Of Ludics & Ludology: Systems of Play as Linear Logic Programs

Chris Martens
Carnegie Mellon University

Presented at NJPLS, May 16, 2014

SPOILER WARNING

takeaway:

linear logic* as a tool for game design
enables **rapid experimentation** and **structural analysis** of a **wide range** of core mechanics.

*… with various extra-logical additives.

THREE ACTS

I. Game Design Vocabulary
Example
Linear logic programming

II. Payoff: Proofs as interaction traces
Generation & Analysis

III. Promise: Interactivity
Invariant Checking

ACT I
Setup: destroy assumptions


DOS - Microsoft Virtual PC 2004
Action Edit CD Floppy Help
Cellar
Scor
You are in the living room. There is a doorway to the
strange gothic lettering to the west, which appears to
trophy case, and a large oriental rug in the center of
Above the trophy case hangs an elvish sword of great a
A battery-powered brass lantern is on the trophy case.
>get lamp
Taken.
>move rug
With a great effort, the rug is mov
dusty cover of a closed trap door.
>open trap door
The door reluctantly opens to revea
darkness.
>d
You have moved into a dark place.
The trap door crashes shut, and you
It is pitch black. You are likely t
Castle
ZZT
MARIO 004250
×05
WORLD 1-1
TIME 283
100
menu
next
challenges
MINECRAFT
SIDE A

what are (digital) games made of?

graphics + sound + mechanics
movement, enemies, levels, bosses, items, characters, endings, ...

most frameworks' starting point:

graphics + sound + mechanics
movement, enemies, levels, bosses, items, characters, endings, ...

languages for programming games:

Unity
Twine
Inform 7
GameMaker, Scratch
StageCast, PuzzleScript
C++?
FRP?

LANGUAGE AFFECTS DESIGN DECISIONS

a better starting point?

graphics + sound + **mechanics**
movement, enemies, levels, bosses, **items**, characters, endings, ...

my starting point:

rules
resources

RULES

Rules of play

~

Rules of logic

linear logic

RULES

Rules of play

~

state change through manipulation of **resources**

linear logic

RULES

Rules of play

~

state change through manipulation of **resources**

~

linear logic

Linear Logic

core judgment:

$$\Gamma; \Delta \vdash A$$

A *persistent* $\in \Gamma$

A *linear* $\in \Delta$

Linear Logic

core judgment:
$\Gamma; \Delta \vdash A$

$A \in \Gamma$: subject to wk, contr, exchg
$A \in \Delta$: "use exactly once"

Linear Logic

A -o B
A * B
!A
1

Linear Logic Programming

fill a signature with predicate declarations
pred <arg_types>

and constant declarations
c : A

Linear Logic Programming

fill a signature with predicate declarations
pred <arg_types>

and constant declarations
r : B -o C

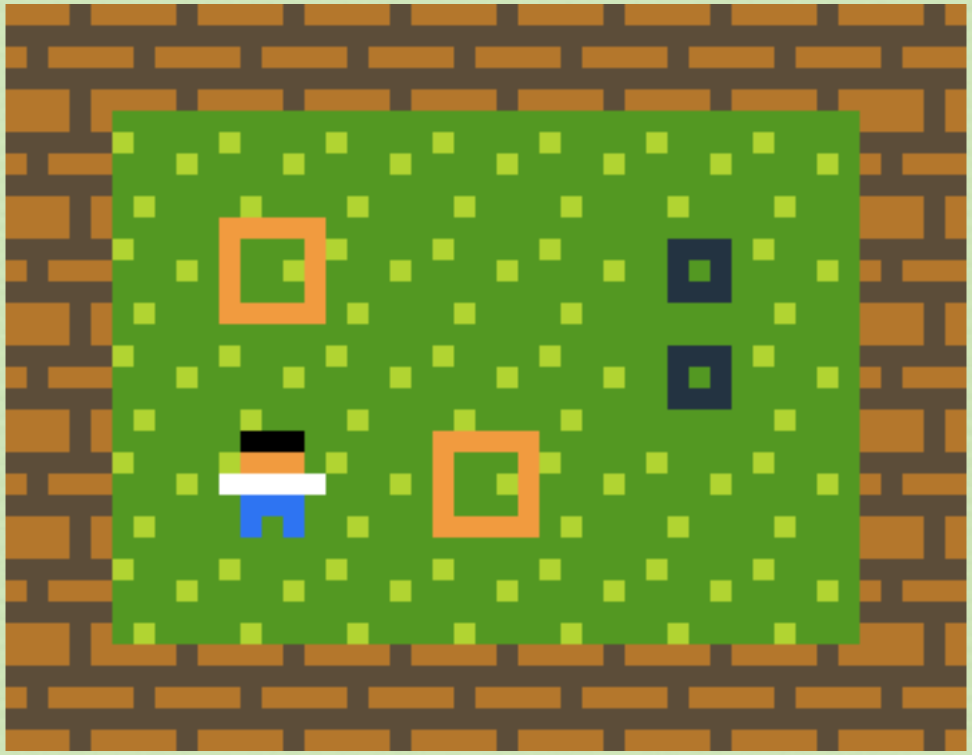
EXAMPLE
2d, turn-based puzzle games

http://www.puzzlescript.net

Hardcoded assumptions:

turn-based
2d grid of adjacent locations
player controls one entity
controls are up, down, left, right, x
single player

