Praxis

A logic-based DSL for modeling social practices
Demo

• Show Marriage Proposal from two angles
• Show Dinner Party, choosing two different characters
Versu is...

• Real-time
• Multiplayer
• Text-based
• Simulation
• Set in Jane Austen’s Regency England
The Simulator

- The world is a set of facts
- The dynamic elements are social practices and agents
The Simulator

• The world is a set of facts
• The dynamic elements are social practices and agents
Social Practices
The Need for Social Practices

• The Sims 1
• My Sim invited his boss over for dinner.
• When he arrived, my Sim let him in - *but then he went to have a bath!* 
• He didn’t understand that certain things were expected of him as a host.
What is a Social Practice?

• It describes what agents *can* do in a social situation
• It also says what agents *should* do
Social Practices

Practices

Agents
Social Practices

Practices

Agents
Social Practices
Social Practices

State A
- Option
- Option

State B
- Option

State C
- Option

State D
- Option
- Option
- Option

Human
- Option
- Option
What is a Social Practice?

• It issues different requests in different circumstances
• It issues different requests to different people
• It notices when requests are satisfied or confounded
Demo

• Show an example of norm-violation.
Multiple Concurrent Practices
300+ Social Practices in Versu

• Dinner Party
• Conversation
• Debate
• Games
• Death
Demo

• Evaluate `process.X` in dinner party and whist game
• Show sub-tree of `process.whist`
Implementation

• A Social Practice is a set of sentences in Exclusion Logic
The Simulator

• The world is a set of facts
• The dynamic elements are social practices and agents
Agents
Implementation

• An agent is just a set of sentences in Exclusion Logic
  – Beliefs
  – Desires
  – Personality quirks
  – Backstory
Demo

• Show sub-terms of brown in the Dinner Party
Agents

• An agent has a set of wants
• He uses utility-based decision-making
Demo

• Show sub-terms of brown.wants in the Dinner Party
• Show the actions Brown is considering, sorted by score
The Simulator

• The world is a set of facts
• The dynamic elements are social practices and agents
• The dynamic elements supervene on the facts
The Simulator

• The world is a set of facts
• The dynamic elements are social practices and agents
• The dynamic elements supervene on the facts
Facts Instantiate Processes
Processes Provide Actions
Agents Perform Actions
Performance Modifies Facts
Exclusion Logic

Praxis is based on a new modal logic called Exclusion Logic
Elementary Propositions

- Jack fell
- Jack likes Jill
Propositional Logic

“Jack likes Jill” → p

• We cannot infer “Jack likes someone”
Predicate Logic

“Jack likes Jill” \(\rightarrow\) Likes(Jack, Jill)

\[\text{Likes}(Jack, Jill) \vdash (\exists x)\text{Likes}(Jack, x)\]
• In Predicate Logic, there are no logical relations between elementary propositions

\[ Likes(Jack, Jill) \vdash (\exists x) Likes(Jack, x) \]
• In Predicate Logic, there are no logical relations between elementary propositions

\[ Likes(Jack, Jill) \vdash (\exists x) Likes(Jack, x) \]
Logical Relations between Elementary Propositions

• “Jack is male” is incompatible with “Jack is female”
• “Jack walks quickly” entails “Jack walks”
Exclusion Logic

• A logic which supports logical relations between elementary propositions
Wittgenstein

• “There are rules for the truth functions which also deal with the elementary part of the proposition”
# Elementary Propositions

<table>
<thead>
<tr>
<th>Logical System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional Logic</td>
<td>An indivisible atomic sentence</td>
</tr>
<tr>
<td>Predicate Logic</td>
<td>Supports inferential relations with <em>compound</em> sentences</td>
</tr>
<tr>
<td>Exclusion Logic</td>
<td>Supports inferential relations with other <em>elementary</em> propositions</td>
</tr>
</tbody>
</table>
Exclusion Logic

\[ E ::= S \mid S \cdot E \mid S!E \]

\[ C ::= E \mid \neg C \mid C \land C \]
\[ E ::= S \mid S.E \mid S!E \] 

- The "." and "!" operators are used to build up trees of information
- \( S.E \) means that \( E \) is one of the ways in which \( S \) is true
- \( S!E \) means that \( E \) is the only way in which \( S \) is true
\[ E ::= S \mid S.E \mid S!E \]

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack.Fell</td>
<td>One of the properties of Jack is that he fell</td>
</tr>
<tr>
<td>Jack.Likes.Jill</td>
<td>One of the people Jack likes is Jill</td>
</tr>
<tr>
<td>Jack.Gender!Male</td>
<td>The (unique!) gender of Jack is male</td>
</tr>
</tbody>
</table>
\[ E ::= S \mid S.E \mid S!E \]

