

Searching for Stories

Yun-Gyung Cheong, Byung-Chull Bae
Procedural Content Generation in
Games, 2013

Outline

- Stories, Games, and Quests
- Story generation systems
- Planning algorithms
- Plan representation
- Generating game world automatically
- Lab: HTN plan

Story and Game

- Stories endow gameplay meanings by providing context and goals
 - Set the mood and general theme
 - Motivate the player to take actions
 - Progress the game (sometimes)
- Stories in games
 - Bioshock Infinite, The last of us, Heavy Rain, L.A. Noire, Wake Alan, To the Moon, The Secret of Monkey Island, The Walking Dead, etc.
 - Serious Games

Story and Quests

- Integrate a storyline with gameplay
 - By giving a player something to do
 - E.g., Retrieve an item, help an NPC, defeat a villain, transfer goods
- Two way interaction
 - Quests are motivated by the storyline
 - Quests advance the story when completed

Why Generate Stories?

- Replayability
- Personalization

Story space explodes



Authoring Bottleneck!!

Endings of Heavy Rain

- +20 endings: 6 different endings for 4 characters



Endings of Heavy Rain

- Ethan's endings
 - Ethan is in prison and hangs himself
 - Ethan kills himself at his apartment when Shaun is not saved
 - Ethan is released from jail and sees Shaun
 - Ethan saves Shaun and lives in an apartment with him
 - Ethan saves Shaun and lives with him and Madison
 - Ethan can also be killed by the Police after saving Shaun but letting Scott live.

Walking Dead: Season 1



<http://venturebeat.com/2013/03/31/the-walking-dead-season-one-plot-graph/>

Story as a Plan

- Story: a sequence of actions that transform the world state to a desired state
- Plan: a sequence of actions to achieve a goal state from an initial state
 - **Action** is an instantiation of an operator
 - E.g, Fly (?x, ?from, ?to)
 - PRECONDITIONS: At(?x, ?from)
 - EFFECTS: At(?x, ?to)
 - E.g., Fly (Justin, CPH, Paris)

Planning-based Story Generation Systems

- Tale-Spin (Meehan, 1976)
 - Simulation of a character who tries to solve a problem
 - Inference, goal-based planning, memory, relationship between characters, character's personality
- Universe (Lebowitz, 1985)
 - Planning (plot fragment containing sub-goals)
- Mimesis (Young, 2000)
 - Partial-order planning
- Interactive Storytelling (Cavazza et al., 2002)
 - HTN (Hierarchical Task Network)
- Façade (Mateas and Stern, 2003)
 - Beat as a dramatic action that encodes goals, preconditions and effects

Planner Generated Story Examples

TALE-SPIN

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe walked to the oak tree. He ate the beehive.

Mis-Spun stories

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe threatened to hit Irving if he didn't tell him where some honey was.

Planner Generated Story Examples

TALE-SPIN

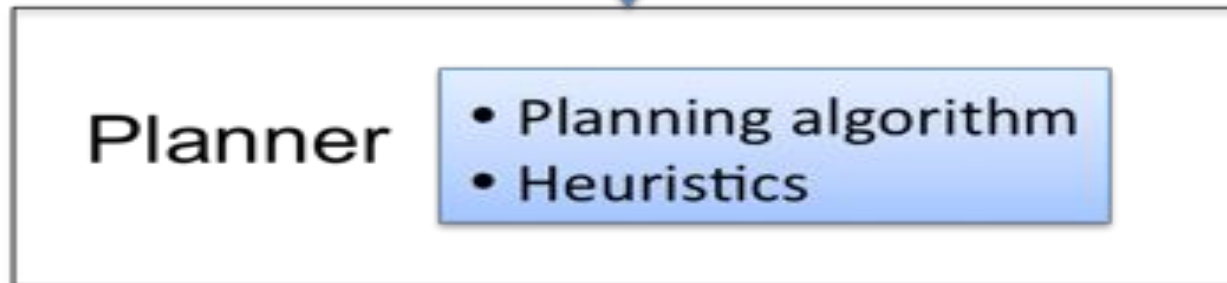
One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe walked to the oak tree. He ate the beehive.

Partial-order Planner

Dr. Evil went to a bank. Dr. Evil withdrew cash from his account to buy a gun. Dr. Evil traveled to a gun store. Dr. Evil bought a gun. Dr. Evil traveled to the White House. Dr. Evil shot the president with his gun.

Planning as a search process

Input: planning problem (consisting of domain description and problem description)



Output: A plan or a set of plans (where a plan = a sequence of actions that will achieve the goal state)

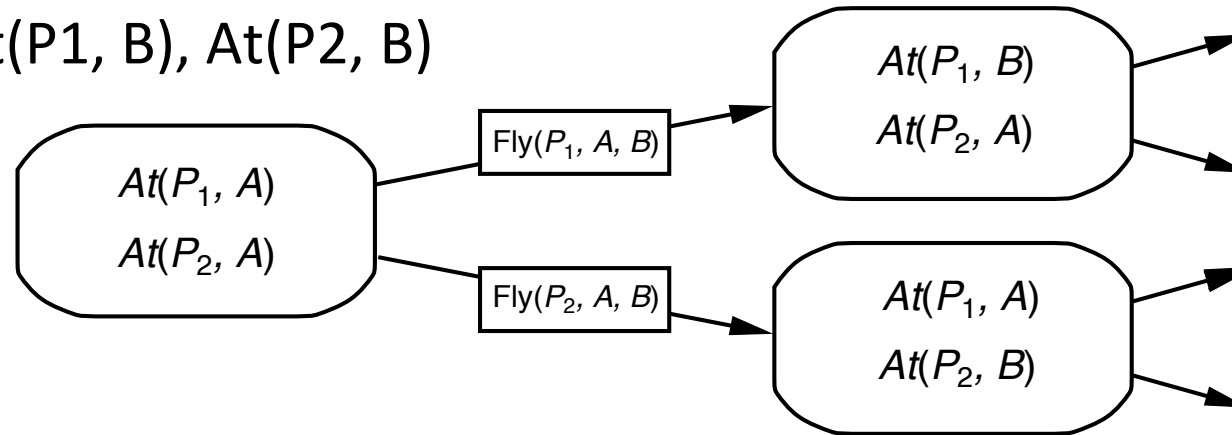
Forward state-space search algorithm

1. construct the root node as the initial state
2. select a non-failure node
 - If such nodes are not found, return 'no solution' and exit
 - if the **goal state is true**, return the path from the initial state up to the current node as a solution and exit
3. select an operator whose preconditions are true
 - if no operators are applicable, mark the node as 'Failure' and go to step 2
4. construct children nodes by applying the operator
 - if the number nodes in the graph exceeds a predefined maximum search nodes, return 'over search limit' and exit
5. go to step 2

