Merits of Hierarchical Story and Discourse Planning with Merged Languages

David R. Winer[1] and **R. Michael Young**[2]
[1,2]School of Computing and [2]Entertainment Arts and Engineering Program
University of Utah
Salt Lake City, UT
drwiner@cs.utah.edu
young@eae.utah.edu

Abstract

A hierarchical, bipartite model can characterize many complex narrative phenomena associated with coordinating plot and communication in storytelling (e.g., cinematography), but the predominant pipeline-based strategy for generating narratives has inadvertently limited the expressiveness of storytelling systems. We introduce computational steps for merging story and discourse languages in plan-based storytelling systems with hierarchical knowledge which avoids this problem and motivates more expressive narrative discourse reasoning.

Narrative intelligence is an interdisciplinary area of research in artificial intelligence leveraging insights from linguistics, cognitive psychology, narratology, and computer science (Schank 1995). This work addresses the problem of generating narrative fiction (e.g., text or film) by using a plan-based language to model the schematic knowledge of 1] storyworld mechanics and 2] communicative plans. The science for merging these two tasks is desirable for generating narrative discourse which has the goal-oriented and hierarchical structure to support expressive storytelling. Integrating scenario generation with communicative reasoning has broad applicability for problem-solving agents which interact with people such as in the context of entertainment and education.

Our approach to automated narrative generation borrows from narrative theory which frequently distinguishes *fabula* (i.e., setting and plot events, story) from discourse (the communicative act of storytelling) (Chatman 1980).

- ***Fabula*** Actions at the fabula level are storyworld mechanics. Story generators arose as the first AI planning algorithms were developed, such as TALE-SPIN (Meehan 1977) in which woodland creatures follow plans to satisfy basic needs. State-of-the-art planners solve multi-agent coordination problems such that characters follow domain-independent rules to behave more believably (Riedl and Young 2010; Ware et al. 2014; Kapadia et al. 2016).
- **Discourse** The communication side of generating narrative content (storytelling) involves modeling the beliefs of a hypothetical viewer and conveying meaning through utterances which refer to the plot (Young 2007), consistent with discourse theories from narratology (Chatman 1980).

Fabula and discourse are hierarchically structured phenomena in their construction and reception by human understanders. At the fabula level, we recognize patterns of human movements as actions, and patterns of actions as tasks or character plans. For example, the task of flying from Point A to Point B can be decomposed into the actions of buying a ticket, getting to the airport, taking your seat, etc. At the discourse level, we recognize the hierarchical structure of films as composed of scenes, which can be further decomposed into camera shots, and further still into frames (other types of patterns may take place simultaneously such as the rise and fall of tension). In practice, plan-based systems can leverage knowledge about hierarchical patterns to narrow the search space by decomposing steps into common patterns of primitive units (Jhala and Young 2010; To, Langley, and Choi 2015).

One of the predominant strategies for designing automated storytelling systems is to adhere to the natural language generation (NLG) pipeline: start with a set of events or a library of information (fabula), generate discourse for conveying the events such as with paragraph and sentence planning, and last make edits to conform to the structural requirements of the medium (Reiter, Dale, and Feng 2000; Callaway and Lester 2002). Systems which borrow this *fabula-then-discourse* architecture typically take fabula as input and form a storytelling plan around some subset of the fabula (Young et al. 2013).

Research Problem

The *fabula-then-discourse* strategy is not designed well for tasks requiring coordinated plot and communication such as generating narrative fiction. An input fabula (or one generated in isolation) may be coherent and believable, but it may not have the desired attributes for good storytelling plans (e.g., take into account the pragmatics of communicating) and thus the quality of storytelling may suffer. Unless directed, a character agent is not likely to arrange itself for a camera shot or hold its positions at favorable moments, and these actions are typically planned at the fabula level.

Another problem occurs when hierarchical knowledge is introduced. If only primitive fabula-level actions are allowed to substitute variables in discourse patterns, then we risk losing context associated with an action's role in the fabula structure. If abstract fabula actions are introduced as substitutions in isolation from their own sub-plans, then we have a *communicative gap*: discourse actions are not equipped with the knowledge to decompose the task of communicating the abstract pattern.

In prior work, the BiPOCL planner (Winer and Young 2016) was introduced which is a novel algorithm architecture for generating narratives with coordinated plot and discourse. The algorithm interleaves fabula and discourse planning rather than generating fabula first; however, the work is limited to leveraging hierarchical knowledge at the discourse level, but not at the fabula level, and thus did not resolve the communicative gap problem.