e.g. Sokoban



in PuzzleScript:

```
[ > Player | Crate ] -> [ > Player | > Crate ]
```

My assumptions

turn-based

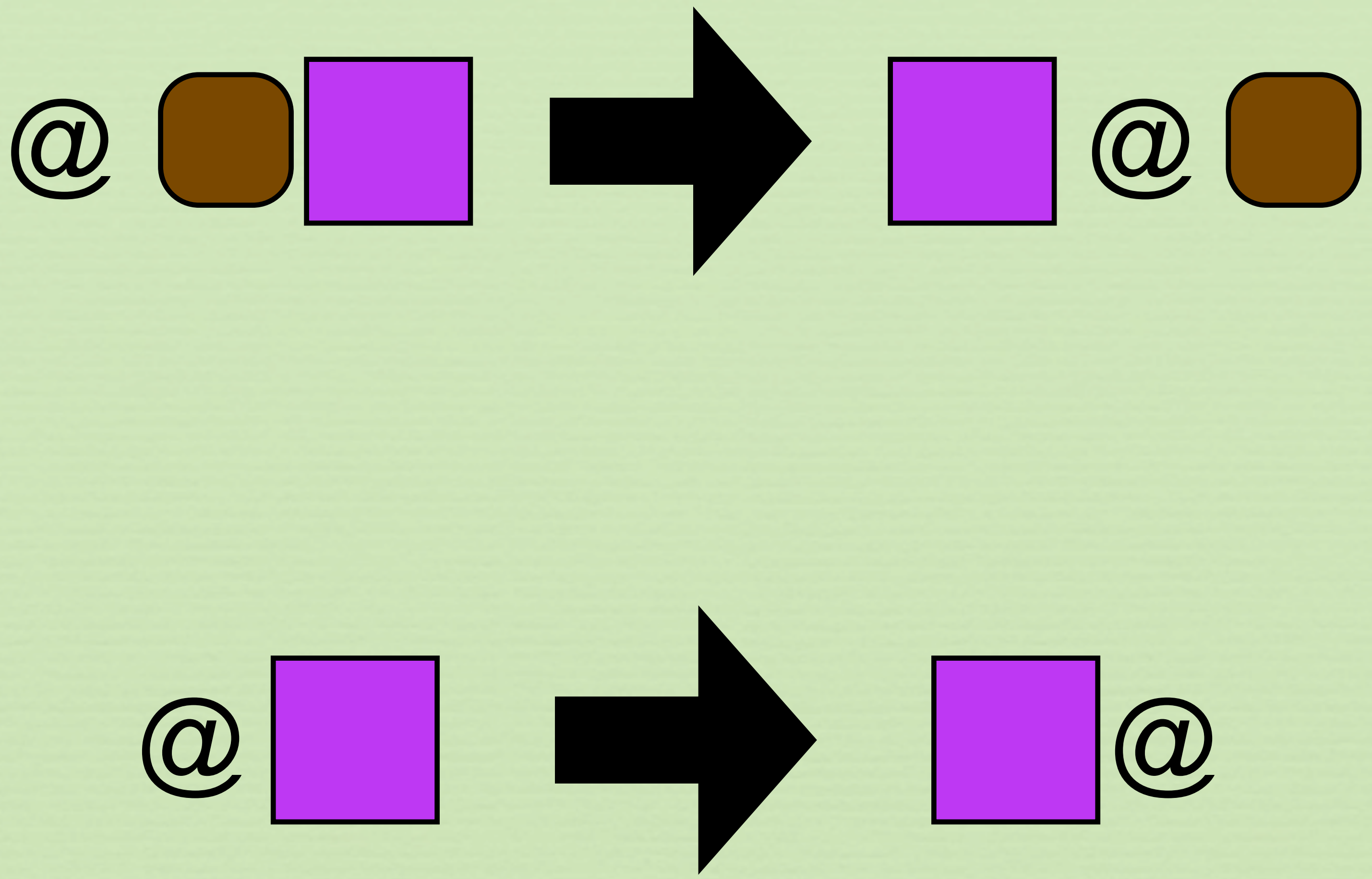
2d grid of adjacent locations

player controls one entity

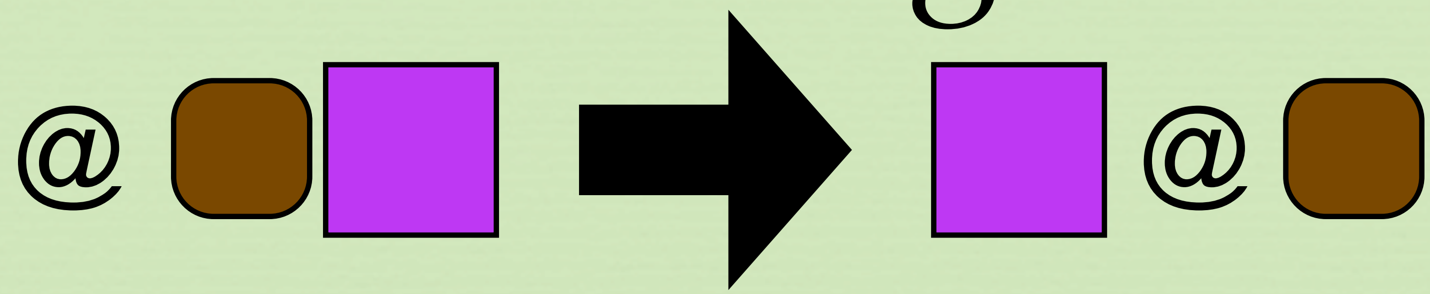
controls are up, down, left, right, x

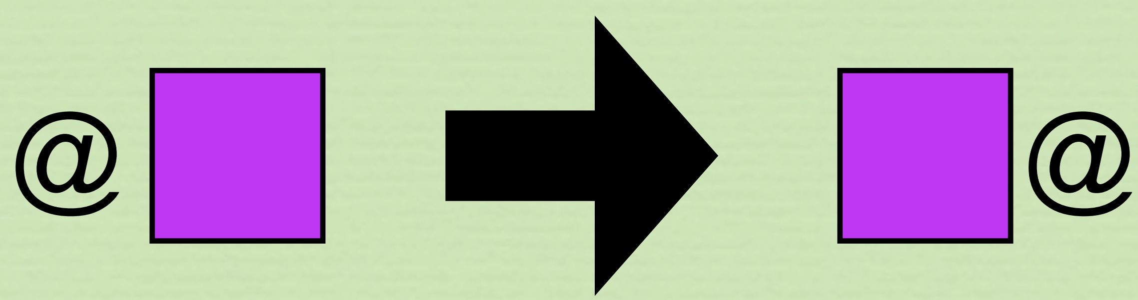
single player

sokoban in linear logic

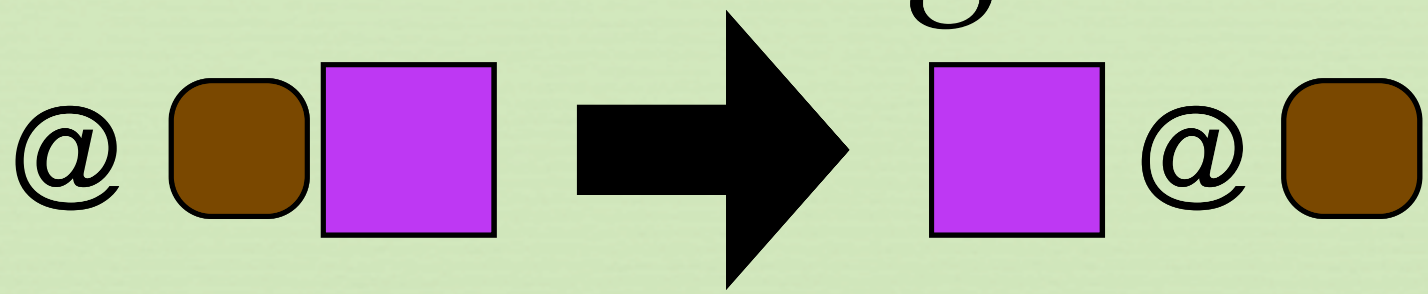


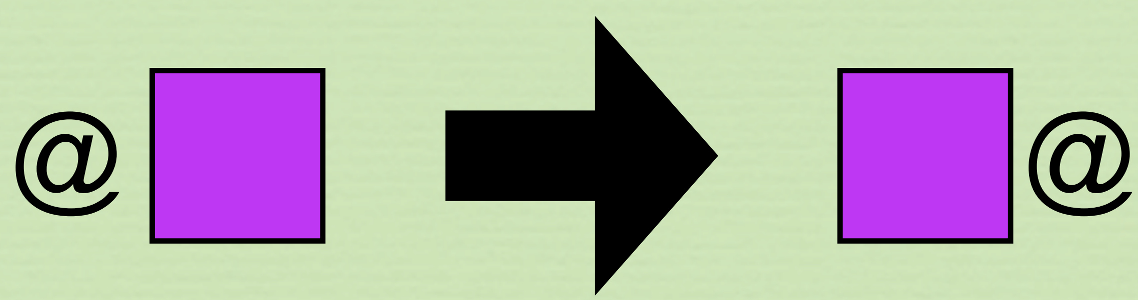
sokoban in linear logic





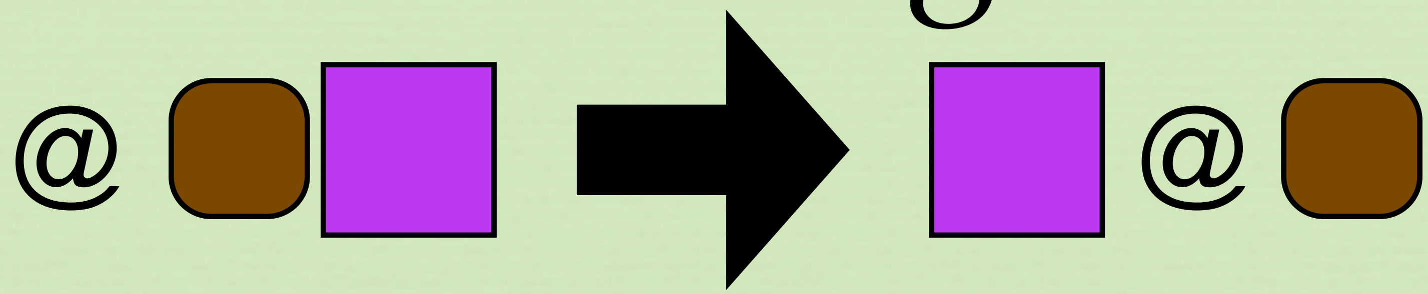
sokoban in linear logic



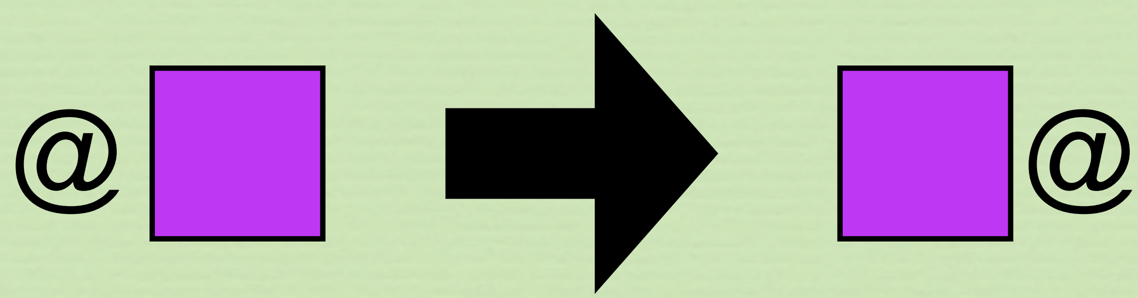



```
move :
loc pusher L * in_dir L Dir L' * empty L'
  -o {empty L * loc pusher L'}.
```