State-space search example

GOAL: $At(P_1, B), At(P_2, B)$

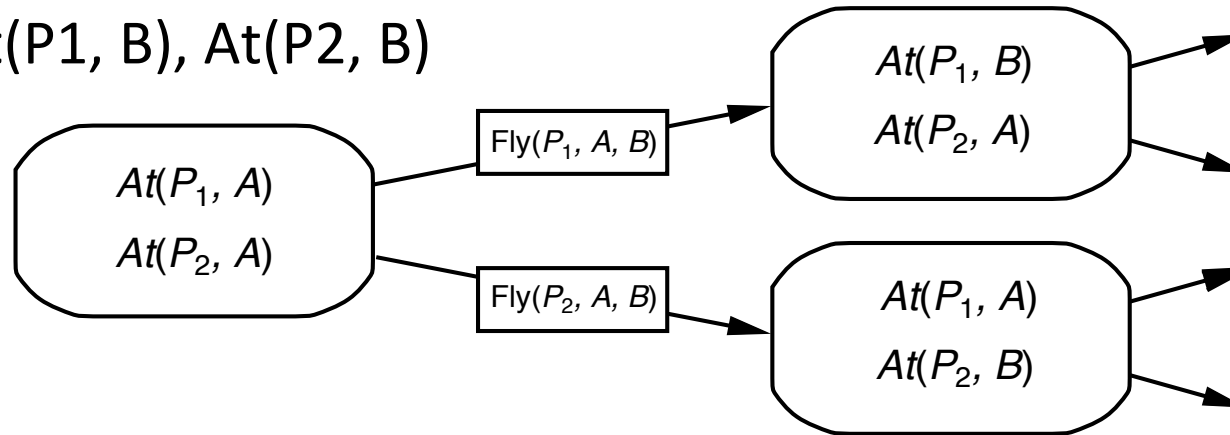
Forward
state-space
search



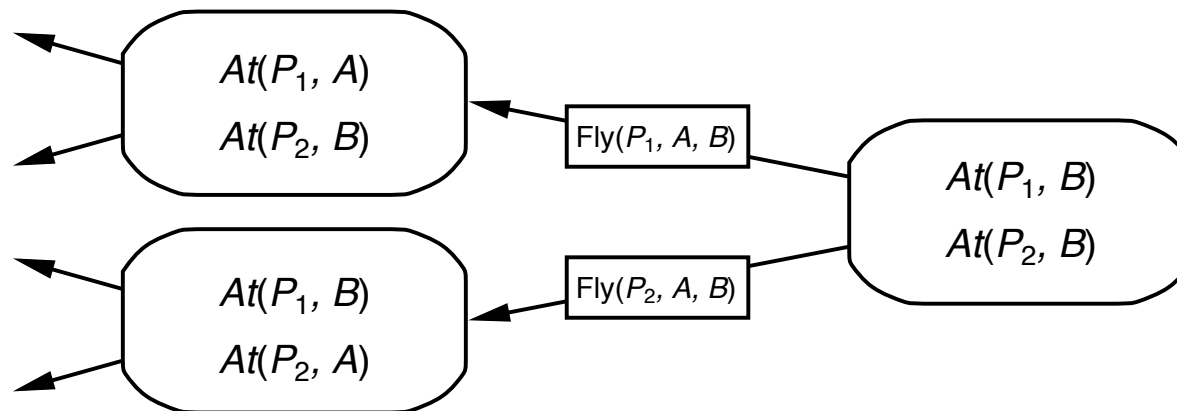
State-space search example

GOAL: $At(P_1, B), At(P_2, B)$

Forward
state-space
search

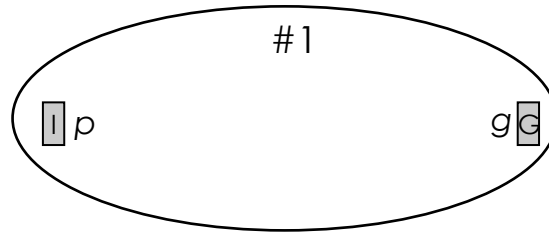


Backward
state-space search

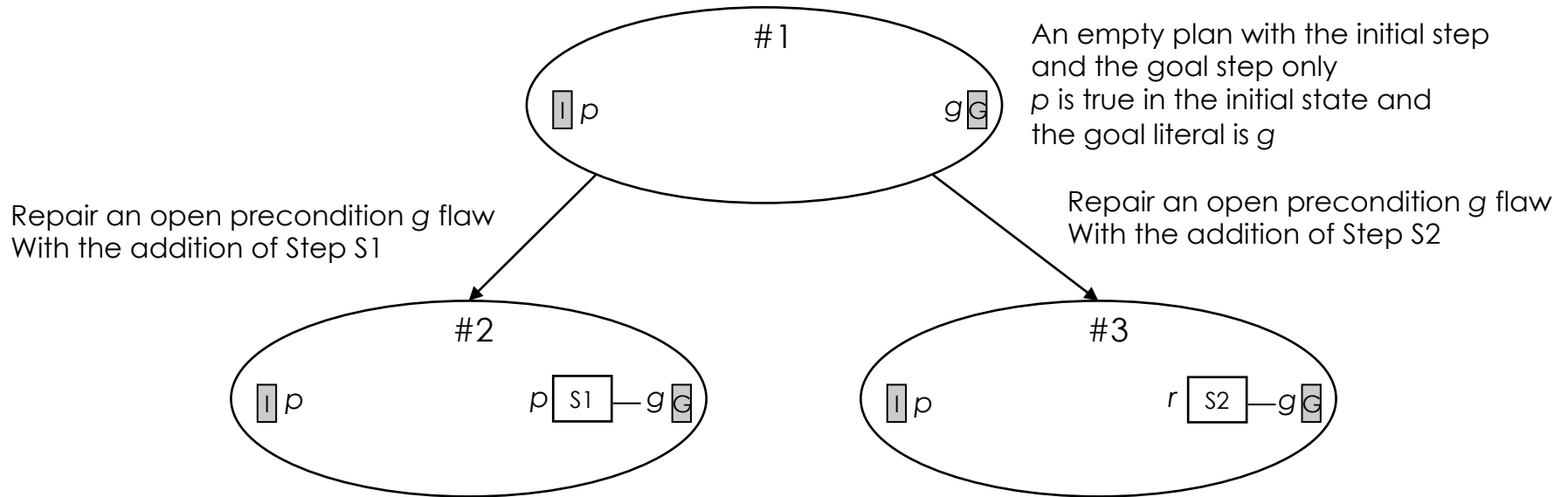


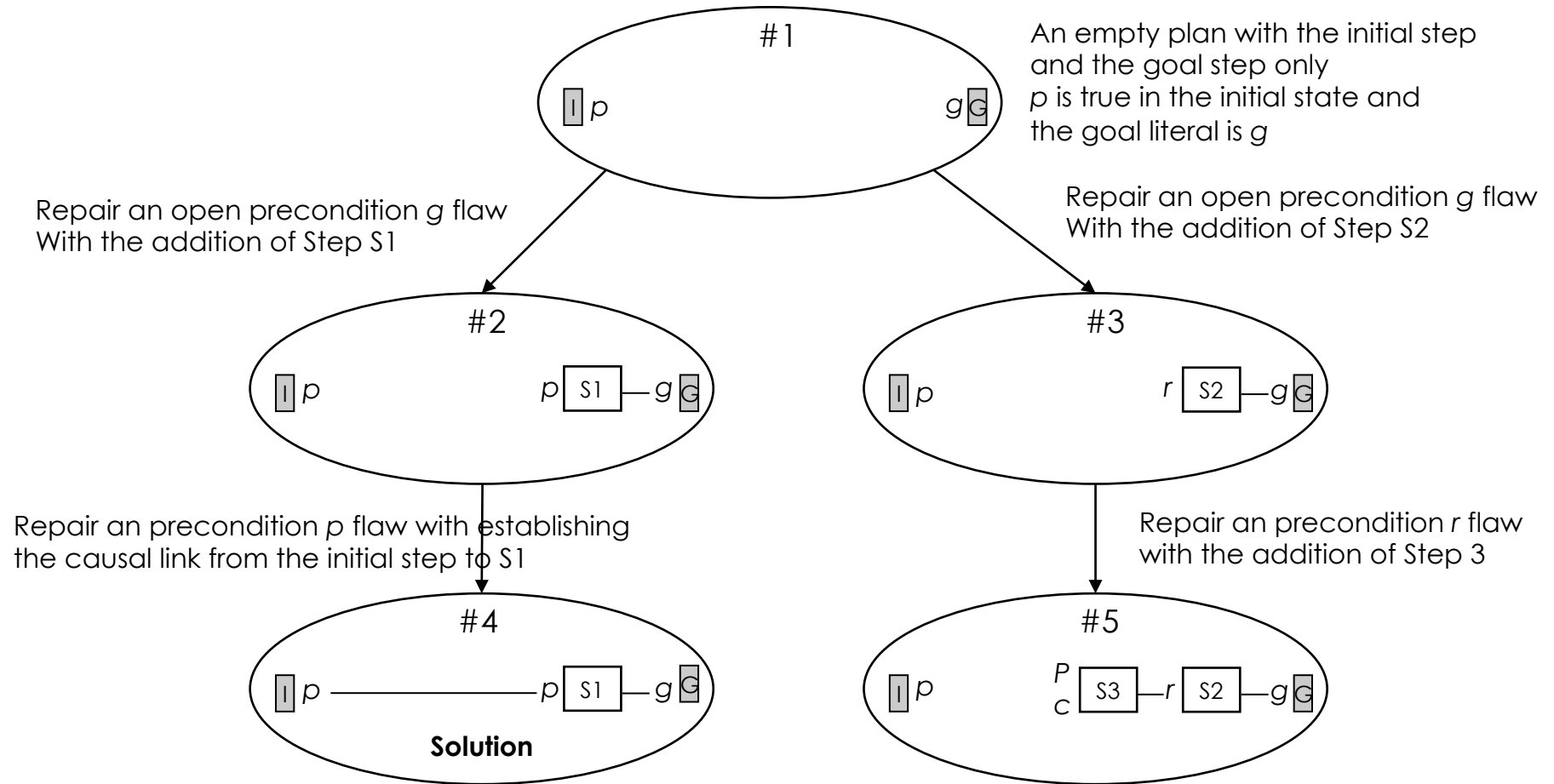
Plan-space Search

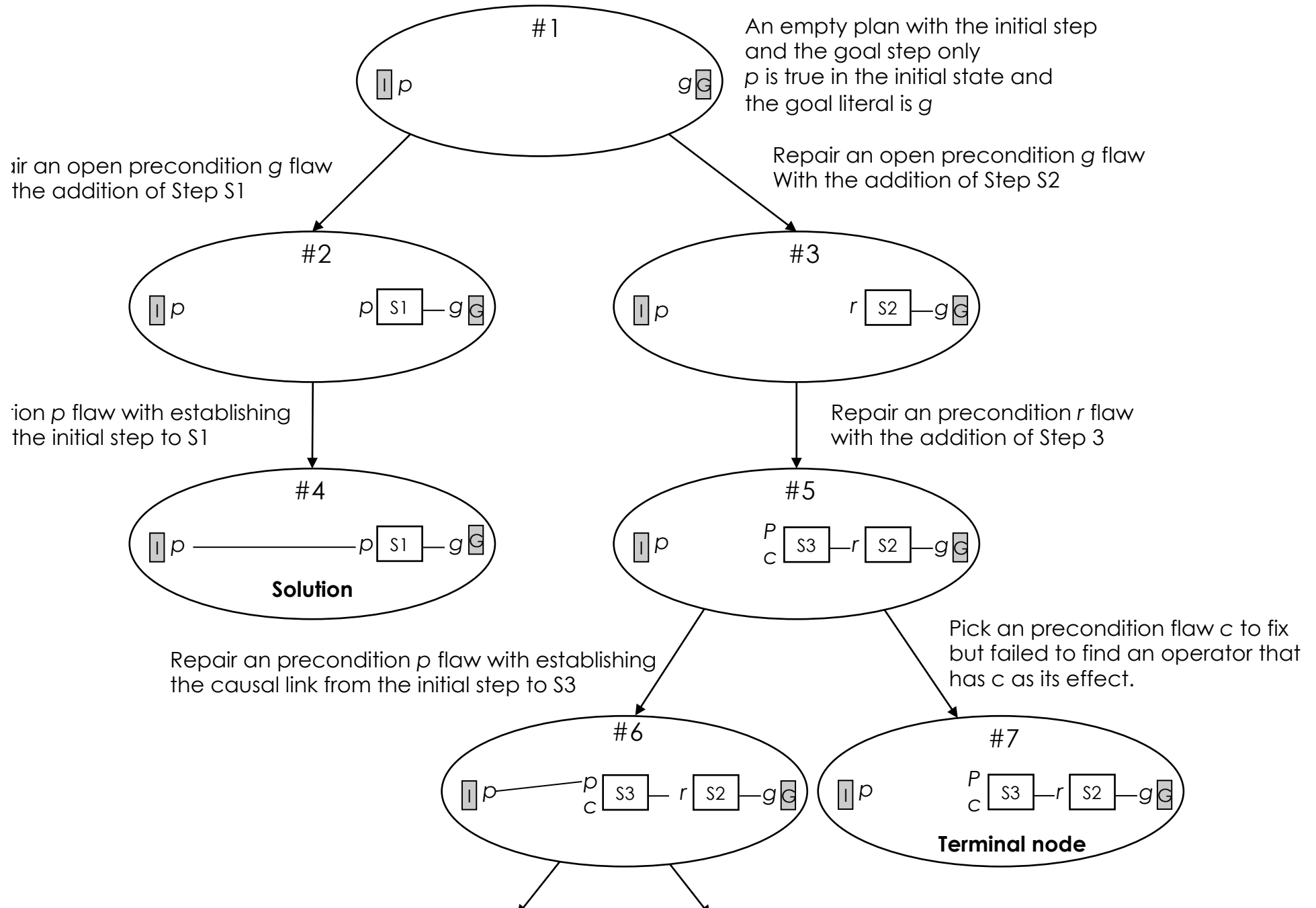
1. construct the root node as the planning problem
2. select a non-failure node (based on its heuristic value)
3. select a flaw in the node
 - if **no flaws are found**, return the node as a solution and exit
4. construct children nodes by repairing the flaw
 - if the flaw is an open precondition
 - a) establishes a causal link from an existing plan step, or
 - b) adds a new plan step whose effects establish the precondition
 - if the flaw is a threat
 - a) add a temporal ordering constraint so that the threatened causal link is not intervened, or
 - b) add a binding constraint to separate the threatening step from the steps involved in the threatened causal link.
 - if the flaw is not repairable, mark the node as 'Failure'
 - if the number nodes in the graph exceeds a predefined maximum search nodes, return 'over the search limit' and exit
5. go to step 2



An empty plan with the initial step
and the goal step only
 p is true in the initial state and
the goal literal is g







Total-order vs. Partial-order plan

- A total-order plan
 - specifies the temporal ordering constraint of every step in the plan
- A partial-order plan
 - specifies only those temporal orderings that must be established to resolve threats.
- Example goal: purchasing milk and bread in a grocery store
 - A total-order plan: a) to purchase milk first and to purchase bread, and b) to purchase bread first and to purchase milk.
