Dataflow Analysis of Computer Game Narratives

Clark Verbrugge
Peng Zhang

McGill University

October 30, 2008
CDP08
What?

- Computer games have narratives
  - A story through events, choices
  - RPG, Adventure, FPS
- Should be consistent
  - Appropriate responses in all eventualities
- Greater scale => greater complexity
  - Large virtual world, multiple interacting events
  - Easy to make mistakes...
Narrative Flaws

- Game narratives often have flaws
  - Non-sequiturs
  - Unwinnable situations
    - “pointlessness”
  - Unanticipated game states
- Would be nice to analyze narratives
  - Detect flaws
  - Ensure good properties
Outline

- Narrative model
- Verification
- Dataflow analysis
- Experiments
- Future Work & Conclusions
Narrative Model

- Need a well-defined target
  - Graphics, input mechanism, etc. secondary
  - Adventure (Interactive Fiction) games
    - “Pure” narrative – limited eye candy
    - Covers essential narrative features
- Formal representation
  - Industry models ad hoc, incomplete
  - Existing IF languages messy
Narrative Model

- (P)NFG
  - Minimalist language for expressing an IF narrative
    - All essential narrative features
  - Simple but expressive grammar
    - Syntactic sugar added
  - Simple operational model
  - “Compiles” down to a well-defined Petri Net model
    - Flexible low-level verification/analysis
Narrative Model

- IF games:
  - Player directs a game avatar through a virtual world
  - World includes objects, locations (rooms)
  - Avatar responds to commands
    - Player input
      - Perform actions in the game world
  - Narrative progress through object, state interactions
Narrative Model

- **Narrative components:**
  - **Objects**
    - State variables
      - Booleans, counters
    - Location (rooms)
      - Tree-containment
  - **Actions**
    - Player-invoked (input event)
    - Test & modify state
      - state/location assignment, sequence, conditionals, output
      - no loops!
Narrative Model

- e.g., an object:
  - object pumpkin {
    state { lit }
  }

- And an action:
  - (you, light, pumpkin) {
    if (!pumpkin.lit) {
      +pumpkin.lit;
      "Is it Halloween already?";
    } else {
      "It's already lit!";
    }
  }

Narrative Model

- Operational behaviour
Outline

- Narrative model
- Verification
- Dataflow analysis
- Experiments
- Future Work & Conclusions
Verification

- Ensure good behaviour
- E.g., “winnability”
  - Basic state search problem
  - Is there a winning path from the current state?
  - List all winning paths.
- Backtracking search for winning states
  - Try all sequences of actions from initial state
- Reaching a winning game state gives game solution
Verification

- Brute force:

for all possible actions
A large state-space to search:

- Deep + large branching factor => not practical
Verification

- Basic searching optimizations
  - Bounded depth
  - Cycle detection
    - Catches empty/null moves as well
  - Error state detection
    - Recognize rejected action sequences
  - Search cache
- Can we do better with high level game information?
Outline

- Narrative model
- Verification
- Dataflow analysis
- Experiments
- Future Work & Conclusions
Dataflow Analysis

- Improve verification with HL game information
  - Dataflow on game narrative
- Heuristic observations
  - Actions are not entirely independent
    - “put on mittens” -> “put on ring”
    - “light candle” when the candle isn't around
  - Cause of excessive backtracking
  - And increased branching factor
    - A large number of action-pairs do not make sense
Dataflow Analysis

- AOT analysis of actions for better search decisions
- Two basic analyses of each game action
  - Post-conditions
  - Pre-conditions
  - Flow abstract game state through, see effects
- Help figure out which actions follow which
- Reduce branching factor in search
Post-Condition Analysis

- What state does the action guarantee?
- Similar to constant propagation:
  - Boolean object states
    - object.state \{bottom, true, false, top\}
  - Enumerated locations
    - object.location \{bottom, loc1, loc2, ..., top\}
Post-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)
  +pumpkin.lit;

"You light the pumpkin";
Post-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)

+pumpkin.lit;

“You light the pumpkin”;

pumpkin:T
pumpkin.lit:T
pumpkin:desk
pumpkin.lit:T
Post-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)

+pumpkin.lit;

"You light the pumpkin";
Post-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)
  +pumpkin.lit;

"You light the pumpkin";
Post-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)
  +pumpkin.lit;
  "You light the pumpkin";

pumpkin:desk
  pumpkin.lit:true
Post-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)

+pumpkin.lit;

"You light the pumpkin";

pumpkin:desk
pumpkin.lit: true
Pre-Condition Analysis

- Pre-condition analysis
  - Backward analogue of post-condition analysis
  - What conditions does an action require?
    - Require for correctness
    - Require in order to have a useful effect? (later)
Pre-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)
    +pumpkin.lit;