sokoban in linear logic




```
push :
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 :
loc pusher L * in_dir L Dir L' * empty L'
  -o {empty L * loc pusher L'}.
```

EXECUTABLE SPECS AS LINEAR LOGIC PROGRAMS

1. specify the predicates needed to track state, e.g.

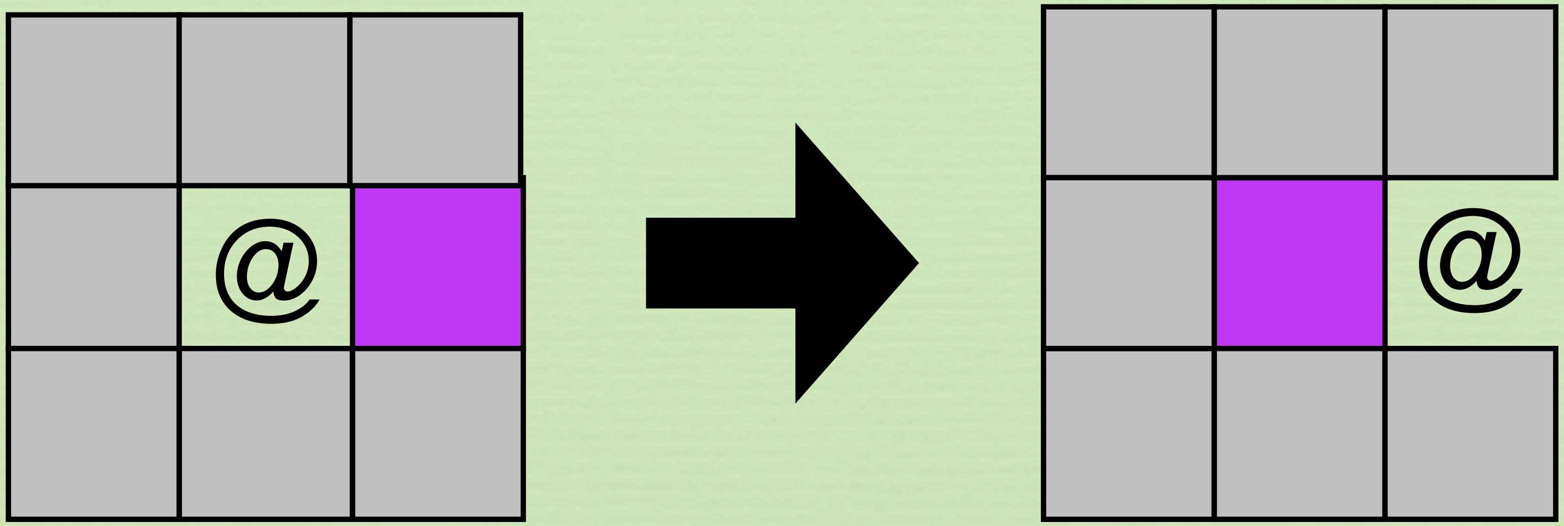
```
loc <entity> <location>
```

2. codify state transitions as linear implications

```
A -o {B}
```

3. specify a query: initial state and expected final state

SEMANTICS OF LINEAR LOGIC PROGRAMS



A -o {B} induces a transition:
forall Δ,

$$\Delta, A \rightarrow \Delta, B$$

SEMANTICS OF LINEAR LOGIC PROGRAMS

the {curly braces} mean
"forward chaining" proof search
(lax modality/monad)

