Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2016)
Ladislav Kavan and Chris Wojtan (Editors)

CANVAS: Computer-Assisted Narrative Animation Synthesis

Mubbasir Kapadia[1,2], Seth Frey[2,3], Alexander Shoulson[2], Robert W. Sumner[2,4], Markus Gross[2,4]

[1]Rutgers University, [2]Disney Research Zürich, [3]Dartmouth College, [4]ETH Zürich

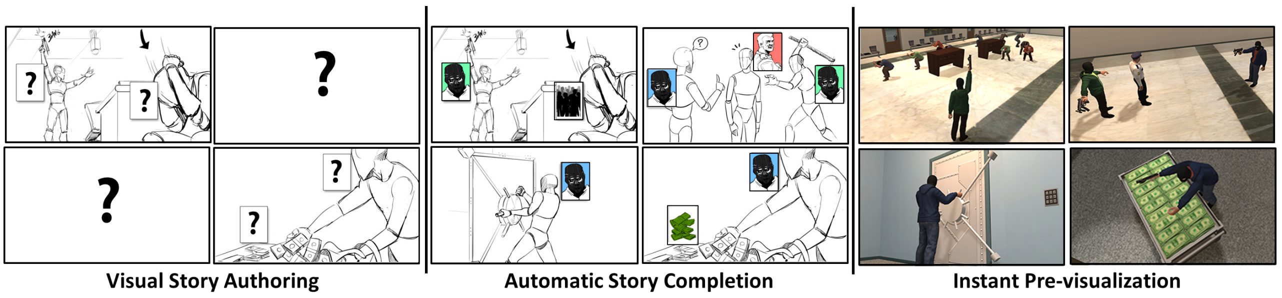


Figure 1: CANVAS Overview. (a) **Visual Story Authoring**: Authors specify key plot points in the narrative as logical interactions between participating actors, which are represented as visual storyboards. (b) **Automatic Story Completion**: CANVAS automatically identifies and resolves incomplete stories 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.

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. 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.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

"No amount of great animation will save a bad story."
– John Lasseter

Character animation has made great strides in recent years, providing mature solutions for animating detailed virtual characters precisely. Somewhat surprisingly, technologies for telling stories with these characters have not kept the same pace. Existing story-authoring systems either burden authors with irrelevant details, or replace creative vision with automation. Authoring stories with animated characters in 3D virtual worlds is presently only feasible for trained artists and experts using proprietary tools. The key challenge is to provide an accessible and expressive interface for narratives that abstracts out distracting details, just as a professional screenwriter leaves the specifics of physical and emotional expression to professional actors.

We present CANVAS, a visual authoring tool that empowers

DOI: 10.2312/sca.20161237

artists, content creators, and even casual users to quickly prototype and synthesize multi-character animations that foster creativity while maintaining formal verifiability. CANVAS uses storyboarding as a natural visual metaphor to enable accessible prototyping and interactive exploration of new narratives. Authors using CANVAS begin with a formal definition of "smart" semantically-labelled virtual actors, and a library of multi-actor interactions, or events. Events are defined as logical context-specific interactions between actors, but end-users interact only with virtual storyboards that insulate them from the details of representation. With CANVAS's simple, intuitive graphical interface, users incrementally build complex story arcs by selecting key plot events and assigning actors to them. Using just this sparse input, CANVAS can instantly synthesize a 3D animation for real-time visual feedback of the narrative.

To further ease the authoring burden, CANVAS automatically identifies and resolves incomplete stories by efficiently exploring the space of partial narratives. This algorithm is guaranteed to be *sound* and *complete*. Automation also preserves creative intent; CANVAS is guaranteed to find a story that satisfies authored constraints, if one exists. Existing authoring systems also offer automation, but at the cost of creative control. In contrast, CANVAS uses automation to facilitate, not replace, the authoring process. Fig. 1 provides an overview of the workflow in CANVAS.