- Jack.Likes.Jill
- Jack.Likes.Josie
\[ E ::= S \mid S.E \mid S!E \]

- Jack.Gender!Male
- Jack.Gender!Female
Inference Rules

\[ X.Y \vdash X \]

\[ X!Y \vdash X \]

\[ X!Y \land X!Z \vdash P \]
Inference Rules

\[ X.Y \vdash X \]
\[ X!Y \vdash X \]
\[ X!Y \land X!Z \vdash P \]

\[ X.Y \not\vdash Y \]
\[ X!Y \not\vdash Y \]
\[ X.Y \land X.Z \not\vdash P \]
Logical Relations between Elementary Propositions

• “Jack is male” is incompatible with “Jack is female”

• “Jack walks quickly” entails “Jack walks”
Logical Relations between Elementary Propositions

• “Jack is male” is incompatible with “Jack is female”

\[ \text{Jack.Gender!Male} \models \neg \text{Jack.Gender!Female} \]
Logical Relations between Elementary Propositions

- “Jack walks quickly” entails “Jack walks”

$Jack.Walks.Quickly \vdash Jack.Walks$
Representing Incompatible Predicates in Predicate Logic

\[ \text{Gender}(Jack) = Male \]

- Requires identity predicate and axiom schema

\[ (\forall x, y) \ x = y \land F(x) \rightarrow F(y) \]
Representing Incompatible Predicates in Predicate Logic

• Brachman and Levesque:

\[(\forall x) \text{Man}(x) \rightarrow \neg \text{Woman}(x)\]
Representing Incompatible Predicates in Predicate Logic

• Brachman and Levesque:

\[(\forall x)\text{Man}(x) \rightarrow \neg\text{Woman}(x)\]

\[(\forall x)\text{SupportsArsenal}(x) \rightarrow \neg\text{SupportsBarnsley}(x) \land
\neg\text{SupportsFulham}(x) \land
\neg\text{SupportsGrimsby}(x) \land \ldots\]
Adverbial Inferences in Predicate Logic

- Davidson analysed “I flew my spaceship to the Morning Star” as:

\[(\exists x)\text{Flew}(I, My\text{Spaceship}, x) \land \text{To}(x, The\text{MorningStar})\]

- “I flew my spaceship to the Morning Star” entails “I flew my spaceship”
Adverbial Inferences in Predicate Logic

• “Jack walks”

\((\exists x)Walks(Jack, x)\)
Predicate Logic vs Exclusion Logic

• Predicate Logic can handle these inferences
• But it can only do so by reinterpreting the sentences as compound
• It uses more complex machinery to get the same results that Exclusion Logic gets directly
Semantics

• We use a labeled rooted tree
• Every vertex is reachable from a starting vertex T
• Each vertex is labeled with a symbol from S
• Each edge is labeled with either ! or *
Labeled Rooted Tree

- (V, E, L, M, R) where
- V: set of vertices
- E: set of edges \((V_1, V_2)\)
- L: vertex labeling \(V \rightarrow S\)
- M: edge labeling \(E \rightarrow \{*,!\}\)
- R: root, member of V
Semantics

– $\mathbf{R} \in \mathbf{V}$ is the vertex for the root of the tree, where $L_r$ and $L_s = \mathbf{T}$ (here, $\mathbf{T}$ is a symbol not occurring in $S$ used to label the root of each tree).

Each element of the tuple yields a corresponding accessor function:

- $\mathbf{X}_v$ is the vertices of $\mathbf{X}$
- $\mathbf{E}_v$ is the edges of $\mathbf{X}$
- $\mathbf{L}_v$ is the vertex labels of $\mathbf{X}$
- $\mathbf{M}_v$ is the edge labels of $\mathbf{X}$ and $\mathbf{R}_v$ is the root vertex of $\mathbf{X}$

Additionally:

- Define $E^* \mathbf{X}$ as the transitive closure of $E \mathbf{X}$

Definition 3.

Not all sets of vertices and edges form a valid labeled rooted tree. An LRT is valid if:

- irredundant: no two children of a vertex share the same label
- respectful of exclusion: if $\mathbf{M}_r x, y \mathbf{s} = \mathbf{h}$ then there are no other edges $r x, z \mathbf{s} \in E$ for some $z \neq y$

Valid LRTs

In these examples, the $\ast$ labels have been suppressed. Note that two vertices may share the same label.