 - A partial-order plan does not specify the ordering constraint and defers the decision until when it is necessary.

Heuristic Function

- Estimates the length or the cost of the solution
 - E.g., the length of the current partial plan + number of flaws
 - E.g., the length of the current plan + number of states that are different from the goal state

POP is computationally expensive



Domain Model

- A library of plan operator templates that encode knowledge in a particular domain
- Example
 - Alex, is on the *rooftop of a building (initial state)*. His goal is to be *on the ground level (goal state)* of the building without being injured (*goal state*).
 - Available options are
 - take a lift (Plan 1)
 - walk down the stairs (Plan 2)
 - jump from the roof (Plan 3)

STRIPS

- Stanford Research Institute Problem Solver (Fikes and Nilsson, 1971)
- A state representation
 - Propositional literals or first-order logic literals
- Closed-world assumption
 - conditions that are not explicitly specified are considered as false
 - only positive literals are used for the description of initial states, goal states, and preconditions
 - Effects may include negative literals

STRIPS representation example

- Problem: Alex on the top of a building wants to be on the ground level
- Initial state representation
 - $\text{At}(\text{Alex}, \text{Rooftop}) \wedge \text{Alive}(\text{Alex}) \wedge \text{Walkable}(\text{Rooftop}, \text{Ground}) \wedge \text{Person}(\text{Alex}) \wedge \text{Place}(\text{Rooftop}) \wedge \text{Place}(\text{Ground})$
- Goal State representation
 - $\text{At}(\text{Alex}, \text{Ground}) \wedge \text{Alive}(\text{Alex})$
- Action representation
 - Action (WalkStairs (p, from, to))
 - PRECONDITION: $\text{At}(p, \text{from}) \wedge \text{Walkable}(\text{from}, \text{to}) \wedge \text{Person}(p) \wedge \text{Place}(\text{from}) \wedge \text{Place}(\text{to})$
 - EFFECT: $\neg \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to})$

ADL (Action Description Language)

