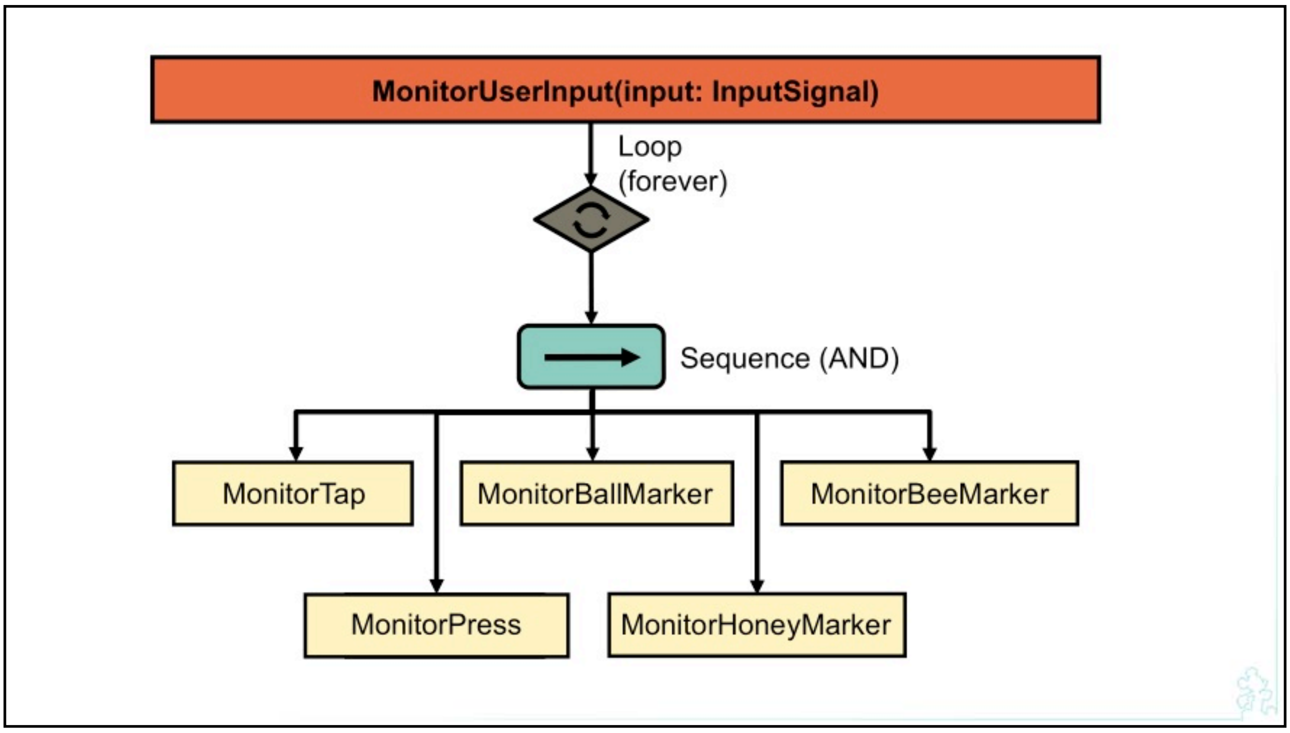
currentArc:	PlayBallArc
selectedObject:	BeachBall
"""	...

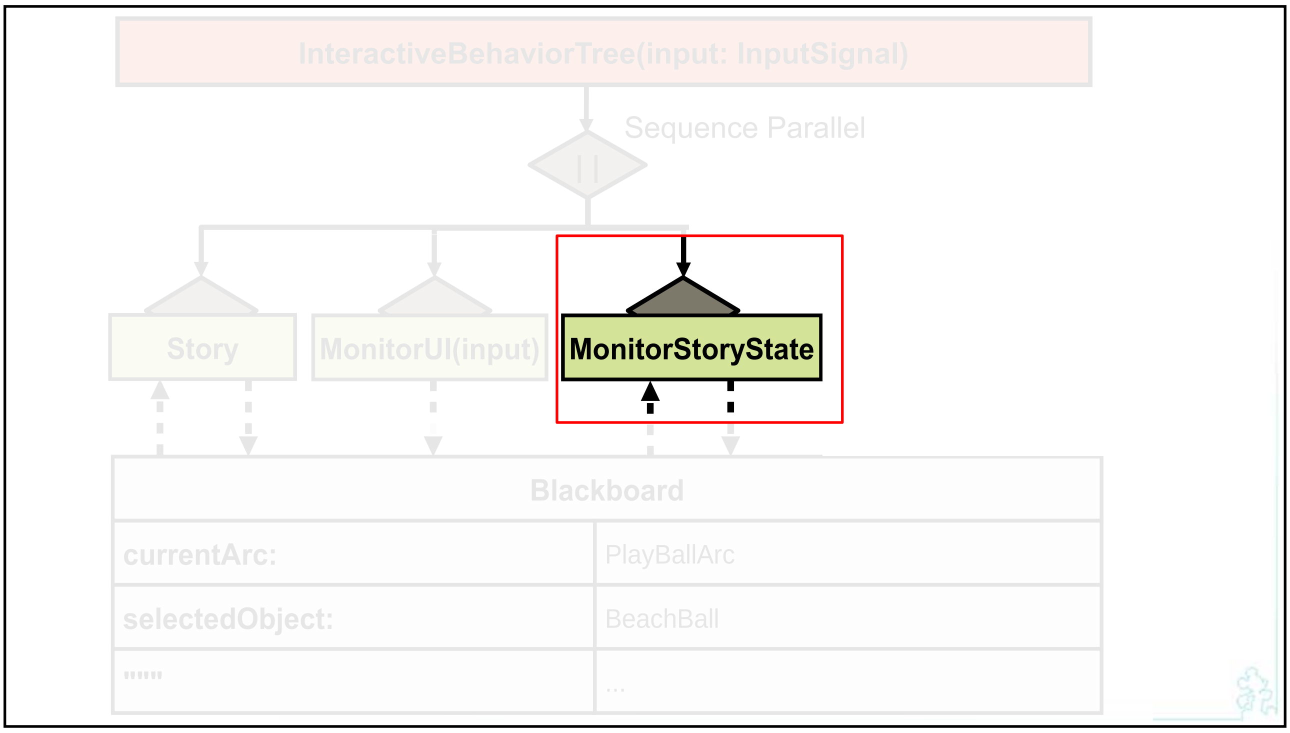

InteractiveBehaviorTree(input: InputSignal)
Sequence Parallel
Story
MonitorUI(input)
MonitorStoryState
Blackboard
currentArc:
PlayBallArc
selectedObject:
BeachBall
......
...

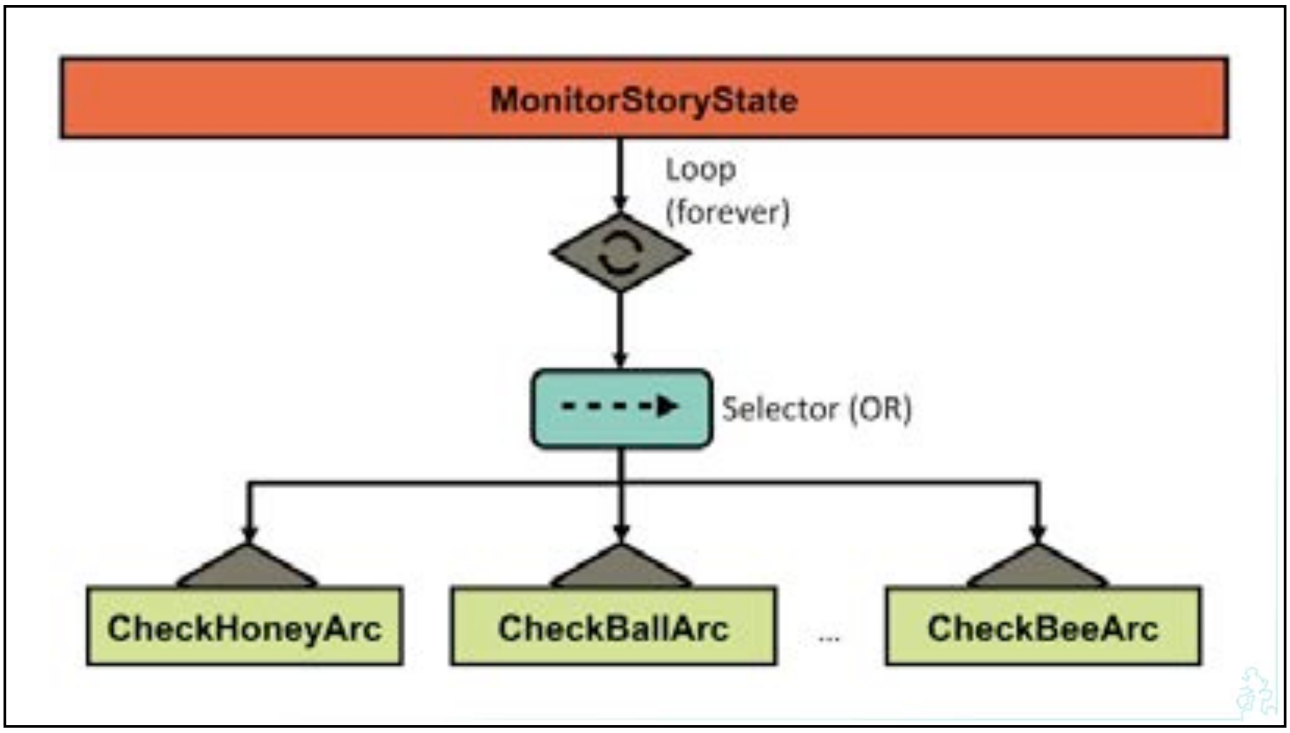

Story
Loop
(forever)
Sequence (AND)
SelectStory
(HoneyArc, honey)
SelectStory
(BallArc, ball)
...
SelectStory
(BeeArc, bee)

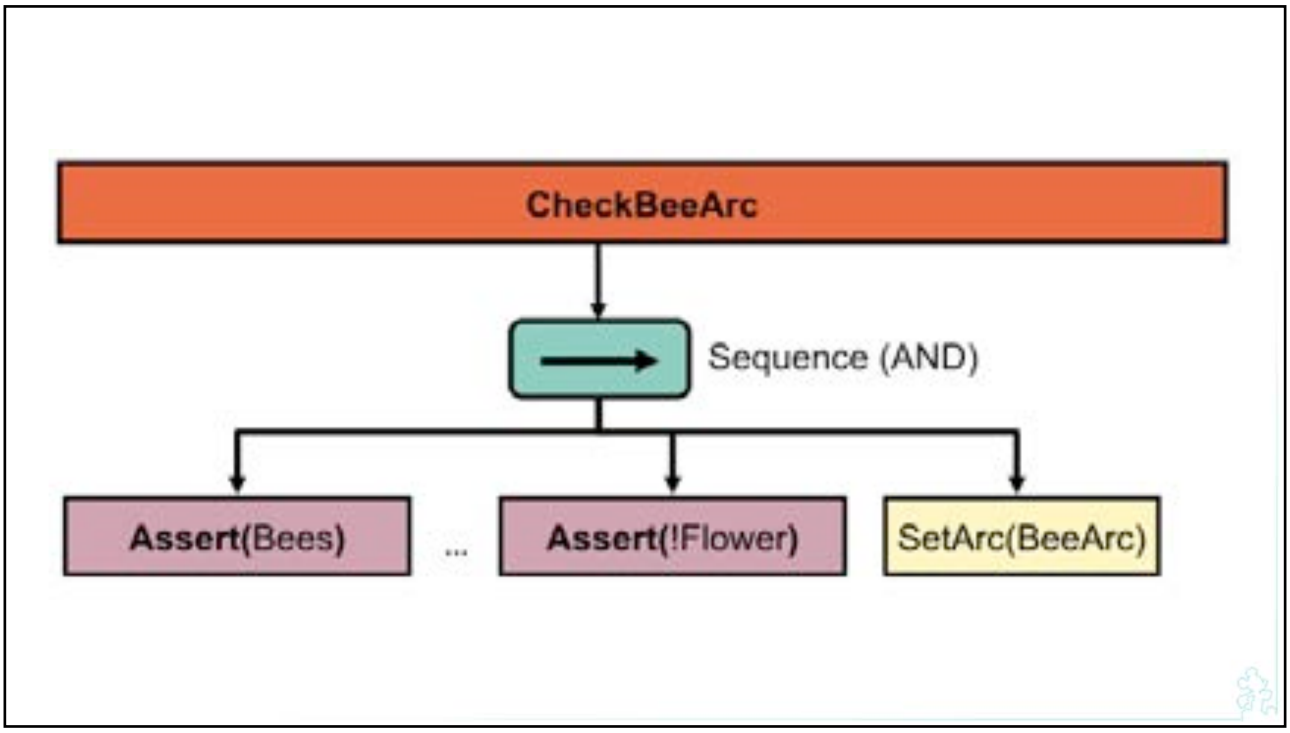

SelectStory(storyArc: BehaviorTree)
Selector Parallel
Loop
(until success)
storyArc
Assert(currentArc != arcID)

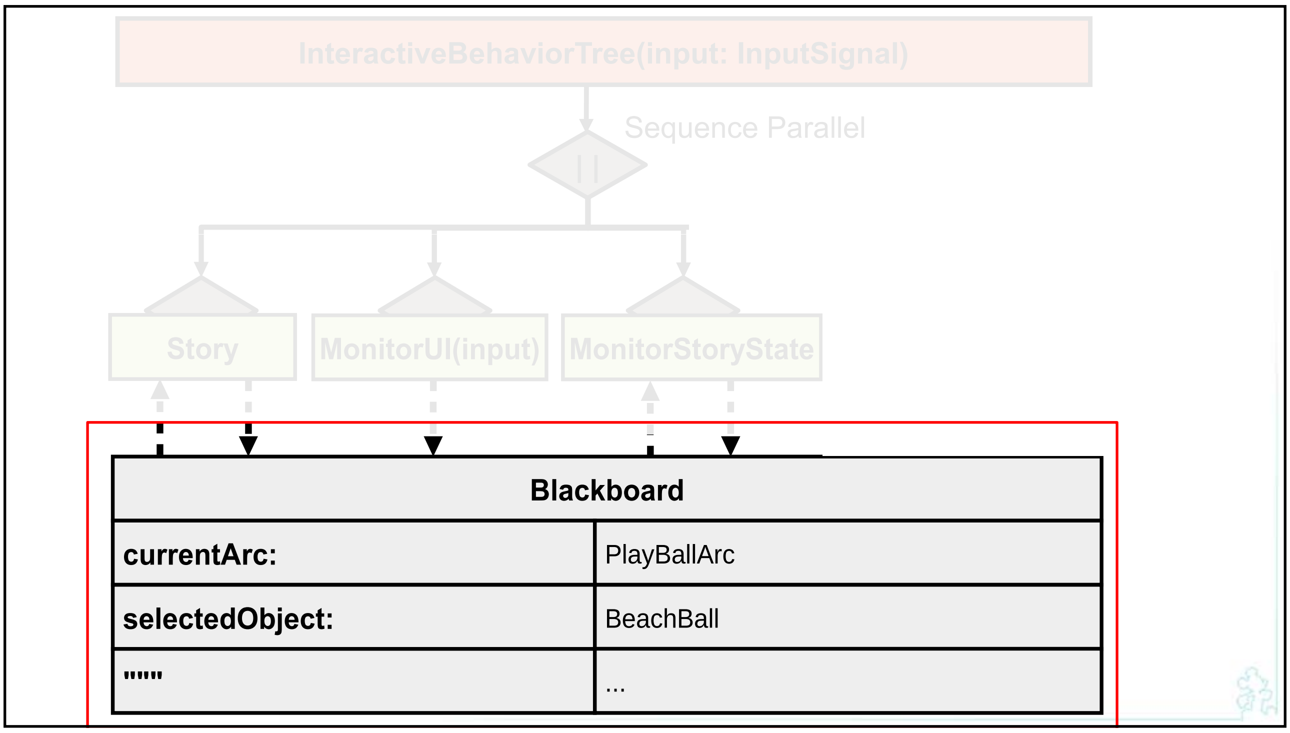

InteractiveBehaviorTree(input: InputSignal)
Sequence Parallel
Story
MonitorUI(input)
MonitorStoryState
Blackboard
currentArc:
PlayBallArc
selectedObject:
BeachBall
""""
…


MonitorUserInput(input: InputSignal)
Loop (forever)
Sequence (AND)
MonitorTap
MonitorBallMarker
MonitorBeeMarker
MonitorPress
MonitorHoneyMarker

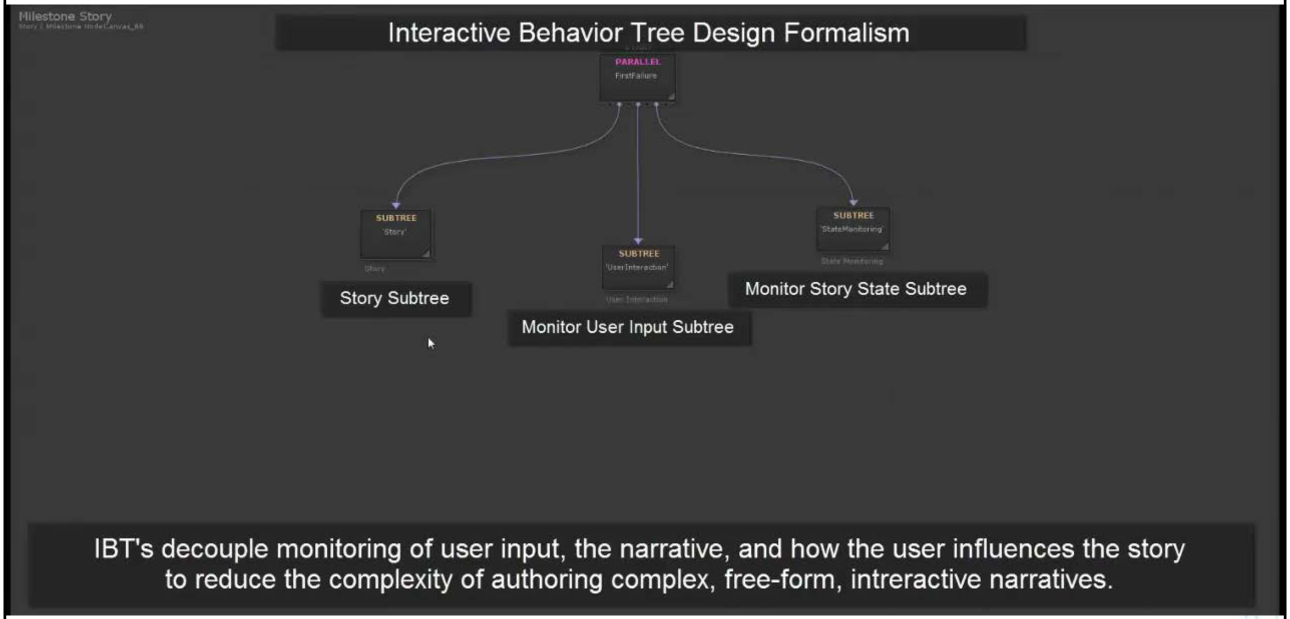

InteractiveBehaviorTree(input: InputSignal)
Sequence Parallel
Story
MonitorUI(input)
MonitorStoryState
Blackboard
currentArc:
PlayBallArc
selectedObject:
BeachBall
""""
...