In this paper, we first introduce hierarchical fabula and discourse languages and provide examples of unified ground models with merged hierarchical structure. State-of-the-art automated planning systems (Vallati et al. 2015) precompile ground models of actions prior to the planning phase for efficiency. We walk through the computational steps needed to precompile hierarchical ground models in a way that is domain-merging. To the best of our knowledge, a formalism for precompiling a hierarchical planning problem is not documented. The ground steps are characterized in a language that the BiPOCL planner can use to generate the bipartite narrative model consisting of fabula and discourse plans (Young 2007). Finally, we discuss how our approach to merging fabula and discourse improves the expressiveness of plan-based narrative generation systems.

Related Work

A range of projects have sought to address aspects of fabula and discourse generation with natural language. Notably, STORYBOOK (Callaway and Lester 2002) is an end-to-end narrative prose generation system; STORYBOOK takes as input a plot from a fabula planner and uses built-in discourse-level directive patterns that serve to segment the fabula, scaffold clause-level structure, and modify the vocabulary and style. The SCHEHERAZADE system (Li et al. 2013) learns script-like knowledge about fabulas from crowd-sourced examples and is used to generate text with a storytelling style by extracting style parameters from the Project Gutenberg book corpus (Li et al. 2014).

A growing trend in narrative discourse reasoning is to approach storytelling as modeling a reader or viewer's beliefs about fabula (Winer et al. 2015). Dramatis (O'Neill and Riedl 2014) rates the suspense of a fabula using automated planning to calculate the likelihood of a protagonist escaping negative outcomes. Ware and Young (2014) model dimensions of conflict in fabula by measuring the degree to which character plans interfere in a multi-agent plan. Recently, Wu and colleagues (2016) tested a belief-memory framework for inserting flashbacks.

The Darshak system (Jhala and Young 2010) has the goal that the viewer observe the actions in a fabula plan provided as input. It represents film scene segments with abstract plan-based actions. The scenes decompose into low-level camera operators and interval planning is used to construct a presentation timeline. The story is played out by avatars in a 3D game environment such that fabula actions are mapped to animation patterns and discourse low-level actions are mapped to virtual camera instructions. Our work adopts a similar representational framework for *discourse generation as automated cinematography* and resolves the problems associated with expanding this work that we mention above.

Fabula and Discourse Languages

A planning language has an object type taxonomy, a set of predicates describing world conditions with typed arguments, and a set of STRIPS-style declarative action models (Fikes and Nilsson 1972) of the form $\langle \alpha, V, A, P, E \rangle$ which have an action type (α), an ordered list of typed parameters (V) 0 or more which may be volunteering agents ($A \subseteq V$), a set of non-ground literal preconditions (P), and a set of non-ground literal effects (E). If s is an action model of the form $\langle \alpha, V, A, P, E \rangle$, then $pre(s) = P$, $eff(s) = E$, $actors(s) = A$, $params(s) = V$, and $arg(s, i) = V_i$. For simplicity, an action with n disjunctive literals as preconditions or effects is considered equivalent to 2^n actions without disjunction.

The action models are found in a domain specification. Actions in the following example *shooting-world* fabula domain specify that a character agent can orient towards an object, unload a gun from its inventory, aim a gun, and fire a gun. Variables are specified with the ? symbol and may be hyphen-labeled with an object type.

```
(:domain shooting-world
(:types char gun - object)
(:predicates (has ?c - char ?g - gun)
        (alive ?c - char) (facing ?c - char ?o - object)
        (holstered ?g - gun) (loaded ?g - gun)
        (shot-at ?g - gun ?o - object)
        (aimed-at ?g - gun ?o - object))
(:action orient
    :parameters (?c - char ?target - object)
    :precondition (alive ?c)
    :effect (facing ?c ?target)
    :consenting-agents (?c))
(:action deholster
    :parameters (?c - char ?g - gun)
    :precondition (has ?c ?g) (alive ?c)
    :effect (not (holstered ?g))
    :consenting-agents (?c))
(:action aim
    :parameters (?c - char ?g - gun ?target - object)
    :precondition (has ?c ?g) (alive ?c)
          (facing ?c ?target) (not (holstered ?g))
    :effect (aimed-at ?g ?target)
    :consenting-agents (?c))
(:action fire
    :parameters (?c - char ?g - gun ?target - object)
    :precondition (alive ?c) (has ?c ?g) (loaded ?g)
                (aimed-at ?g ?target)
    :effect (shot-at ?g ?t)
    :consenting-agents (?c)))
```

The language afforded by a domain depends on a particular planning problem which specifies the following:

- Instances of typed objects of the form $\langle obj, type \rangle$ such as $\langle$Jane, agent$\rangle$, $\langle$revolver, gun$\rangle$.
- A complete closed-world initial state
- A set of goal conditions

The language of the domain and problem is the set of steps which can be constructed by substituting variables in actions with consistently-typed objects. A step is a ground action instance whose arguments are object instances and whose preconditions and effects are function-free ground literals. Steps are created by swapping each typed variable argument with a typed object instance. A special dummy step d_i is created whose effects are the initial conditions, and a dummy step d_g is created whose preconditions are the goal conditions.