With the following technical contributions, we overcome limitations of both author-centric and automation-centric authoring tools: a formal representation of narrative elements, a story specification that is accessible to diverse authors, and algorithms for completing stories. Rather than compromising between the relative benefits and limitations of authors and automation, we use the opportunities in each to address the limitations of the other in novel ways. With multiple possibly-contradictory goal constraints, plot points sketched by authors pose a challenging constraint satisfaction problem for story completion, because locally-consistent stories between successive plot points may not satisfy the plot constraints of the entire narrative. We formulate this problem as a search in the space of partial stories, adding new elements only when absolutely necessary. This avoids unnecessary computations and expensive backtracking, and the authored plot points help constrain the otherwise combinatorial search space to provide efficient algorithms for story completion. The resulting automation then frees authors to focus on narrative-relevant details, without artificial technical restrictions that preclude creative expression.

CANVAS introduces, for the first time, the means for untrained users to create complex narrative-driven animations within minutes. This is made possible with two equally important contributions: (1) A visual storyboarding interface that provides the right level of abstraction for expressing an authors narrative intent without sacrificing creativity or adding overwhelming complexity, and (2) provably sound, complete computational tools that can automatically fill in partially specified stories without violating author constraints. Our approach does not relax guarantees by using heuristics to accelerate searches. It ensures that a complete story will be found if one exists, without violating any author constraints. These requirements are essential for an author-centric tool.

CANVAS can be used in a variety of ways, depending on the author's expertise and goals. Artists and directors with a clear vision can synthesize 3D pre-visualizations of storyboards for interactive story development. Casual users may explore and modify different automatically synthesized narratives for rapid content creation and dissemination. CANVAS is ideally suited for story writers to explore narratives for episodic content (e.g., long running TV series) as well as for consumer applications such as story creation tools in video games. In both these cases, professionally produced assets and domain knowledge have already been created and need only be specified once. CANVAS makes it possible for traditional content consumers to create their own digital stories. It promises to propel the field of digital storytelling towards developing new and accessible metaphors for user-generated animated content.

2. Related Work

The maturity of research in the simulation and animation of virtual characters has expanded the possibilities for authoring complex multi-character animations [KGP02, Lee10]. These methods represent tradeoffs between user-driven specification and automatic behavior generation.

The work of Kwon *et al.* [KLLT08], and Kim *et al.* [KHKL09, KSKL14] synthesizes synchronized multi-character motions and crowd animations by editing and stitching motion capture clips. "Motion patches" [LCL06] annotate motion semantics in environments and can be concatenated [KHHL12] or precomputed [SKSY08] to synthesize complex multi-character interactions. Recent work [WLH*14] 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 [JPCC14] where the consistency of the interactions between characters toward an overarching narrative is less relevant.

In contrast, behavior-centric approaches use logical constructs [VCC*07, SSH*10] and complex models [YT07] to represent knowledge and action selection in agents. Parameterized Behavior Trees (PBT's) [SGJ*11] 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 [Loy97, Mat02] 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 [PG96], LIVE [Men01], 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 [WB97] and storyboards are a natural metaphor for specifying narratives [KSS96]. Game-design systems [KPK07, Ros14] facilitate the authoring of game logic by providing visual analogies to programming constructs.

Skorupski and Mateas [SM10] propose a storyboarding tool for novice users to author interactive comics. LongBoard [JRMY08] provides a hybrid sketch and scripting interface for rendering an animated 3D scene. Physics Storyboards [HMLP13] 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 [FN71, HNR72] are promising for automated behavior generation [FTT99, KSRF11, SGKB13, RB13]. 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 [JRMY08] with small numbers of agents. Partial-order planners [Sac75] relax the strict ordering constraints during planning to potentially handle more complex domains, and have been applied to accommodate player agency in interactive narratives [KFZ*15, CCM02]. The work in [PTPC11, HPS04] introduces the concept of "landmarks" to allow authors to specify additional narrative constraints for automated narrative synthesis.

Comparison to Prior Work. Existing authoring systems [KSRF11, RB13] 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. CANVAS users author plot points that must be enforced in the story. This maps to a constraint satisfaction problem with multiple, possibly contradicting goal constraints, where locally-consistent stories between successive plot points may not satisfy the plot constraints of the entire narrative, and cannot be solved by previous methods.