MonitorStoryState
Loop (forever)
Selector (OR)
CheckHoneyArc
CheckBallArc
...
CheckBeeArc


CheckBeeArc
Sequence (AND)
Assert(Bees)
...
Assert(!Flower)
SetArc(BeeArc)

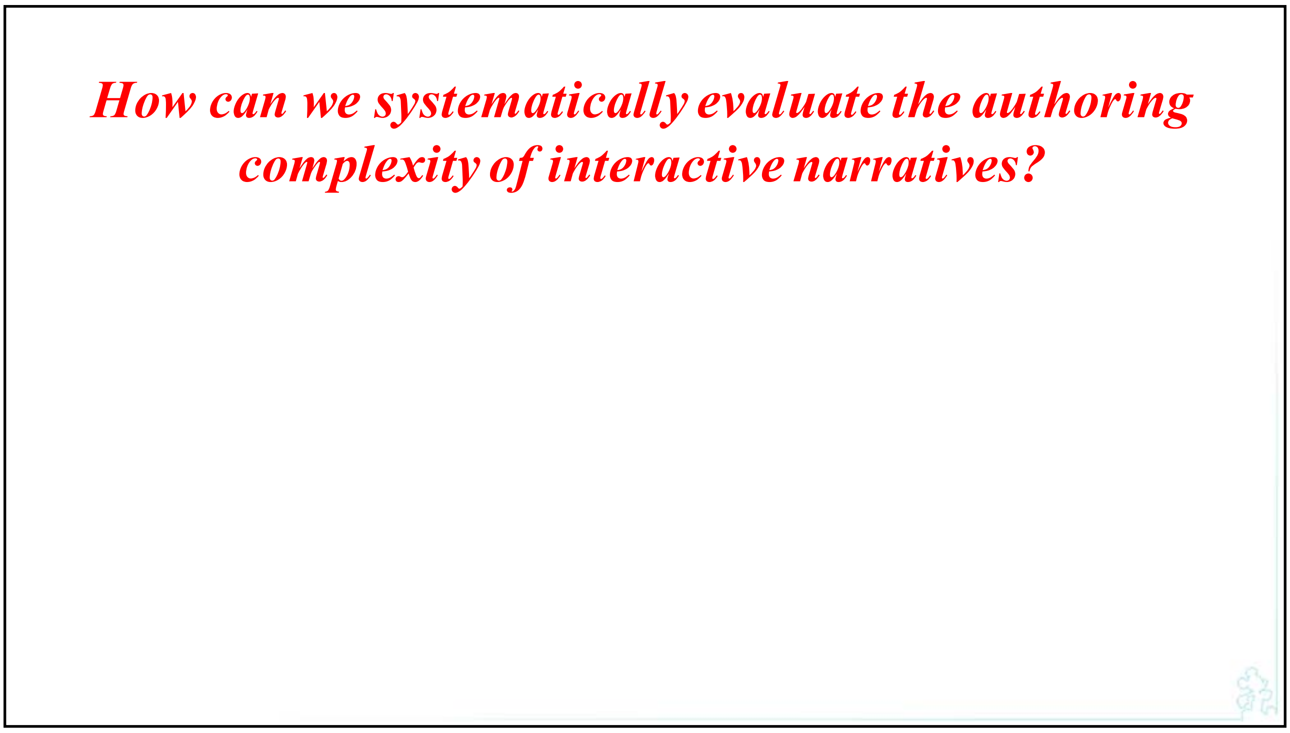


Blackboard	
currentArc:	PlayBallArc
selectedObject:	BeachBall
""""	...

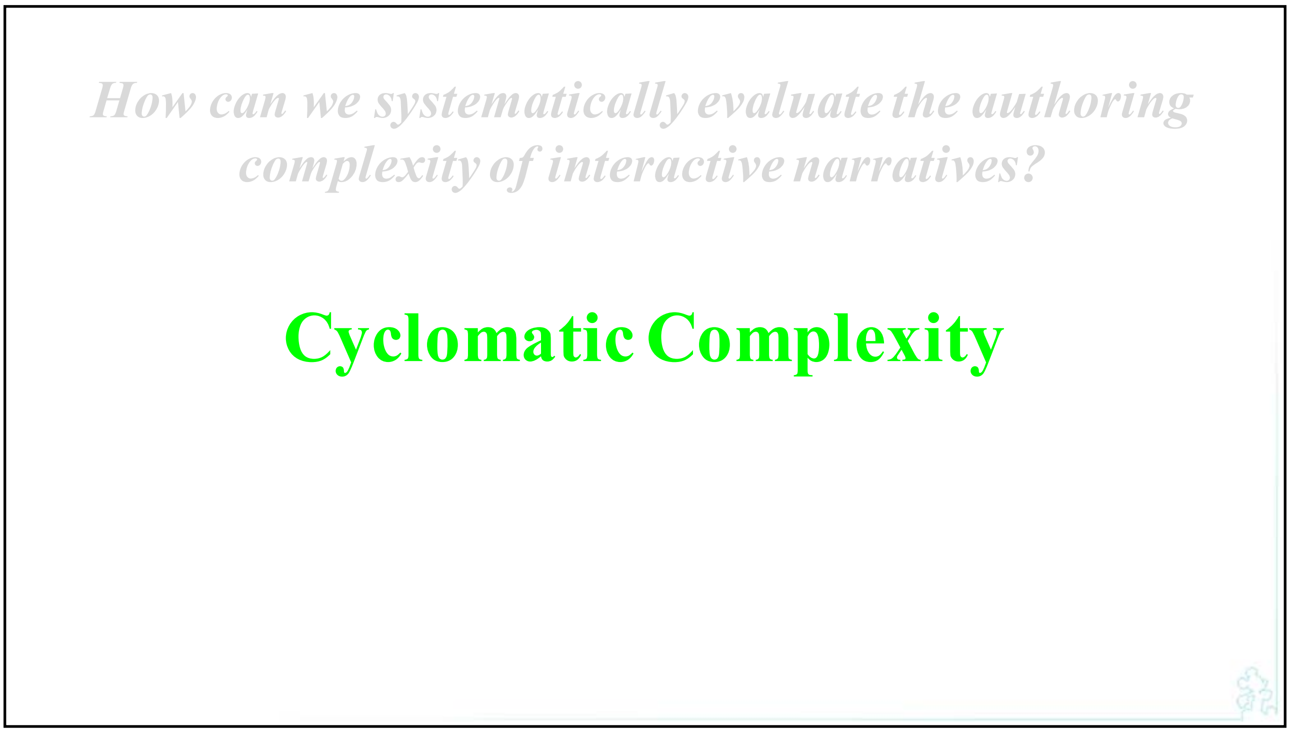


Interactive Behavior Tree Design Formalism

IBT's decouple monitoring of user input, the narrative, and how the user influences the story to reduce the complexity of authoring complex, free-form, intreractive narratives.



How can we systematically evaluate the authoring complexity of interactive narratives?



How can we systematically evaluate the authoring complexity of interactive narratives?

Cyclomatic Complexity

How can we systematically evaluate the authoring complexity of interactive narratives?

Cyclomatic Complexity

$$c(\mathbf{g}) = p(\mathbf{g}) + 1$$

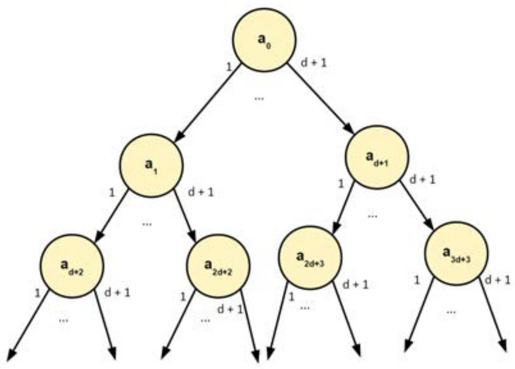
How can we systematically evaluate the authoring complexity of interactive narratives?

Cyclomatic Complexity

$$c(\mathbf{g}) = p(\mathbf{g}) + 1$$

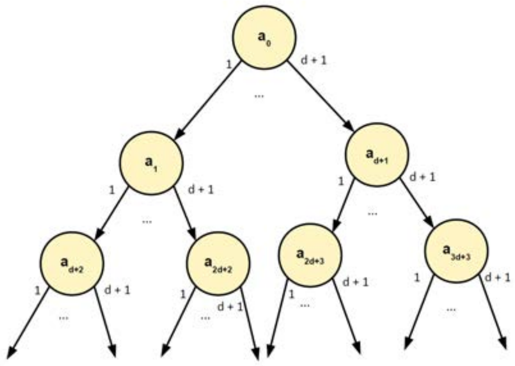
$$c(\mathbf{g}) = p(\mathbf{g}) - s(\mathbf{g}) + 2$$

Story Graphs



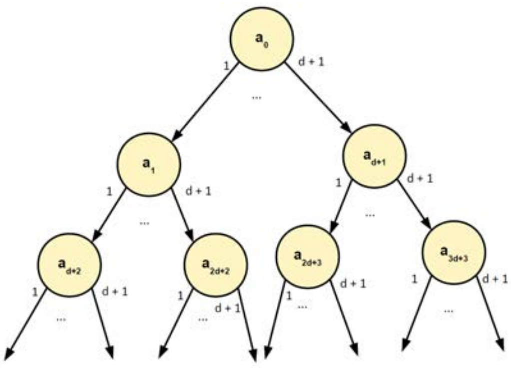
$$c(\mathbf{g}_\mathrm{s}) = d \cdot \left(\sum_{i=1}^{m} |a_i| \right) - s(\mathbf{g}_\mathrm{s}) + 2$$

Story Graphs



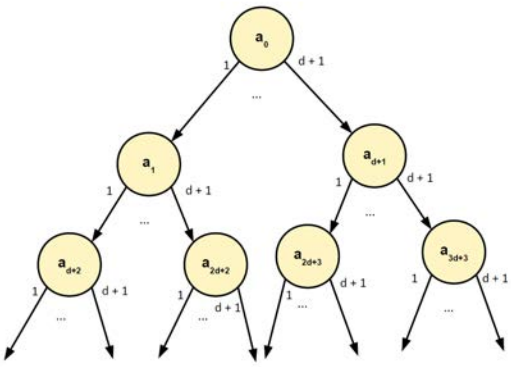
$$c(\mathbf{g}_\mathrm{s}) = \boxed{d \cdot} \left(\sum_{i=1}^{m} |a_i| \right) - s(\mathbf{g}_\mathrm{s}) + 2$$

Story Graphs



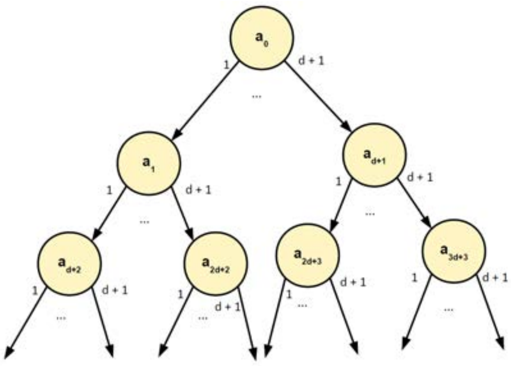
$$c(\mathbf{g}_s) = d \cdot \left(\sum_{i=1}^{\boxed{m}} |a_i| \right) - s(\mathbf{g}_s) + 2$$

Story Graphs



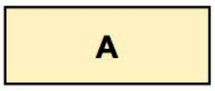
$$c(\mathbf{g}_s) = d \cdot \left(\sum_{i=1}^{m} \boxed{|a_i|} \right) - s(\mathbf{g}_s) + 2$$

Story Graphs



$$c(\mathbf{g}_\mathrm{s}) = d \cdot \left(\sum_{i=1}^{m} |a_i| \right) - s(\mathbf{g}_\mathrm{s}) + 2$$

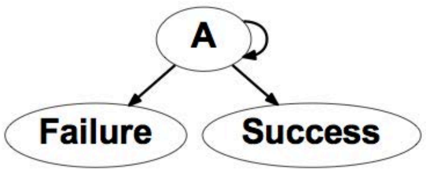
Leaf Node

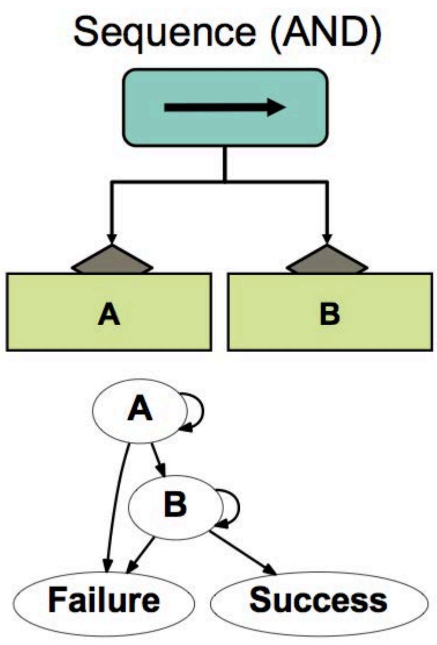


$$p(\mathbf{t}_\mathrm{leaf}) = \{0, 1, 2\}$$

$$s(\mathbf{t}_\mathrm{leaf}) = \{1, 2\}$$

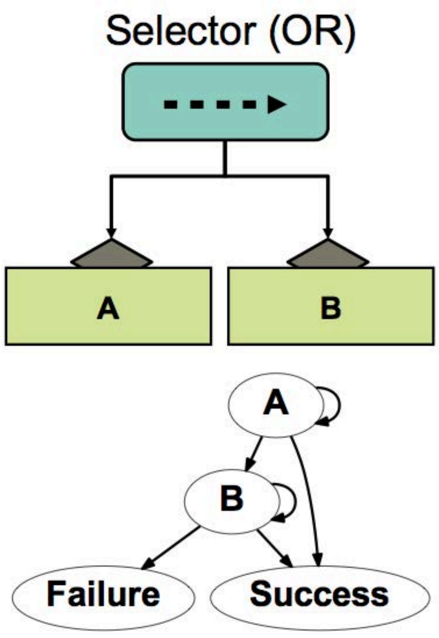
$$c(\mathbf{t}_\mathrm{leaf}) = 2$$





$$p(\mathbf{t}_{\text{seq}}) = \sum_{i=1}^{m} p(\mathbf{t}_i),\ s(\mathbf{t}_{\text{seq}}) = 2,$$

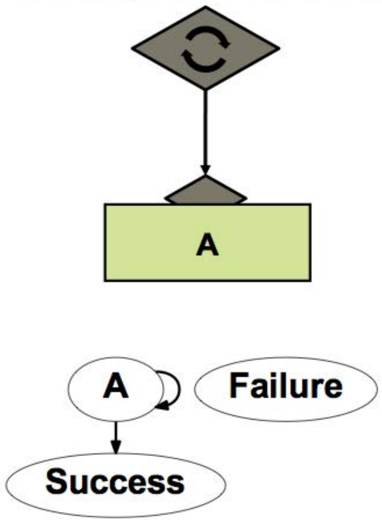
$$\therefore\ c(\mathbf{t}_{\text{seq}}) = \sum_{i=1}^{m} p(\mathbf{t}_i)$$



$$c(\mathbf{t}_{\text{sel}}) = c(\mathbf{t}_{\text{seq}})$$

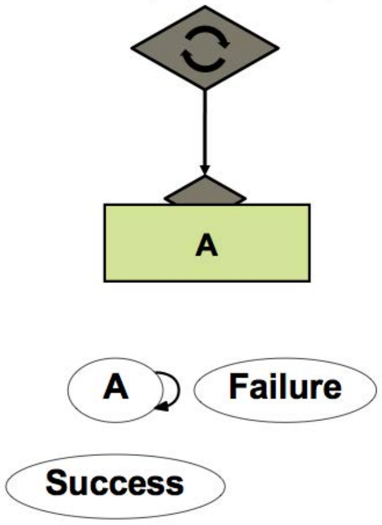
$$= \sum_{i=1}^{m} p(\mathbf{t}_i)$$

Loop (until Success)

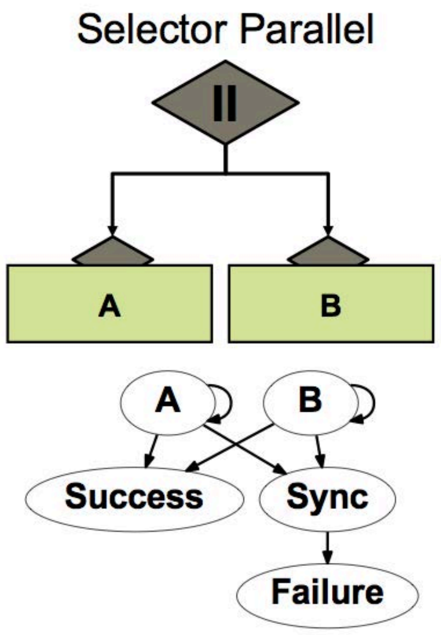


$$p(\mathbf{t}_{\text{loop}}) = 1 + p(\mathbf{t}_{\text{c}}),\ s(\mathbf{t}_{\text{loop}}) = 1,$$
$$\therefore\ c(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}_{\text{c}}) + 2$$

Loop (forever)



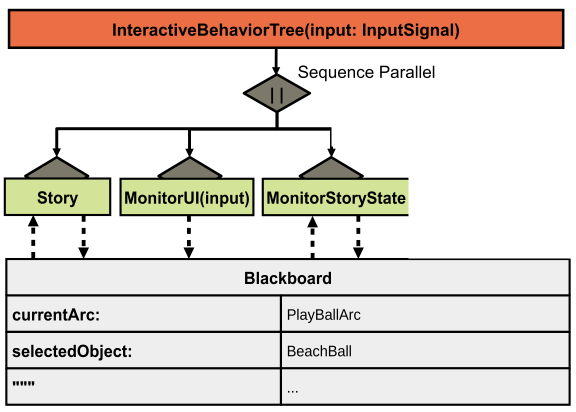
$$p(\mathbf{t}_{\text{loop}}) = p(\mathbf{t}),\ s(\mathbf{t}_{\text{loop}}) = 0,$$
$$\therefore\ c(\mathbf{t}_{\text{loop}}) = c(\mathbf{t})$$



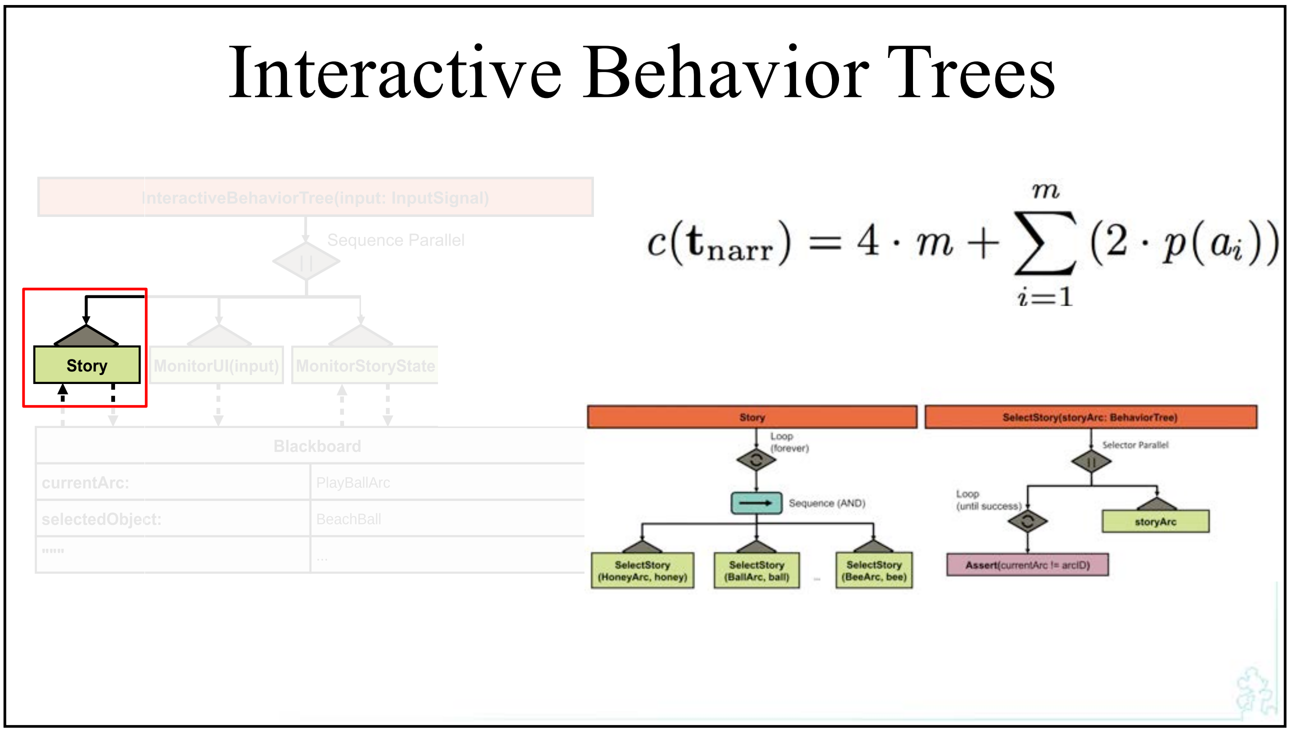
$$p(\mathbf{t}_{\text{par}}) = 2 \cdot \sum_{i=1}^{m} p(\mathbf{t}_i),\ s(\mathbf{t}_{\text{par}}) = 2,$$

$$\therefore\ c(\mathbf{t}_{\text{par}}) = 2 \cdot \sum_{i=1}^{m} p(\mathbf{t}_i)$$