A **plan** is a partially ordered set of steps. A plan starts out with an ordering $d_i \prec d_g$, and then adds new steps from the language to repair flaws. The classic partial-order planning algorithm (Penberthy, Weld, and others 1992) begins with **open condition flaws** for each precondition of d_g. Open condition flaws are resolved by adding steps and ordering them before they are needed. As part of the intermediate representation of plans, we denote links between steps called **causal links**. If s_{need} is a step with an open condition flaw for precondition p, and s is a step s.t. $p \in \texttt{eff}(s)$, then to denote that s resolves open condition flaw $\langle p, s_{need} \rangle$, we add a *causal link* to the plan of the form $s \xrightarrow{p} s_{need}$; this way, if there is some other step s_{threat} s.t. $\neg p \in \texttt{eff}(s_{threat})$ and s_{threat} is possibly ordered after s and before s_{need} in some total ordering of steps, then the algorithm can protect the repaired flaw by detecting the causal link is *threatened* and adding an ordering $s_{threat} \prec s$ or $s_{need} \prec s_{threat}$ (or fail) so that p is protected. A step s is executable when for every total ordering of steps in the plan, all of its preconditions are protected.

Abstract Actions

An *abstract action* is an action with built-in hierarchical knowledge about how to decompose that action into substeps in a *subplan*. The preconditions of an abstract action are guaranteed initial conditions of its subplan, and the effects are the goal conditions of its subplan. A subplan characterizes authored plot points which help constraint the set of possible plans for establishing an action's effect conditions. For simplicity, an action with $n > 0$ subplans is equivalent to n actions each with 1 subplan each.

The plan components (steps, orderings, and causal-links) hold as domain-independent properties of plans. These are needed for specifications on the subplans of abstract actions. To extend the domain above to include abstract actions, we add a few extra types and specialized predicates corresponding to plan-based elements. Below are some examples:

Predicate	Description
`(has-effect s l)`	$l \in eff(s)$
`(has-precond s l)`	$l \in pre(s)$
`(has-actor s a)`	$a \in actors(s)$
`(linked-by s t l)`	$s \xrightarrow{l} t$
`(before s t)`	$s \prec t$
`(has-arg s v)`	$v \in param(s)$
`(has-arg-n s v j)`	$v = arg(s, j)$

For example, predicate `(has-effect s l)` indicates that for any step which binds to s and literal which binds to l, then $l \in \texttt{eff}(s)$.

An abstract action's *decomp* characterizes a partial subplan. The following action to draw and fire a weapon is achieved with a subplan to grab, lift, and fire the weapon.

```
step lit-elm-object
(:action draw
    :parameters (?c - char ?g - gun ?obj - object)
    :precondition (has ?c ?g) (holstered ?g)
                  (loaded ?g) (alive ?c)
                  (not (= ?c ?obj)) (not (= ?g ?obj))
    :effect (shot-at ?c ?g ?obj)
    :decomp (?grab ?lift ?fire - step)
        (linked-by ?grab ?lift (not (holstered ?g)))
        (linked-by ?lift ?fire (aimed-at ?g ?obj))
        (has-effect ?fire (shot-at ?g ?obj))
    :consenting-agents (?c))
```

Let $\langle$Jane, agent$\rangle$, $\langle$revolver, gun$\rangle$, $\langle$Cole, agent$\rangle$, and $\langle$bottle, object$\rangle$ be the objects in a planning problem. Using primitive fabula actions to substitute the step parameters, the possible subplans for the `draw` step are shown below:

- (deholster Jane revolver) (aim Jane revolver Cole) (fire Jane revolver Cole)
- (deholster Cole revolver) (aim Cole revolver Jane) (fire Cole revolver Jane)
- (deholster Jane revolver) (aim Jane revolver bottle) (fire Jane revolver bottle)
- (deholster Cole revolver) (aim Cole revolver bottle) (fire Cole revolver bottle)

The step parameters at the subplan level may also be substituted by abstract actions. For example, a draw-typed action could satisfy all of the conditions of a draw action subplan and substitute one or more substep parameters. As part of the methodology introduced in the next section, an abstract ground step cannot be its own substep, preventing pointless recursion. Subplans may be incomplete; in a plan-space planning algorithm, actions can be inserted between sub-steps to resolve open conditions. A step s can be referenced in a subplan without being added as a substep using the `(not-occurs s)`, where condition $\neg$`(not-occurs` s) is assumed as part of the closed world at the subplan level.