The work in [KFZ*15] uses a partial-order planner to detect and resolve local inconsistencies in existing story specifications. However, this approach cannot resolve global inconsistencies, such as the simple example highlighted in Figure 4. Our proposed approach for automatic story completion addresses these limitations by searching in the space of partial narratives for the whole story arc, not just between two beats. The formulation of generating a complete and consistent story arc from an incomplete story specification, cast as a partial-order planning problem, is a novel contribution of this work. Our work demonstrates the practical utility of POP-like solvers for solving heavily constrained problems in unstructured, high-dimensional search spaces, while preserving theoretical guarantees. No other approach can provide these theoretical guarantees while maintaining interactivity.

3. Domain Knowledge

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

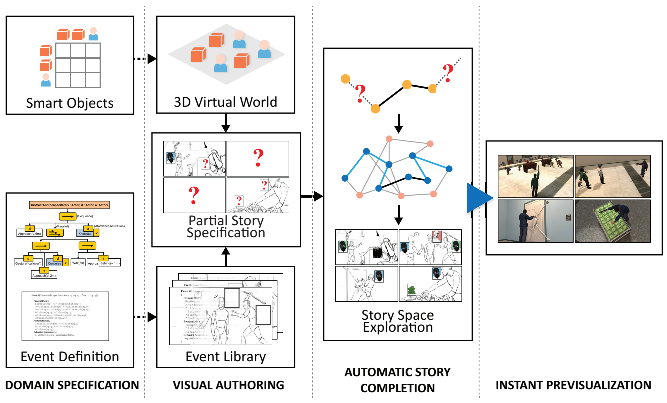


Figure 2: CANVAS Framework Overview.

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 [RB13] in this regard. However, the cost of specifying domain knowledge is 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 [PKS*15].

Smart Objects and Actors. The virtual world $\mathcal{W}$ consists of smart objects [KT99] 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) [SGJ*11] 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 $\Phi : \mathbf{s_w} \leftarrow \{\mathtt{TRUE}, \mathtt{FALSE}\}$ 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 $\Phi_e(\mathbf{w}) = \mathtt{TRUE}$. The event postcondition $\Omega : \mathbf{s} \to \mathbf{s}'$ 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 $\mathbf{w}$. $\Phi_e(\mathbf{w}) = \mathtt{TRUE}$. The event postcondition $\Omega : \mathbf{s} \to \mathbf{s}'$ 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 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.

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 $\alpha = (\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_m)$ is an ordered sequence of beats representing a story, where events can occur both sequentially and simultaneously throughout that story's execution.

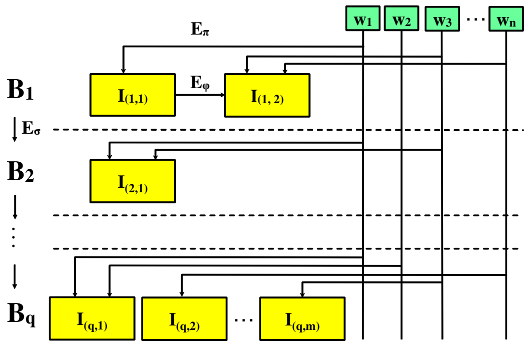


Figure 3: Representation of a story arc as a story sequence diagram.

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. 3 illustrates a generic story arc represented as a story sequence diagram.

4.1. 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 in Section 5 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 [SKMB14] 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. Computer-Assisted Story Authoring

5.1. 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:

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

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 [Sus75].