Interactive Behavior Trees

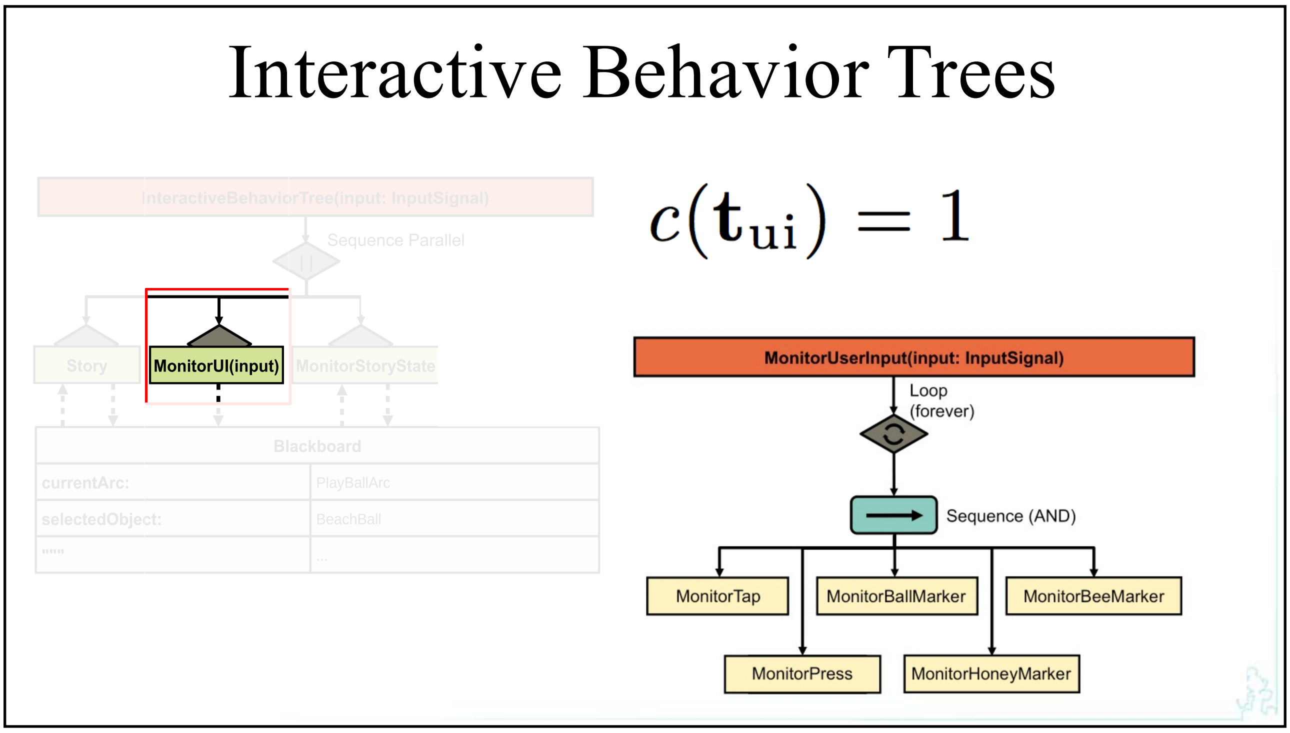


$$c(\mathbf{t}_{\text{IBT}}) = c(\mathbf{t}_{\text{ui}}) + c(\mathbf{t}_{\text{state}}) + c(\mathbf{t}_{\text{narr}})$$



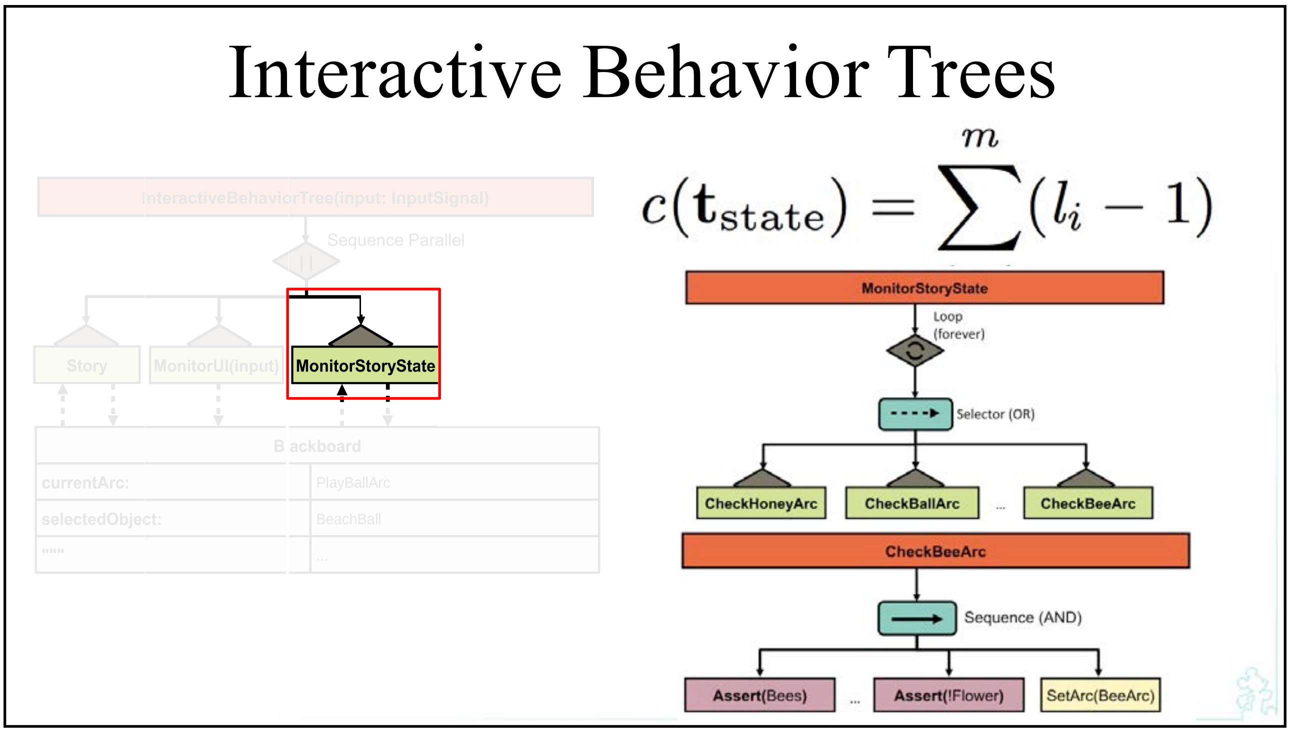
Interactive Behavior Trees

$$c(\mathbf{t}_{\mathrm{narr}}) = 4 \cdot m + \sum_{i=1}^{m} (2 \cdot p(a_i))$$



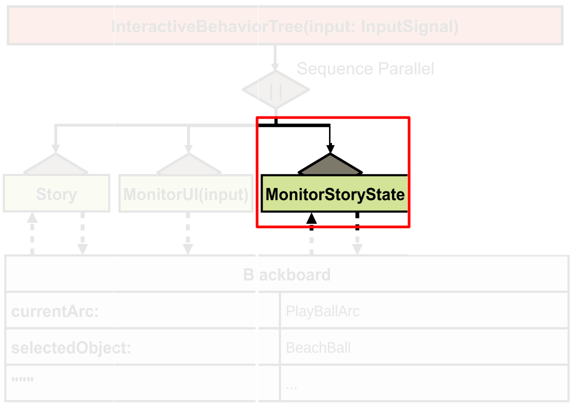
Interactive Behavior Trees

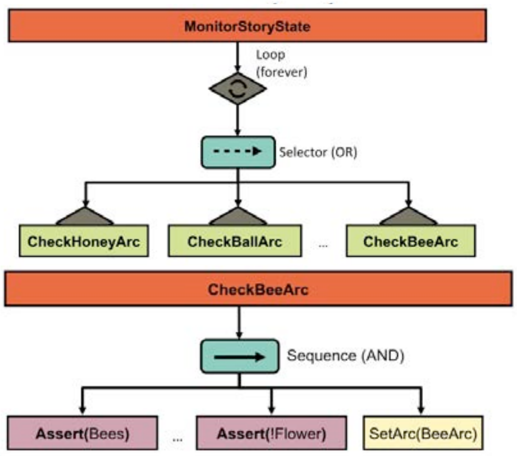
$$c(\mathbf{t}_{\mathrm{ui}}) = 1$$

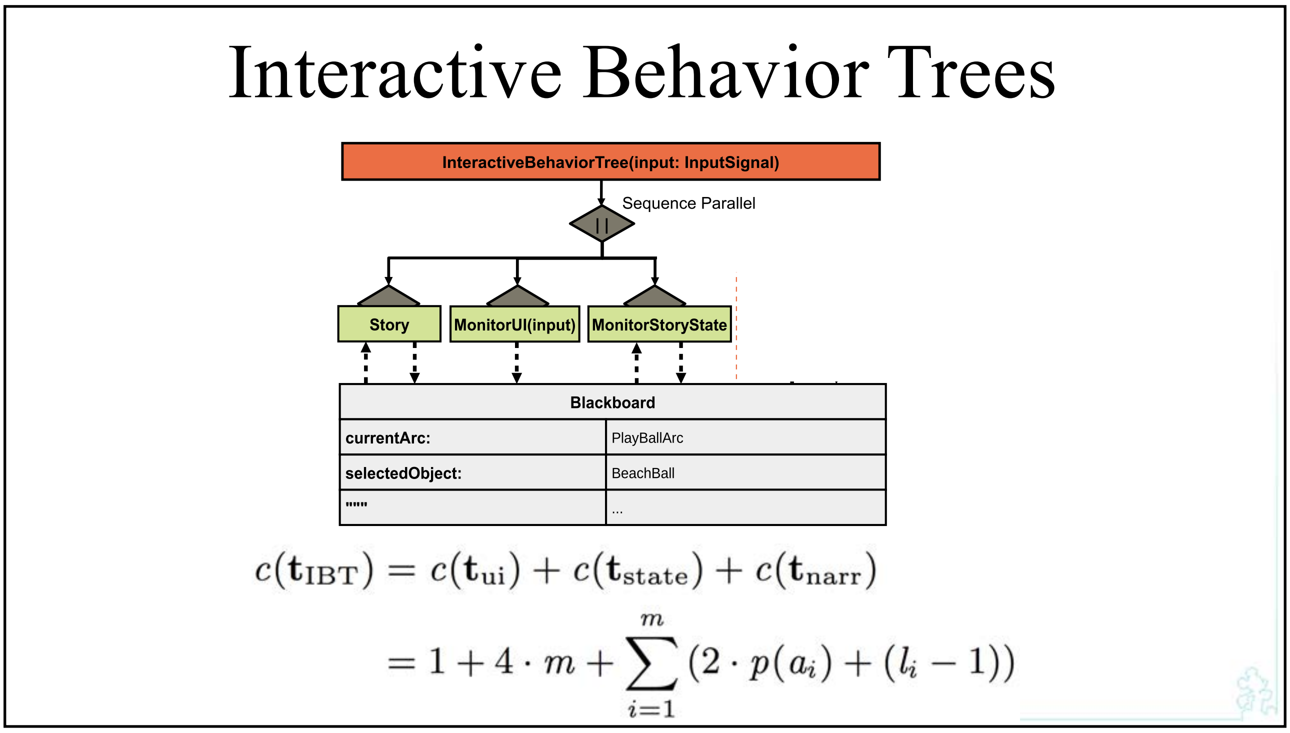


Interactive Behavior Trees

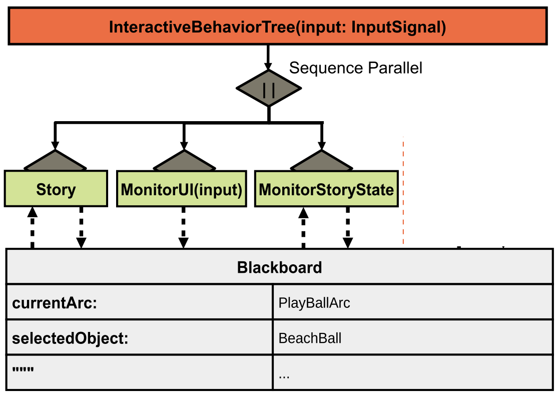
$$c(\mathbf{t}_{\text{state}}) = \sum^{m} (l_i - 1)$$







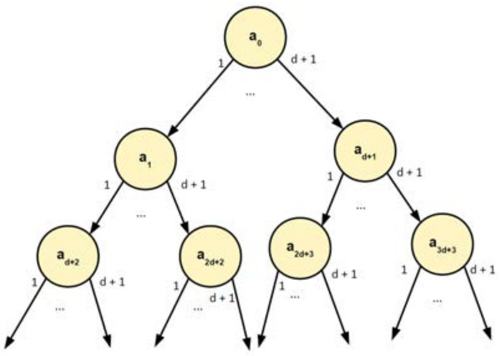
Interactive Behavior Trees

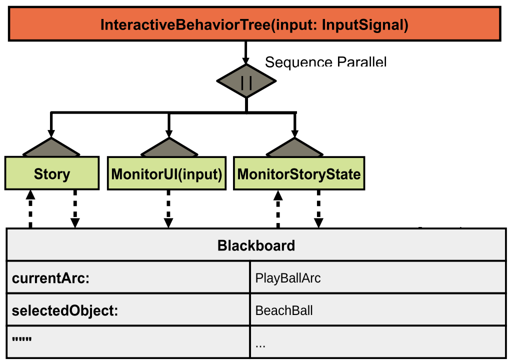


$$c(\mathbf{t}_{\text{IBT}}) = c(\mathbf{t}_{\text{ui}}) + c(\mathbf{t}_{\text{state}}) + c(\mathbf{t}_{\text{narr}})$$

$$= 1 + 4 \cdot m + \sum_{i=1}^{m} (2 \cdot p(a_i) + (l_i - 1))$$

Cyclomatic Complexity

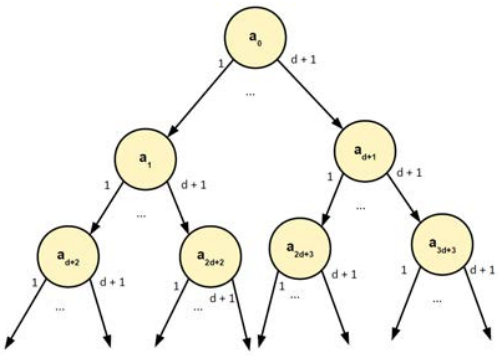




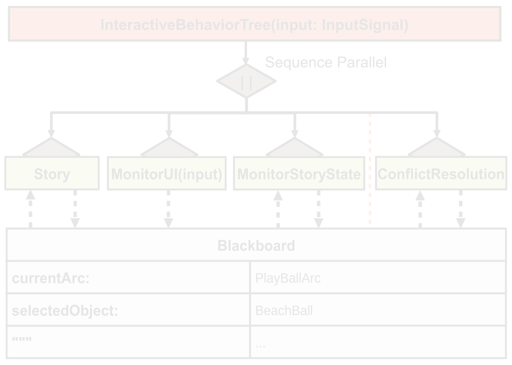
$$c(\mathbf{g}_s) = d \cdot \left(\sum_{i=1}^{m} |a_i|\right) - s(\mathbf{g}_s) + 2$$

$$c(\mathbf{t}_{\mathrm{IBT}}) = c(\mathbf{t}_{\mathrm{ui}}) + c(\mathbf{t}_{\mathrm{state}}) + c(\mathbf{t}_{\mathrm{narr}})$$
$$= 1 + 4 \cdot m + \sum_{i=1}^{m} (2 \cdot p(a_i) + (l_i - 1))$$

Cyclomatic Complexity

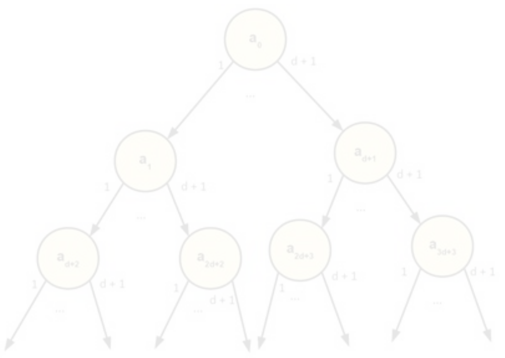


$$c(\mathbf{g}_s) = d \cdot \left(\sum_{i=1}^{m} |a_i|\right) - s(\mathbf{g}_s) + 2$$

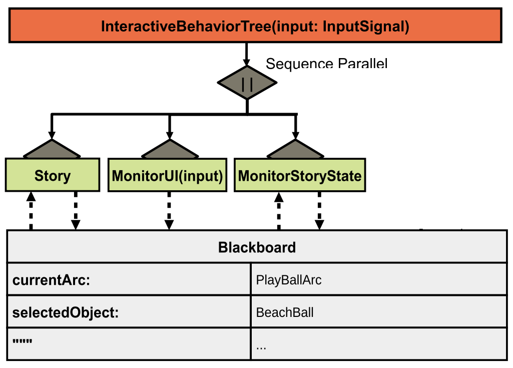


$$c(\mathbf{t}_{\mathrm{IBT}}) = c(\mathbf{t}_{\mathrm{ui}}) + c(\mathbf{t}_{\mathrm{state}}) + c(\mathbf{t}_{\mathrm{narr}})$$
$$= 1 + 4 \cdot m + \sum_{i=1}^{m} (2 \cdot p(a_i) + (l_i - 1))$$

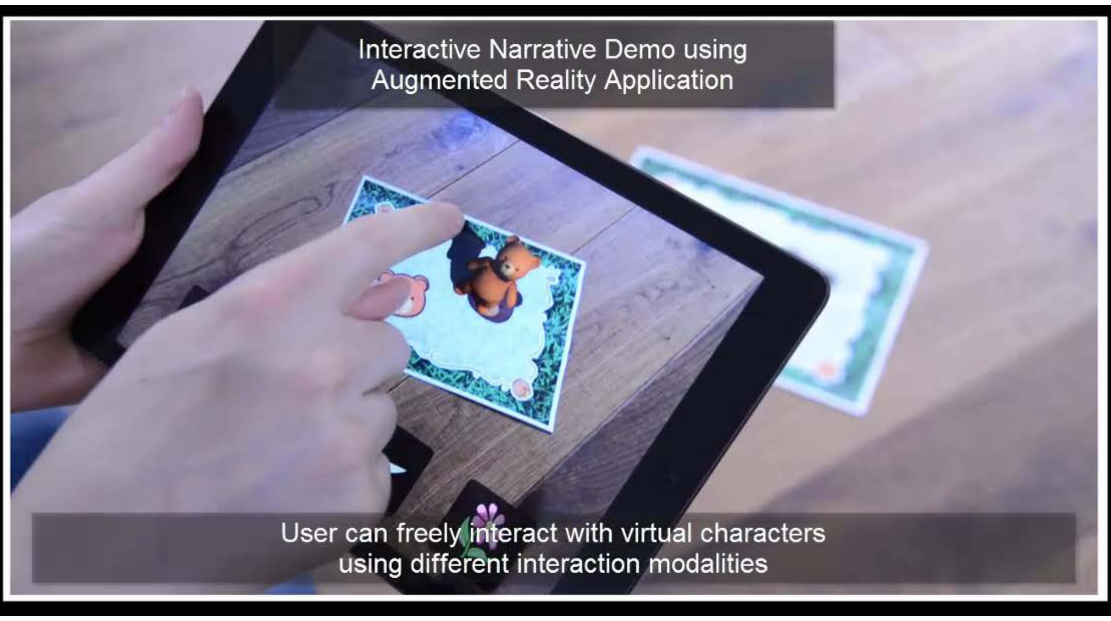
Cyclomatic Complexity



$$c(\mathbf{g}_s) = d \cdot \left(\sum_{i=1}^{m} |a_i| \right) - s(\mathbf{g}_s) + 2$$



$$c(\mathbf{t}_{\text{IBT}}) = c(\mathbf{t}_{\text{ui}}) + c(\mathbf{t}_{\text{state}}) + c(\mathbf{t}_{\text{narr}})$$
$$= 1 + 4 \cdot m + \sum_{i=1}^{m} (2 \cdot p(a_i) + (l_i - 1))$$