Discourse Actions

Discourse actions are goal-oriented utterances by a speaker to a listener. We adopt the model of a director as speaker whose utterances are camera actions which show fabula to a viewer (i.e., viewers observe and form beliefs about the

content of the storyworld). We designate the predicates `obs` and `bel` for this meaning (e.g., `obs` (p) indicates that the viewer observes p. Observations and beliefs are grounded in the presentation time of the story using specialized interval predicates associated with Allen's interval logic (Allen and Ferguson 1994).

Symbol	Literal	Allen notation
$i \vdash$	`alu` i	i' s.t. (<m i i')
$\dashv i$	`fol` i	i' s.t. (<m i' i)
$\vdash i \dashv$	`dur` i	i' s.t. (c i i')
$< i$	$< i$	i' s.t. (< i' i)

The interval $\dashv i$, read as *at-least-until* (alu), is a function which returns an interval i' where either i' *overlaps* i or i' *meets* i. If `(obs` p $\dashv i$`)` is a precondition of a discourse action, then p is observed in the presentation timeline right at the moment that i begins (and possibly through i). The interval $i \vdash$, read as *following* (fol), refers to an interval i' where either (i' *met-by* i) or (i' *overlapped-by* i). The interval $\vdash i \dashv$ is an interval contained in i, and for any two intervals j, k contained in i, then j and k do not overlap as a default setting. This interval notation allows us to easily characterize observations and beliefs occurring before, during, and after a discourse action.

The predicate `(bel` e i`)` indicates that the reader believes element e during interval i, where i may have unbound endpoints. Beliefs and observations do not exist in the initial state and must be introduced by an action. If s is a step whose precondition is satisfied by ignorance of the form ($\neg$`bel` $(x) \dashv i$) then that ignorance is established by the initial state. A belief at interval i in the effect of an action is threatened only by an inconsistent belief during k where k is during or overlapping i. The threat is resolved by assigning endpoints to k.

If `(obs` a i`)` is a condition and a is an abstract action, then a must be presented during i without any *observation gaps*; meaning that for each goal condition of a's subplan, the viewer must observe the fabula-level actions which establish them if they are not in a's initial conditions. During compilation, any literal in this form is decomposed into a conjunction of observation literals for each substep of a. These observations can be established with other discourse actions. An observation of the form `(obs` a i`)` is considered *threatened* if $p \in pre(a)$ and `(obs` $(\neg p)$ $\dashv i$`)`. It's resolved by a discourse action which shows an action s s.t. $\neg p \in pre(s)$ and $p \in eff(s)$ inserted into the timeline such that `(obs` $(\neg p)$ $\dashv i'$`)`, `(obs` (s) i'`)`, and `(obs` (a) $i' \vdash$`)`. This strategy resolves the communicative gap problem by decomposing the task of communicating a fabula action's subplan such that changes to world conditions are explained.

A *duel* is something that both occurs in a storyworld and is a concept communicated to the viewer through camera shots, and thus serves as a good example of coordinated plot and communication. Shot types extreme close-up (ECU) and medium (MED) are constants referring to shot scale in a camera shot.

```
ECU MED - scale - object
(:action cam-shot
    :parameters (?s - step ?sc - scale ?i - intrvl)
    :precondition ()
    :effect (obs ?s ?i) (bel ?s (fol ?i)))
(:action anticipate-duel
    :parameters (?c1 ?c2 - char ?g1 ?g2 - gun ?i - intrvl)
    :precondition(obs (has ?c1 ?g1) (alu ?i))
                 (obs (has ?c2 ?g2) (alu ?i))
    :effect (bel (draw ?c1 ?g1) (fol ?i))
    :decomp (?gunthreat ?stare1 ?stare2 - step)
            (has-effect ?gunthreat (aimed-at ?g2 ?c1))
            (has-effect ?stare1 (facing ?c1 ?c2))
            (has-effect ?stare2 (facing ?c2 ?c1))
            (cam-shot ?gunthreat MED (dur ?i))
            (cam-shot ?stare1 ECU (dur ?i))
            (cam-shot ?stare2 ECU (dur ?i)))
```

An abstract discourse action specifies what the viewer should observe or believe in the presentation timeline at least until the action's interval, what the viewer will observe during the interval, and what the viewer will believe following the interval. Real value durations for minimum or maximum intervals could be provided as part of the input. The `anticipate-duel` action's subplan characterizes a scenario where two characters are armed and staring at eachother and one is aiming a gun. As a result of this pattern, the viewer anticipates that the other character will draw his or her gun and fire. The `cam-shot` action has a *step*-typed argument which is substituted during instantiation (and could be abstract). If s is a ground step, then a `cam-shot` action is compiled with argument s, and if s is abstract then abstract `cam-shot` actions are calculated for each possible subplan which presents a subplan of s.