Fig. 3: Valid LRTs

4.1 Defining $\leq$ on LRTs

We are going to interpret expressions in a lattice of LRTs so we start with a partial ordering on LRTs. We will say $A \leq B$ if $A$ contains at least as much information as $B$ if $B$ is a subgraph of $A$ which has the same root.

Now it might seem more natural to define $A \leq B$ if $A$ is a subgraph of $B$ but we want to preserve the semantic convention that $X$ entails $Y$ iff $[X] \leq [Y]$.
A Partial Ordering on LRTs

We will define a partial ordering \( \leq \) on LRTs using subgraph isomorphism. We introduce the concept of the signature of a vertex, and say that two vertices from two different graphs are equivalent if they share the same signature.

Definition 4. The signature \( s(Xv) \) of a vertex \( v \) in LRT \( X \) is the ordered list of vertex symbols associated with the path from \( R_X \) to \( v \). Because the LRT is a tree, each vertex has only one path from \( R_X \).

Definition 5. Vertex \( v \) in \( X \) and vertex \( v' \) in \( Y \) are equivalent, written \( X,v \equiv Y,v' \), if \( s(Xv) = s(Yv') \). Edges are equivalent if both vertices are equivalent.

Definition 6. A \( \leq \) B if every edge in \( B \) has a corresponding edge in \( A \), and edge-labels of \( A \) are at least as specific as the labels in \( B \).

Note that it is possible for \( A \leq B \) even if \( A \) and \( B \) have different labels on the same edge, as long as the label on \( A \)'s edge is more specific than \( B \)'s label for the corresponding edge.

Examples of \( \leq \)

Here, again, the \( * \) labels have been omitted from the diagram to reduce clutter.
A Partial Ordering on LRTs

We will define a partial ordering $\leq$ on LRTs using subgraph isomorphism. We introduce the concept of the signature of a vertex, and say that two vertices from two different graphs are equivalent if they share the same signature.

Definition 4. The signature $s(X,v)$ of a vertex $v$ in LRT $X$ is the ordered list of vertex symbols associated with the path from $R_X$ to $v$. Because the LRT is a tree, each vertex has only one path from $R_X$.

Definition 5. Vertex $v$ in $X$ and vertex $v'$ in $Y$ are equivalent, written $v \equiv v'$, if $s(X,v) = s(Y,v')$. Edges are equivalent if both vertices are equivalent.

Definition 6. $A \leq B$ if every edge in $B$ has a corresponding edge in $A$, and edge-labels of $A$ are at least as specific as the labels in $B$.

Note that it is possible for $A \leq B$ even if $A$ and $B$ have different labels on the same edge, as long as the label on $A$'s edge is more specific than $B$'s label for the corresponding edge.
Exclusion Logic 9

Proof. The simplest example which shows this is:

\[ \text{T A T} \]
\[ \text{!} \]
\[ \text{T B} \]
\[ (a) \]
\[ (b) \]
\[ (c) \]

Fig 6: Incompatibility is not transitive

Here is incompatible with \( s_b \) and \( s_b \) is incompatible with \( s_c \) but \( s_a \) is not incompatible with \( s_c \).

4.3 LRTs form a lattice

We will use LRTs to model formulae in ELw and \( \perp \) to model conjunctions of expressions. So we need some sort of LRT to model incompatible conjunctions and stipulate that for all \( X \)
\[ \perp \leq X \]

Similarly, the lattice needs a top element \( \top \) such that for all \( X \)
\[ X \leq \top \]

We define \( \top \) as
\[ \{ 1 \}, \{ \}, \{ s_1, T \}, \{ \}, 1 \]

Now, with these elements in place, we are ready to define greatest lower bound.

**Definition 8.** For any LRTs \( X \) and \( Y \), define \( X \triangleright Y \) as \( \perp \) if \( X \) and \( Y \) are incompatible; otherwise:

\[ \forall V \in X \cup V \in Y \triangleright, E \in X \cup E \in Y \triangleright, L \in X \cup L \in Y \triangleright, \min \leq M \in X \cup M \in Y \triangleright t, R \in X \cup R \in Y \triangleright \]

where
\[ \min \leq s m \in X \]
\[ v \mapsto v' \text{ if } v = v' \]