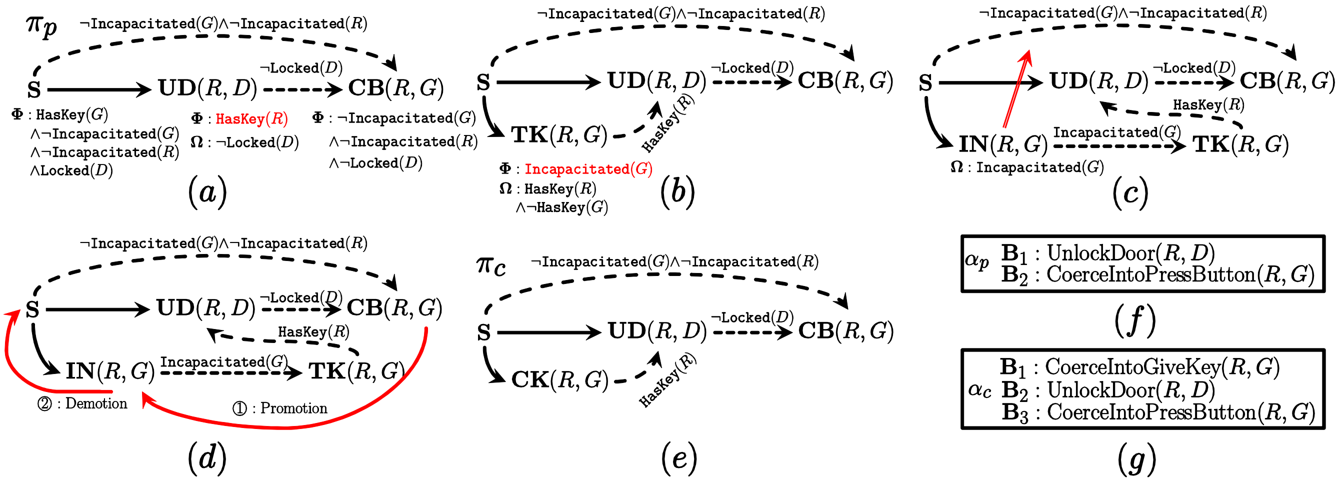
Consider a simple scenario with a guard and robber. The robber has a weapon for coercion or incapacitation and the guard has the key to the room with a button to open the vault. We author an incomplete story where $\mathbf{B}_1$: the robber will unlock the door to the room (which requires the key) and $\mathbf{B}_2$: 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 [Sac75], 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 [FN71, HNR72] 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.* [MDBP92]. Section 5.2 introduces relevant terminology and Section 5.3 describes a provably sound, complete algorithm for automatically filling in partial story specifications using a plan-space approach.

5.2. 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



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 4: Execution of Algorithm 1 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.

two event instances implies that I_2 must execute some time after I_1 in the story α, though not necessarily immediately after. Formally, $I_1 \prec I_2 \Rightarrow I_1 \in \mathbf{B}_i, I_2 \in \mathbf{B}_j,\ \exists\, \mathbf{B}_i, \mathbf{B}_j \in \alpha$ 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* 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. 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 $\mathcal{O}_c$. Alg. 1 describes 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 $I = \langle e, \mathbf{w} \rangle \in \alpha$ 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 $\mathbf{x}_i = \{x \mid x \in \mathcal{W}, r_i(x) =$

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 (Lines 17-19). 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 (Line 21). $\mathcal{P}$ 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 (Lines 22-23).

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 $\mathcal{A}$ (Lines 2-8). 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 $\mathcal{L}$ (Line 32).

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 (Lines 26-30). 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 (Lines 34-36).

Event Instance Selection. An event instance I_s is selected that satisfies ϕ_c, either directly from $\mathcal{I}$ (Line 37), or by non-deterministically picking a new instance and adding it to $\mathcal{I}$ (Line 39). A new causal link $I_s \xrightarrow{\phi_c} I_c$ is established to indicate that I_s satisfies ϕ_c for I_c (Line 44). A new ordering constraint $I_s \prec I_c$ is added to ensure that I_c executes after I_s (Line 45). 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 $\mathcal{I}_{\text{end}}$ (Line 42). (b) $\mathcal{A}$ is updated to include all precondition clauses $\phi_s \in \Phi_s$ that are not satisfied (Line 43). If no event instance exists that satisfies the precondition clause, we recursively roll back to the previous choice point (Line 36).

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 (Lines 46-52).

Recursive Invocation. This process is recursively repeated until the termination condition is met, or no solution is found (Line 54). 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 (Lines 55-67).

Let us revisit the example of the robber and guard, discussed earlier. Fig. 4(a) illustrates the initial partial plan π_p for the incomplete arc (f). The start node **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 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. 4(c)). This introduces a threat: $\mathbf{IN}(R,G)$ conflicts with the causal link, as indicated by the red arrow in Fig. 4(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. 4(g)).

The constraint-satisfaction solver (Alg. 1) 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 non-deterministic 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.