SEMANTICS OF LINEAR LOGIC PROGRAMS

A -o {B} induces a transition:
forall Δ,

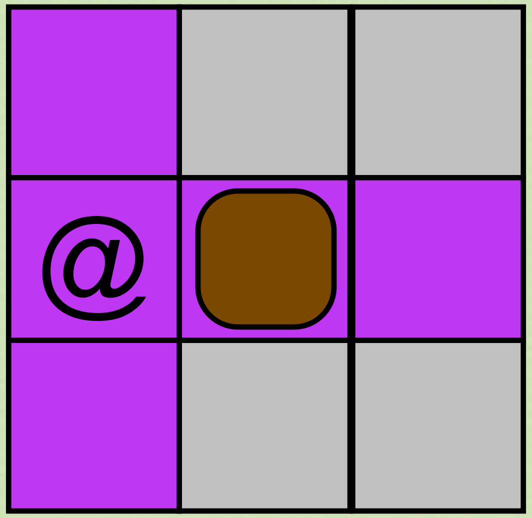
$$\Delta, A \rightarrow \Delta, B$$

meaning this is admissible:

$$\frac{\Delta, B \rightarrow *}{\Delta, A \rightarrow *}$$

COMMITTED CHOICE

when there are multiple choices available,
pick one and commit
(NO BACKTRACKING)



these ideas are implemented in frameworks like Celf, LolliMon, Lygon

(I use Celf)

ACT II

Payoff: More interesting examples, Proofs & Analysis

GOAL: interactive fiction with complex character interaction.

shakespearean tragedy world

state components:
character **location**, **possession**,
sentiment toward other characters,
goals

shakespearean tragedy world

at <character> <location>
has <character> <object>
anger <character> <character>
philia <character> <character>
depressed <character>

shakespearean tragedy world

!dead <character>
!killed <character> <character>

```
do/insult :
at C L * at C′ L * anger C C′
-o {at C L * at C′ L * anger C C′
     anger C′ C * depressed C′}.
```

```
do/compliment :
at C L * at C′ L * philia C C′
-o {at C L * at C′ L *
    philia C C′ * philia C′ C}.
```

```
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 * !dead C’ * !murdered C C’ *
    has C weapon}.
```

```
do/mourn :
at C L * philia C C′ * dead C′
-o {philia C C′ * at C L *
    depressed C * depressed C}.
```

```
do/becomeSuicidal :
at C L * depressed C * depressed C *
depressed C * depressed C
-o {at C L * suicidal C *
    wants C weapon}.
```

```
do/loot
   : at C L * dead C' * has C' O *
     wants C O
     -o {at C L * has C O}.
```

```
do/comfort
: at C L * at C' L *
  suicidal C' * philia C C' * philia C' C
  -o {at C L * at C' L *
       philia C C' * philia C' C *
       philia C' C}.
```

initial state

```
story_start :
init -o
{ at romeo town * at montague mon_house *
  at capulet cap_house * at mercutio town *
  at nurse cap_house * at juliet town *
  at tybalt town * at apothecary town *

  has tybalt weapon * has romeo weapon *
  has apothecary weapon *
 ...
```

```
... *
anger montague capulet * anger capulet montague *
anger tybalt romeo * anger capulet romeo *
anger montague tybalt *

philia mercutio romeo * philia romeo mercutio *
philia montague romeo * philia capulet juliet *
philia juliet nurse * philia nurse juliet *

neutral nurse romeo * neutral mercutio juliet *
neutral juliet mercutio *
neutral apothecary nurse *
neutral nurse apothecary}.
```

final state

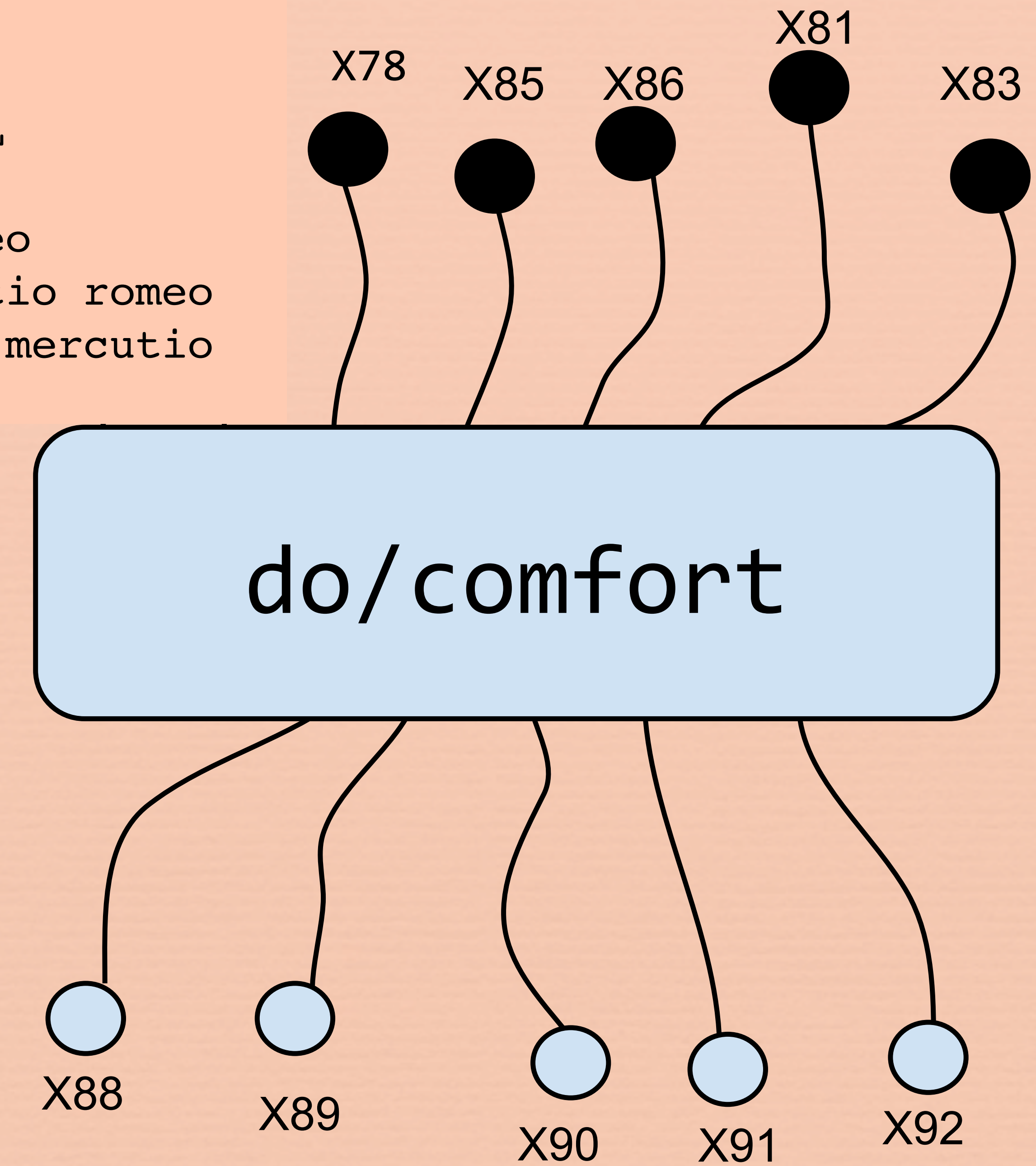
```
ending_happy : nonfinal *
actor C * actor C' *
at C L * at C' L * married C C' -o {final}.


ending_vengeance : nonfinal *
actor C1 * actor C2 * actor C3 *
killed C1 C2 * philia C3 C2 * killed C3 C1
-o {final}.
```