Interactive Narrative Demo using Augmented Reality Application
User can freely interact with virtual characters using different interaction modalities

User Study

- 12 subjects
  - 6 Experts, 6 Novices
- Task: Author same story using Story Graphs and IBT's in random order

Metrics

- **Independent Variables**
  - Authoring Method (Story Graphs, IBT's)
  - Authoring Proficiency (Expert, Novice)
- **Dependent Variables**
  - Time to author (minutes)
  - Authoring effort (number of mouse clicks)
  - Subjective difficulty (5 point Likert scale)

MANOVA Results

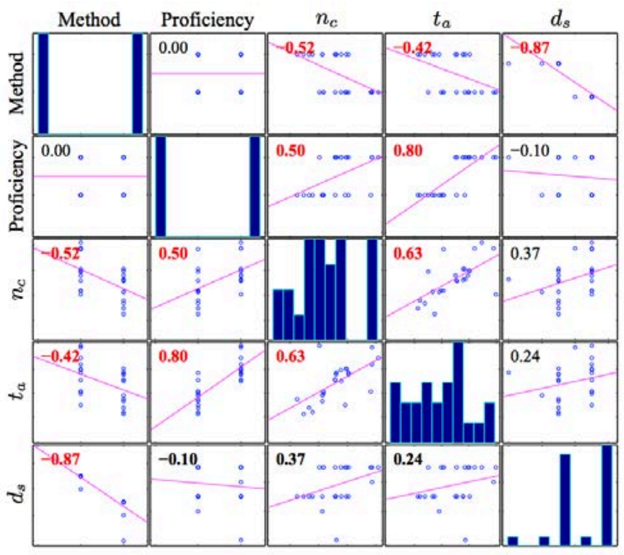
Statistically significant difference in user performance based on authoring method

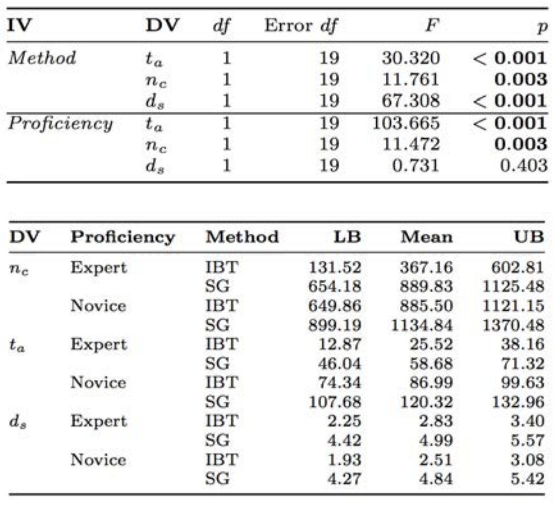
Roy’s Largest Root = 6.282, F(3,17) = 35.587, $p < 0.001$, partial $\eta^2 = 0.863$

Statistically significant difference in user performance based Proficiency

Roy’s Largest Root = 5.525, F(3,17) = 31.309, $p < 0.001$, partial $\eta^2 = 0.847$

User Study – ANOVA Results





IV	DV	*df*	Error *df*	*F*	*p*
Method	t_a	1	19	30.320	**< 0.001**
	n_c	1	19	11.761	**0.003**
	d_s	1	19	67.308	**< 0.001**
Proficiency	t_a	1	19	103.665	**< 0.001**
	n_c	1	19	11.472	**0.003**
	d_s	1	19	0.731	0.403

DV	Proficiency	Method	LB	Mean	UB
n_c	Expert	IBT	131.52	367.16	602.81
		SG	654.18	889.83	1125.48
	Novice	IBT	649.86	885.50	1121.15
		SG	899.19	1134.84	1370.48
t_a	Expert	IBT	12.87	25.52	38.16
		SG	46.04	58.68	71.32
	Novice	IBT	74.34	86.99	99.63
		SG	107.68	120.32	132.96
d_s	Expert	IBT	2.25	2.83	3.40
		SG	4.42	4.99	5.57
	Novice	IBT	1.93	2.51	3.08
		SG	4.27	4.84	5.42

Discussion

- ***Interactive Behavior Trees:*** facilitate modular authoring of interactive narratives
- ***Story graphs:*** used as a baseline for comparison
  - Comprehensive coverage with other representations across story types is important to understand benefits and tradeoffs
- ***Cyclomatic Complexity:*** a suitable static indicator of authoring complexity
  - Complementary to existing measures [Chen *et al.* 2009] that account for high-level story structure
- ***Preliminary User study:*** IBT's reduce authoring time and effort
  - *Extend scope to more users and more complex narratives*

Additional Challenges

- Inconsistent Stories
- Conflicting User Actions
- User Inactions
- Incomplete Stories

Partial-Order Planning

- **Problem Formulation**
- Partial Plan
- POP

$$\mathbf{P} = \langle \mathbf{s}_0, \mathbf{\Phi}_g, \mathbf{A} \rangle$$

$$\mathbf{A} = \{\mathbf{h}_i\}$$

Partial-Order Planning

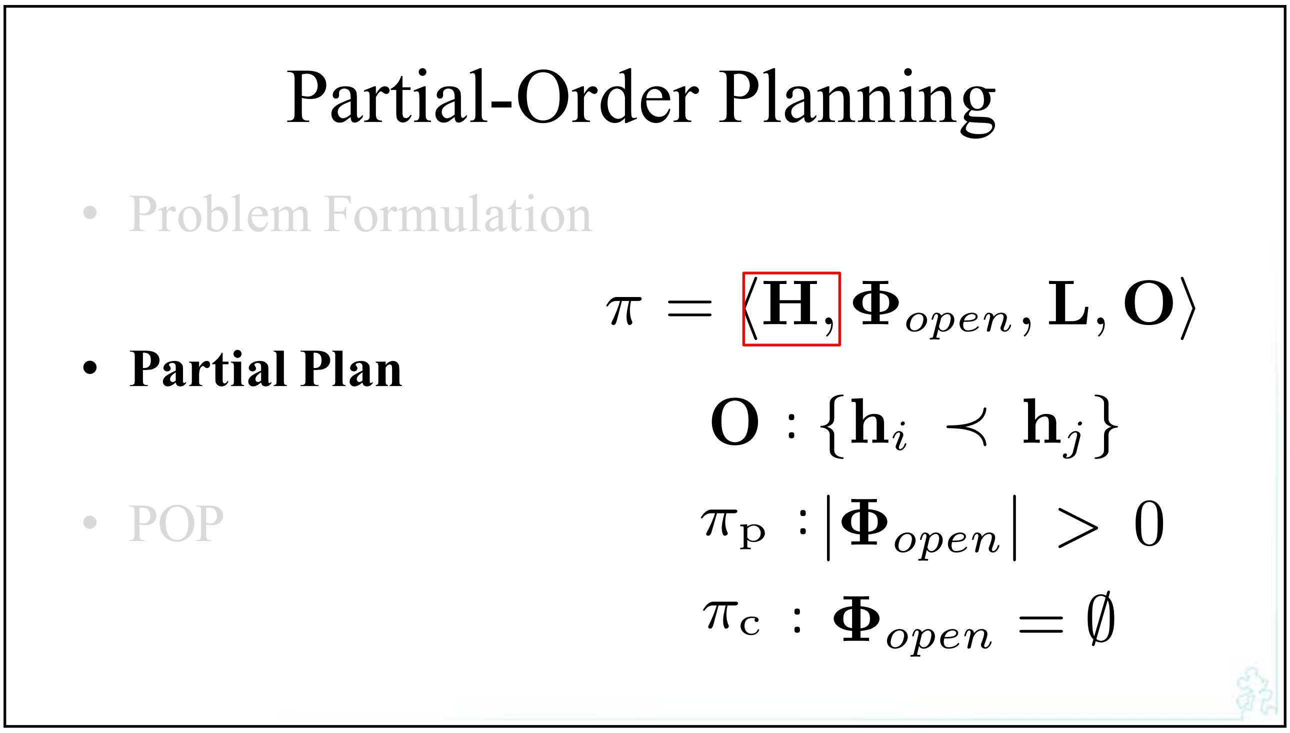
- Problem Formulation
- **Partial Plan**
- POP

$$\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \mathbf{L}, \mathbf{O} \rangle$$

$$\mathbf{O} : \{\mathbf{h}_i \prec \mathbf{h}_j\}$$

$$\pi_{\mathrm{p}} : |\mathbf{\Phi}_{open}| > 0$$

$$\pi_{\mathrm{c}} : \mathbf{\Phi}_{open} = \emptyset$$



Partial-Order Planning

- Problem Formulation
- **Partial Plan**

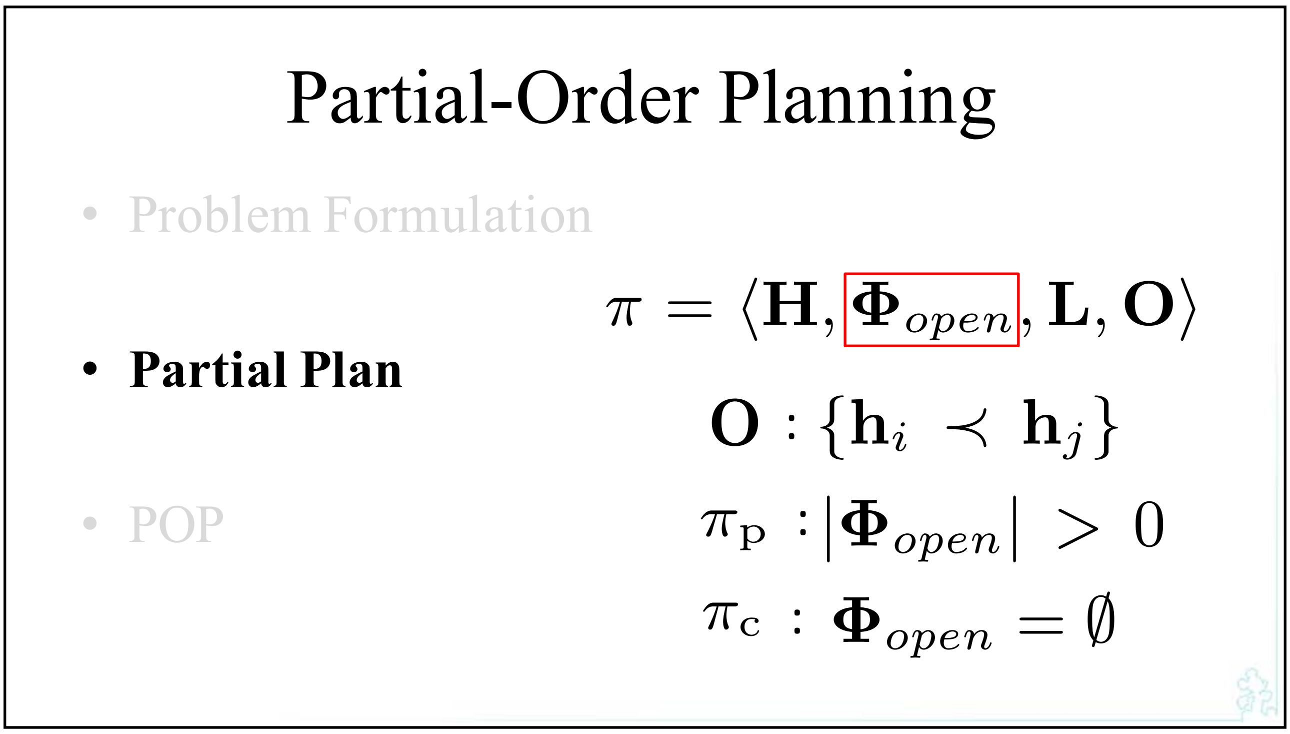
$$\pi = \langle \boxed{\mathbf{H},} \mathbf{\Phi}_{open}, \mathbf{L}, \mathbf{O} \rangle$$

$$\mathbf{O} : \{\mathbf{h}_i \prec \mathbf{h}_j\}$$

$$\pi_\mathrm{p} : |\mathbf{\Phi}_{open}| > 0$$

$$\pi_\mathrm{c} : \mathbf{\Phi}_{open} = \emptyset$$

- POP



Partial-Order Planning

- Problem Formulation
- **Partial Plan**

$$\pi = \langle \mathbf{H}, \boxed{\mathbf{\Phi}_{open}}, \mathbf{L}, \mathbf{O} \rangle$$

$$\mathbf{O} : \{\mathbf{h}_i \prec \mathbf{h}_j\}$$

$$\pi_\mathrm{p} : |\mathbf{\Phi}_{open}| > 0$$

$$\pi_\mathrm{c} : \mathbf{\Phi}_{open} = \emptyset$$

- POP

Partial-Order Planning

- Problem Formulation
- **Partial Plan**
- POP

$$\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \boxed{\mathbf{L},} \mathbf{O} \rangle$$

$$\mathbf{O} : \{\mathbf{h}_i \prec \mathbf{h}_j\}$$

$$\pi_{\mathrm{p}} : |\mathbf{\Phi}_{open}| > 0$$

$$\pi_{\mathrm{c}} : \mathbf{\Phi}_{open} = \emptyset$$

Partial-Order Planning

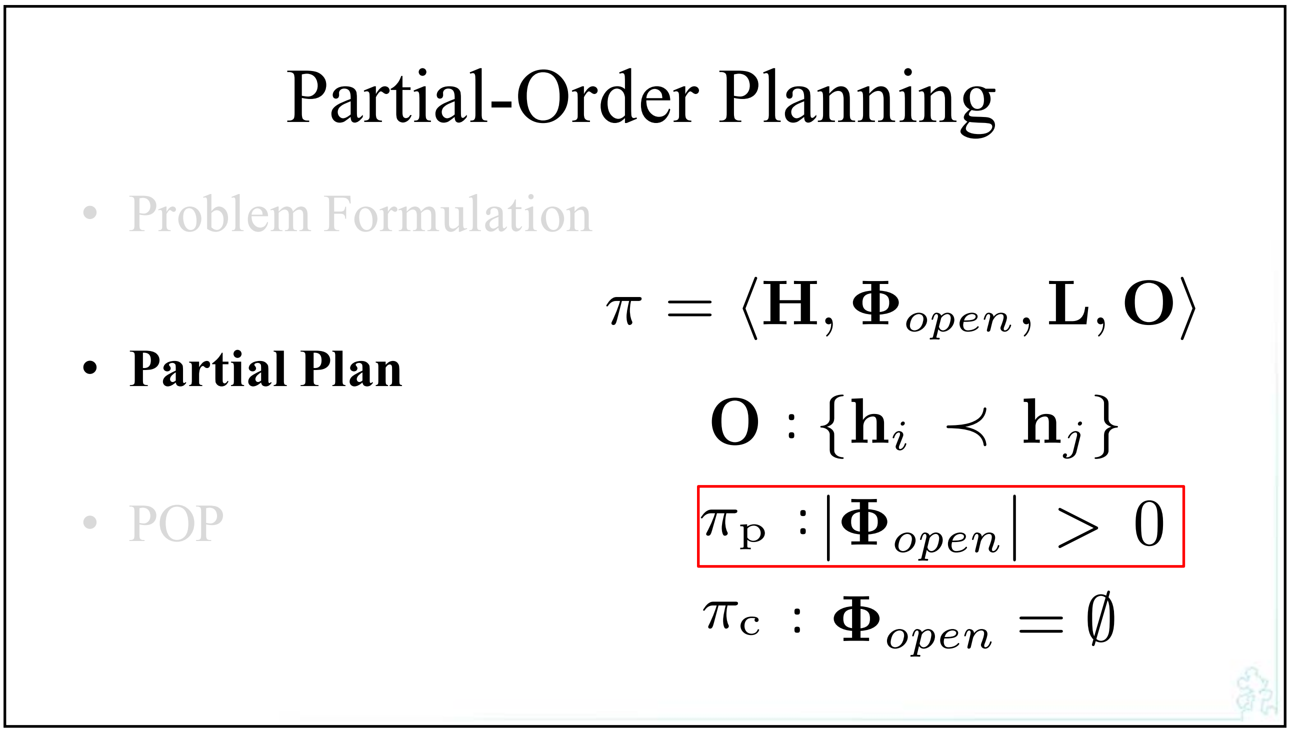
- Problem Formulation
- **Partial Plan**
- POP

$$\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \mathbf{L}, \boxed{\mathbf{O} \rangle}$$

$$\mathbf{O} : \{\mathbf{h}_i \prec \mathbf{h}_j\}$$

$$\pi_{\mathrm{p}} : |\mathbf{\Phi}_{open}| > 0$$

$$\pi_{\mathrm{c}} : \mathbf{\Phi}_{open} = \emptyset$$



Partial-Order Planning

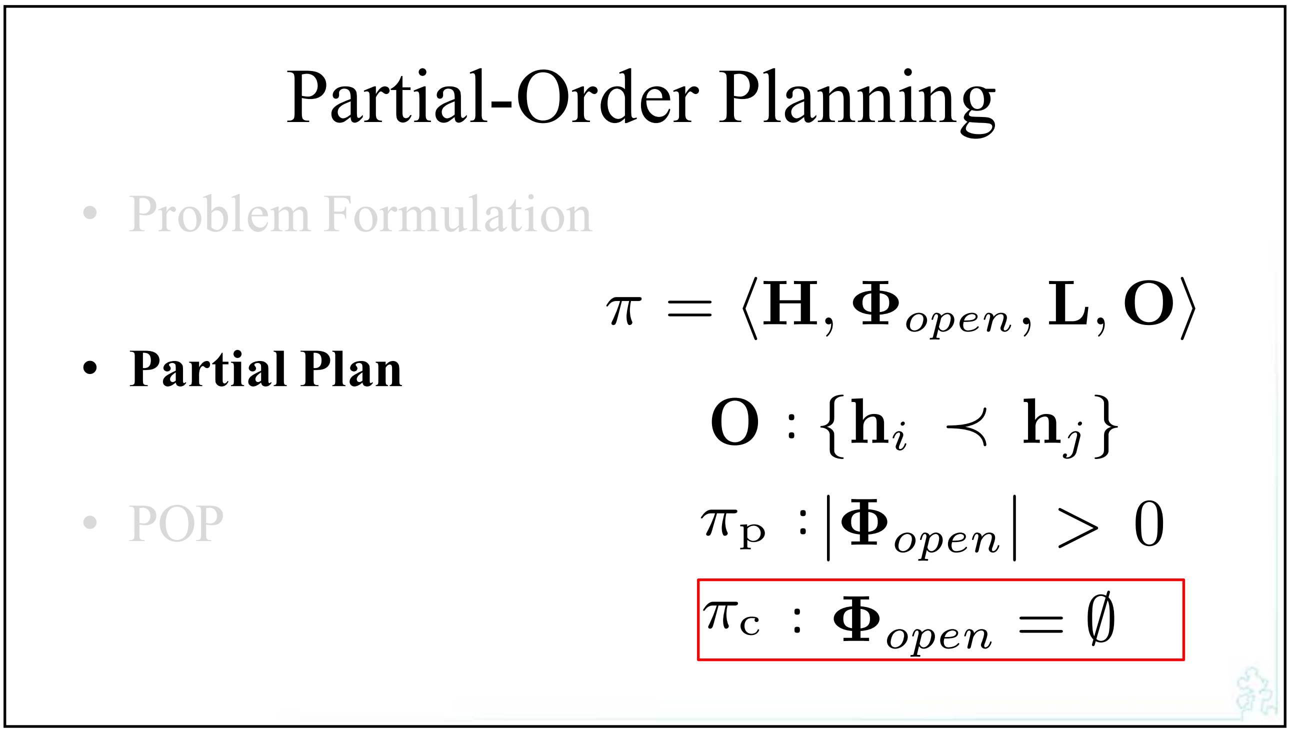
- Problem Formulation
- **Partial Plan**
- POP

$$\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \mathbf{L}, \mathbf{O} \rangle$$

$$\mathbf{O} : \{\mathbf{h}_i \prec \mathbf{h}_j\}$$

$$\boxed{\pi_{\mathrm{p}} : |\mathbf{\Phi}_{open}| > 0}$$

$$\pi_{\mathrm{c}} : \mathbf{\Phi}_{open} = \emptyset$$



Partial-Order Planning

- Problem Formulation
- **Partial Plan**
- POP

$$\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \mathbf{L}, \mathbf{O} \rangle$$

$$\mathbf{O} : \{\mathbf{h}_i \prec \mathbf{h}_j\}$$

$$\pi_{\mathrm{p}} : |\mathbf{\Phi}_{open}| > 0$$

$$\boxed{\pi_{\mathrm{c}} : \mathbf{\Phi}_{open} = \emptyset}$$

Partial-Order Planning

- Problem Formulation
- Partial Plan
- **POP**

$$\pi_c = \textbf{Plan}(\mathbf{P})$$

[Sacerdoti 1975]