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 $\langle k_1, |\mathcal{O}_I| \rangle$

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
        $I_s \leftarrow \langle e, \mathbf{w} \rangle \mid e \in \mathcal{E}, \mathbf{w} \in \mathcal{P}(\langle e, \{w_i = \varnothing \mid \forall i \in (1, |\mathbf{w}|)\} \rangle) \wedge \phi_c \in \Omega_I$
        if $I_s = \varnothing$ then return fail
        $\mathcal{I} \leftarrow \mathcal{I} \cup I_s$
        $\mathcal{O} \leftarrow \mathcal{O} \cup (I_0 \prec I_s) \cup \{(I_s \prec I) \mid \forall\, I \in \mathcal{I}_{\text{end}}\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \langle I_s, \phi_s \rangle\ \exists\, \phi_s \in \Phi_{I_s}, \phi_s = \texttt{FALSE}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \langle I_s, \phi_c, I_c \rangle$
    $\mathcal{O} \leftarrow \mathcal{O} \cup (I_s \prec I_c)$
    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 ($\pi_p = \langle \mathcal{I}, \mathcal{O}, \mathcal{L} \rangle, \mathcal{A}, \mathcal{E}$)

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 $\alpha_c$
```

6. 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 6.1, and Section 6.2 describes some authored narratives. Please refer to the supplementary video for additional details.

6.1. 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

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.

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 6.2. 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 person 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.

6.2. 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 pre-visualize their stories and iterate. Table 2 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 (Table 2a) 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 (Table 2b), and CANVAS automatically selects a suitable combination of actors and smart objects (highlighted in bold) to complete the story and ensure its consistency. In Table 2c, 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. Table 2d illustrates 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

(a)

B$_1$	IncapacitateStealthily(r_1, g_1); IncapacitateStealthily(r_2, g_2)
B$_2$	DistractAndIncapacitate(r_1, r_3, g_3); WarningShot(r_2, cr)
B$_3$	Flee(cr); TakeKeyFromIncapacitated(g_3, r_1)
B$_4$	UnlockDoor(r_1, d_t); Converse(r_2, r_3)
B$_5$	PressButton(r_1, b_t); Converse(r_2, r_3)
B$_6$	IncapacitateStealthily(r_3, r_1)
B$_7$	CoerceIntoUnlockDoor(r_2, m, d_m); PressButton(r_3, b_m); Escape(t_1) Escape(t_1)
B$_8$	Escape(m); OpenDoor(r_3, d_v)
B$_9$	Argue(r_2, r_3)
B$_{10}$	IncapacitateStealthily(r_2, r_3)
B$_{11}$	TakeMoney(r_2)
B$_{12}$	Escape(r_2)

(b)

B$_1$	IncapacitateStealthily($\mathbf{r}_1$, g_1); IncapacitateStealthily($\mathbf{r}_2$, g_2); IncapacitateStealthily($\mathbf{r}_3$, g_3)
B$_2$	WarningShot($\mathbf{r}_3$, **cr**)
B$_3$	CoerceIntoPressButton($\mathbf{r}_1$, t_1, b_t); Flee(**cr**); Converse($\mathbf{r}_2$, $\mathbf{g}_3$)
B$_4$	CoerceIntoUnlockDoor($\mathbf{r}_1$, m, d_m);
B$_5$	PressButton($\mathbf{r}_2$, b_m)
B$_6$	UnlockDoor($\mathbf{r}_2$, d_v); Flee(t_1); Flee(t_2); IncapacitateStealthily($\mathbf{r}_1$, m);
B$_7$	TakeMoney($\mathbf{r}_1$); TakeMoney($\mathbf{r}_2$); TakeMoney($\mathbf{r}_3$)
B$_8$	Escape($\mathbf{r}_1$); Escape($\mathbf{r}_2$); Escape($\mathbf{r}_3$)

(c)

B$_1$	**CoerceIntoGiveKey($\mathbf{r}_1$, $\mathbf{g}_1$)**