proofs as stories

proof of
init -o {final}

λx:init.
let [xs] = r [ys] in … end

```
X78 : at mercutio L
X85 : at romeo L
X86 : suicidal romeo
X81 : philia mercutio romeo
X83 : philia romeo mercutio
```




```
X88 : at mercutio L
X89 : at romeo L
X90 : philia mercutio romeo
X91 : philia romeo mercutio
X92 : philia romeo mercutio
```

concurrent equality

let x1 = M1 in let x2 = M2 in M

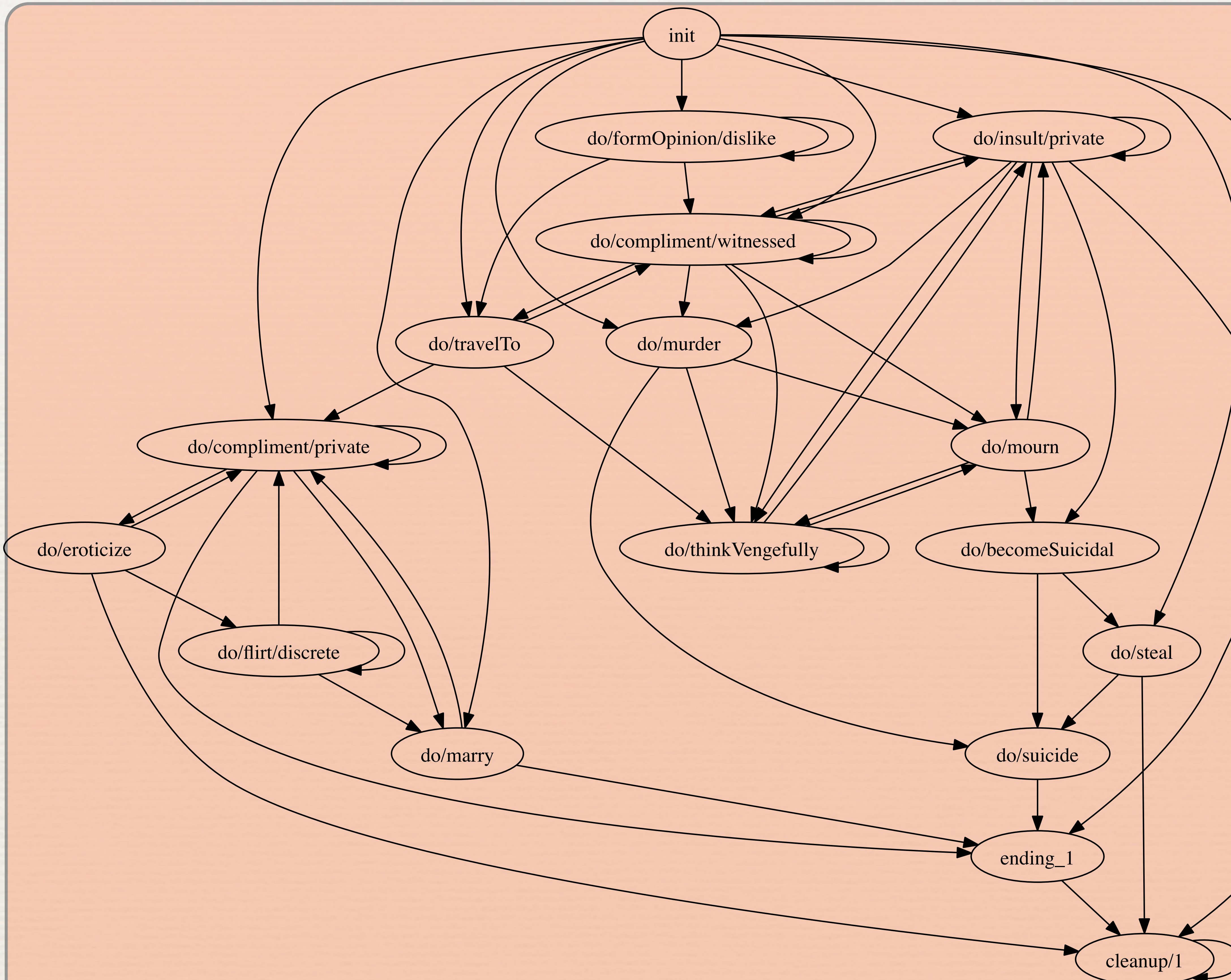
~

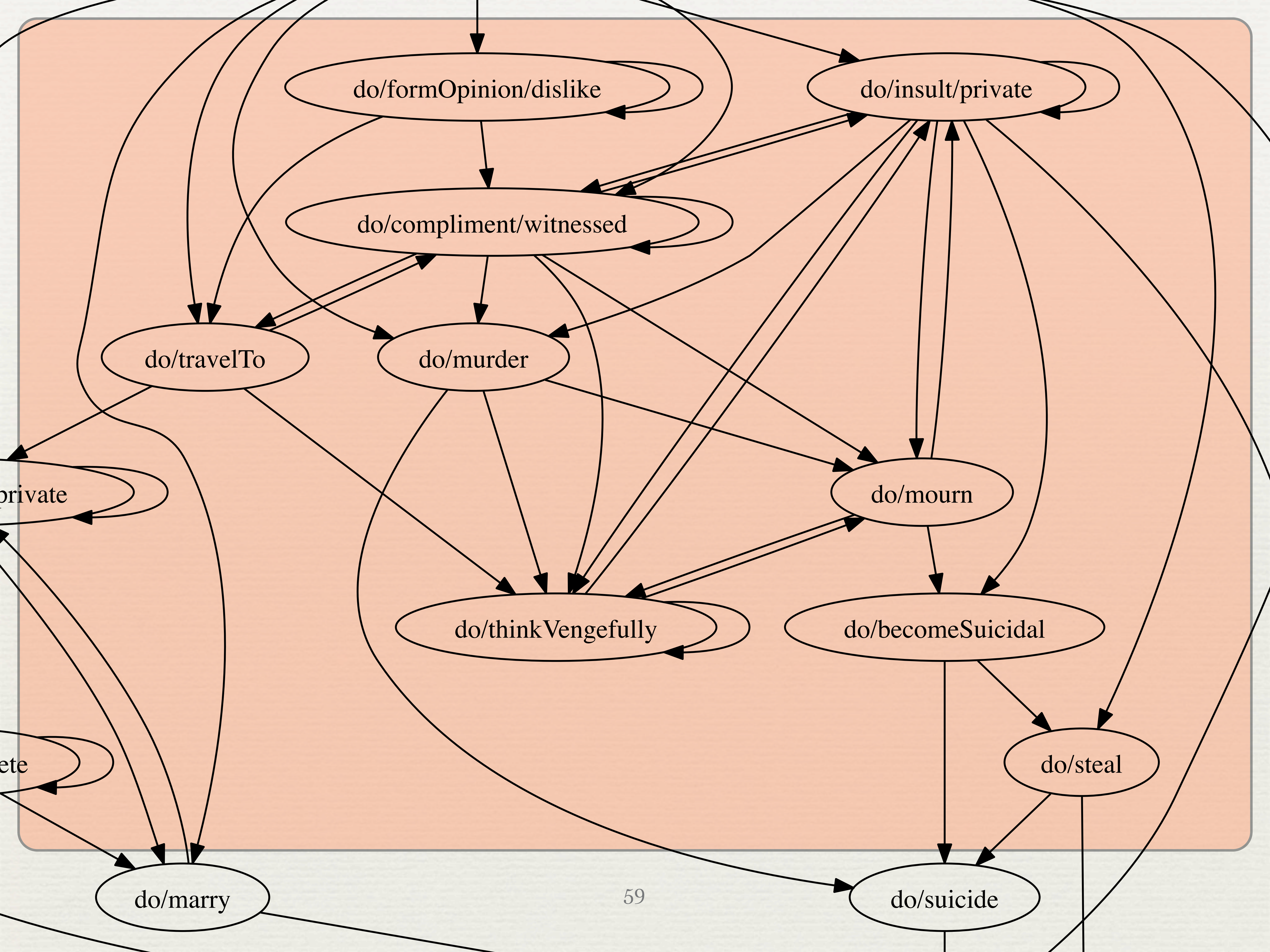
let x2 = M2 in let x1 = M2 in M

iff the inputs of M2 are separate from the outputs of M1.

```
...
let {[X73, [X74, [X75, [X76, X77]]]]}

= do/insult/private [a-tybalt, [a-romeo, [X68, [X66, X72]]]] in

let {[X85, [X86, X87]]}

= do/becomeSuicidal [a-romeo, [X79, [X41, [X59, [X52, X77]]]]] in

let {[X88, [X89, [X90, [X91, X92]]]]}

= do/comfort [a-mercutio, [a-romeo, [X78, [X85, [X86, [X81, X83]]]]]] in

let {[X101, [!X102, [!X103, X104]]]}

= do/murder [a-romeo, [a-tybalt, [X58, [X40, [X76, [X51, [X94, [X96, X27]]]]]]]] in

let {[X105, [X106, [X107, X108]]]}

= do/compliment/private [a-nurse, [a-juliet, [X46, [X47, X30]]]] in

let {[X109, [X110, [X111, X112]]]}