Fig 7: Example of \( \triangleright \)
Greatest Lower Bound

Definition 9. For any LRTs $X$ and $Y$, the least upper bound $X \triangleright Y$ is

$$\left( V_X \cap V_Y, E_X \cap E_Y, L_X \cap L_Y, \max \leq (M_X, M_Y) \right)$$

where

$$\max \leq (X, Y) = \{ (E, !) | (E, !) \in X \land (E, !) \in Y \} \cup \{ (E, \ast) | (E, \ast) \in X \land (E, \ast) \in Y \} \lor (E, \ast) \in X \land (E, !) \in Y \lor (E, !) \in X \land (E, \ast) \in Y \}$$

and $Y'$ is a renaming of the vertices of $Y$, as before.
Greatest Lower Bound

**Fig. 8:** Example of ⊥

**Fig. 9:** Example of ⊗

**Proposition 4.**

**Definition 9.** For any LRTs \( X \) and \( Y \), the least upper bound \( X \oplus Y \) is

\[
(V_X \cap V_\oplus Y, E_X \cap E_\oplus Y, L_X \cap L_\oplus Y, \max(\min(M_X, M_Y)))
\]

where

\[
\max(\min(X, Y)) = \{(E, !) | (E, !) \in X \land (E, !) \in Y \} \cup \{(E, *) | ((E, *) \in X \land (E, !) \in Y) \lor ((E, !) \in X \land (E, *) \in Y) \lor ((E, !) \in X \land (E, *) \in Y) \}
\]

and \( Y' \) is a renaming of the vertices of \( Y \), as before.
Satisfaction

\[ \text{Sat}(X, v, L, S) \text{ iff } \exists v' : (v, v') \in E_X \]
\[ L_X(v') = S \text{ and } \]
\[ M_X(v, v') = L \]

\[ \text{Sat}(X, v, L, S!E) \text{ iff } \exists v' : (v, v') \in E_X \]
\[ L_X(v') = S \]
\[ M_X(v, v') = L \text{ and } \]
\[ \text{Sat}(X, v', !, E) \]

\[ \text{Sat}(X, v, L, S.E) \text{ iff } \exists v' : (v, v') \in E_X \]
\[ L_X(v') = S \text{ and } \]
\[ M_X(v, v') = L \text{ and } \]
\[ \text{Sat}(X, v', *, E) \]
Decision Procedure

• Define \([x]\) as the set of LRTs which satisfy \(x\)

\[
[x] = \{ M \mid \models_M x \}
\]

• Because the LRTs form a lattice, this set has a least upper bound:

\[ \bigvee[x] \]
decision procedure

\[ X \models Y \iff \forall M \models_M X \Rightarrow \models_M Y \]

\[ \iff [X] \subseteq [Y] \]

\[ \iff \bigcup [X] \leq \bigcup [Y] \]
Computing the LUB

\[ m(A \land B) = m(A) \cap m(B) \]

\[ m(A:B) = (V_{m(A)} \cup \{v\}, E_{m(A)} \cup \{(v',v)\}, L_{m(A)} \cup (v,B), M_{m(A)} \cup \{((v',v),*)\}) \]

\[ m(A.B) = (V_{m(A)} \cup \{v\}, E_{m(A)} \cup \{(v',v)\}, L_{m(A)} \cup (v,B), M_{m(A)} \cup \{((v',v),!)\}) \]
Hennessy-Milner Logic

• Let $A$ be a set of constants
• Let $B = \{*,!\}$ be a two-point set

$$C ::= \langle \alpha, b \rangle C \mid C \land C \mid \top$$

where $\alpha \in A, b \in B$
Hennessy-Milner Logic

- A model is a rooted graph where transitions are labeled with constants from $A$
- Satisfaction in a graph $T$ rooted at $r$:

$$T \models \langle \alpha, * \rangle C \text{ if } \exists t, r \xrightarrow{\alpha} t \wedge T(t) \models C$$

$$T \models \langle \alpha, ! \rangle C \text{ if } \exists t, r \xrightarrow{\alpha} t \wedge T(t) \models C \wedge \text{out}(t) = 1$$

$$T \models A \wedge B \text{ if } T \models A \wedge T \models B$$
Praxis

Using Exclusion Logic as a Logic Programming Language
Praxis: Evolution

1) Roll-my-own procedural language
   • Spent a lot of time implementing basic language features
   • No debugger; no visualisation of state

2) Thin DSL on top of LUA
   • Untyped

3) Coded practices directly in C#
   •Verbose, error-prone

4) Practices encoded in Deontic Logic

5) Praxis
The Query Language

\[ E ::= T \mid T . E \mid T ! E \]

\[ Q ::= E \mid \neg Q \mid Q \land Q \mid Q \lor Q \mid Q \rightarrow Q \mid \forall X, Q \mid \exists X, Q \]
Typing

• Praxis is strongly typed and statically typed
• It has sub-typing
• It uses type-inference
Type Inference

function define_characters
    if global.playable.N!X
    then
        insert global.is_playing.X
        insert X.at!front_yard
    ...