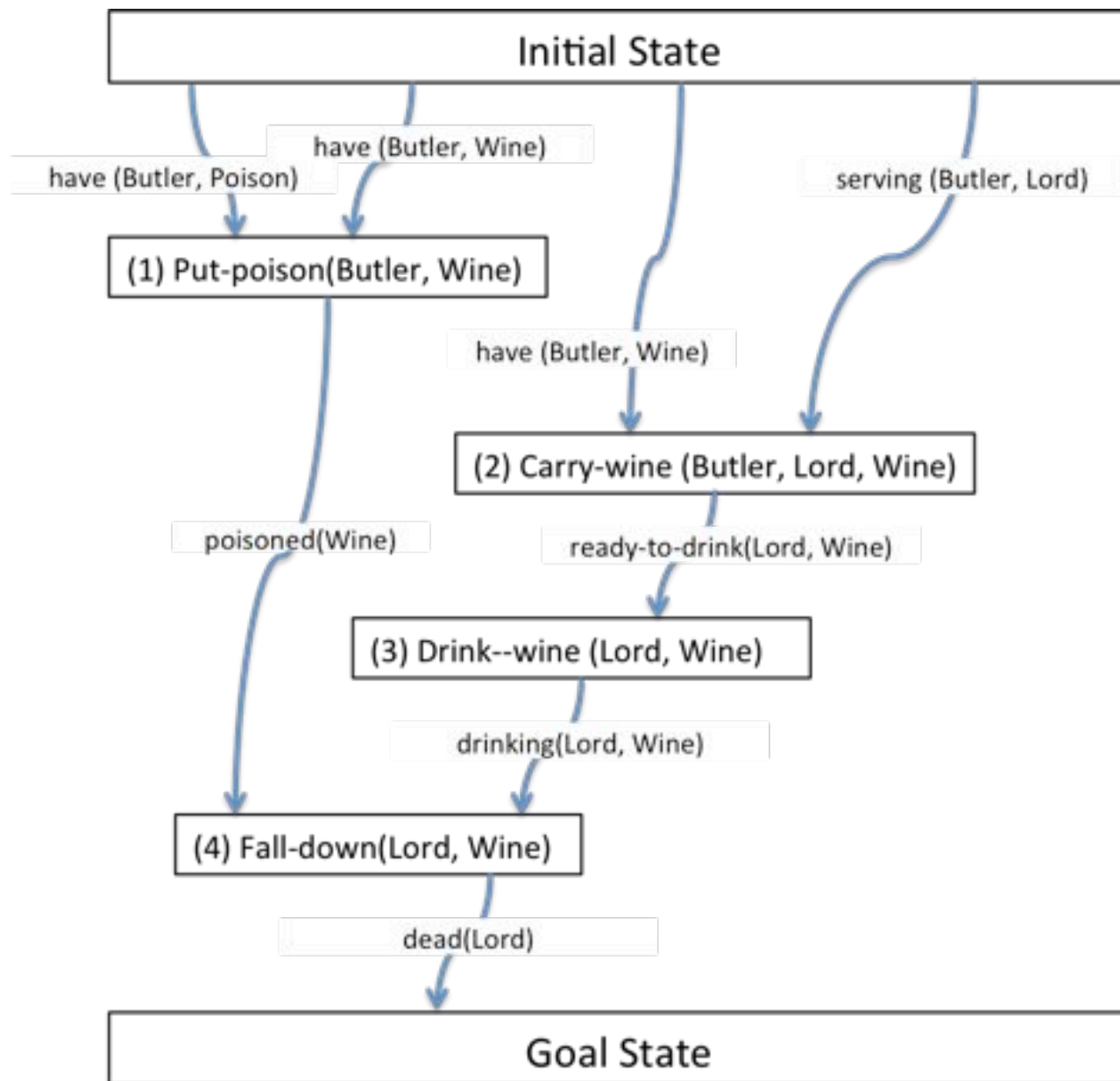
- More expressive than STRIPS
- Open-world assumption
 - Both positive and negative literals are allowed
- Quantified variables: All, existential
- Combination of conjunction and disjunction are allowed in the goal state description.
- Conditional effects are allowed.
- Equality and non-equality predicates and type in variable
 - e.g., (from \neq to)
 - e.g., (p: Person), (from: Location))

ADL representation example

- Initial state representation
 - $\text{At}(\text{Alex}, \text{Rooftop}) \wedge \neg \text{Dead}(\text{Alex}) \wedge \text{Walkable}(\text{Rooftop}, \text{Ground}) \wedge \text{Person}(\text{Alex}) \wedge \text{Place}(\text{Rooftop}) \wedge \text{Place}(\text{Ground}) \wedge \text{Wearing}(\text{Alex}, \text{Parachute}) \wedge \neg \text{Injured}(\text{Alex}) \wedge \text{Thing}(\text{Parachute})$
- Goal State representation
 - $\text{At}(\text{Alex}, \text{Ground}) \wedge \neg \text{Dead}(\text{Alex}) \wedge \neg \text{Injured}(\text{Alex})$
- Action representation
 - Action (WalkStairs (p: Person, from: Place, to: Place))
 - PRECONDITION: $\text{At}(p, \text{from}) \wedge (\text{from} \neq \text{to}) \wedge (\text{Walkable}(\text{from}, \text{to}) \vee \neg \text{Working}(\text{Lift}))$
 - EFFECT: $\neg \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to})$
 - Action (JumpFromRooftop (p: Person, from: Place, to: Place, sth:Thing))
 - PRECONDITION: $\text{At}(p, \text{from}) \wedge (\text{from} \neq \text{to}) \wedge \text{Emergent}(p)$
 - EFFECT: $\neg \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to}) \wedge (\text{when } \text{Wearing}(p, \text{Parachute}): \neg \text{Dead}(p) \wedge \neg \text{Injured}(p))$

A Story Plan

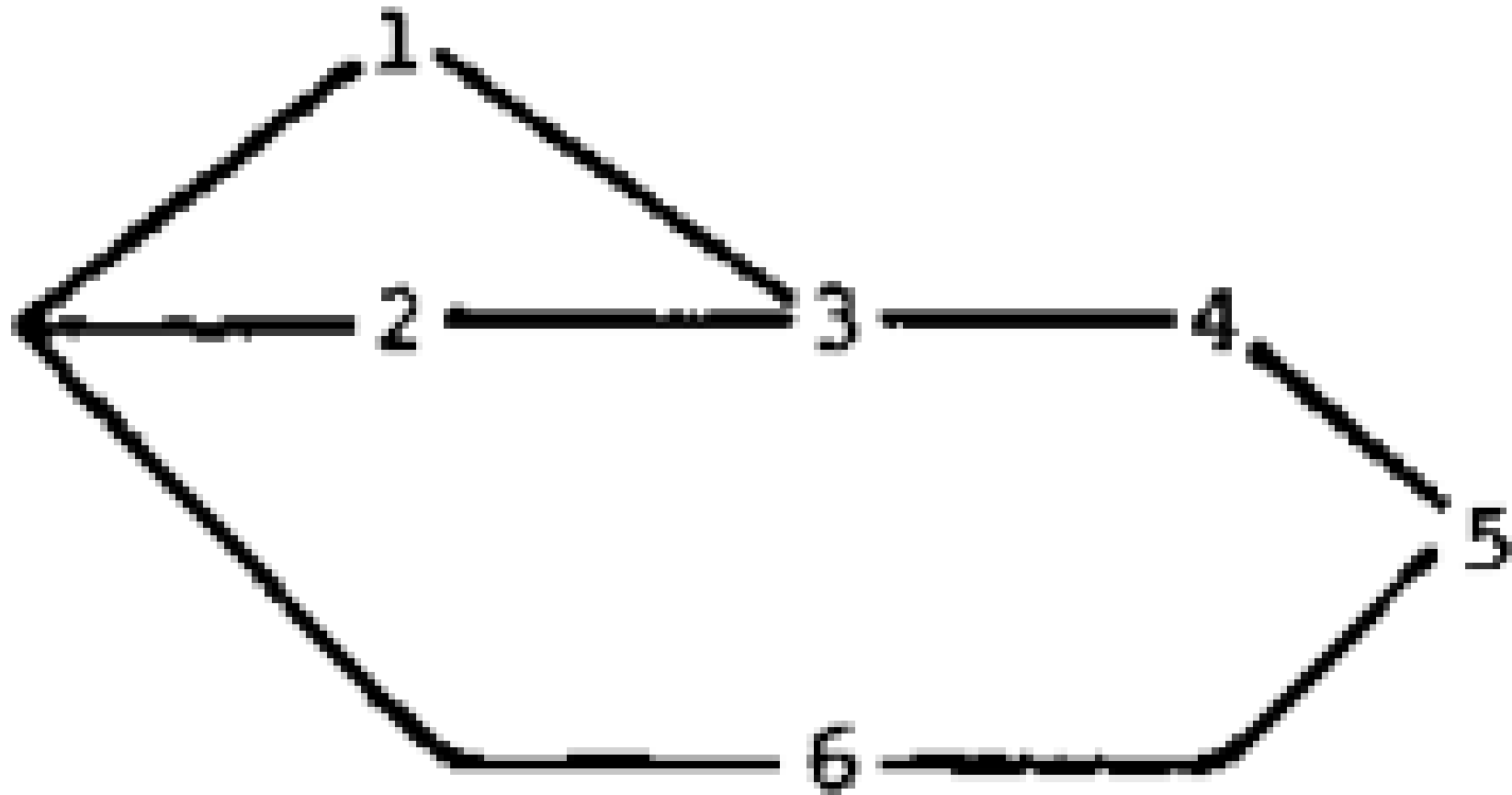
- A story can be represented as a partial-order plan, a tuple $\langle S, O, C \rangle$ where
 - S is a series of events (i.e., instantiated plan operators),
 - O is temporal ordering information represented as $(s1 < s2)$ where $s1$ precedes $s2$,
 - C is a list of causal links where a causal link is represented by $(s, t; c)$ notating a plan step s establishes c , a precondition of a step t