= do/compliment/private [a-juliet, [a-nurse, [X106, [X105, X108]]]] in

let {[X113, X114]}
= do/loot [a-romeo, [a-tybalt, [X101, [X102, [X26, X87]]]]] in

...
```

graphical representation of traces


init
do/formOpinion/dislike
do/insult/private
do/compliment/witnessed
do/travelTo
do/murder
do/compliment/private
do/mourn
do/eroticize
do/thinkVengefully
do/becomeSuicidal
do/flirt/discrete
do/steal
do/marry
do/suicide
ending_1
cleanup/1


do/formOpinion/dislike
do/insult/private
do/compliment/witnessed
do/travelTo
do/murder
private
do/mourn
do/thinkVengefully
do/becomeSuicidal
ete
do/steal
do/marry
do/suicide

queries on sets of traces

> exists ending_1

> exists do/thinkVengefully &&
~link do/thinkVengefully do/murder

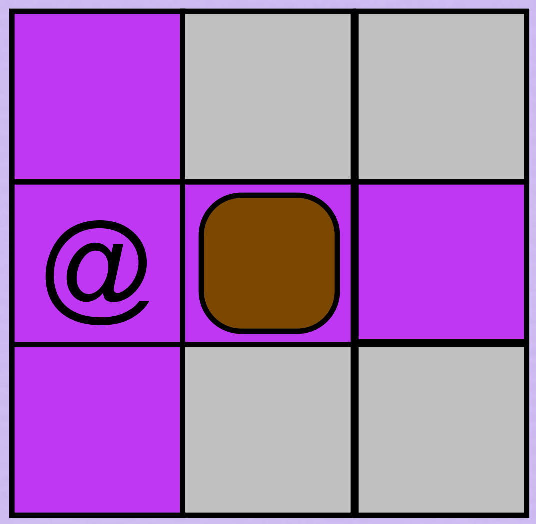
Martens, Ferreira, Bosser
"Generative Story Worlds as Linear Logic Programs"
accepted to INT 2014

ACT III

Promise: Interactivity; Invariant Checking

sokoban, reprise

interactivity, version 1:
at choice points (multiple rules apply), present all available options to player



move
Dir = up, down;
L = …, L’ = ...

push
Dir = right
L, L’, L”..