Algorithm 1 Partial Order Planner

```
function Plan ($\mathbf{P} = \langle \mathrm{s}_0, \mathbf{\Phi}_g, \mathbf{A} \rangle$)
  $\mathbf{\Phi}_{open} = \{\langle \mathbf{h}_{\mathbf{\Phi}_g}, \phi \rangle | \; \forall \phi \in \mathbf{\Phi}_g\}$
  $\mathbf{H} = \{\mathbf{h}_{\mathbf{s}_0}, \mathbf{h}_{\mathbf{\Phi}_g}\}$
  $\mathbf{O} = \{\mathbf{h}_{\mathbf{s}_0} \prec \mathbf{h}_{\mathbf{\Phi}_g}\}$
  $\mathbf{L} = \emptyset$
  while $\mathbf{\Phi}_{open} \neq \emptyset$ do
    $\langle \mathbf{h}_c, \phi_c \rangle$ = SelectAndRemoveCondition($\mathbf{\Phi}_{open}$)
    if $\phi_c(\mathbf{h}) = \texttt{FALSE} \; \forall \; \mathbf{h} \in \mathbf{H}$ then
      $\mathbf{h}_s = \exists \mathbf{h} \in \mathbf{A}$ s.t. $\phi_c(\mathbf{h}) = \texttt{TRUE}$
      $\mathbf{H} = \mathbf{H} \cup \mathbf{h}_s$
      $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_{\mathbf{s}_0} \prec \mathbf{h}_s)$
      for all $\mathbf{l} \in \mathbf{L}$ do
        $\mathbf{O}$ = Protect($\mathbf{l}, \mathbf{h}_s, \mathbf{O}$)
      $\mathbf{\Phi}_{open} = \mathbf{\Phi}_{open} \cup \{\langle \mathbf{h}_s, \phi_s \rangle | \; \forall \; \phi_s \in \mathbf{\Phi}_s\}$
    else
      $\mathbf{h} = \exists \; \mathbf{h} \in \mathbf{H}$ s.t. $\phi(\mathbf{h}) = \texttt{TRUE}$
    $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_s \prec \mathbf{h}_c)$
    $\mathbf{L} = \mathbf{L} \cup \langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle$
    for all $\mathbf{h} \in \mathbf{H}$ do
      $\mathbf{O}$ = Protect($\langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle, \mathbf{h}, \mathbf{O}$)
  $\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \mathbf{L}, \mathbf{O} \rangle$
  return $\pi$
```

- Initialization
- Open Precondition Selection
- Precondition Resolution
- Causal Link Protection
- Plan Completion

Algorithm 1 Partial Order Planner

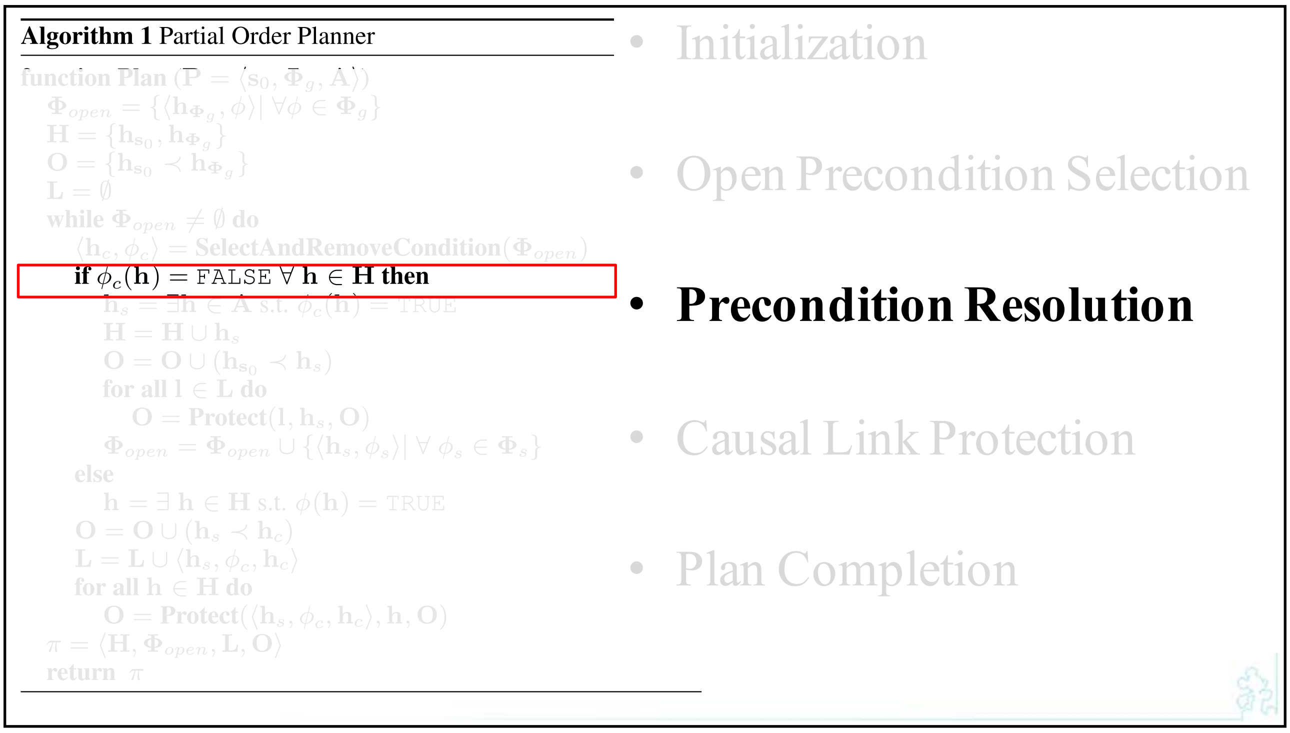
```
function Plan ($\mathbf{P} = \langle \mathbf{s}_0, \mathbf{\Phi}_g, \mathbf{A} \rangle$)
    $\mathbf{\Phi}_{open} = \{\langle \mathbf{h}_{\mathbf{\Phi}_g}, \phi \rangle | \forall \phi \in \mathbf{\Phi}_g\}$
    $\mathbf{H} = \{\mathbf{h}_{\mathbf{s}_0}, \mathbf{h}_{\mathbf{\Phi}_g}\}$
    $\mathbf{O} = \{\mathbf{h}_{\mathbf{s}_0} \prec \mathbf{h}_{\mathbf{\Phi}_g}\}$
    $\mathbf{L} = \emptyset$
    while $\Phi_{open} \neq \emptyset$ do
        $\langle \mathrm{h}_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
        if $\phi_c(\mathrm{h}) = \mathtt{FALSE}\ \forall\ \mathrm{h} \in \mathrm{H}$ then
            $\mathrm{h}_s = \exists \mathrm{h} \in \mathrm{A}$ s.t. $\phi_c(\mathrm{h}) = \mathtt{TRUE}$
            $\mathrm{H} = \mathrm{H} \cup \mathrm{h}_s$
            $\mathrm{O} = \mathrm{O} \cup (\mathrm{h}_{s_0} \prec \mathrm{h}_s)$
            for all $\mathrm{l} \in \mathrm{L}$ do
                $\mathrm{O}$ = Protect($\mathrm{l}, \mathrm{h}_s, \mathrm{O}$)
            $\Phi_{open} = \Phi_{open} \cup \{\langle \mathrm{h}_s, \phi_s \rangle | \forall \phi_s \in \Phi_s\}$
        else
            $\mathrm{h} = \exists\ \mathrm{h} \in \mathrm{H}$ s.t. $\phi(\mathrm{h}) = \mathtt{TRUE}$
        $\mathrm{O} = \mathrm{O} \cup (\mathrm{h}_s \prec \mathrm{h}_c)$
        $\mathrm{L} = \mathrm{L} \cup \langle \mathrm{h}_s, \phi_c, \mathrm{h}_c \rangle$
        for all $\mathrm{h} \in \mathrm{H}$ do
            $\mathrm{O}$ = Protect($\langle \mathrm{h}_s, \phi_c, \mathrm{h}_c \rangle, \mathrm{h}, \mathrm{O}$)
    $\pi = \langle \mathrm{H}, \Phi_{open}, \mathrm{L}, \mathrm{O} \rangle$
    return $\pi$
```

- **Initialization**
- Open Precondition Selection
- Precondition Resolution
- Causal Link Protection
- Plan Completion

Algorithm 1 Partial Order Planner

```
function Plan ($\mathrm{P} = \langle \mathrm{s}_0, \Phi_g, \mathrm{A} \rangle$)
    $\Phi_{open} = \{\langle \mathrm{h}_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
    $\mathrm{H} = \{\mathrm{h}_{\mathrm{s}_0}, \mathrm{h}_{\Phi_g}\}$
    $\mathrm{O} = \{\mathrm{h}_{\mathrm{s}_0} \prec \mathrm{h}_{\Phi_g}\}$
    $\mathrm{L} = \emptyset$
    while $\mathbf{\Phi}_{open} \neq \emptyset$ do
        $\langle \mathbf{h}_c, \phi_c \rangle$ = SelectAndRemoveCondition($\mathbf{\Phi}_{open}$)
        if $\phi_c(\mathrm{h}) = \mathtt{FALSE}\ \forall\ \mathrm{h} \in \mathrm{H}$ then
            $\mathrm{h}_s = \exists \mathrm{h} \in \mathrm{A}$ s.t. $\phi_c(\mathrm{h}) = \mathtt{TRUE}$
            $\mathrm{H} = \mathrm{H} \cup \mathrm{h}_s$
            $\mathrm{O} = \mathrm{O} \cup (\mathrm{h}_{s_0} \prec \mathrm{h}_s)$
            for all $\mathrm{l} \in \mathrm{L}$ do
                $\mathrm{O}$ = Protect($\mathrm{l}, \mathrm{h}_s, \mathrm{O}$)
            $\Phi_{open} = \Phi_{open} \cup \{\langle \mathrm{h}_s, \phi_s \rangle | \forall \phi_s \in \Phi_s\}$
        else
            $\mathrm{h} = \exists\ \mathrm{h} \in \mathrm{H}$ s.t. $\phi(\mathrm{h}) = \mathtt{TRUE}$
        $\mathrm{O} = \mathrm{O} \cup (\mathrm{h}_s \prec \mathrm{h}_c)$
        $\mathrm{L} = \mathrm{L} \cup \langle \mathrm{h}_s, \phi_c, \mathrm{h}_c \rangle$
        for all $\mathrm{h} \in \mathrm{H}$ do
            $\mathrm{O}$ = Protect($\langle \mathrm{h}_s, \phi_c, \mathrm{h}_c \rangle, \mathrm{h}, \mathrm{O}$)
    $\pi = \langle \mathrm{H}, \Phi_{open}, \mathrm{L}, \mathrm{O} \rangle$
    return $\pi$
```

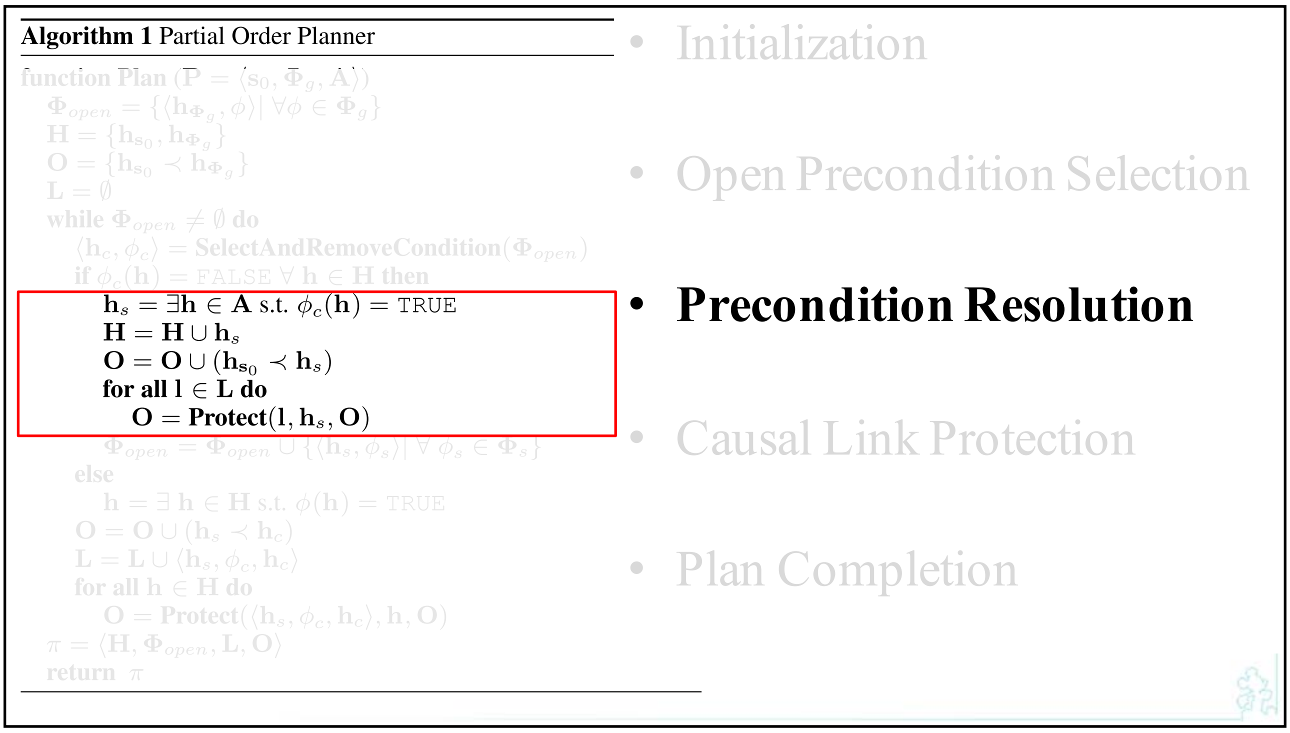
- Initialization
- **Open Precondition Selection**
- Precondition Resolution
- Causal Link Protection
- Plan Completion



Algorithm 1 Partial Order Planner

```
function Plan ($\mathbf{P} = \langle s_0, \Phi_g, \mathbf{A} \rangle$)
    $\Phi_{open} = \{\langle \mathbf{h}_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
    $\mathbf{H} = \{\mathbf{h}_{s_0}, \mathbf{h}_{\Phi_g}\}$
    $\mathbf{O} = \{\mathbf{h}_{s_0} \prec \mathbf{h}_{\Phi_g}\}$
    $\mathbf{L} = \emptyset$
    while $\Phi_{open} \neq \emptyset$ do
        $\langle \mathbf{h}_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
        if $\phi_c(\mathbf{h}) = \texttt{FALSE}\ \forall\ \mathbf{h} \in \mathbf{H}$ then
            $\mathbf{h}_s = \exists \mathbf{h} \in \mathbf{A}$ s.t. $\phi_c(\mathbf{h}) = \texttt{TRUE}$
            $\mathbf{H} = \mathbf{H} \cup \mathbf{h}_s$
            $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_{s_0} \prec \mathbf{h}_s)$
            for all $\mathbf{l} \in \mathbf{L}$ do
                $\mathbf{O}$ = Protect($\mathbf{l}, \mathbf{h}_s, \mathbf{O}$)
            $\Phi_{open} = \Phi_{open} \cup \{\langle \mathbf{h}_s, \phi_s \rangle | \forall \phi_s \in \Phi_s\}$
        else
            $\mathbf{h} = \exists\ \mathbf{h} \in \mathbf{H}$ s.t. $\phi(\mathbf{h}) = \texttt{TRUE}$
        $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_s \prec \mathbf{h}_c)$
        $\mathbf{L} = \mathbf{L} \cup \langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle$
        for all $\mathbf{h} \in \mathbf{H}$ do
            $\mathbf{O}$ = Protect($\langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle, \mathbf{h}, \mathbf{O}$)
    $\pi = \langle \mathbf{H}, \Phi_{open}, \mathbf{L}, \mathbf{O} \rangle$
    return $\pi$
```

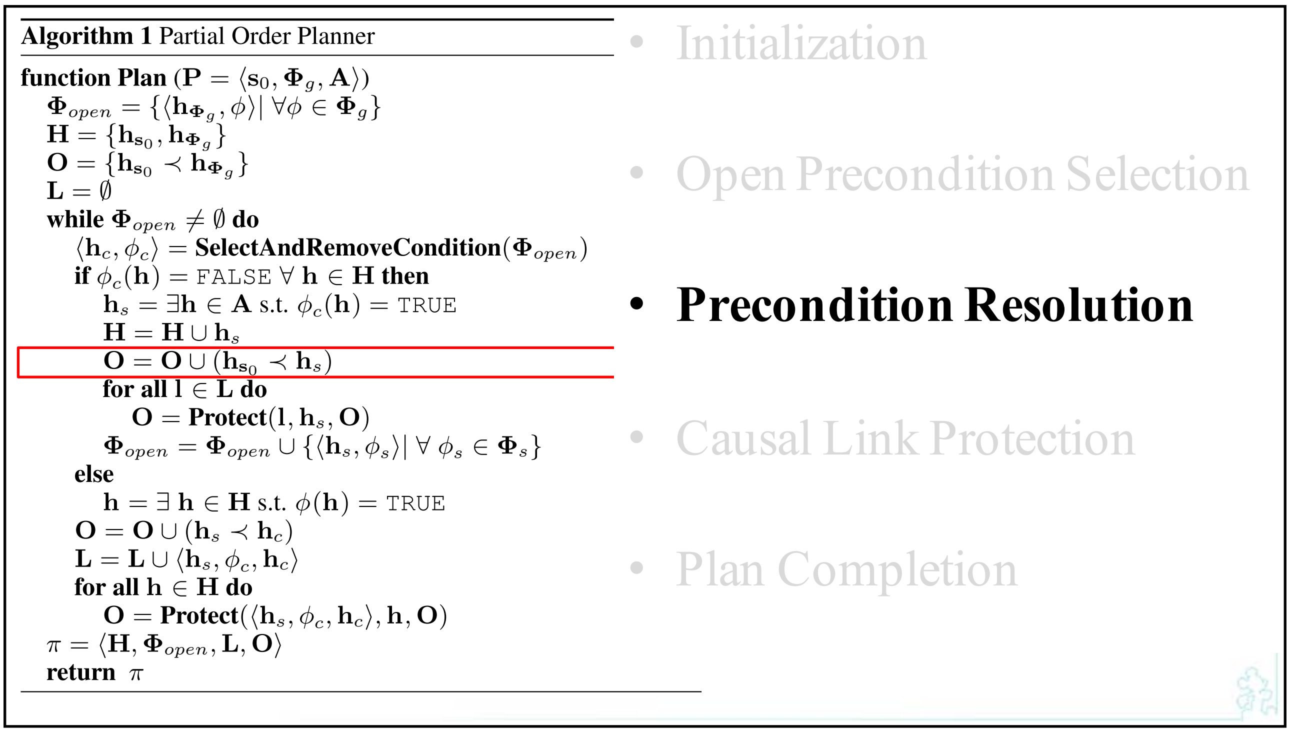
- Initialization
- Open Precondition Selection
- **Precondition Resolution**
- Causal Link Protection
- Plan Completion