HTN

1. construct the root node with an abstract operator which performs the given goal task
 2. select an abstract operator of which preconditions are true
 - if no such abstract operator is found, return failure
 3. decompose the abstract operator into subtasks as encoded in the action schema
 - if all the children of the node are primitive actions, return the primitive actions as a solution
 4. go to step 2
- Fast, practical, straightforward ➔ popular in industry
 - Recipes, interaction between subtasks

HTN Representation example (Tate, 1977)



NONLIN: Generating Project Network, Austin Tate (1977)

ACTSCHEMA DECOR

PATTERN «DECORATE»

EXPANSION

1 ACTION <<FASTEN PLASTER AND PLASTER BOARD»

2 ACTION «POUR BASEMENT FLOOR>>

3 ACTION <<LAY FINISHED FLOORING»

4 ACTION <<FINISH CARPENTRY>>

5 ACTION «SAND AND VARNISH FLOORS>>

6 ACTION «PAINT»

ORDERINGS 1——>3 6——>5 SEQUENCE 2 TO 5

CONDITIONS

UNSUPERVISED «ROUGH PLUMBING INSTALLED» AT 1

UNSUPERVISED «ROUGH WIRING INSTALLED» AT 1

UNSUPERVISED «AIR CONDITIONING INSTALLED» AT 1

UNSUPERVISED «DRAINS INSTALLED» AT 2

UNSUPERVISED «PLUMBING FINISHED» AT 6

SUPERVISED «PLASTERING FINISHED>> AT 3 FROM 1

SUPERVISED <<BASEMENT FLOOR LAYED» AT 3 FROM 2

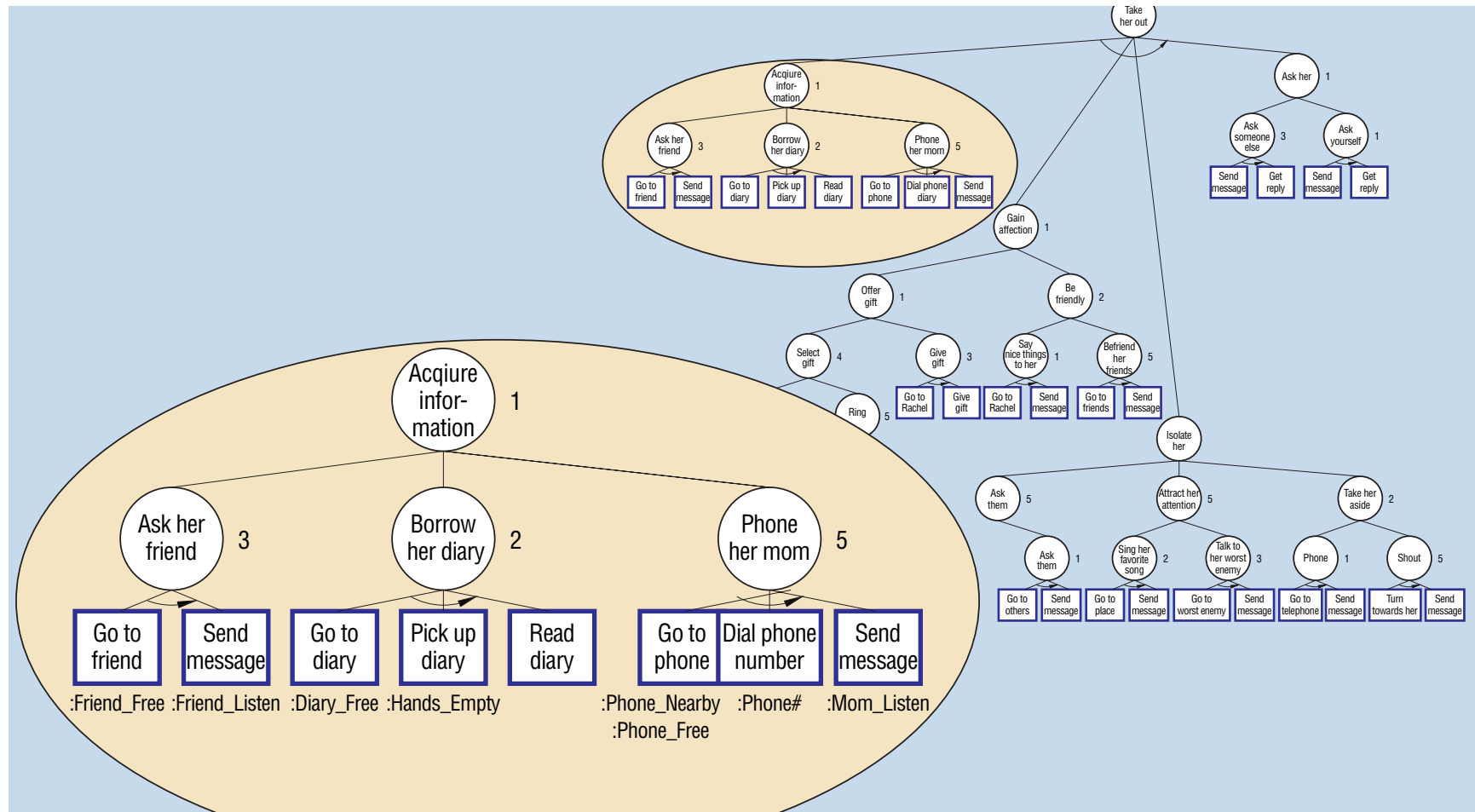
SUPERVISED «FLOORING FINISHED» AT 4 FROM 3

SUPERVISED «CARPENTRY FINISHED» AT 5 FROM 4

SUPERVISED <<PAINTED» AT 5 FROM 6.