PROBLEM: not all parts of the program should be manipulable by the player

interactivity, version 2:
give a **language of interaction**

```
dir : type.
u, d, l, r : dir.

act : type.
move <dir> : act.
```

-- new piece of state:

```
action <act>
```

augment rules w/extra premise:

```
push :
action (move 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 (move Dir) *
loc pusher L * in_dir L Dir L' * empty L'
-o {empty L * loc pusher L'}.
```

but when to introduce
“action A”?

PHASES

Phases

Block-delimited subsignatures

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

phase player = {
  rule : player_turn -o {…}
}
```

Phases

Connected by specification of *quiescence* behavior

```
phase world = {...}

phase player = {...}

quiesced world -o
  {player_turn * phase player}.

quiesced player -o {phase world}.
```

Phases

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

```
quiesced P * State -o {phase P' * State'}.
```

arbitrarily many phases

looping + branching

```
rock * paper -o {paper}.
paper * scissors -o {scissors}.
scissors * rock -o {rock}.

rock * rock_count N -o {rock_count N+1}.
paper * paper_count N -o {paper_count N+1}.
scissors * scissors_count N -o {scissors_count N+1}.

init -o {rock_count 0 * paper_count 0 * scissors_count 0
         * rock * rock * rock * paper * paper * scissors}.
```

```
rock * paper -o {paper}.
paper * scissors -o {scissors}.
scissors * rock -o {rock}.

rock * rock_count N -o {rock_count N+1}.
paper * paper_count N -o {paper_count N+1}.
scissors * scissors_count N -o {scissors_count N+1}.

init -o {rock_count 0 * paper_count 0 * scissors_count 0
         * rock * rock * rock * paper * paper * scissors}.
```

```
rock * paper -o {paper}.
paper * scissors -o {scissors}.
scissors * rock -o {rock}.

rock * rock_count N -o {rock_count N+1}.
paper * paper_count N -o {paper_count N+1}.
scissors * scissors_count N -o {scissors_count N+1}.

init -o {rock_count 0 * paper_count 0 * scissors_count 0
        * rock * rock * rock * paper * paper * scissors}.
```

```
phase rps = {
  rock * paper -o {paper}.
  paper * scissors -o {scissors}.
  scissors * rock -o {rock}.

  init -o {rock * rock * rock * paper * paper * scissors}.
}

phase count = {
  init -o {rock_count 0 * paper_count 0 * scissors_count 0}.

  rock * rock_count N -o {rock_count N+1}.
  paper * paper_count N -o {paper_count N+1}.
  scissors * scissors_count N -o {scissors_count N+1}.
}
```

```
phase rps = {
  rock * paper -o {paper}.
  paper * scissors -o {scissors}.
  scissors * rock -o {rock}.

  init -o {rock * rock * rock * paper * paper * scissors}.
}

phase count = {
  init -o {rock_count 0 * paper_count 0 * scissors_count 0}.