Algorithm 1 Partial Order Planner

```
function Plan ($\mathbf{P} = \langle s_0, \Phi_g, \mathbf{A} \rangle$)
    $\Phi_{open} = \{\langle \mathbf{h}_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
    $\mathbf{H} = \{\mathbf{h}_{s_0}, \mathbf{h}_{\Phi_g}\}$
    $\mathbf{O} = \{\mathbf{h}_{s_0} \prec \mathbf{h}_{\Phi_g}\}$
    $\mathbf{L} = \emptyset$
    while $\Phi_{open} \neq \emptyset$ do
        $\langle \mathbf{h}_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
        if $\phi_c(\mathbf{h}) = \texttt{FALSE}\ \forall\ \mathbf{h} \in \mathbf{H}$ then
            $\mathbf{h}_s = \exists \mathbf{h} \in \mathbf{A}$ s.t. $\phi_c(\mathbf{h}) = \texttt{TRUE}$
            $\mathbf{H} = \mathbf{H} \cup \mathbf{h}_s$
            $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_{s_0} \prec \mathbf{h}_s)$
            for all $\mathbf{l} \in \mathbf{L}$ do
                $\mathbf{O}$ = Protect($\mathbf{l}, \mathbf{h}_s, \mathbf{O}$)
            $\Phi_{open} = \Phi_{open} \cup \{\langle \mathbf{h}_s, \phi_s \rangle | \forall \phi_s \in \Phi_s\}$
        else
            $\mathbf{h} = \exists\ \mathbf{h} \in \mathbf{H}$ s.t. $\phi(\mathbf{h}) = \texttt{TRUE}$
        $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_s \prec \mathbf{h}_c)$
        $\mathbf{L} = \mathbf{L} \cup \langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle$
        for all $\mathbf{h} \in \mathbf{H}$ do
            $\mathbf{O}$ = Protect($\langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle, \mathbf{h}, \mathbf{O}$)
    $\pi = \langle \mathbf{H}, \Phi_{open}, \mathbf{L}, \mathbf{O} \rangle$
    return $\pi$
```

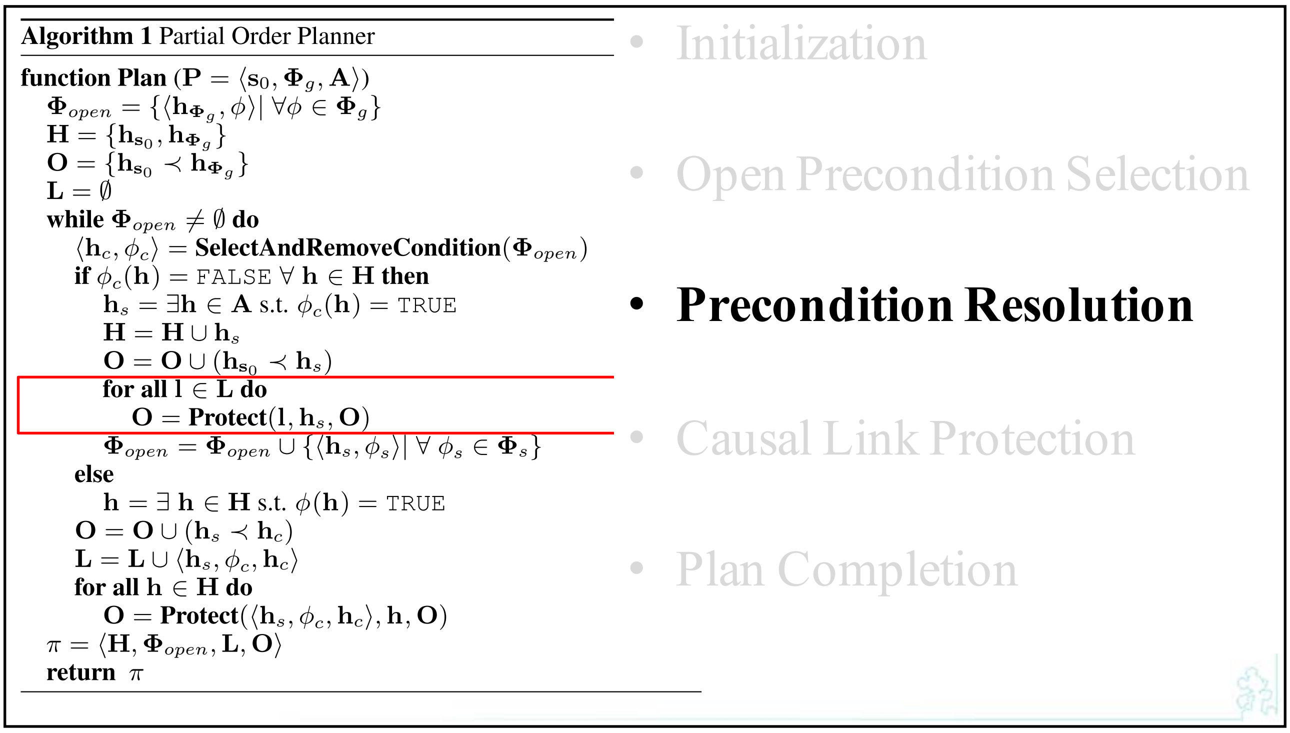
- Initialization
- Open Precondition Selection
- **Precondition Resolution**
- Causal Link Protection
- Plan Completion



Algorithm 1 Partial Order Planner

```
function Plan ($\mathbf{P} = \langle s_0, \mathbf{\Phi}_g, \mathbf{A} \rangle$)
    $\mathbf{\Phi}_{open} = \{\langle \mathbf{h}_{\mathbf{\Phi}_g}, \phi \rangle | \ \forall \phi \in \mathbf{\Phi}_g\}$
    $\mathbf{H} = \{\mathbf{h}_{s_0}, \mathbf{h}_{\mathbf{\Phi}_g}\}$
    $\mathbf{O} = \{\mathbf{h}_{s_0} \prec \mathbf{h}_{\mathbf{\Phi}_g}\}$
    $\mathbf{L} = \emptyset$
    while $\mathbf{\Phi}_{open} \neq \emptyset$ do
        $\langle \mathbf{h}_c, \phi_c \rangle$ = SelectAndRemoveCondition($\mathbf{\Phi}_{open}$)
        if $\phi_c(\mathbf{h}) = \texttt{FALSE} \ \forall \ \mathbf{h} \in \mathbf{H}$ then
            $\mathbf{h}_s = \exists \mathbf{h} \in \mathbf{A}$ s.t. $\phi_c(\mathbf{h}) = \texttt{TRUE}$
            $\mathbf{H} = \mathbf{H} \cup \mathbf{h}_s$
            $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_{s_0} \prec \mathbf{h}_s)$
            for all $\mathbf{l} \in \mathbf{L}$ do
                $\mathbf{O}$ = Protect($\mathbf{l}, \mathbf{h}_s, \mathbf{O}$)
            $\mathbf{\Phi}_{open} = \mathbf{\Phi}_{open} \cup \{\langle \mathbf{h}_s, \phi_s \rangle | \ \forall \ \phi_s \in \mathbf{\Phi}_s\}$
        else
            $\mathbf{h} = \exists \ \mathbf{h} \in \mathbf{H}$ s.t. $\phi(\mathbf{h}) = \texttt{TRUE}$
        $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_s \prec \mathbf{h}_c)$
        $\mathbf{L} = \mathbf{L} \cup \langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle$
        for all $\mathbf{h} \in \mathbf{H}$ do
            $\mathbf{O}$ = Protect($\langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle, \mathbf{h}, \mathbf{O}$)
    $\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \mathbf{L}, \mathbf{O} \rangle$
    return $\pi$
```

- Initialization
- Open Precondition Selection
- **Precondition Resolution**
- Causal Link Protection
- Plan Completion



Algorithm 1 Partial Order Planner

```
function Plan ($\mathbf{P} = \langle s_0, \mathbf{\Phi}_g, \mathbf{A} \rangle$)
    $\mathbf{\Phi}_{open} = \{\langle \mathbf{h}_{\mathbf{\Phi}_g}, \phi \rangle | \ \forall \phi \in \mathbf{\Phi}_g\}$
    $\mathbf{H} = \{\mathbf{h}_{s_0}, \mathbf{h}_{\mathbf{\Phi}_g}\}$
    $\mathbf{O} = \{\mathbf{h}_{s_0} \prec \mathbf{h}_{\mathbf{\Phi}_g}\}$
    $\mathbf{L} = \emptyset$
    while $\mathbf{\Phi}_{open} \neq \emptyset$ do
        $\langle \mathbf{h}_c, \phi_c \rangle$ = SelectAndRemoveCondition($\mathbf{\Phi}_{open}$)
        if $\phi_c(\mathbf{h}) = \texttt{FALSE} \ \forall \ \mathbf{h} \in \mathbf{H}$ then
            $\mathbf{h}_s = \exists \mathbf{h} \in \mathbf{A}$ s.t. $\phi_c(\mathbf{h}) = \texttt{TRUE}$
            $\mathbf{H} = \mathbf{H} \cup \mathbf{h}_s$
            $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_{s_0} \prec \mathbf{h}_s)$
            for all $\mathbf{l} \in \mathbf{L}$ do
                $\mathbf{O}$ = Protect($\mathbf{l}, \mathbf{h}_s, \mathbf{O}$)
            $\mathbf{\Phi}_{open} = \mathbf{\Phi}_{open} \cup \{\langle \mathbf{h}_s, \phi_s \rangle | \ \forall \ \phi_s \in \mathbf{\Phi}_s\}$
        else
            $\mathbf{h} = \exists \ \mathbf{h} \in \mathbf{H}$ s.t. $\phi(\mathbf{h}) = \texttt{TRUE}$
        $\mathbf{O} = \mathbf{O} \cup (\mathbf{h}_s \prec \mathbf{h}_c)$
        $\mathbf{L} = \mathbf{L} \cup \langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle$
        for all $\mathbf{h} \in \mathbf{H}$ do
            $\mathbf{O}$ = Protect($\langle \mathbf{h}_s, \phi_c, \mathbf{h}_c \rangle, \mathbf{h}, \mathbf{O}$)
    $\pi = \langle \mathbf{H}, \mathbf{\Phi}_{open}, \mathbf{L}, \mathbf{O} \rangle$
    return $\pi$
```

- Initialization
- Open Precondition Selection
- **Precondition Resolution**
- Causal Link Protection
- Plan Completion

Algorithm 1 Partial Order Planner

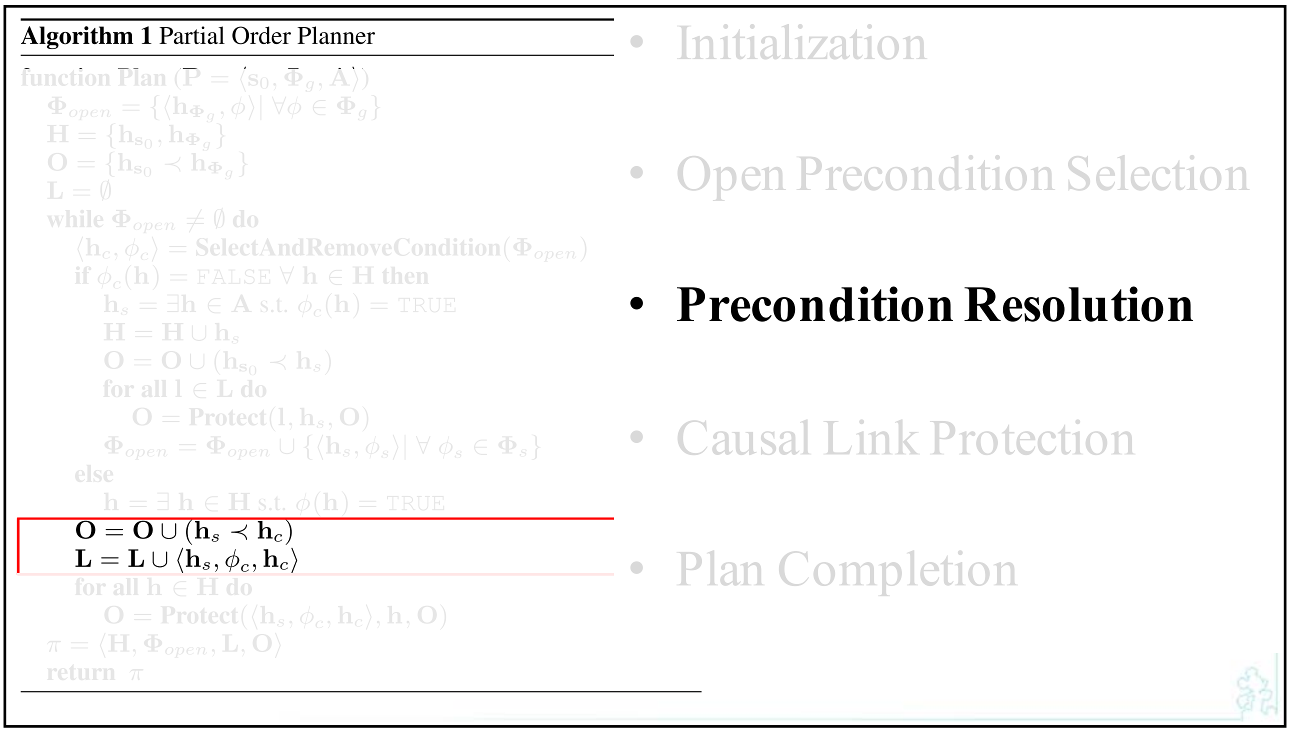
```
function Plan ($P = \langle s_0, \Phi_g, A \rangle$)
  $\Phi_{open} = \{\langle h_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
  $H = \{h_{s_0}, h_{\Phi_g}\}$
  $O = \{h_{s_0} \prec h_{\Phi_g}\}$
  $L = \emptyset$
  while $\Phi_{open} \neq \emptyset$ do
    $\langle h_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
    if $\phi_c(h) = \texttt{FALSE}\ \forall\ h \in H$ then
      $h_s = \exists h \in A$ s.t. $\phi_c(h) = \texttt{TRUE}$
      $H = H \cup h_s$
      $O = O \cup (h_{s_0} \prec h_s)$
      for all $l \in L$ do
        $O$ = Protect($l, h_s, O$)
      $\Phi_{open} = \Phi_{open} \cup \{\langle h_s, \phi_s \rangle | \forall\ \phi_s \in \Phi_s\}$
    else
      $h = \exists\ h \in H$ s.t. $\phi(h) = \texttt{TRUE}$
    $O = O \cup (h_s \prec h_c)$
    $L = L \cup \langle h_s, \phi_c, h_c \rangle$
    for all $h \in H$ do
      $O$ = Protect($\langle h_s, \phi_c, h_c \rangle, h, O$)
  $\pi = \langle H, \Phi_{open}, L, O \rangle$
  return $\pi$
```

- Initialization
- Open Precondition Selection
- **Precondition Resolution**
- Causal Link Protection
- Plan Completion

Algorithm 1 Partial Order Planner

```
function Plan ($P = \langle s_0, \Phi_g, A \rangle$)
  $\Phi_{open} = \{\langle h_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
  $H = \{h_{s_0}, h_{\Phi_g}\}$
  $O = \{h_{s_0} \prec h_{\Phi_g}\}$
  $L = \emptyset$
  while $\Phi_{open} \neq \emptyset$ do
    $\langle h_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
    if $\phi_c(h) = \texttt{FALSE}\ \forall\ h \in H$ then
      $h_s = \exists h \in A$ s.t. $\phi_c(h) = \texttt{TRUE}$
      $H = H \cup h_s$
      $O = O \cup (h_{s_0} \prec h_s)$
      for all $l \in L$ do
        $O$ = Protect($l, h_s, O$)
      $\Phi_{open} = \Phi_{open} \cup \{\langle h_s, \phi_s \rangle | \forall\ \phi_s \in \Phi_s\}$
    else
      $h = \exists\ h \in H$ s.t. $\phi(h) = \texttt{TRUE}$
    $O = O \cup (h_s \prec h_c)$
    $L = L \cup \langle h_s, \phi_c, h_c \rangle$
    for all $h \in H$ do
      $O$ = Protect($\langle h_s, \phi_c, h_c \rangle, h, O$)
  $\pi = \langle H, \Phi_{open}, L, O \rangle$
  return $\pi$
```

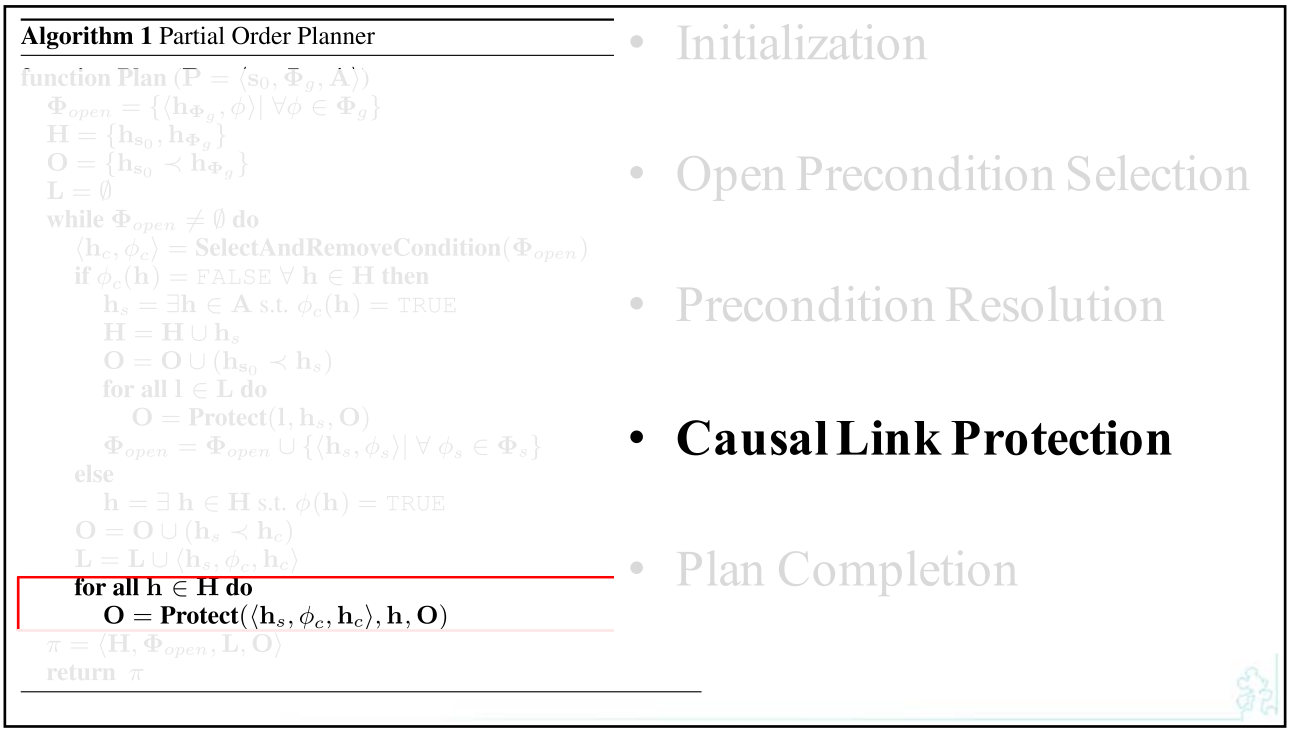
- Initialization
- Open Precondition Selection
- **Precondition Resolution**
- Causal Link Protection
- Plan Completion



Algorithm 1 Partial Order Planner

```
function Plan ($P = \langle s_0, \Phi_g, A \rangle$)
  $\Phi_{open} = \{\langle h_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
  $H = \{h_{s_0}, h_{\Phi_g}\}$
  $O = \{h_{s_0} \prec h_{\Phi_g}\}$
  $L = \emptyset$
  while $\Phi_{open} \neq \emptyset$ do
    $\langle h_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
    if $\phi_c(h) = \texttt{FALSE}\ \forall\ h \in H$ then
      $h_s = \exists h \in A$ s.t. $\phi_c(h) = \texttt{TRUE}$
      $H = H \cup h_s$
      $O = O \cup (h_{s_0} \prec h_s)$
      for all $l \in L$ do
        $O$ = Protect($l, h_s, O$)
      $\Phi_{open} = \Phi_{open} \cup \{\langle h_s, \phi_s \rangle | \forall\ \phi_s \in \Phi_s\}$
    else
      $h = \exists\ h \in H$ s.t. $\phi(h) = \texttt{TRUE}$
    $O = O \cup (h_s \prec h_c)$
    $L = L \cup \langle h_s, \phi_c, h_c \rangle$
    for all $h \in H$ do
      $O$ = Protect($\langle h_s, \phi_c, h_c \rangle, h, O$)
  $\pi = \langle H, \Phi_{open}, L, O \rangle$
  return $\pi$
```

- Initialization
- Open Precondition Selection
- **Precondition Resolution**
- Causal Link Protection
- Plan Completion