B$_2$	**UnlockDoor($\mathbf{r}_1$, $\mathbf{d}_\mathrm{t}$)**
B$_3$	**CoerceIntoMove($\mathbf{r}_1$, $\mathbf{g}_1$, $\mathbf{f}_\mathrm{t}$)**
B$_4$	**CoerceIntoPressButton($\mathbf{r}_1$, $\mathbf{r}_1$, $\mathbf{b}_\mathrm{t}$)**
B$_5$	**LockDoor($\mathbf{r}_1$, $\mathbf{d}_\mathrm{t}$)**
B$_6$	**UnlockDoor($\mathbf{r}_1$, $\mathbf{d}_\mathrm{m}$)**
B$_7$	**PressButton($\mathbf{r}_1$, $\mathbf{b}_\mathrm{m}$)**
B$_8$	**OpenVault($\mathbf{r}_1$)**
B$_9$	TakeMoney(r_1)
B$_{10}$	Escape(r_1)

(d)

B$_1$	**CoerceIntoDropWeapon($\mathbf{r}_1$, $\mathbf{g}_1$)**
B$_2$	**CoerceIntoGiveKey($\mathbf{r}_1$, $\mathbf{g}_1$)**
B$_3$	WarningShot(r_2, **cr**)
B$_4$	UnlockDoor(r_1, $\mathbf{d}_\mathrm{t}$)
B$_5$	**CoerceIntoMove($\mathbf{r}_1$, $\mathbf{g}_1$, $\mathbf{f}_\mathrm{t}$)**
B$_6$	CoerceIntoPressButton(r_1, g_1, b_t); **PickUp(c, w)**
B$_7$	LockDoor(r_1, d_t); Escape(r_2)
B$_8$	CoerceIntoSurrender(c, r_1)

Table 2: A selection of narratives authored using CANVAS. The event parameters and event instances that were automatically inserted are highlighted in **bold**. Event parameter symbols: Robbers (r_1, r_2, r_3), Guards (g_1,g_2,g_3), Individual customer (c), Group of customers (cr), Bank tellers (t_1, t_2), Manager (m), Teller, manager, and vault doors (d_t, d_m, d_v).

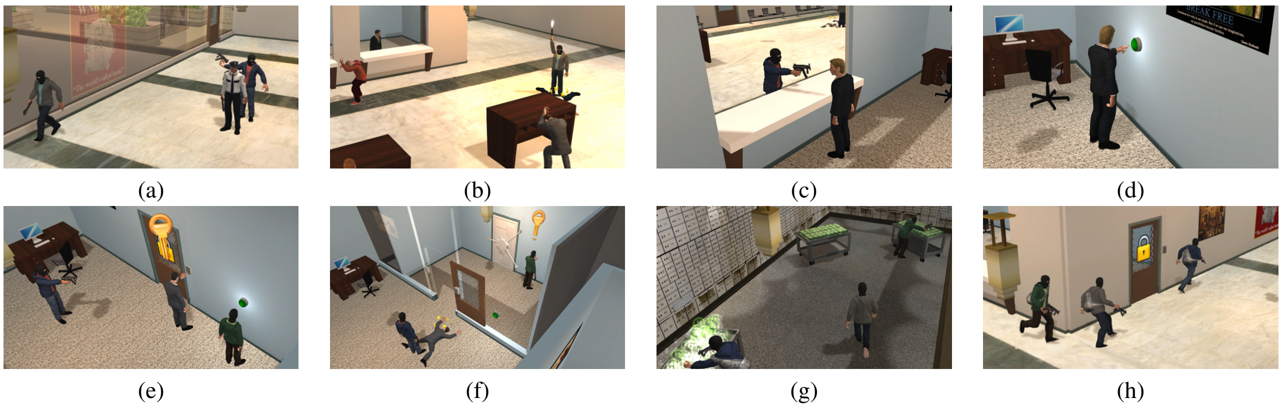


Figure 5: A complex narrative authored using CANVAS. (a) Robbers enter the bank from the back door and begin incapacitating guards. (b) A robber fires a shot into the air to intimidate the crowd. (c) A second robber coerces the teller to (d) press a button behind his desk to unlock the vault. (e) The 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.

events are removed or modified during automation. The original intent of the author is thus always preserved.

6.3. 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 [KSRF11]. For a single-threaded C# implementation, the computation times for automating stories b, c, and d of Table 2 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 pre-visualization is instantaneous and the video is a real-time recording of our system.

7. Conclusions and Future Work

We present CANVAS, a visual authoring tool that allows expert or novice users to quickly prototype and synthesize story-driven animations. We use a *storyboard metaphor* for visual authoring in which authors specify key events in the story as parameterized storyboard elements that are automatically translated into an animated 3D scene. CANVAS identifies and resolves incomplete story definitions to produce a consistent and complete narrative, while meeting the author's original vision. Our work demonstrates the practical utility of POP-like solvers for solving heavily constrained problems in unstructured, high-dimensional search spaces, while preserving theoretical guarantees. It has the potential for improving constraint-satisfaction problem tasks in other graphics domains.

Limitations and Future Work. 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. For the results described in this paper, we use a graphical tool [PKS*15] to minimize the one-time cost of domain specification. Additionally, a promising direction for future work is to automatically create knowledge bases from existing or crowd-sourced data [LR15] which can be used for narrative synthesis. The narratives produced using CANVAS are suitable for pre-visualization or cut scenes but cannot be used for interactive applications such as games. Extending our framework to robustly handle free-form user interactions is an exciting avenue for future exploration.