Computing Abstract Steps

The merged language is a library of ground abstract actions compiled using our methodology. In this section, we present technical instructions and definitions to calculate abstract actions with multiple levels of decomposition. The formalism can be used to merge any number of hierarchical domains and planning problems provided as input,. Compiling the planning language in advance is beneficial for efficiency because no time is wasted grounding the same action models, and the causal relationships between steps are available to inform heuristics (Helmert 2006; Vallati et al. 2015). Compiling hierarchical domains for this purpose is has not been documented but is not trivial because abstract actions may be nested as sub-steps. This process shows that hierarchical planning such as with merged languages can leverage the same efficiency benefits.

Problem Definition: Given a set of planning problems, generate the library of ground steps.

We skip the subtask of generating primitive steps (steps which have no subplan) and the ground literals which compose the preconditions and effects of these steps, as this portion of the task is a trivial substitution of variable parameters with consistent-typed object instances. The abstract actions may have sub-steps which themselves are abstract, so we start with level $i = 1$: substituting sub-steps with primitive steps, and work our way up to some cutoff level $i = h$ or until there's no sub-step substitutable by a step at level $i - 1$.

Consider a subplan for an abstract step. There are regular object variables, some sub-step variables, and possibly literal variables as well. It's helpful to package these together as follows:

Definition 1 (Subplan Variable Triple) *A **subplan variable triple** is a triple of the form $\langle V_o, V_l, V_s \rangle$ where V_o is a set of object-variables, V_l is a set of non-ground literals, and V_s is a set of step variables whose preconditions and effects are in V_l.*

The subplan variable triple can be found as part of the subplan constraints in an abstract action.

Definition 2 (Abstract Action) *An **abstract action** is a tuple of the form $\langle \alpha, V, P, E, U \rangle$ where $\langle \alpha, V, P, E \rangle$ is an action of type α with parameters V, preconditions P and effects E, and U is a subplan problem, a tuple of the form $\langle U_v, U_r, U_\omega, U_\lambda \rangle$ where U_v is a subplan-variable triple representing sub-parameters, U_r is a set of requirements between variables in U_v, U_ω is a set of ordering constraints between sub-step variables in U_v, and U_λ is a set of causal-links between steps in U_v whose dependencies may be in U_v.*

The goal is to pick some object instances, ground literals, and ground steps from any planning problem to substitute the variables of the subplan.

Definition 3 (Ground Library Triple) *A **ground library triple** is a triple of the form $\langle G_o, G_l, G_s \rangle$ where G_o is a set of object instances, G_l is a set of ground literals, and G_s is a set of ground steps whose preconditions and effects are in G_l.*

Since we start the task having already generated all primitive ground steps from all problems, we start with a ground library triple which includes all of the object instances, ground preconditions, ground effects, and ground steps from this process. To achieve the goal of substituting the variables in the subplan, the task is to find a set of bindings to map variables to ground elements.

Definition 4 (Subplan Binding Triple) *Given a subplan variable triple $\langle V_o, V_l, V_s \rangle$ and a ground library triple $\langle G_o, G_l, G_s \rangle$, a **subplan binding triple** is a triple of the form $\langle B_o, B_l, B_\Sigma \rangle$ such that $\forall v_o \in V_o, v_l \in V_l, \sigma \in V_s$, $\exists g_o \in G_o, g_l \in G_l, s \in G_s$, where $\langle v_o, g_o \rangle \in B_o$, $\langle v_l, g_l \rangle \in B_l$, and $\langle \sigma, s \rangle \in B_\Sigma$. The subplan binding triple is **valid** just when no variable is in two bindings.*

The subplan binding triple is the intermediate goal. To reach this goal, construct a list where each entry in the list corresponds to a sub-step variable in the subplan. Start by substituting each sub-step variable with a ground step, and add the bindings which make the substitution possible to that entry in the list.

Definition 5 (Subplan Step Binding List) *Given a subplan variable triple $\langle V_o, V_l, V_s \rangle$ and a ground library triple $\langle G_0, G_l, G_s \rangle$, a **subplan step binding list** B_S is a list of the form $B_1, ..., B_k$ where $k = |V_s|$ and $\forall B_i \in B_S$, B_i is a subplan binding triple of the form $\langle B_o, B_l, \langle \sigma_i, s_i \rangle \rangle$ s.t. $\sigma_i \in V_s$, $s \in G_s$, and B_o, B_l are bindings between all variable arguments in σ_i and ground arguments in s_i, and $\forall \sigma' \in V_s$, $\exists \langle B'_o, B'_l, \langle \sigma', s' \rangle \rangle \in B_S$.*

This list is used to solve the subplan binding problem by checking to see if the configuration of step substitutions are consistent.