Algorithm 1 Partial Order Planner

```
function Plan ($P = \langle s_0, \Phi_g, A \rangle$)
  $\Phi_{open} = \{\langle h_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
  $H = \{h_{s_0}, h_{\Phi_g}\}$
  $O = \{h_{s_0} \prec h_{\Phi_g}\}$
  $L = \emptyset$
  while $\Phi_{open} \neq \emptyset$ do
    $\langle h_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
    if $\phi_c(h) = \texttt{FALSE}\ \forall\ h \in H$ then
      $h_s = \exists h \in A$ s.t. $\phi_c(h) = \texttt{TRUE}$
      $H = H \cup h_s$
      $O = O \cup (h_{s_0} \prec h_s)$
      for all $l \in L$ do
        $O$ = Protect($l, h_s, O$)
      $\Phi_{open} = \Phi_{open} \cup \{\langle h_s, \phi_s \rangle | \forall\ \phi_s \in \Phi_s\}$
    else
      $h = \exists\ h \in H$ s.t. $\phi(h) = \texttt{TRUE}$
    $O = O \cup (h_s \prec h_c)$
    $L = L \cup \langle h_s, \phi_c, h_c \rangle$
    for all $h \in H$ do
      $O$ = Protect($\langle h_s, \phi_c, h_c \rangle, h, O$)
  $\pi = \langle H, \Phi_{open}, L, O \rangle$
  return $\pi$
```

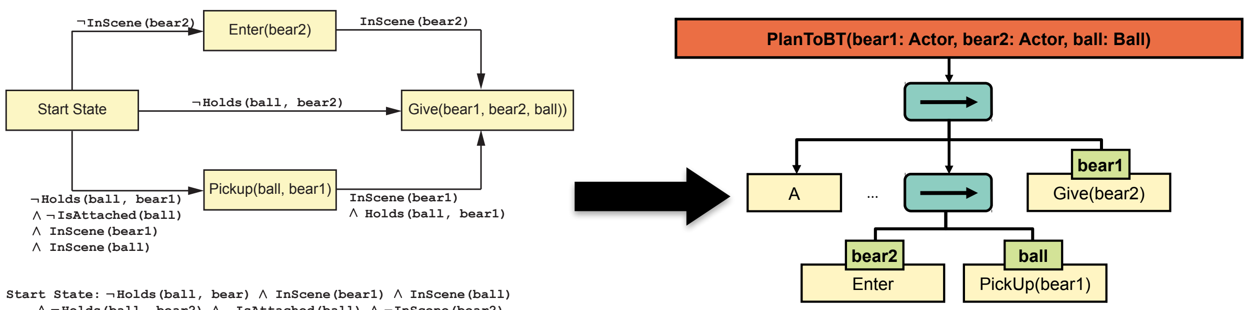
- Initialization
- Open Precondition Selection
- Precondition Resolution
- **Causal Link Protection**
- Plan Completion

Algorithm 1 Partial Order Planner

```
function Plan ($P = \langle s_0, \Phi_g, A \rangle$)
  $\Phi_{open} = \{\langle h_{\Phi_g}, \phi \rangle | \forall \phi \in \Phi_g\}$
  $H = \{h_{s_0}, h_{\Phi_g}\}$
  $O = \{h_{s_0} \prec h_{\Phi_g}\}$
  $L = \emptyset$
  while $\Phi_{open} \neq \emptyset$ do
    $\langle h_c, \phi_c \rangle$ = SelectAndRemoveCondition($\Phi_{open}$)
    if $\phi_c(h)$ = FALSE $\forall$ h $\in$ H then
      $h_s = \exists h \in A$ s.t. $\phi_c(h)$ = TRUE
      $H = H \cup h_s$
      $O = O \cup (h_{s_0} \prec h_s)$
      for all $l \in L$ do
        $O$ = Protect($l, h_s, O$)
      $\Phi_{open} = \Phi_{open} \cup \{\langle h_s, \phi_s \rangle | \forall \phi_s \in \Phi_s\}$
    else
      $h = \exists h \in H$ s.t. $\phi(h)$ = TRUE
    $O = O \cup (h_s \prec h_c)$
    $L = L \cup \langle h_s, \phi_c, h_c \rangle$
    for all $h \in H$ do
      $O$ = Protect($\langle h_s, \phi_c, h_c \rangle, h, O$)
  $\pi = \langle H, \Phi_{open}, L, O \rangle$
  return $\pi$
```

- Initialization
- Open Precondition Selection
- Precondition Resolution
- Causal Link Protection
- **Plan Completion**

Plan Integration


¬InScene(bear2)
Enter(bear2)
InScene(bear2)
Start State
¬Holds(ball, bear2)
Give(bear1, bear2, ball))
Pickup(ball, bear1)
¬Holds(ball, bear1) ∧ ¬IsAttached(ball) ∧ InScene(bear1) ∧ InScene(ball)
InScene(bear1) ∧ Holds(ball, bear1)
PlanToBT(bear1: Actor, bear2: Actor, ball: Ball)
A
...
bear1
Give(bear2)
bear2
Enter
ball
PickUp(bear1)

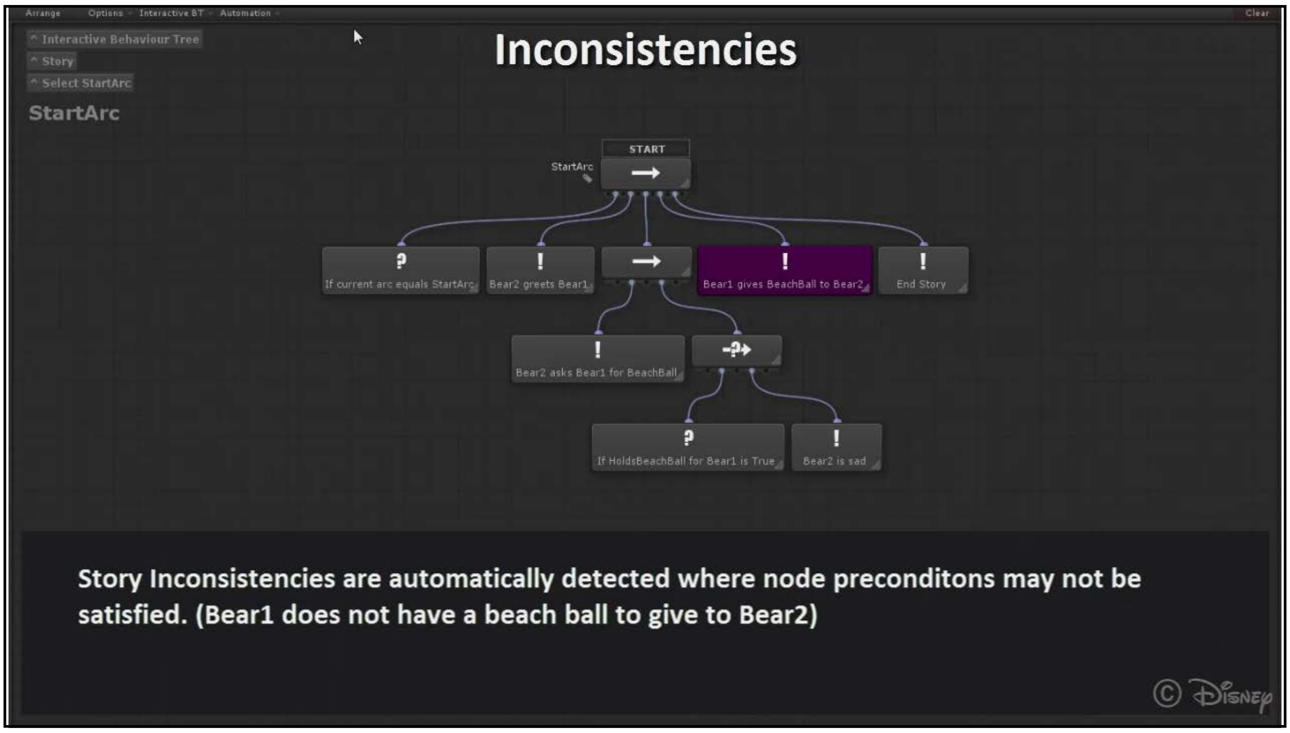
```
Start State: ¬Holds(ball, bear) ∧ InScene(bear1) ∧ InScene(ball)
    ∧ ¬Holds(ball, bear2) ∧ ¬IsAttached(ball) ∧ ¬InScene(bear2)

Goal State: Holds(ball, bear1) ∧ InScene(bear1)
    ∧ ¬Holds(ball, Bear2) ∧ InScene(bear2)
```

Additional Challenges

- **Inconsistent Stories**
- Conflicting User Actions
- User Inactions
- Incomplete Stories

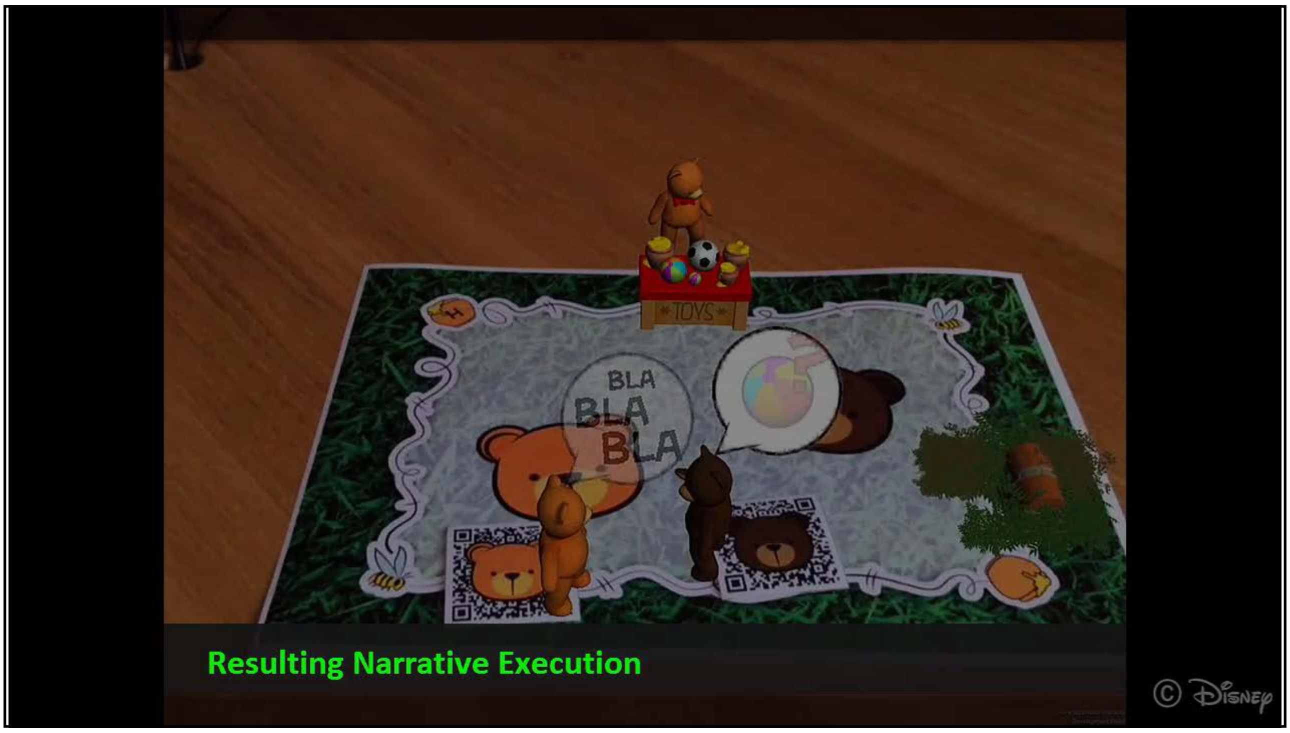
Algorithm 2 Inconsistency Detection Resolution

```
function DetectAndResolveInconsistencies ($\mathbf{t}_{\mathrm{IBT}}$)
    for all $\mathbf{t}^{\mathrm{arc}} \in \mathbf{t}_{\mathrm{narr}}$ do
        for all $\mathbf{t} \in \mathbf{t}^{\mathrm{arc}}$ do
            $\mathbf{b} =$ ComputeBeliefState$(\mathbf{t}, \mathbf{t}_{\mathrm{IBT}})$
            for all $\mathbf{s} \in \mathbf{b}$ do
                if $\mathbf{\Phi_t}(\mathbf{s}) == \mathtt{FALSE}$ then
                    $\mathbf{P} = \langle \mathbf{s}, \mathbf{\Phi_t}, \mathbf{A} \rangle$
                    $\pi =$ Plan$(\mathbf{P})$
                    $\mathbf{t}_{\mathrm{IBT}} =$ IntegratePlan$(\pi, \mathbf{t}, \mathbf{t}_{\mathrm{IBT}})$
    return $\mathbf{b}$
```


Arrange
Options
Interactive BT
Automation
Clear
Interactive Behaviour Tree
Story
Select StartArc
StartArc
START
StartArc
If current arc equals StartArc
Bear2 greets Bear1
Bear1 gives BeachBall to Bear2
End Story
Bear2 asks Bear1 for BeachBall
If HoldsBeachBall for Bear1 is True
Bear2 is sad
© Disney

Inconsistencies

Story Inconsistencies are automatically detected where node preconditons may not be satisfied. (Bear1 does not have a beach ball to give to Bear2)



Resulting Narrative Execution

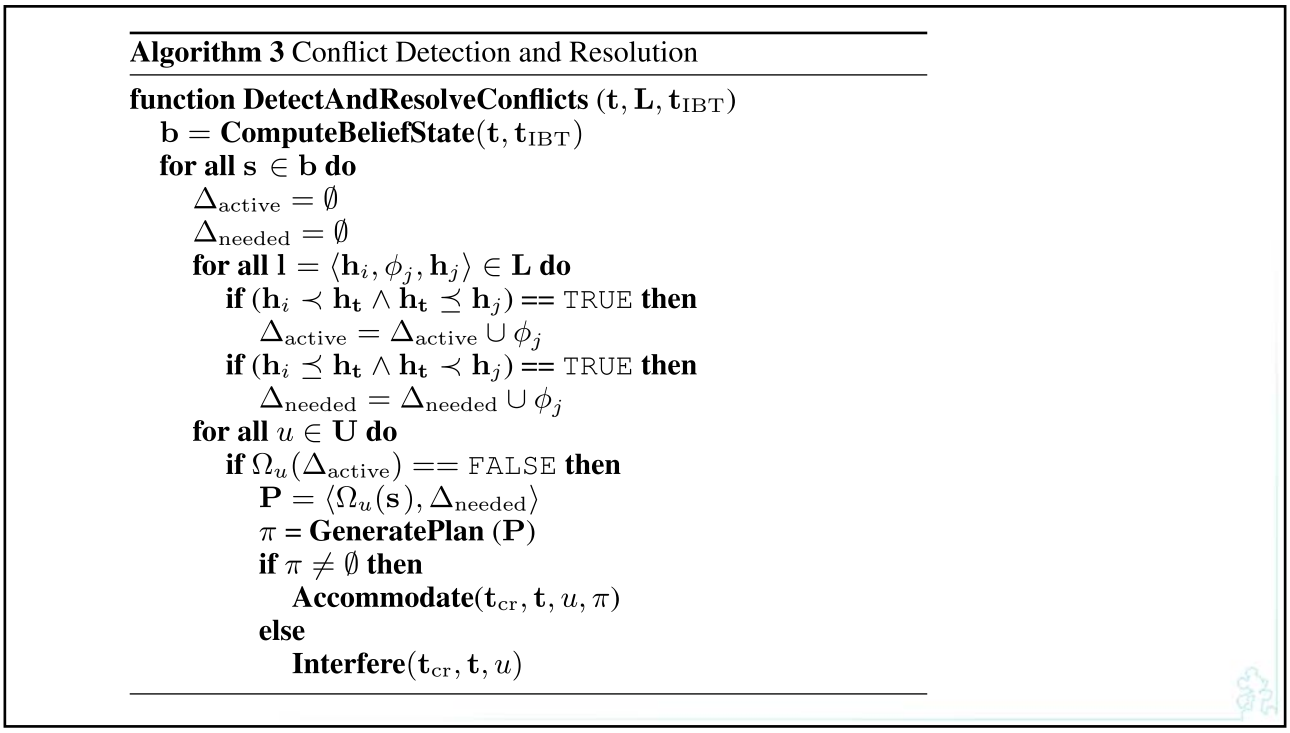
Additional Challenges

- Inconsistent Stories
- **Conflicting User Actions**
- User Inactions
- Incomplete Stories

Challenges

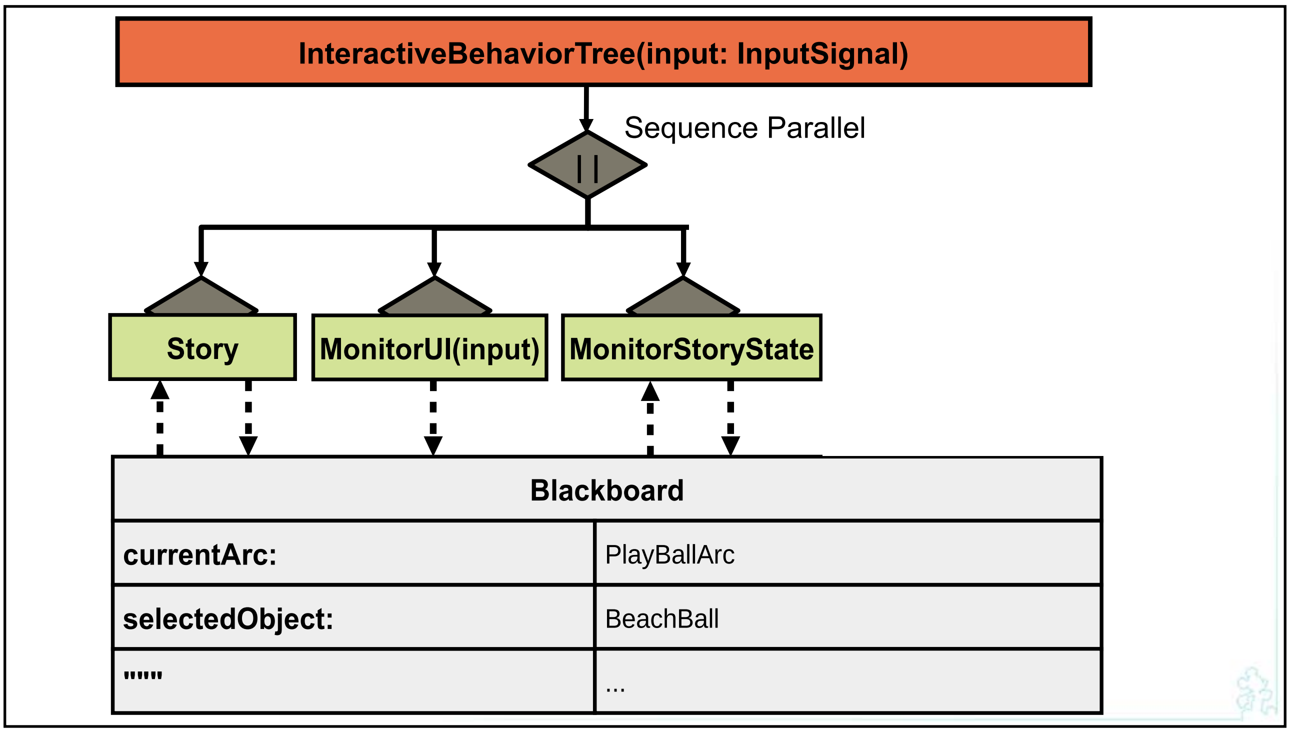
- Inconsistent Stories
- **Conflicting User Actions**
- User Inactions
- Incomplete Stories