References

[CCM02] Cavazza M., Charles F., Mead S. J.: Character-based interactive storytelling. *IEEE Intelligent Systems 17*, 4 (July 2002), 17–24. 3

[FN71] Fikes R. E., Nilsson N. J.: Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* (1971). 3, 5

[FTT99] Funge J., Tu X., Terzopoulos D.: Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *ACM SIGGRAPH* (1999), pp. 29–38. 3

[HMLP13] Ha S., McCann J., Liu C. K., Popović J.: Physics storyboards. *Computer Graphics Forum 32*, 2pt2 (2013), 133–142. 3

[HNR72] Hart P. E., Nilsson N. J., Raphael B.: Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Bull.*, 37 (1972), 28–29. 3, 5

[HPS04] Hoffmann J., Porteous J., Sebastia L.: Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR) (JAIR) 22* (2004), 215–278. URL: http://dx.doi.org/10.1613/jair.1492. 3

[JPCC14] Jordao K., Pettré J., Christie M., Cani M.-P.: Crowd Sculpting: A space-time sculpting method for populating virtual environments. *Computer Graphics Forum 33*, 2 (Apr. 2014). 2

[JRMY08] Jhala A., Rawls C., Munilla S., Young R. M.: Longboard: A sketch based intelligent storyboarding tool for creating machinima. In *FLAIRS Conference* (2008), AAAI Press, pp. 386–390. 3

[KFZ*15] Kapadia M., Falk J., Zünd F., Marti M., Sumner R. W., Gross M.: Computer-assisted authoring of interactive narratives. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2015), i3D '15, ACM, pp. 85–92. 3

[KGP02] Kovar L., Gleicher M., Pighin F.: Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 473–482. 2

[KHHL12] Kim M., Hwang Y., Hyun K., Lee J.: Tiling motion patches. In *ACM SIGGRAPH / Eurographics SCA* (2012), pp. 117–126. 2

[KHKL09] Kim M., Hyun K., Kim J., Lee J.: Synchronized multi-character motion editing. In *ACM SIGGRAPH* (2009). 2

[KLLT08] Kwon T., Lee K. H., Lee J., Takahashi S.: Group motion editing. In *ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 80:1–80:8. 2

[KPK07] Kelleher C., Pausch R., Kiesler S.: Storytelling alice motivates middle school girls to learn computer programming. In *ACM SIGCHI Conference on Human Factors in Computing Systems* (2007), pp. 1455–1464. 2

[KSKL14] Kim J., Seol Y., Kwon T., Lee J.: Interactive manipulation of large-scale crowd animation. *ACM Transactions on Graphics (SIGGRAPH 2014, To Appear) 33* (2014). 2

[KSRF11] Kapadia M., Singh S., Reinman G., Faloutsos P.: A behavior-authoring framework for multiactor simulations. *Computer Graphics and Applications, IEEE 31*, 6 (nov.-dec. 2011), 45 –55. 3, 10

[KSS96] Kurlander D., Skelly T., Salesin D.: Comic chat. In *ACM SIGGRAPH* (1996), pp. 225–236. 2

[KT99] Kallmann M., Thalmann D.: Direct 3d interaction with smart objects. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (New York, NY, USA, 1999), VRST '99, ACM, pp. 124–130. 3

[LCL06] Lee K. H., Choi M. G., Lee J.: Motion patches: building blocks for virtual environments annotated with motion data. In *ACM SIGGRAPH* (2006), pp. 898–906. 2

[Lee10] Lee J.: Introduction to data-driven animation: Programming with motion capture. In *ACM SIGGRAPH ASIA 2010 Courses* (New York, NY, USA, 2010), SA '10, ACM, pp. 4:1–4:50. 2