Definition 6 (Subplan Binding Problem) *The **subplan binding problem** is a tuple $\langle V, G \rangle$ where $V = \langle V_o, V_l, V_s \rangle$ is a subplan variable triple and $G = \langle G_0, G_l, G_s \rangle$ is a ground library triple. A **solution** is a valid subplan binding triple $\langle B_o, B_l, B_\Sigma \rangle$ s.t. $\langle B^0_o, B^0_l, b_0 \rangle, ..., \langle B^k_o, B^k_l, b_k \rangle$ is a subplan step binding list and $\langle B_o, B_l, B_\Sigma \rangle = \langle \bigcup_{i=0}^k B^i_o, \bigcup_{i=0}^k B^i_l, \bigcup_{i=0}^k \{b_i\} \rangle$.*

The ground subplan is formed by adding causal links and orderings between ground steps which substitute the sub-step variables. Once all step substitutions are made, it's possible that some causal links cannot be formed (such as causal links where the dependency was not specified). At this stage, new candidate subplans are added for every valid dependency condition.

Definition 7 (Subplan Causal-Link Problem) *The **subplan causal-link problem** is a triple $\langle V, B, \Lambda \rangle$ where $V = \langle V_o, V_l, V_s \rangle$ is a subplan variable triple, $B = \langle B_o, B_l, B_\Sigma \rangle$ is a subplan binding triple, and Λ is a set of causal links of the form $\sigma_u \xrightarrow{p} \sigma_v$ where $\sigma_u, \sigma_v \in V_s$. A **solution** is a set of causal links L of the form $s_u \xrightarrow{l} s_v$ s.t. $\forall \langle \sigma_u \xrightarrow{p} \sigma_v \rangle \in \Lambda$, $\exists \langle s_u \xrightarrow{l} s_v \rangle \in L$ where $\langle \sigma_u, s_u \rangle, \langle \sigma_v, s_v \rangle \in B_\Sigma$, $l \in G_l$, and if $p \in V_l$ then $\langle p, l \rangle \in B_l$.*

The `Linkify` Algorithm (2) substitutes causal links between step variables to generate all possible causal links between corresponding ground steps.

Definition 8 (Subplan Problem) *The **subplan problem** is a tuple $\langle V, G, \Omega, \Lambda \rangle$ where $V = \langle V_o, V_l, V_s \rangle$ is a subplan variable triple, $G = \langle G_o, G_l, G_s \rangle$ is a ground library triple, Ω is a set of ordering constraints between variables in V_s, and Λ is a set of causal links between variables in V_s. A **solution** is called a **subplan**, a tuple of the form $\langle B, S, O, L \rangle$ s.t. $B = \langle B_o, B_l, B_\Sigma \rangle$ is a subplan binding triple which solves the subplan binding problem $\langle V, G \rangle$, $S = \{s : \langle \sigma, s \rangle \in B_\Sigma\}$, O is a set of ordering constraints over sub-steps in S, and L is a solution to the subplan causal-link problem $\langle V, B, \Lambda \rangle$.*

The `Subplannify` Algorithm (1) takes a subplan problem as input and returns all possible subplans.

Finally, new ground abstract steps are added to the library for every subplan by propagating the ground subplan parameters to the step's arguments, preconditions, and effects. If there are leftover non-ground arguments, we create new abstract steps for every possible consistently-typed substitution from the language.

Definition 9 (Abstract Step) *Given a ground library triple $G = \langle G_o, G_l, G_s \rangle$ and an abstract action $\langle \alpha, V, P, E, \langle U_v, U_r, U_\omega, U_\lambda \rangle \rangle$, an **abstract step** is a tuple $\langle V_G, P_G, E_G, \pi_{sub} \rangle$ where V_G are ground arguments, P_G, E_G are sets of ground preconditions and effects corresponding to P and E, respectively, whose arguments are in V_G, and π_{sub} is a subplan which solves the subplan problem $\langle U_v, G, U_\omega, U_\lambda \rangle$.*

Algorithm 1 `Subplannify`

Input: $\langle V, G, \Omega, \Lambda, \Upsilon, i\rangle$ where $\langle V, G, \Omega, \Lambda\rangle$ is a subplan problem, $\Upsilon = \{\}$, and i is an integer.
Output: The set of solutions to the subplan problem.