END;

HTN example (Cavazza et al., 2002)



Dating with Rachel (Cavazza et al., 2002)

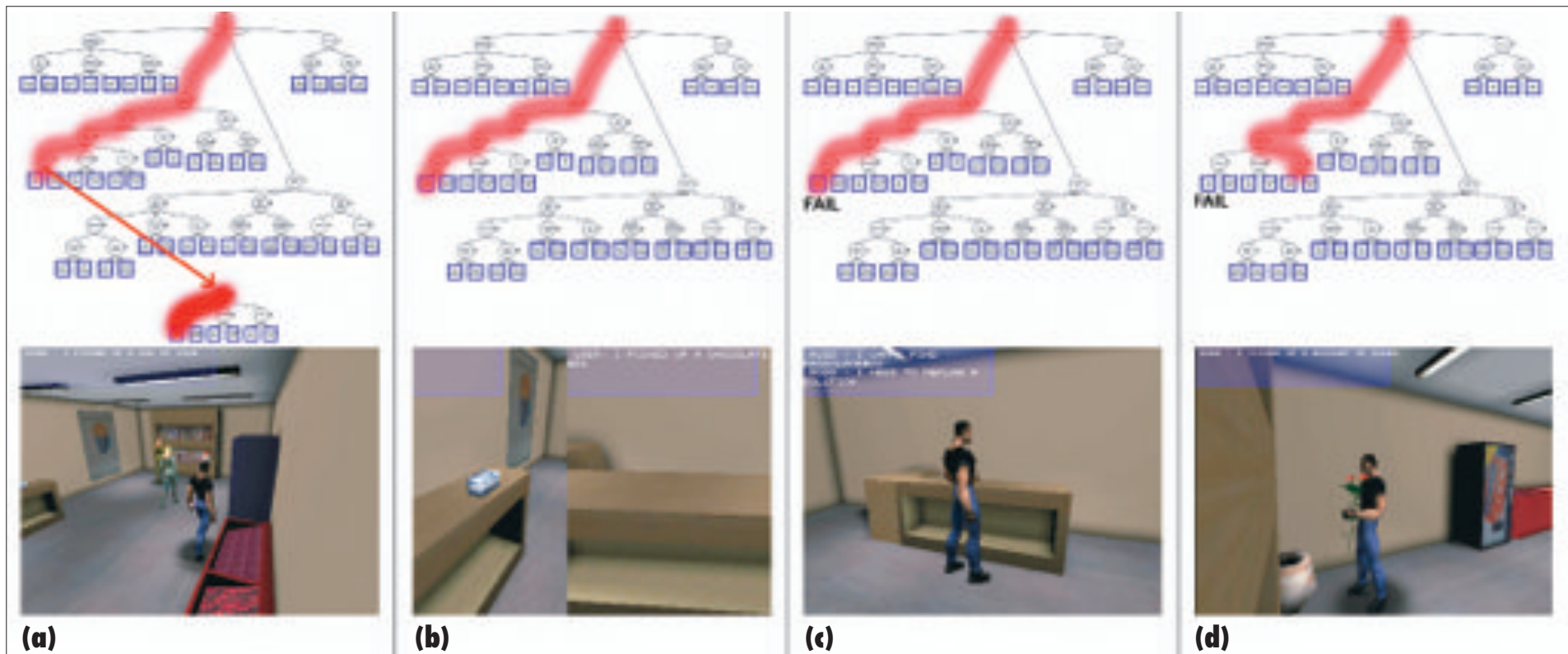


Figure 3. User intervention: (a) Ross goes to get a box of chocolates. (b) The user sees this and steals the chocolates. (c) Ross can't find them, so he (d) replans and gets roses instead.

Generating Game Worlds and Stories Together

Mark O. Riedl

Georgia Tech

A Story Plan

1. **Take** (paladin, water-bucket, palace)
2. **Kill** (paladin, baba-yaga, water-bucket, graveyard1)
3. **Drop** (baba-yaga, ruby-slippers, graveyard1)
4. **Take** (paladin, shoes, graveyard1)
5. **Gain-Trust** (paladin, king-alfred, shoes, palace)
6. **Tell-About** (king-alfred, treasure, treasure-cave, paladin)
7. **Take** (paladin, treasure, treasure-cave)
8. **Trap-Closes** (paladin, treasure-cave)
9. **Solve-Puzzle** (paladin, treasure-cave)
10. **Trap-Opens** (paladin, treasure-cave)

Story play and open-ended play

- **Story play** progresses the game world via a sequence of narrative events towards a desired conclusion
- **Open-ended play** encompasses player activities that do not progress (nor inhibit) the story plan
 - E.g., exploring the spatial environment, encountering random enemies, collecting items

Integrating play into a story

- Challenges
 - A story plan only contains the essential steps to progress toward a goal situation
 - Plan steps are abstract events e.g., solve puzzle
- Goal
 - Input: a list of events that reference **locations** of known types
 - Generate a game world that allows a linear progression through the events

Mapping from story to space

- Metaphor of islands and bridges
 - **Islands:** spatial areas where events occur
 - **Bridges:** spatial areas between islands where open-ended game play occurs
 - Note: Bridges can branch. The player does not necessarily need to visit in the course of the story.

Game world generation: 3 steps

- (1) Parse a story plan for location information referenced by events
- (2) Generate an intermediate representation of the space
- (3) Visualize the space graphically

Step 1: Create Islands

- Extract a sequence of locations from a plan
- Each location becomes an island
- Requirement
 - The story plan must be fully ordered
 - Each event must be associated with a location
 - Each referenced location must have a type
 - Can be found in the initial state

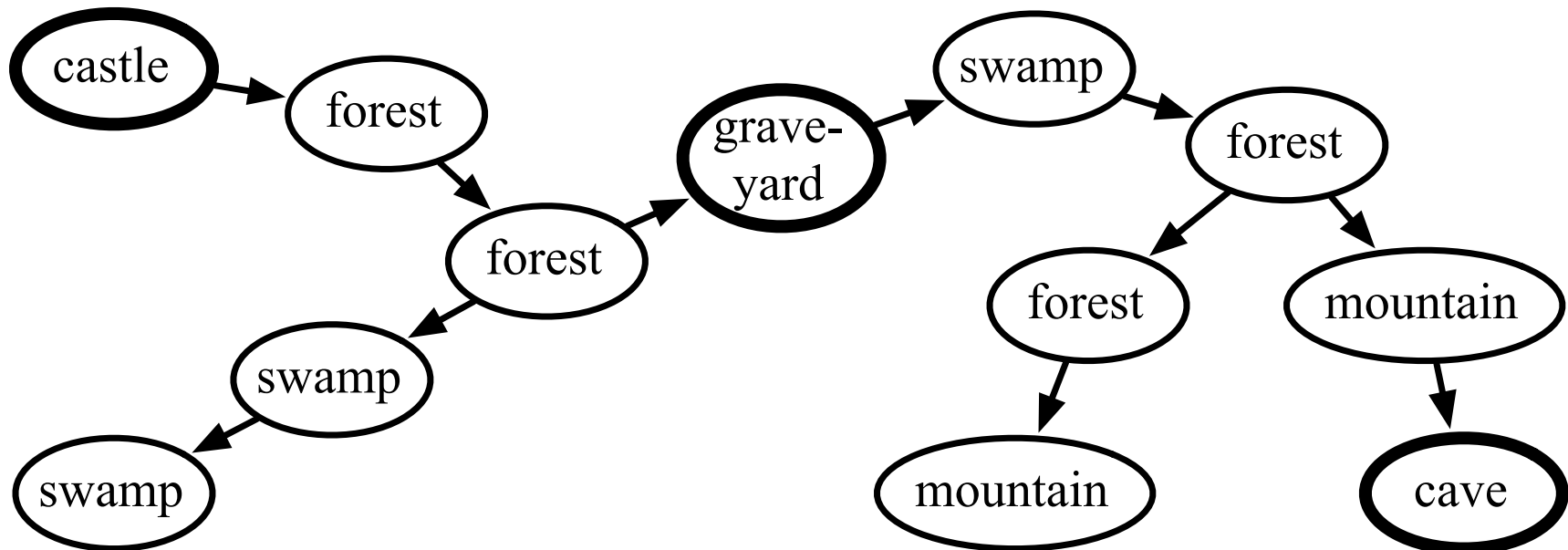
A Story Plan

1. **Take** (paladin, water-bucket, palace)
 2. **Kill** (paladin, baba-yaga, water-bucket, graveyard1)
 3. **Drop** (baba-yaga, ruby-slippers, graveyard1)
 4. **Take** (paladin, shoes, graveyard1)
 5. **Gain-Trust** (paladin, king-alfred, shoes, palace)
 6. **Tell-About** (king-alfred, treasure, treasure-cave, paladin)
 7. **Take** (paladin, treasure, treasure-cave)
 8. **Trap-Closes** (paladin, treasure-cave)
 9. **Solve-Puzzle** (paladin, treasure-cave)
 10. **Trap-Opens** (paladin, treasure-cave)
-

Hero (paladin), NPC (baba-yaga), NPC (king-alfred), Place (palace),
Place (graveyard1), Place (treasure-cave), Thing (water-bucket),
Thing (treasure), Thing (ruby-slippers), Type (baba-yaga, witch),
Type (king-alfred, king), Type (palace, castle),
Type (graveyard1, graveyard), Type (treasure-cave, cave),
Type (water-bucket, bucket), Type (ruby-slippers, shoes),
Type (treasure, gold), Evil (baba-yaga) ...