"You light the pumpkin";
move pumpkin from player to desk;

if (!pumpkin.lit)

+pumpkin.lit;

"You light the pumpkin";
Pre-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)

+pumpkin.lit;

"You light the pumpkin";
Pre-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)

+pumpkin.lit;

"You light the pumpkin";
Pre-Condition Analysis

move pumpkin from player to desk;

if (!pumpkin.lit)
++pumpkin.lit;

"You light the pumpkin";
Pre-Condition Analysis

```
move pumpkin from player to desk;
if (!pumpkin.lit)
pumpkin.lit;
"You light the pumpkin";
```
Dataflow Analysis

- Pre/post-conditions constrain which actions can follow which
Dataflow Analysis

- Can we make this more precise?
- Control flow often exists to give meaningful responses, even if there are errors
  - (you, wear, ring) {
    if (you contains mittens) {
      "you can't put on a ring while wearing mittens!"
    } else {
      move ring from desk to you;
    }
  }
- No constraints in "wear mittens; wear ring"
Accurate Match

- Accurate match analysis:
- Detect necessary internal predicate for any state change
  - Outermost conditional(s):
    ```
    if (you contains mittens)
    ```
- If one branch has no state changes
  - Pre-requisites for the action are preserved from the other branch
    - Gives conditions required for meaningful action sequencing
Winning Set

- Not all actions are required to win the game
  - Actions containing `+game.win`
  - Actions containing statements which enable reaching `win`
    - ...and so on, recursively
  - “Enable” is based on state reachability

```java
(you, do, something) { 
  if (x) { 
    if (y) { 
      +game.win; 
    } 
  } 
}
```

Action (you, do, something) is part of the winning set.
Winning Set

- Not all actions are required to win the game
  - Actions containing `+game.win`
  - Actions containing statements which enable reaching win
    - ...and so on, recursively
- “Enable” is based on state reachability
  - `(you,do,something) {`  
    - `if (x) {`
      - `if (y) {`  
        - `+game.win;`  
      - `
    - `}`
  - `}`

Winning enabled by actions which can make either x true or y true.
Winning Set

- Not all actions are required to win the game
  - Actions containing \texttt{+game.win}
  - Actions containing statements which enable reaching win
    - ...and so on, recursively
- “Enable” is based on state reachability
  - (you, do, somethingElse) {
    - if (w) {
      - if (z) {
        - \texttt{x} is enabled by action (you, do, somethingElse).
        - Now we also need actions which can make either w true or z true...
      }
    }
  }
Winning Set

- Not all actions are required to win the game
  - Actions containing `+game.win`
  - Actions containing statements which enable reaching win
    - ...and so on, recursively
- Result is a closed subset of actions
  - All actions required to win the game
    - No “useless” actions
  - Reduces branching factor
Useless Objects

- Winning set establishes “useless” actions
  - Actions that do not contribute to search goal
- Useless objects exist too
  - Window dressing, promote interesting gameplay
- But still have non-trivial state interactions
  - Conservatively kept in state space
  - Searching their states adds overhead
Useless Objects

- Useless objects
  - Useless state variable
    - Read/write only by useless actions
    - Read/write otherwise only to reassign itself
    - \[ \text{if} \ (\text{object.firstUse}) \ \{ \]
      \[-\text{object.firstUse}; \]
      \[\text{“Some text you only see once.”}; \]
    \[
  \]
  - Useless location
    - Similar constraints
Outline

- Narrative model
- Verification
- Dataflow analysis
- Experiments
- Future Work & Conclusions
Variety of test narratives
  - Two chapters from Return to Zork
    - Prior work using LL analysis unable to solve
    - Non-trivial student narratives
      - Three Little Pigs
      - Complexity requirement
  - Look for first winning path solution
    - various maximum search depths
  - See effect of optimizations
Experiments

- Small narrative (dpomer)
Experiments

- Larger narrative (mccheva)
Experiments

- sdesja8 and RTZtask02
Dataflow Experiments

- mcheva
Dataflow Experiments

- sdesja8 and RTZtask02
Outline

- Narrative model
- Verification
- Dataflow analysis
- Experiments
- Future Work & Conclusions
Conclusions

- Previous work used low-level analysis
  - Hard to scale to larger narratives
  - Hours to days to analyze very small narratives
    - Even with state-of-the-art heuristic solvers
- Our approach has orders of magnitude improvement
- High-level info from dataflow analysis
  - Combined with efficient searching
Future Work

- Extend the experimentation
  - Model and analyze full commercial narratives
- Techniques to ease analysis
  - Syntax to better identify semantic structure
- Other dataflow analyses
  - context-sensitive
- Metrics
  - Quality, complexity, understanding
Thanks

Questions?