$\mathcal{B} :=$ `Find-Subplan-Bindings`(V, G)
for each $\langle B_o, B_l, B_\Sigma\rangle \in \mathcal{B}$, **do**
  Skip if $|B_\Sigma| < k$ or $\neg\exists s \in G^i_s, \langle\sigma, s\rangle \in B_\Sigma$
  $S := \{s : \langle\sigma, s\rangle \in B_\Sigma\}$.
  $O := \{s_u \prec s_v : \langle\sigma_u, s_u\rangle, \langle\sigma_v, s_v\rangle \in B_\Sigma$ and $\sigma_u \prec \sigma_v \in \Omega\}$
  $\mathcal{L} :=$ `Linkify`(V, B, Λ)
  $\Upsilon+ = \{\langle\langle B_o, B_l, B_\Sigma\rangle, S, O, L\rangle$ for each $L \in \mathcal{L}\}$.
end for
return Υ

Algorithm 2 `Linkify`

Input: $\langle\langle B_o, B_l, B_s\rangle, \Lambda\rangle$, a subplan causal-link problem.
Output: The solutions to the subplan causal-link problem.

$L :=$ empty nested list
for each $\langle\sigma_u \xrightarrow{p} \sigma_v\rangle_i \in \Lambda$ **do**
  $L[i]$.append($\langle s_u \xrightarrow{q} s_v\rangle$) **if** $\exists\langle p, q\rangle \in B_l$
  Else:
  $Q := \{q : q \in eff(s_u) \cap pre(s_v), \langle\sigma_u, s_u\rangle, \langle\sigma_v, s_v\rangle \in B_\Sigma\}$
  $L[i]$.append($\langle s_u \xrightarrow{q} s_v\rangle$ for each q in Q)
end for
return $\mathcal{L} := \prod_i^{|\Lambda|} L[i]$

Abstract steps are compiled bottom-up (with increasing depth of recursion in the hierarchical structure). For $i = 1 : h$, where i is the level of depth such that a step at $i = 1$ has only primitive sub-steps, we compute steps at level i such that at least one sub-step at level $i-1$ until some cutoff level h to avoid infinite recursion.

When abstract steps are added to a plan during plan generation, steps in its subplan are either added or substituted by steps already in the plan. For completeness, all valid substitutions are considered.

Comparative Analysis

The performance of a narrative discourse generation approach is often judged based on the kinds of examples that can be formulated. We use a comparative analysis to evaluate our approach with respect to kinds of utterances related to narrating. Recall that BiPOCL (Winer and Young 2016) can use the ground unified actions computed by our methodology in section 4 to generate fabula and discourse plans. We use BiPOCL as a representative for our approach in the context of narrative generation. First, we compare our approach to the Darshak system (Jhala and Young 2010) which was introduced in related work. Then, we introduce a typology of narration actions which form the basis for our comparison with other systems.

We use a representational framework for discourse planning which is similar to Darshak. Because Darshak takes fabula as input, it can fail to generate a scene architecture when an abstract discourse step is needed in the solution but its instantiated fabula-level constraints are inconsistent with the input. This situation is avoided by our approach because the fabula constraints found in the Darshak model are analogous to the decomp requirements which motivate fabula generation. The fabula-first strategy inadvertently penalizes discourse actions which characterize complex fabula-level constraints, yet these constraints are critical for coordinating fabula and discourse. Our approach eliminates this balancing act and incentivizes fabula-level constraints from an efficiency standpoint because constraints help limit the breadth of the search space (fewer abstract steps will exist with consistent subplans).

Storytelling Acts

Coordination in fabula and discourse is central for extending the kinds of utterances supported in automated narration. We outline a typology of actions as a proposed extension to the *narrate* speech act (Searle 1969).

Narrate Act	arg_0	arg_1	arg_2
`show`	e	i	
`inform`	e	$\prec$	i
`withhold`	e	i	i'
`suggest`	e	i	i'
`describe`	$desc$	e	
`relate`	rel	e_0	e_1
`evaluate`	$\{+,-\}$	e	

Arguments are fabula elements (e) (e.g., entities, literals, steps, etc.), intervals (i, i'), partial ordering ($\prec$), symbols representing valence in set $\{+,-\}$, and predicates for description of an arg ($desc$) and for relations between args (rel). Narration themselves have intervals for their duration in the presentation (the first 3 are shown transparent).

- The `show` act lets the story unfold before the eyes/ears of the reader (e.g., `obs`).
- An action `informs` when it has an effect which is a belief whose content is not observed, such as because it is learned indirectly. A plan action `informs` b if it has preconditions `(not (obs` b`) (alu ?i))` and `(not (bel` b`) (alu ?i))`, subplan requirement `(not (obs` b`) (dur ?i))`, and effect `(bel` b `(fol ?i))`.
- `Withholding` is when the viewer is required not to observe or believe an element during some interval.
- The `suggest` act is when the viewer is led to anticipate that some element will be observed at a future interval.
- A `describe` act is when the viewer recognizes or is led to believe a subjective description about an element.
- The `relate` act is when the viewer recognizes or is led to believe a subjective relationship between two elements.
- An `evaluate` act is a `describe` act specific to valence or sentiment.

