Thesis Proposal: Logical Interactive Programming for Narrative Worlds

Chris Martens
December 6, 2013
My interest: supporting the design & analysis of game mechanics at a linguistic level.
# Talk Outline

<table>
<thead>
<tr>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrative Worlds</td>
<td>define my target domain</td>
</tr>
<tr>
<td>Example: Blocks World</td>
<td>describe how CLF specification works</td>
</tr>
<tr>
<td>Supporting Interactivity and</td>
<td>describe my language extensions (phases,</td>
</tr>
<tr>
<td>Analysis</td>
<td>generative properties)</td>
</tr>
<tr>
<td>Narrative Worlds, revisited</td>
<td>give more examples to show breadth of scope</td>
</tr>
<tr>
<td>Proposed Work &amp; Evaluation</td>
<td>establish a plan to justify my thesis statement</td>
</tr>
<tr>
<td>Strategy</td>
<td></td>
</tr>
</tbody>
</table>

*thesis statement*
## Talk Outline

<table>
<thead>
<tr>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrative Worlds</td>
<td>define my target domain</td>
</tr>
<tr>
<td>Example: Blocks World</td>
<td>describe how CLF specification works</td>
</tr>
<tr>
<td>Supporting Interactivity and Analysis</td>
<td>describe my language extensions (phases, generative properties)</td>
</tr>
<tr>
<td>Narrative Worlds, revisited</td>
<td>give more examples to show breadth of scope</td>
</tr>
<tr>
<td>Proposed Work &amp; Evaluation Strategy</td>
<td>establish a plan to justify my thesis statement</td>
</tr>
</tbody>
</table>
Narrative Worlds
You are in the living room. There is a doorway
strange gothic lettering to the west, which appears
trophy case, and a large oriental rug in the cen-
Above the trophy case hangs an elvish sword of g-
A battery-powered brass lantern is on the trophy

> get lamp
  Taken.

> move rug
  With a great effort, the rug is moved to one side
  dusty cover of a closed trap door.

> open trap door
  The door reluctantly opens to reveal a rickety
  darkness.
emphasis on narrative
“vs.”
emphasis on (open) worlds

“ludonarrative” = ludo (game/play) + narrative (story)
Puzzle Structure of Legend of Zelda: Ocarina of Time

http://garethrees.org/2004/12/01/ocarina-of-time/
Narrative Structures

(Madame Bovary)

(Martens, Bosser, Ferreira, Cavazza ’13)
Leon falls in love with Emma
Emma marries Charles
Emma reads romantic novels
Emma is invited to an aristocrats ball
Rodolphe decides to seduce Emma
Emma and Leon meet again
The Bovary go to the ball
Bored, Emma contracts debts
Emma realises Leon's love
Emma pushes Leon away
Emma accepts Leon's advances
Emma reimburses some of the debt
Emma swallows the arsenic and dies
Emma offers a gift to Rodolphe
Emma's love for Leon falters
Mr Homais informs Emma about Mr Bovary's death
The tribunal pronounces the Bovary's ruin
Emma Purchases Gift:
Emma and Lheureux are present,
Rodolphe & Emma are Together
→ debt, gift
Shared structure:
plots & puzzles create resource dependencies
<table>
<thead>
<tr>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrative Worlds</td>
<td>define my target domain</td>
</tr>
<tr>
<td>Example: Blocks World</td>
<td>describe how CLF specification works</td>
</tr>
<tr>
<td>Supporting Interactivity and Analysis</td>
<td>describe my language extensions (phases, generative properties)</td>
</tr>
<tr>
<td>Narrative Worlds, revisited</td>
<td>give more examples to show breadth of scope</td>
</tr>
<tr>
<td>Proposed Work &amp; Evaluation Strategy</td>
<td>establish a plan to justify my thesis statement</td>
</tr>
</tbody>
</table>
Simple Example: Blocks World
Representation of Individual States

\{\text{arm\_free}, \text{on\_table\ b}, \text{on\_table\ c}, \text{clear\ c}, \text{on\ a\ b}, \text{clear\ a}\ \}
Representation of Action Rules

\[
pickup\_from\_table : \\
on\_table X * clear X * arm\_free \\
-o \{ arm\_holding X \}.
\]
Blocks world cont’d

pickup_from_block :
  on X Y * clear X * arm_free
  -o {clear Y * arm_holding X}.

put_on_table :
  arm_holding X -o
  {on_table X * clear X * arm_free}.

put_on_block :
  arm_holding X * clear Y
  -o {on X Y * clear X * arm_free}. 
Local state change
Celf Specification Framework

based on CLF (Watkins, Cervesato, Pfenning, Walker ’02)

implements linear logic as a logic programming language
(execution as proof search)

still many open questions about operational semantics
Committed Choice
Committed Choice
Celf