  rock * rock_count N -o {rock_count N+1}.
  paper * paper_count N -o {paper_count N+1}.
  scissors * scissors_count N -o {scissors_count N+1}.
} %% expects: all rock, all paper, or all scissors.
```

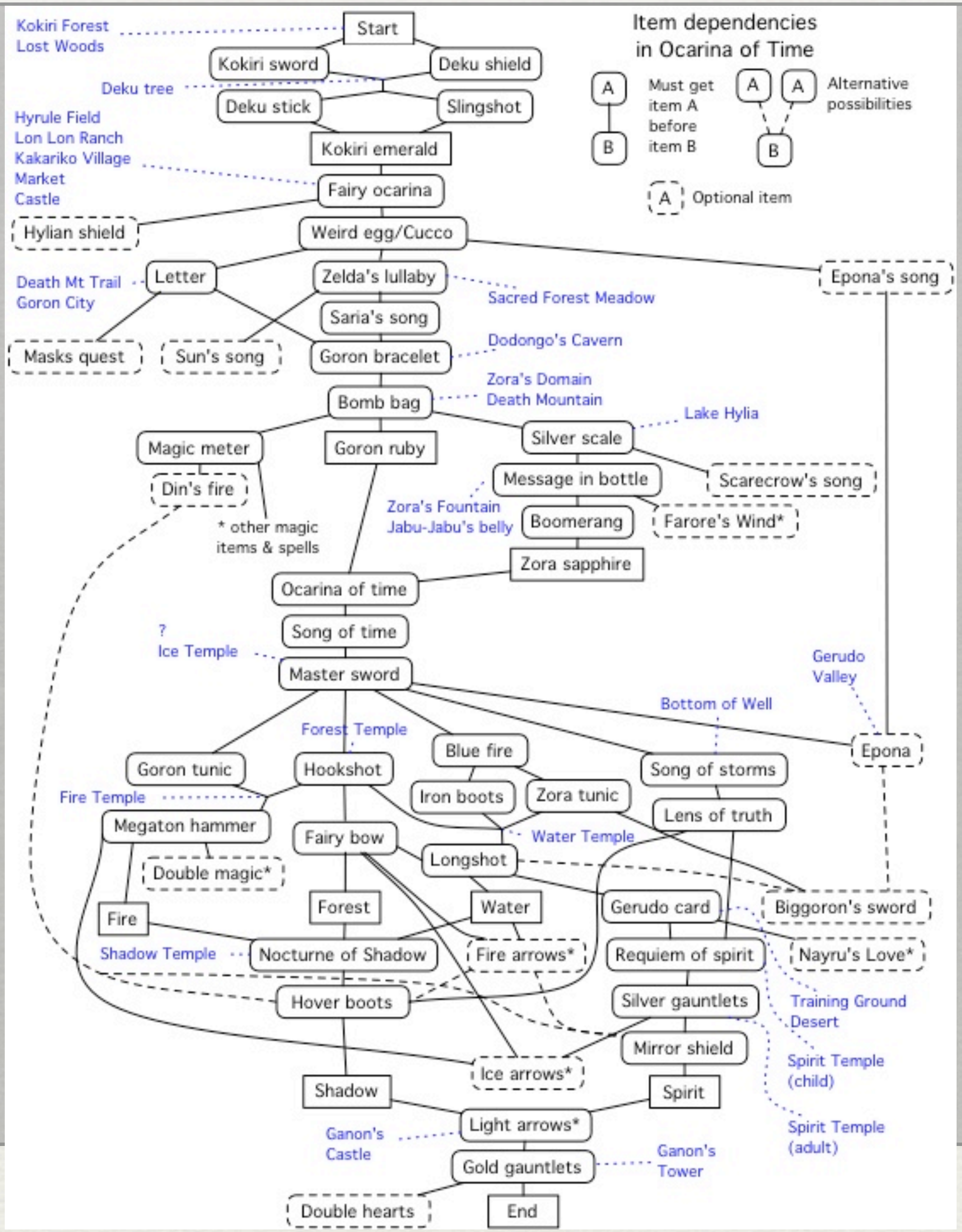
Compiling Phases

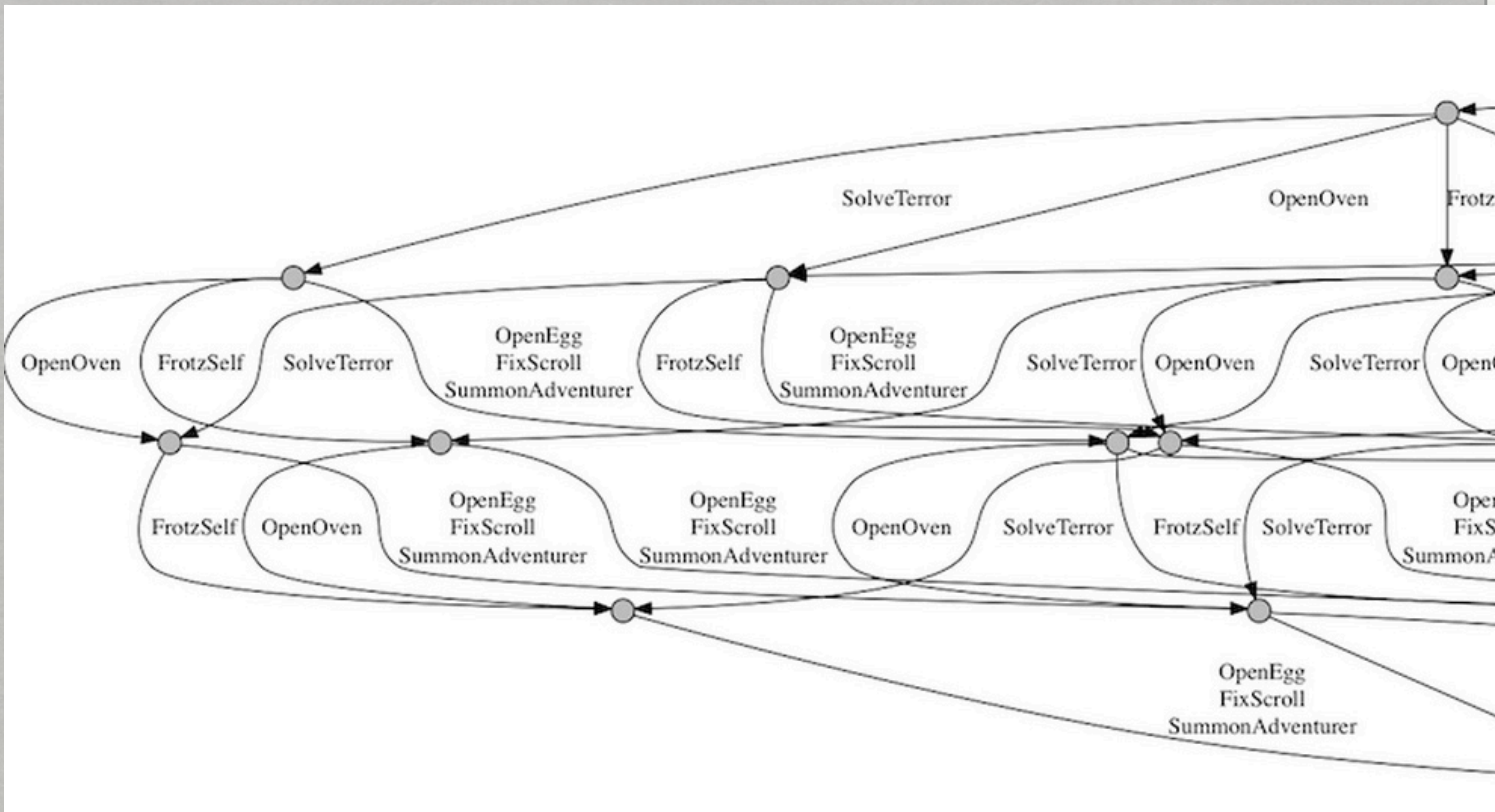
We can interpret phase-structured programs as programs with higher-order, mixed-chaining rules in Celf.

Compiling Phases

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

FINALE


Item dependencies
in Ocarina of Time
A
B
Must get item A before item B
A
A
B
Alternative possibilities
A
Optional item
Start
Kokiri Forest
Lost Woods
Kokiri sword
Deku shield
Deku tree
Deku stick
Slingshot
Kokiri emerald
Hyrule Field
Lon Lon Ranch
Kakariko Village
Market
Castle
Fairy ocarina
Hylian shield
Weird egg/Cucco
Epona's song
Letter
Zelda's lullaby
Death Mt Trail
Goron City
Sacred Forest Meadow
Saria's song
Masks quest
Sun's song
Goron bracelet
Dodongo's Cavern
Zora's Domain
Death Mountain
Bomb bag
Lake Hylia
Magic meter
Goron ruby
Silver scale
Message in bottle
Scarecrow's song
Din's fire
Zora's Fountain
Jabu-Jabu's belly
Boomerang
Farore's Wind*
* other magic items & spells
Zora sapphire
Ocarina of time
Song of time
?
Ice Temple
Master sword
Gerudo Valley
Bottom of Well
Forest Temple
Blue fire
Epona
Goron tunic
Hookshot
Song of storms
Iron boots
Zora tunic
Fire Temple
Lens of truth
Megaton hammer
Water Temple
Fairy bow
Longshot
Double magic*
Forest
Water
Gerudo card
Biggoron's sword
Fire
Shadow Temple
Nocturne of Shadow
Fire arrows*
Requiem of spirit
Nayru's Love*
Hover boots
Silver gauntlets
Training Ground
Desert
Mirror shield
Spirit Temple (child)
Ice arrows*
Shadow
Spirit
Light arrows*
Spirit Temple (adult)
Ganon's Castle
Gold gauntlets
Ganon's Tower
Double hearts
End


SolveTerror
OpenOven
OpenOven
FrotzSelf
SolveTerror
OpenEgg
FixScroll
SummonAdventurer
FrotzSelf
OpenEgg
FixScroll
SummonAdventurer
SolveTerror
OpenOven
SolveTerror
FrotzSelf
OpenOven
OpenEgg
FixScroll
SummonAdventurer
OpenEgg
FixScroll
SummonAdventurer
OpenOven
SolveTerror
FrotzSelf
SolveTerror
OpenEgg
FixScroll
SummonAdventurer

Minecraft Production Web

Key -
- Used to obtain
- Created With
- Manufacured with
- Obtained by drop
- Produced by / Found by

how designs fail

FINALE

takeaway:

linear logic with phases
as a DSL for game design enables
rapid experimentation and **structural analysis**
of a wide range of core ludical mechanics.