Step 2: Intermediate representation

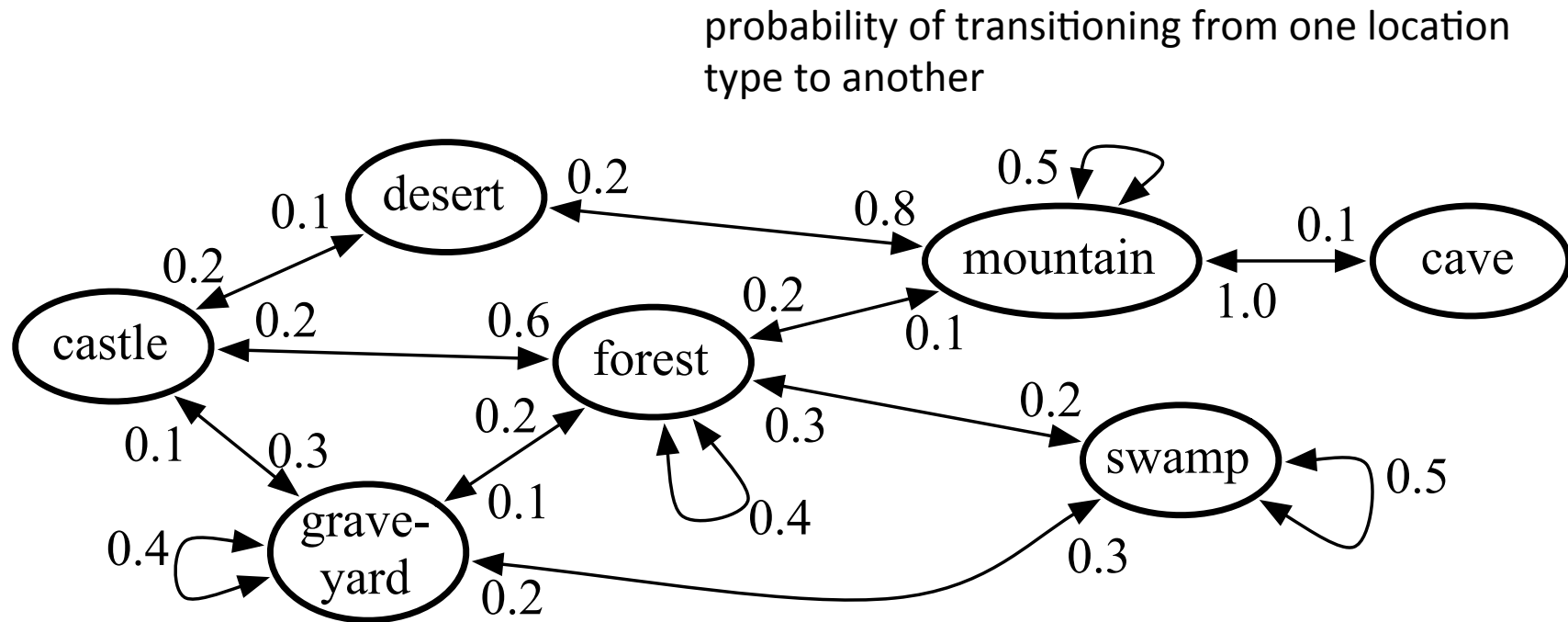
- Game world as a tree of location types
 - Indicates the size of the game world, the number of unique locations, and which locations are adjacent to each other



where story plan events are to occur

Environment Transition Graph

captures the **game designer's** beliefs about good **environment type** transitions

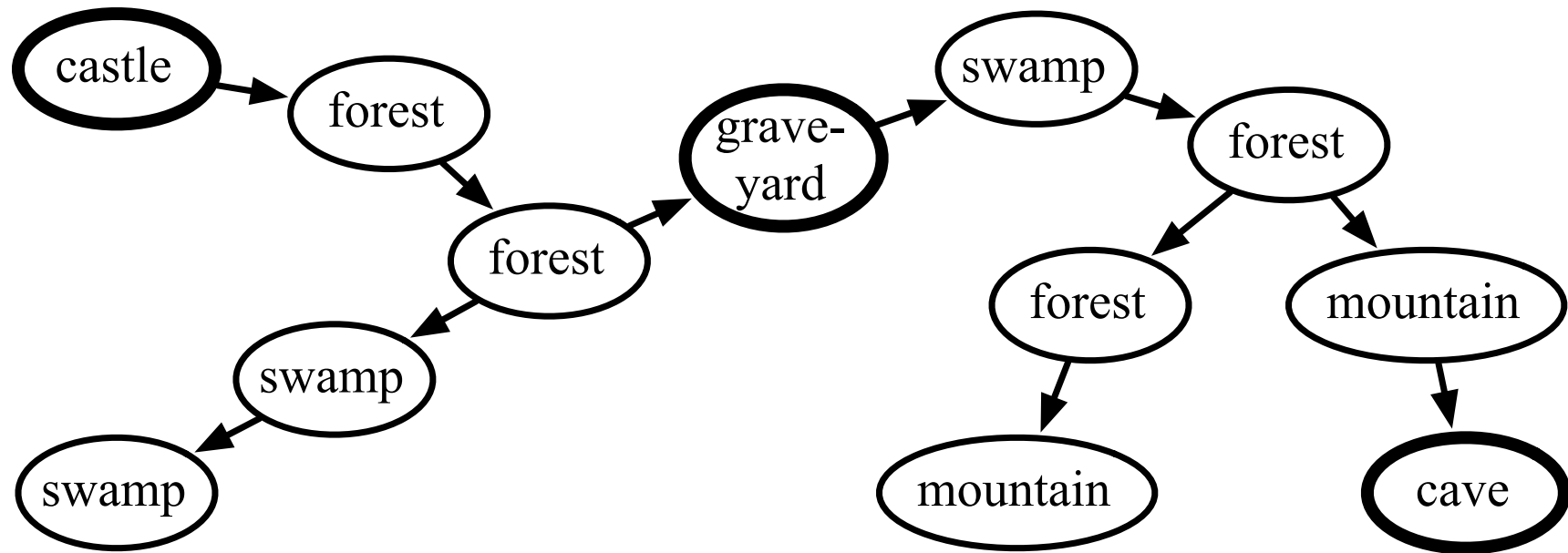


Space Tree Generation

- GA starts with initial population of random trees
- Optimized based on
 - Degree to which the number of bridges nodes in the space tree between islands have the **preferred length**
 - Whether the bridges have the preferred **branching factor**
 - Degree to which the length of side paths—branch nodes that are not directly between two islands—matches the preferred **side path length**
 - How closely environment type **transitions between adjacent nodes match the environment transition graph probabilities**