[Loy97] Loyall A. B.: *Believable agents: building interactive personalities*. PhD thesis, Pittsburgh, PA, USA, 1997. 2

[LR15] Li B., Riedl M. O.: Scheherazade: Crowd-powered interactive narrative generation. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015). 10

[Mat02] Mateas M.: *Interactive drama, art and artificial intelligence*. PhD thesis, Pittsburgh, PA, USA, 2002. 2

[MDBP92] Minton S., Drummond M., Bresina J. L., Philips A. B.: Total order vs. partial order planning: Factors influencing performance. In *KR* (1992), Morgan Kaufmann, pp. 83–92. 5

[Men01] Menou E.: Real-time character animation using multi-layered scripts and spacetime optimization. In *ICVS* (2001), pp. 135–144. 2

[PG96] Perlin K., Goldberg A.: Improv: A system for scripting interactive actors in virtual worlds. In *ACM SIGGRAPH* (1996), pp. 205–216. 2

[PKS*15] Poulakos S., Kapadia M., Schupfer A., Zund F., Sumner R., Gross M.: Towards an accessible interface for story world building, 2015. 3, 10

[PTPC11] Porteous J., Teutenberg J., Pizzi D., Cavazza M.: 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* (2011). URL: `http://aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/view/2681`. 3

[RB13] Riedl M. O., Bulitko V.: Interactive narrative: An intelligent systems approach. *AI Magazine 34*, 1 (2013), 67–77. 3

[Ros14] Rosini R.: Storybricks. Namaste Entertainment Inc., 2014. 2

[Sac75] Sacerdoti E. D.: The nonlinear nature of plans. In *IJCAI* (1975), pp. 206–214. 3, 5

[SGJ*11] Shoulson A., Garcia F. M., Jones M., Mead R., Badler N. I.: Parameterizing behavior trees. In *Motion in Games* (2011), pp. 144–155. 2, 4

[SGKB13] Shoulson A., Gilbert M. L., Kapadia M., Badler N. I.: An event-centric planning approach for dynamic real-time narrative. In *Proceedings of Motion on Games* (New York, NY, USA, 2013), MIG '13, ACM, pp. 99:121–99:130. 3

[SKMB14] Shoulson A., Kapadia M., Marshak N., Badler N. I.: Adapt: The agent development and prototyping testbed. *IEEE Transactions on Visualization and Computer Graphics 99* (2014), 1. 5

[SKSY08] Shum H. P. H., Komura T., Shiraishi M., Yamazaki S.: Interaction patches for multi-character animation. In *ACM SIGGRAPH Asia* (2008), pp. 114:1–114:8. 2

[SM10] Skorupski J., Mateas M.: Novice-friendly authoring of plan-based interactive storyboards. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE* (2010). 3

[SSH*10] Stocker C., Sun L., Huang P., Qin W., Allbeck J. M., Badler N. I.: Smart events and primed agents. In *IVA* (2010), vol. 6356, pp. 15–27. 2

[Sus75] Sussman G. J.: *A computer model of skill acquisition*. Artificial intelligence series. American Elsevier Pub. Co., New York, 1975. 5

[VCC*07] 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., Werf R. J.: The behavior markup language: Recent developments and challenges. In *Intelligent Virtual Agents* (2007), pp. 99–111. 2

[WB97] Whitley K. N., Blackwell A. F.: Visual programming: The outlook from academia and industry. In *Seventh Workshop on Empirical Studies of Programmers* (1997), ESP '97, ACM, pp. 180–208. 2

[WLH*14] Won J., Lee K., Hodgins J., O'Sullivan C., Lee J.: Generating and ranking diverse multi-character interactions. In *ACM SIGGRAPH Asia* (2014). 2

[YT07] Yu Q., Terzopoulos D.: A decision network framework for the behavioral animation of virtual humans. In *ACM SIGGRAPH/Eurographics SCA* (2007), pp. 119–128. 2