#query 10 (init -o {end_condition})
Celf

#query 10 (init -o \{end_condition\})
Celf

#query 10 (init -o {end_condition})
#query 10 (init -o {end_condition})

state at quiescence
Celf

on a b * on b c * on_table c * arm_free
-o \{end_condition\}
Celf

#query 10 (init -o {end_condition})

... let \{X_{13}\} = \textbf{pickup\_from\_table} [X_{10}, [X_{11}, X_{12}]] in let \{[X_{14}, [X_{15}, X_{16}]]\} = \textbf{put\_on\_table} X_{13} in let \{X_{17}\} = \textbf{pickup\_from\_table} [X_{3}, [X_{6}, X_{16}]] in let \{[X_{18}, [X_{19}, X_{20}]]\} = \textbf{put\_on\_block} [X_{17}, X_{8}] in ...
let \{X_{13}\} = \text{pickup\_from\_table} \ [X_{10}, [X_{11}, X_{12}]] \text{ in}
let \{X_{13}\} = \texttt{pickup\_from\_table} [X_{10}, [X_{11}, X_{12}]] \text{ in}
Proofs-as-traces: structural artifacts that we can analyze, e.g. for causal dependency.
Celf

Proofs-as-traces: structural artifacts that we can analyze, e.g. for causal dependency.

c.f: PlotEx
http://eblong.com/zarf/plotex/

GraphPlan
http://www.cs.cmu.edu/~avrim/graphplan.html
Proto-Thesis Statement

Linear logic programming can form the basis of a framework for specifying simulation mechanics.
Proto-Thesis Statement

Linear logic programming can form the basis of a framework for specifying simulation mechanics.
Proto-Thesis Statement

[
Linear logic programming]+ can form the basis of a framework for [specifying]+ [simulation]+ mechanics.
## Talk Outline

<table>
<thead>
<tr>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrative Worlds</td>
<td>define my target domain</td>
</tr>
<tr>
<td>Example: Blocks World</td>
<td>describe how CLF specification works</td>
</tr>
<tr>
<td>Supporting Interactivity and</td>
<td>describe my language extensions (phases, generative properties)</td>
</tr>
<tr>
<td>Analysis</td>
<td></td>
</tr>
<tr>
<td>Narrative Worlds, revisited</td>
<td>give more examples to show breadth of scope</td>
</tr>
<tr>
<td>Proposed Work &amp; Evaluation</td>
<td>establish a plan to justify my thesis statement</td>
</tr>
<tr>
<td>Strategy</td>
<td></td>
</tr>
</tbody>
</table>
Adding interactivity to blocks world

action : type.

**pickup** : block → action.

**putdown_on** : block → action.

**putdown_table** : action.

**stop** : action.
pickup_from_block:

current (pickup X)
* on X Y * clear X * arm_free
-o {clear Y * arm_holding X}. 

Interactivity cont’d

Where does “**current**” come from?
The engine & player should “take turns.”

```plaintext
  current (pickup X) * ... -o {... * player_turn}
  current (putdown_table X) * ... -o {... * player_turn}
  ...

  player_turn -o {ForAny a:action. current a}
```
Phases

Block-delimited subsignatures

phase **world** = {
    rule1 : current Action * … -o {...}.
    rule2 : current Action * … -o {...}.
}

phase **player** = {
    rule : player_turn -o {...}
}

39
Phases

Connected by specification of quiescence behavior

\[
\text{phase world} = \{\ldots\}
\]

\[
\text{phase player} = \{\ldots\}
\]

\[
\text{quiesced world} -o
\{\text{player\_turn} \ast \text{phase player}\}.
\]

\[
\text{quiesced player} -o \{\text{phase world}\}.
\]
Phases

Connected by specification of quiescence behavior

phase world = {...}

phase player = {...}

quiesced world -o {player_turn * phase player}.

quiesced player -o {phase world}.

Related: “sensing” and “action” atoms in Meld (Claytronics)
Phases

...are block-delimited subsignatures connected by specifications of quiescence behavior.

\[
\textit{quiesced } P \ast \textit{ State } \dasho \{\textit{phase } P' \ast \textit{ State' }\}.
\]

arbitrarily many phases
looping + branching
Compiling Phases

We can interpret phase-structured programs as programs in Celf.
We can interpret phase-structured programs as programs with higher-order, mixed-chaining rules in Celf.
Compiling Phases

We can interpret phase-structured programs as programs with higher-order, mixed-chaining rules in Celf.