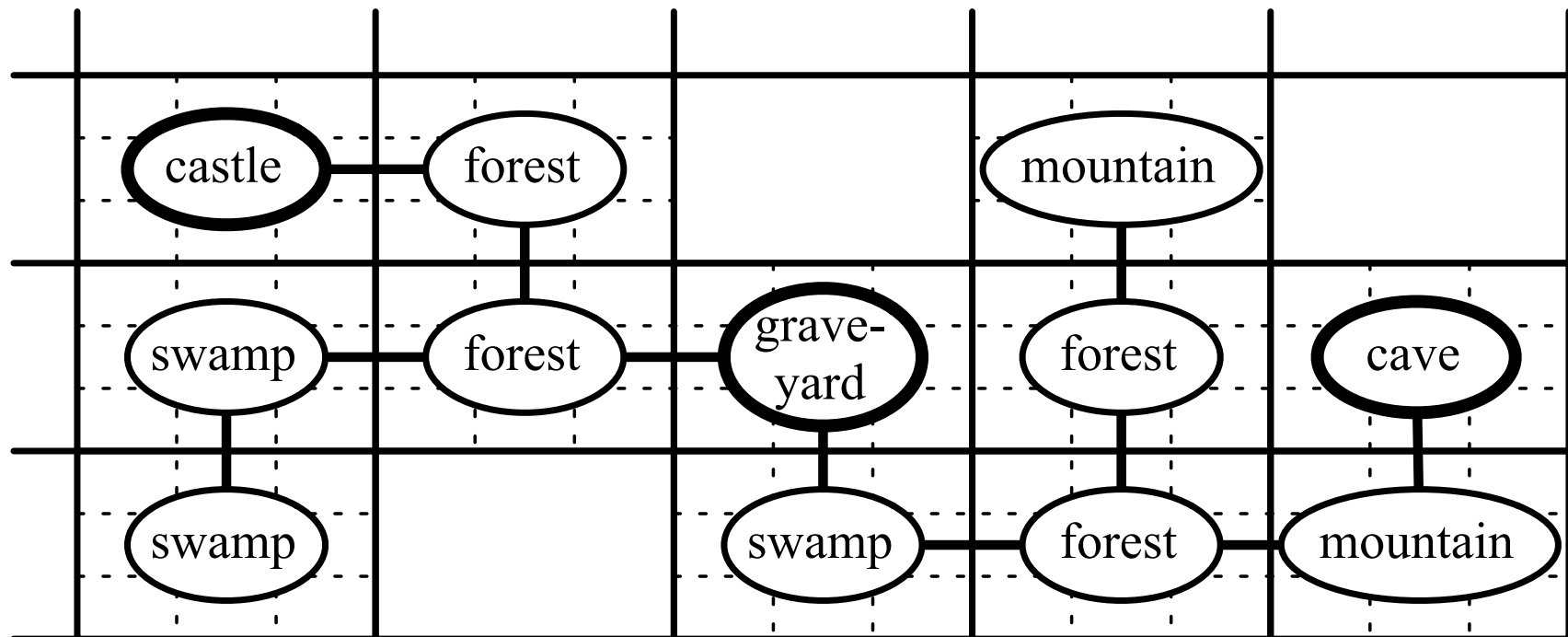
Step 3: Graphical Realization

- Visualize a game world described by a space tree



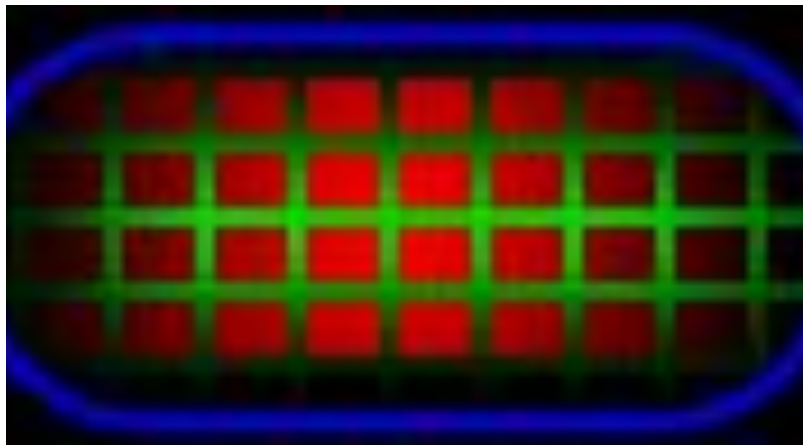
Step 3: Graphical Realization

- 2-D, top-down, tile- based
- Depth-first traversal of the space tree, placing **each child adjacent to its parent** on a grid
- If mapping is not feasible, generate a new space tree



Graphical instantiation of Location

- Location type determines graphical assets
 - E.g, town: buildings, paving stones, towers
- Placing each asset by
 - Custom distribution
 - Gaussian distribution

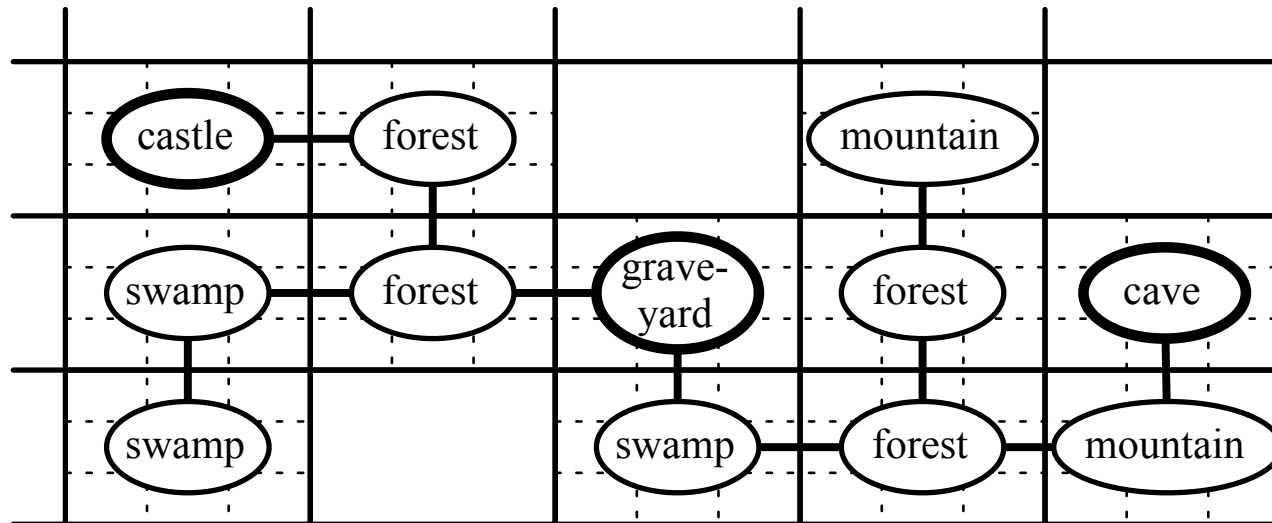


Graphical instantiation

- Location type determines graphical assets
 - E.g, forest: trees, bushes, grass
- Placing each asset by
 - Custom distribution
 - Gaussian distribution

Gaussian
distribution





Different worlds for the same plot

Greater Branching parameter



Little Branching parameter

More Issues

- (a) the world must be populated with NPCs
 - Parse the story plan and instantiate sprites (based on NPC types) in the location of the event they first participate in
- (b) the NPCs must act out the story
 - Reactive script, similar to HTN (AND-OR tree structure)
 - Narrative directive behaviors
 - Local autonomous behaviors

Video

- <http://www.youtube.com/watch?v=8xeJn7JCrgE>

Lab Session – Red Riding Hood

```
(defdomain red2
  (
    (:method (visit ?who ?whom)
      by-walk
      ((at ?who ?x) (at ?whom ?y) (walkable ?x ?y))
      (!!walk-to ?who ?x ?y)))

    (:method (eat ?who ?what)
      ((hungry ?who) (predator ?who) (alive ?who) (alive ?what) (not (equal ?who ?what)))
      (!!eat-alive ?who ?what)))

    (:operator (!walk-to ?p ?here ?there)
      ((at ?p ?here))
      ((at ?p ?here))
      ((at ?p ?there)))

    (:operator (!eat-alive ?who ?whom)
      ((at ?who ?loc1) (at ?whom ?loc1) (alive ?whom) (hungry ?who))
      ((alive ?whom) (hungry ?who))
      ((full ?who))))
```

Problem definition

```
(defproblem problem red2
  ((at Red RedHouse)
   (at Granny GrannyHouse)
   (at Wolf GrannyHouse)
   (alive Red) (alive Wolf)
   (predator Wolf) (hungry Wolf)
   (walkable RedHouse GrannyHouse))

  ( (visit Red Granny) (eat Wolf Red)))
```

Run the program

- `cd examples/red2; rm red2.java; rm red2.txt; rm problem.java; rm *.class`
- `administrators-MacBook-Air-4:jshop2 byuc$
xcrun make 12`
- `cd examples/red2; java JSHOP2.InternalDomain
red2`
- `cd examples/red2; java JSHOP2.InternalDomain -
ra problem`
- `cd examples/red2; javac problem.java`
- `cd examples/red2; java problem`

Thanks!