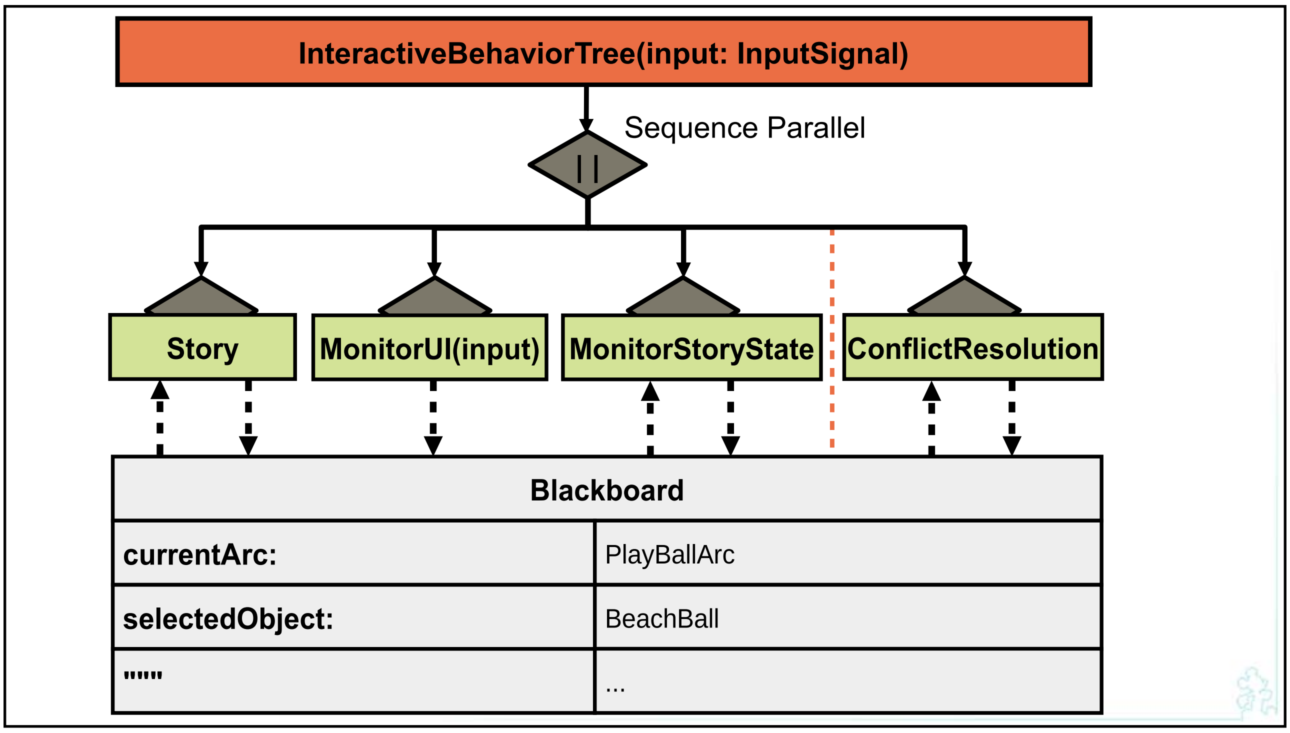
$$\begin{tikzcd}
 & \text{Accommodation} \\
\textbf{Conflicting User Actions} \arrow[ur] \arrow[dr] & \\
 & \text{Interference}
\end{tikzcd}$$

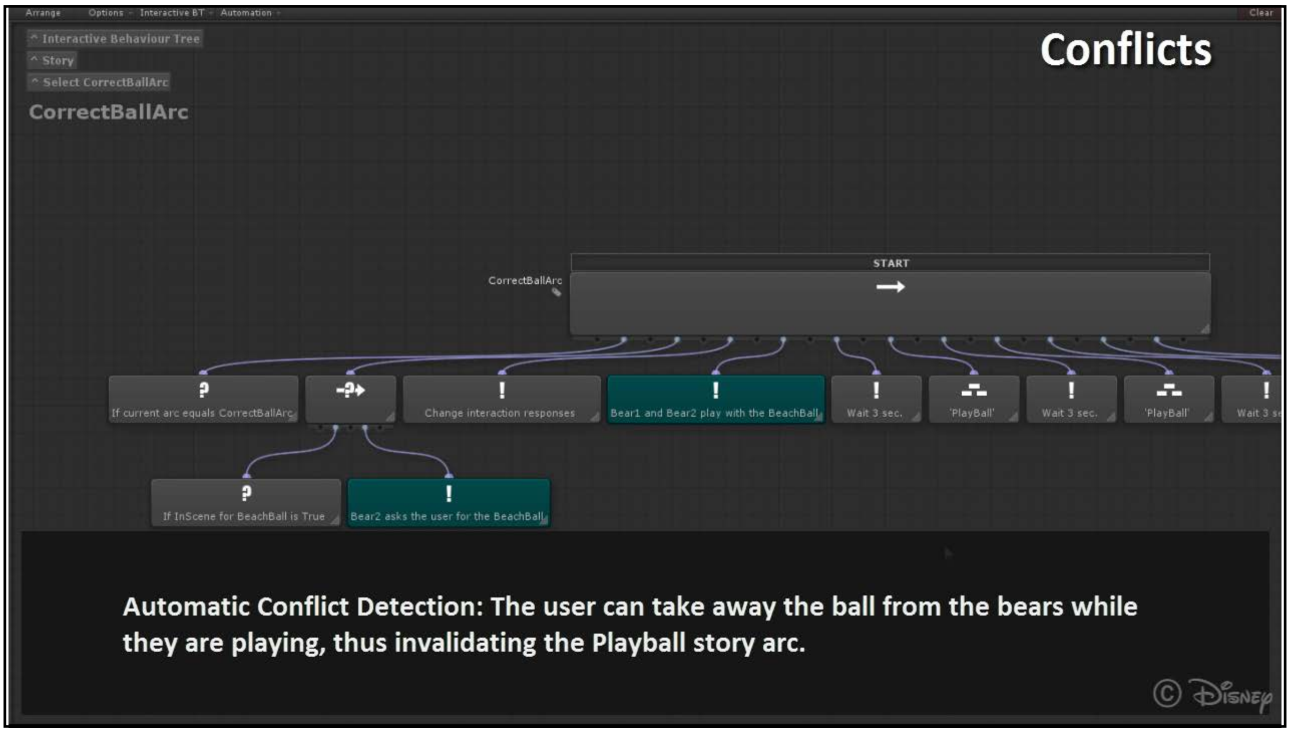


Algorithm 3 Conflict Detection and Resolution

```
function DetectAndResolveConflicts ($\mathbf{t}, \mathbf{L}, \mathbf{t}_{\text{IBT}}$)
  b = ComputeBeliefState($\mathbf{t}, \mathbf{t}_{\text{IBT}}$)
  for all s $\in$ b do
    $\Delta_{\text{active}} = \emptyset$
    $\Delta_{\text{needed}} = \emptyset$
    for all l $= \langle \mathbf{h}_i, \phi_j, \mathbf{h}_j \rangle \in$ L do
      if ($\mathbf{h}_i \prec \mathbf{h}_\mathbf{t} \wedge \mathbf{h}_\mathbf{t} \preceq \mathbf{h}_j$) == TRUE then
        $\Delta_{\text{active}} = \Delta_{\text{active}} \cup \phi_j$
      if ($\mathbf{h}_i \preceq \mathbf{h}_\mathbf{t} \wedge \mathbf{h}_\mathbf{t} \prec \mathbf{h}_j$) == TRUE then
        $\Delta_{\text{needed}} = \Delta_{\text{needed}} \cup \phi_j$
    for all $u \in$ U do
      if $\Omega_u(\Delta_{\text{active}})$ == FALSE then
        $\mathbf{P} = \langle \Omega_u(\mathbf{s}), \Delta_{\text{needed}} \rangle$
        $\pi$ = GeneratePlan ($\mathbf{P}$)
        if $\pi \neq \emptyset$ then
          Accommodate($\mathbf{t}_{\text{cr}}, \mathbf{t}, u, \pi$)
        else
          Interfere($\mathbf{t}_{\text{cr}}, \mathbf{t}, u$)
```






Arrange
Options
Interactive BT
Automation
Clear
Interactive Behaviour Tree
Story
Select CorrectBallArc
CorrectBallArc
START
CorrectBallArc
If current arc equals CorrectBallArc
Change interaction responses
Bear1 and Bear2 play with the BeachBall
Wait 3 sec.
'PlayBall'
Wait 3 sec.
'PlayBall'
If InScene for BeachBall is True
Bear2 asks the user for the BeachBall

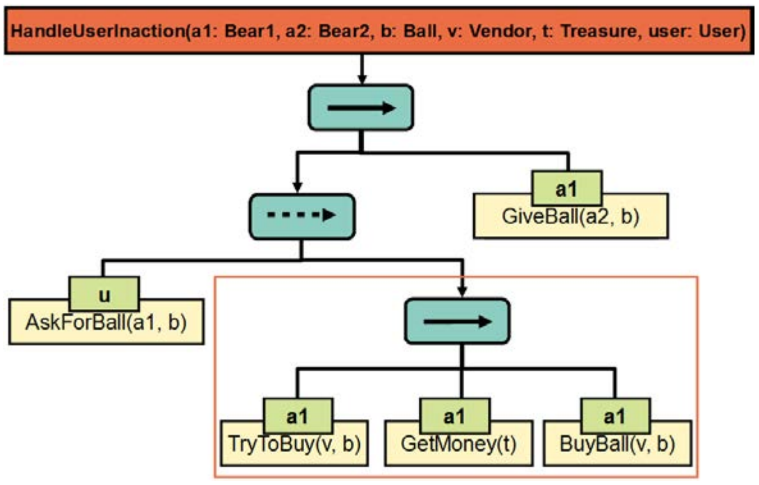
Conflicts

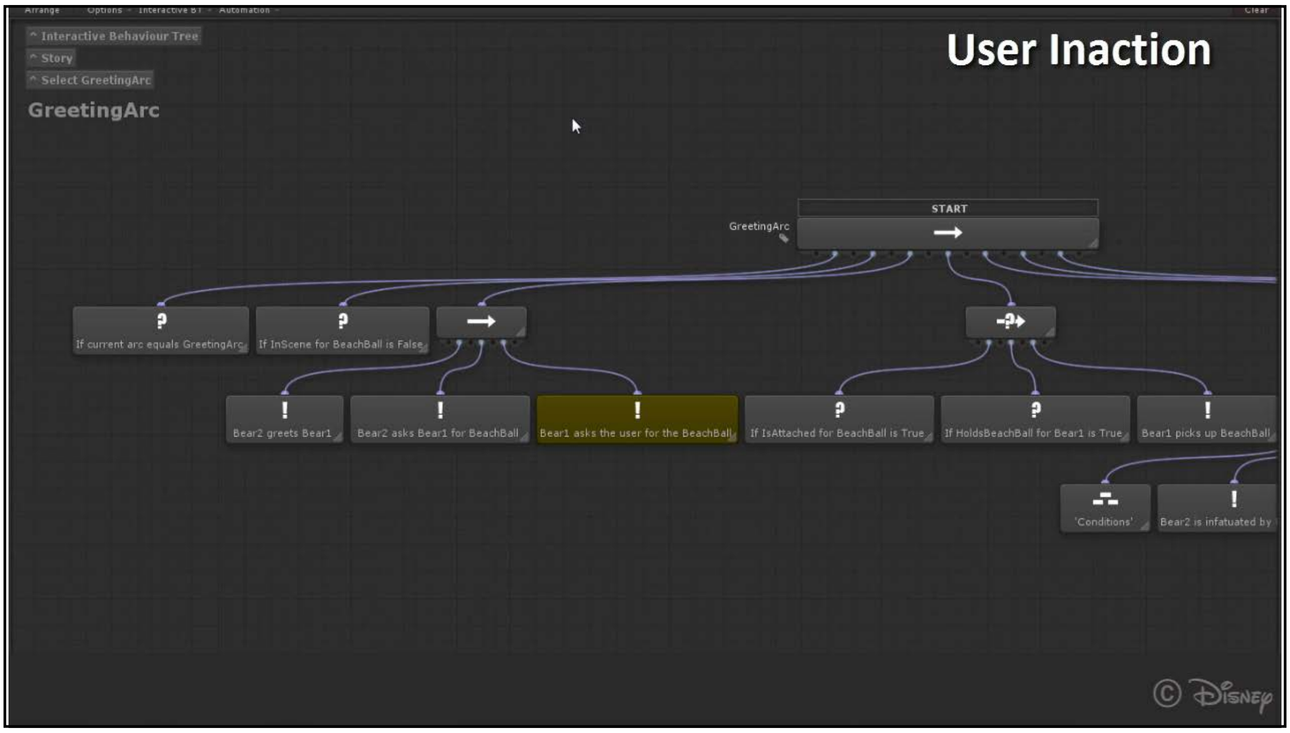
Automatic Conflict Detection: The user can take away the ball from the bears while they are playing, thus invalidating the Playball story arc.

Additional Challenges

- Inconsistent Stories
- Conflicting User Actions
- **User Inactions**
- Incomplete Stories

User Inaction

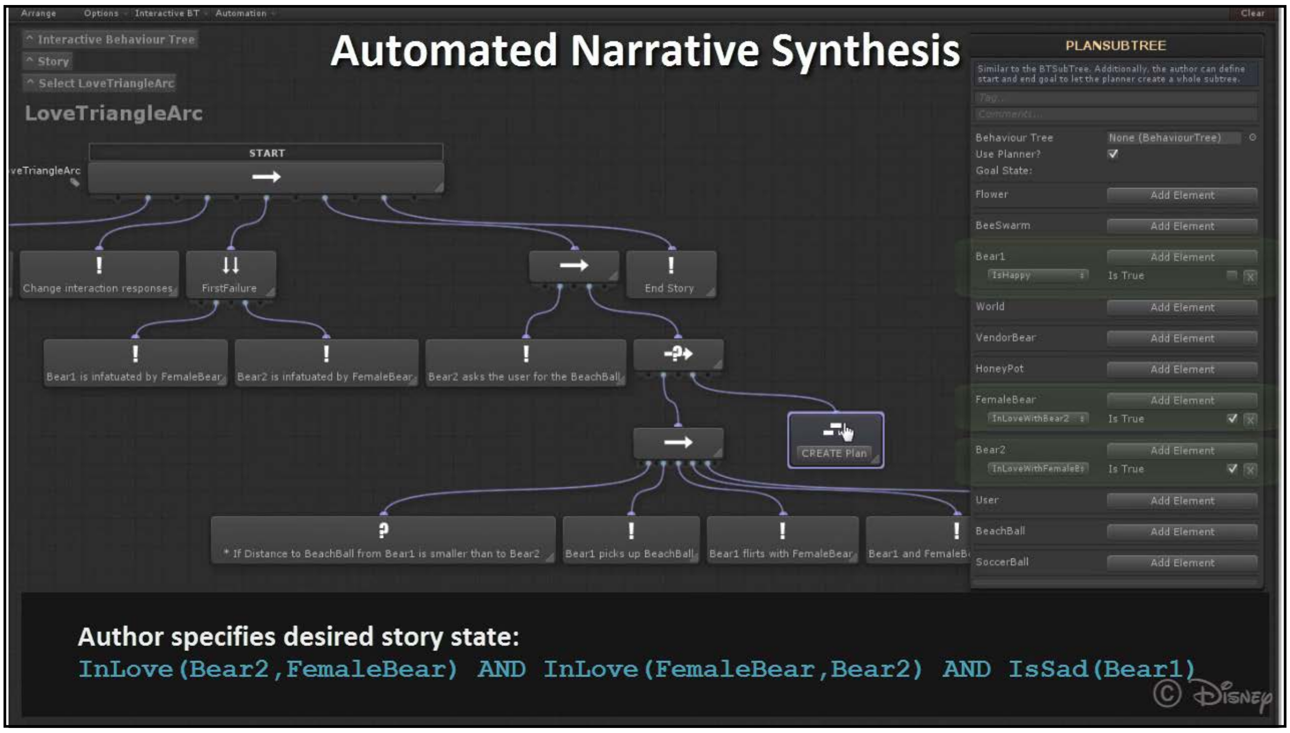

HandleUserInaction(a1: Bear1, a2: Bear2, b: Ball, v: Vendor, t: Treasure, user: User)
a1
GiveBall(a2, b)
u
AskForBall(a1, b)
a1
TryToBuy(v, b)
a1
GetMoney(t)
a1
BuyBall(v, b)


Arrange
Options
Interactive BT
Automation
Clear
Interactive Behaviour Tree
Story
Select GreetingArc
GreetingArc
START
GreetingArc
If current arc equals GreetingArc
If InScene for BeachBall is False
Bear2 greets Bear1
Bear2 asks Bear1 for BeachBall
Bear1 asks the user for the BeachBall
If IsAttached for BeachBall is True
If HoldsBeachBall for Bear1 is True
Bear1 picks up BeachBall
'Conditions'
Bear2 is infatuated by
© Disney

User Inaction

Additional Challenges

- Inconsistent Stories
- Conflicting User Actions
- User Inactions
- **Incomplete Stories**



Automated Narrative Synthesis

Arrange
Options
Interactive BT
Automation
Clear
^ Interactive Behaviour Tree
^ Story
^ Select LoveTriangleArc
LoveTriangleArc
START
veTriangleArc
Change interaction responses
FirstFailure
End Story
Bear1 is infatuated by FemaleBear
Bear2 is infatuated by FemaleBear
Bear2 asks the user for the BeachBall
CREATE Plan
* If Distance to BeachBall from Bear1 is smaller than to Bear2
Bear1 picks up BeachBall
Bear1 flirts with FemaleBear
Bear1 and FemaleB

PLANSUBTREE

Similar to the BTSubTree. Additionally, the author can define start and end goal to let the planner create a whole subtree.

Tag...
Comment...

Behaviour Tree: None (BehaviourTree)
Use Planner? ✓
Goal State:

Flower	Add Element		
BeeSwarm	Add Element		
Bear1	Add Element		
IsHappy	Is True		
World	Add Element		
VendorBear	Add Element		
HoneyPot	Add Element		
FemaleBear	Add Element		
InLoveWithBear2	Is True	✓	
Bear2	Add Element		
InLoveWithFemaleB	Is True	✓	
User	Add Element		
BeachBall	Add Element		
SoccerBall	Add Element		

Author specifies desired story state:

`InLove(Bear2,FemaleBear) AND InLove(FemaleBear,Bear2) AND IsSad(Bear1)`

© Disney

CANVAS: Computer-Assisted Narrative Animation Synthesis. Mubbasir Kapadia, Seth Frey, Alexander Shoulson, Robert W. Sumner, Markus Gross. *ACM SIGGRAPH Symposium on Computer Animation (SCA) 2016.*

Evaluating Accessible Graphical Interfaces for Building Story Worlds. S Poulakos, M Kapadia, GM Maiga, F Zünd, M Gross, RW Sumner, *Interactive Storytelling: 9th International Conference on Interactive Digital Storytelling, 2016*

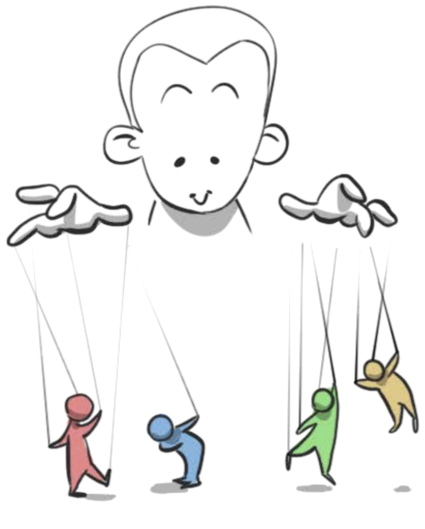
An event-centric approach to authoring stories in crowds. M Kapadia, A Shoulson, C Steimer, S Oberholzer, RW Sumner, M Gross, *ACM SIGGRAPH Conference on Motion in Games, 15-24*

Towards an Accessible Interface for Story World Building, *S Poulakos, M Kapadia, A Schüpfer, F Zünd, RW Sumner, M Gross. Eleventh Artificial Intelligence and Interactive Digital Entertainment, 2015*

Computer-Assisted Authoring of Interactive Narratives. Mubbasir Kapadia, Jessica Falk, Fabio Zund, Marcel Marti, Robert W. Sumner, Markus Gross. *ACM SIGGRAPH Interactive 3D Graphics and Games (I3D), 2015.*

Augmented creativity: bridging the real and virtual worlds to enhance creative play. F Zünd, M Ryffel, S Magnenat, A Marra, M Nitti, M Kapadia, G Noris. *ACM SIGGRAPH Asia 2015 Mobile Graphics and Interactive Applications, 2016*

Evaluating the Authoring Complexity of Interactive Narratives with Interactive Behavior Trees. Mubbasir Kapadia, Jessica Falk, Fabio Zund, Marcel Marti, Robert W. Sumner, Markus Gross. *Foundations of Digital Games (FDG), 2015.*



Thank You!