(see proposal document for details)
Compiling Phases

We can interpret phase-structured programs as programs with higher-order, mixed-chaining rules in Celf.

**Ongoing work:** Check that the source-level semantics corresponds to compiled semantics.
Phase-structured linear logic programming can form the basis of a framework for specifying, testing, and inventing ludonarrative mechanics.
Checked Metatheory
Dungeon A: may become unsolvable

Dungeon B: always solvable
If the arm is holding a block, it is not empty.

If block A is on the table it is not on any other block.

If block A is on block B, block B is not clear.
Generative Properties

a way of stating and checking programmer intent
Generative Properties

based on Generative Invariants (Simmons ’12)
Generative Invariants

To prove an invariant of a signature $\Sigma$:

Describe a signature $\Sigma_{\text{gen}}$ with a distinguished start state (usually an atom “gen”)

and prove that

- initial states of $\Sigma$ are in (could be generated by) $\Sigma_{\text{gen}}$
- every rule in $\Sigma$ preserves membership in $\Sigma_{\text{gen}}$
Generative Invariants

\[ \text{gen} \quad -o \quad \{ \text{genArm} \land \neg \text{genBlocks} \} \].

\[ \text{genArm} \quad -o \quad \{ \text{arm}_{-} \text{free} \} \].

\[ \text{genArm} \quad -o \quad \{ \text{arm}_{-} \text{holding} \, X \} \].

\[ \text{genBlocks} \quad -o \quad \{ \text{on}_{-} \text{table} \, X \land \neg \text{genTop} \, X \} \].

\[ \text{genBlocks} \quad \land \quad \text{genTop} \, Y \quad -o \quad \{ \text{on} \, XY \land \neg \text{genTop} \, X \} \].

\[ \text{genTop} \, X \quad -o \quad \{ \text{clear} \, X \} \].
Quiescence & Activity

Quiescence: no rules can fire

Activity: at least one rule can fire
Activity Generator for Blocks World

\texttt{act} -o \{arm\_holding X * !actBlocks\}.
\texttt{act} -o \{arm\_free * clear Y * !actBlocks\}.
\texttt{actBlocks} -o \{on\_table X\}.
\texttt{actBlocks} -o \{on X Y\}.
\texttt{actBlocks} -o \{clear X\}.
Ongoing Work:

Work out how to mechanically check these properties.

Show applicability to invariant properties of game worlds.
<table>
<thead>
<tr>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrative Worlds</td>
<td>define my target domain</td>
</tr>
<tr>
<td>Example: Blocks World</td>
<td>describe how CLF specification works</td>
</tr>
<tr>
<td>Supporting Interactivity and Analysis</td>
<td>describe my language extensions (phases, generative properties)</td>
</tr>
<tr>
<td>Narrative Worlds, revisited</td>
<td>give more examples to show breadth of scope</td>
</tr>
<tr>
<td>Proposed Work &amp; Evaluation Strategy</td>
<td>establish a plan to justify my thesis statement</td>
</tr>
</tbody>
</table>
Narrative Worlds, Revisited
As a player, you get to **select a character, guide their choices, watch other characters react** to what you've chosen, and **accomplish (or fail at) your chosen goals**.
Generalized Narrative Structures

do/murder :
    anger C C' * anger C C' * anger C C' * anger C C'
    * at C L * at C' L * has C weapon -o
{at C L * has C weapon * !dead C' * !murdered C C'}. 

do/thinkVengefully :
    loves C C' * !murdered K C'
    -o {loves C C' * anger C K * anger C K}. 

Generalized Narrative Structures

do/murder : **do C (murder C')** *

  anger C C' * anger C C' * anger C C' * anger C C' *
  at C L * at C' L * has C weapon -o

{at C L * has C weapon * !dead C' * !murdered C C'}. 


do/thinkVengefully : **do C (thinkVenge K)** *

  loves C C' * !murdered K C' 
  -o {loves C C' * anger C K * anger C K}.
Generalized Narrative Structures

Ongoing work: figure out how to specify failure conditions when preconditions for an action are not met.
Generalized Narrative Structures

Inform 7 action processing:

Implementable as phases!
Puzzle games

Scope: PuzzleScript

http://www.puzzlescript.net
Sokoban
In PuzzleScript:

[ > Player | Crate ] -> [ > Player | > Crate ]
Sokoban Rules

push:

loc pusher \( \mathbf{L} \) * in_dir \( \mathbf{L} \) Dir \( \mathbf{L}' \) * loc block \( \mathbf{L}' \)

* in_dir \( \mathbf{L}' \) Dir \( \mathbf{L}'' \) * empty \( \mathbf{L}'' \)