Figure 2 shows a comparison of BiPOCL to other systems on the basis of their support of narrate acts. BiPOCL has

system	whold	inform	suggest	desc	rel	eval
BiPOCL	yes	yes	yes	yes	yes	yes
Darshak	no	no	limited	limited	limited	yes
SBOOK	no	no	no	yes	yes	yes
SCHZAD	no	yes	yes	no	no	yes

Figure 1: Narration acts supported by generation systems

a characterization of observations, beliefs, and fabula constraints which makes these narrate acts possible. Darshak cannot characterize gaps of observation and has limited deployment of fabula-level constraints as discussed, but can depict emotion using camera features like shot angle. STORYBOOK (Callaway and Lester 2002) lacks observation and belief modeling but has built-in directives for controlling descriptions, relations, and valence. SCHEHERAZADE (Li et al. 2014) uses word choice to influence valence but lacks the same goal-oriented influence for description and relation through its style parameters. Its script representation of fabula can be used to prompt or leverage expectations about next events at the discourse level.

Conclusion

Our work describes the merits of merging fabula and discourse for narrative planning with abstract actions. The pipeline bottleneck caused by the fabula-then-discourse approach has inadvertently limited the expressiveness of plan-based narratives in practice. Our methodology merges fabula and discourse planning which alleviates the problem. We present a formalism for this method which is consistent with a modular approach to fabula and discourse and can be used to support efficiency gains associated with precompiling steps. The merits of our approach are compared to other systems on the basis of a new typology of narrate acts. We hope our work encourages more sophisticated models of narrative discourse reasoning.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1654651, for which the authors are thankful.

References

Allen, J. F., and Ferguson, G. 1994. Actions and events in interval temporal logic. *Journal of logic and computation* 4(5):531–579.

Callaway, C. B., and Lester, J. C. 2002. Narrative prose generation. *Artificial Intelligence* 139(2):213–252.

Chatman, S. B. 1980. *Story and discourse: Narrative structure in fiction and film*. Cornell University Press.

Fikes, R. E., and Nilsson, N. J. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Jhala, A., and Young, R. M. 2010. Cinematic visual discourse: Representation, generation, and evaluation. *TCIAIG* 2(2):69–81.

Kapadia, M.; Frey, S.; Shoulson, A.; Sumner, R. W.; and Gross, M. H. 2016. CANVAS: computer-assisted narrative animation synthesis. In *Symposium on Computer Animation*, 199–209.

Li, B.; Lee-Urban, S.; Johnston, G.; and Riedl, M. 2013. Story Generation with Crowdsourced Plot Graphs. In *AAAI*.

Li, B.; Thakkar, M.; Wang, Y.; and Riedl, M. O. 2014. Storytelling with adjustable narrator styles and sentiments. In *ICIDS*, 1–12. Springer.

Meehan, J. R. 1977. Tale-spin, an interactive program that writes stories. In *IJCAI*, volume 77, 91–98.

O'Neill, B., and Riedl, M. 2014. Dramatis: A computational model of suspense. In *AAAI*, volume 2, 944–950.

Penberthy, J. S.; Weld, D. S.; et al. 1992. UCPOP: A sound, complete, partial order planner for ADL. *KR* 92:103–114.

Reiter, E.; Dale, R.; and Feng, Z. 2000. *Building natural language generation systems*, volume 33. MIT Press.

Riedl, M. O., and Young, R. M. 2010. Narrative planning: Balancing plot and character. *JAIR* 39(1):217–268.

Schank, R. C. 1995. *Tell me a story: Narrative and intelligence*. Northwestern University Press.

Searle, J. R. 1969. *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press.

To, S. T.; Langley, P.; and Choi, D. 2015. A unified framework for knowledge-lean and knowledge-rich planning. In *ACS*, 20.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.

Ware, S. G.; Young, R. M.; Harrison, B.; and Roberts, D. L. 2014. A computational model of plan-based narrative conflict at the fabula level. *TCIAIG* 6(3):271–288.

Winer, D. R., and Young, R. M. 2016. Discourse-driven narrative generation with bipartite planning. In *INLG Conference*.

Winer, D. R.; Amos-Binks, A. A.; Barot, C.; and Young, R. M. 2015. Good timing for computational models of narrative discourse. In *Workshop on Computational Models of Narrative*, volume 45. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Wu, H.-Y.; Young, R. M.; and Christie, M. 2016. A cognitive-based model of flashbacks for computational narratives. In *AIIDE*.

Young, R. M.; Ware, S.; Cassell, B.; and Robertson, J. 2013. Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives. *Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative* 37(1-2):41–64.

Young, R. M. 2007. Story and discourse: A bipartite model of narrative generation in virtual worlds. *Interaction Studies* 8(2):177–208.