-o \{empty \( \mathbf{L} \) * loc pusher \( \mathbf{L}' \) * loc block \( \mathbf{L}'' \)\}.

move:

loc pusher \( \mathbf{L} \) * in_dir \( \mathbf{L} \) Dir \( \mathbf{L}' \) * empty \( \mathbf{L}' \)

-o \{empty \( \mathbf{L} \) * loc pusher \( \mathbf{L}' \)\}. 
push :

**action (arrow Dir)***
loc pusher L * in_dir L Dir L' * loc block L' *
    * in_dir L' Dir L'' * empty L''
-o {empty L * loc pusher L' * loc block L'"}.

move :

**action (arrow Dir)***
loc pusher L * in_dir L Dir L' * empty L'  
-o {empty L * loc pusher L'}. 

Many more examples (some in progress):

https://github.com/chrisamaphone/interactive-lp/tree/master/examples
## Talk Outline

<table>
<thead>
<tr>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrative Worlds</td>
<td>define my target domain</td>
</tr>
<tr>
<td>Example: Blocks World</td>
<td>step through all the pieces of my proposal</td>
</tr>
<tr>
<td>Narrative Worlds, revisited</td>
<td>show the intended scope of those ideas</td>
</tr>
<tr>
<td>Proposed Work &amp; Evaluation Strategy</td>
<td>establish a plan to justify my thesis statement</td>
</tr>
</tbody>
</table>
## Proposed Work

<table>
<thead>
<tr>
<th>Shortcoming of Existing Framework</th>
<th>Proposed Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sometimes we want to impose partial orderings among rules.</td>
<td>Language proposal with phases.</td>
</tr>
<tr>
<td>Programming with state is hard to reason about!</td>
<td><strong>Machine-checked</strong> invariants and other characterizations of states; analysis tools such as causality and dependency graphs.</td>
</tr>
<tr>
<td>Non-interactive, low-feedback programming workflow.</td>
<td><strong>Visual state editor and trace rendering.</strong></td>
</tr>
<tr>
<td>Lack of access to common game programming libraries for e.g. graphical rendering, text parsing, etc.</td>
<td><strong>Implement compatibility between the language and existing game frameworks</strong> (e.g. Twine)</td>
</tr>
</tbody>
</table>
Evaluation
Phase-structured linear logic programming can form the basis of a framework for specifying, testing, and inventing ludonarrative mechanics.

How will I determine success?
Phase-structured linear logic programming can form the basis of a framework for specifying, testing, and inventing ludonarrative mechanics.

Develop several examples in the framework.
Phase-structured linear logic programming can form the basis of a framework for specifying, testing, and inventing ludonarrative mechanics.

Prove correspondence and build prototype.
Design UI & tooling, including visual rendering.

Phase-structured linear logic programming can form the basis of a framework for specifying, testing, and inventing ludonarrative mechanics.
Phase-structured linear logic programming can form the basis of a framework for specifying, testing, and inventing ludonarrative mechanics.

Generative properties and graphical analysis tools
Key Contributions

To game design:
- simple, uniform logical formalism
- executable specs (“sketching” systems)
- reasoning and intent-checking tools as an integrated part of design process
Key Contributions

To logical frameworks:
- exploration of a new domain as evidence for its generality
- new or alternative answers to open questions about semantics
- establishment of metatheoretic tools
Timeline

• **Spring 2014:** Finish working out theoretical concerns (language semantics, proofs, and sketch of generative property checking)

• **Summer-Fall 2014:** Implementation of prototype and development of examples

• **Spring 2015:** Write dissertation

• **Summer 2015:** Defend dissertation

Thank You!
extra slides
Phase links for Inform7 action processing graph:

qui read -o {phase parse}.
qui parse * outcome none
    -o {message defaultParseError * phase report}.
qui parse * outcome failure -o {phase report}.
qui parse * outcome success -o {phase check1}.
qui check1 -o {phase check2}.
qui check2 * outcome success
    -o {phase carryout}.
qui check2 * outcome failure -o {phase report}.
qui check2 * outcome none
    -o {message default * phase report}.
qui carryout -o {phase report}. 
Phases Inform7 action processing:

phase check1 = {
- : init * outcome X -o {outcome none}.

- : $action (take Obj) * inventory Obj
  -o {outcome failure
      * message "You already have it."}.

- : $action look -o {outcome success}.
}

phase check2 = {
- : $action (take Obj) * outcome none
  * visible Obj -o {outcome success}.
}

phase carryout = {
- : action (take Obj) * in Obj C
  -o {inventory Obj * message "taken"}.
- : action look * $in player R * $description R D
  -o {message D}.
}