    global.playable.Index(number)!Agent(agent)
Instantiating Practices

process.greet.X(agent).Y(agent)
action "Greet"
  preconditions
  Actor = X
  Actor.in!L
  postconditions
  text "[X] says 'hullo' to [Y obj]" if Recipient.in!L
  call update_conversation.L.Actor.greet.Y.respond_to_greet
  insert process.respond_to_greet.Y.X
  delete Self
end
Instantiating Practices

process.greet.X(agent).Y(agent)
  action "Greet"
  preconditions
  Actor = X
  Actor.in!L
  postconditions
  text "[X] says 'hullo' to [Y obj]" if Recipient.in!L
  call update_conversation.L.Actor.greet.Y.respond_to_greet
  insert process.respond_to_greet.Y.X
  delete Self

end

process.greet.jack.jill
Instantiating Practices

process.greet.X(agent).Y(agent)
  action "Greet"
    preconditions
      Actor = X
      Actor.in!L
    postconditions
      text "[X] says 'hullo' to [Y obj]" if Recipient.in!L
      call update_conversation.L.Actor.greet.Y.respond_to_greet
      insert process.respond_to_greet.Y.X
      delete Self
  end

process.greet.jack.jill
  Jack/X, Jill/Y
Practices are HFSMs

process.ticTacToe.Player1(agent).Player2(agent)
...
state!whoseMove!Mover(agent)!Other(agent)
  action "Tic Tac Toe | Row [R] | Place [Piece] at [C],[R]"
  preconditions
    Actor = Mover
    Parent.board.C.R!empty
    Parent.piece.Mover!Piece
    Parent.piece.Other!OtherPiece
  postconditions
    text "[Mover] place[s] an [Piece] at [C], [R]." if Parent.board.C.R!Piece
    call updateBoardOnMove.Mover.Other.C.R.Piece.OtherPiece
    insert Parent.state!whoseMove!Other!Mover
...

80
Practices have constructors

```plaintext
process.ticTacToe.Player1(agent).Player2(agent)
  start
    insert Self.participants.Player1
    insert Self.participants.Player2
    insert Self.viewers.Player1
    insert Self.viewers.Player2
    text "You are playing 'X'" to Player1
    text "You are playing '0'" to Player2
    insert Self.piece.Player1!x
    insert Self.piece.Player2!o
  ...
```
Practices provide actions

action "The game of whist...[Trump with the [RT] of [S]]"
preconditions
  Actor = Player
  Actor.in!L
  Parent.trumps!S
  Parent.cards.Actor.R.S
  data.cards.rank.R!RV!RT
  Parent.leading_suit!LeadingSuit
  LeadingSuit = S
  not Parent.cards.Actor.Any.LeadingSuit
postconditions
  text "[Actor] trump[s] with the [RT] of [S]"
  call norm_respecting.Actor
  insert Parent.trick.Actor!S
  delete Parent.cards.Actor.R.S
  call evaluate_trump.Actor
  if N = 4 then
    insert Parent.state!evaluate_trick
  else
    if Parent.next.Actor!Next and NextN = N+1 then
      insert Parent.state!follow!NextN!Next
  end
end
action "The game of whist...|Trump with the [RT] of [S]"
preconditions
  Actor = Player
  Actor.in!L
  Parent.trumps!S
  Parent.cards.Actor.R.S
  data.cards.rank.R!RV!RT
  Parent.leading_suit!LeadingSuit
  LeadingSuit $\sim S$
  not Parent.cards.Actor.Any.LeadingSuit
postconditions
  text "[Actor] trump[s] with the [RT] of [S]"
  call norm_respecting.Actor
  insert Parent.trick.Actor!R!S
  delete Parent.cards.Actor.R.S
  call evaluate_trump.Actor
  if N = 4 then
    insert Parent.state!evaluate_trick
  else
    if Parent.next.Actor!Next and NextN = N+1 then
      insert Parent.state!follow!NextN!Next
  end
end
Updating the Database

• When adding a sentence $p$ to the database, we first remove all information which is incompatible with $p$

• This is a non-monotonic update
Updating the Database

6.3 Update Rules

What makes Exclusion Logic update nonmonotonic is this: when adding a formula \( f \) of Exclusion Logic to a database \( D \) we first remove all information from \( D \) which is incompatible with \( f \). For example, if our database is the LRT for \( A \land B \rightarrow C \) and then we add \( B \land D \):

\[
\begin{array}{c}
T \\
A \\
B \\
C \\
\end{array}
\quad \xrightarrow{B \land D}
\quad \begin{array}{c}
T \\
A \\
B \\
D \\
\end{array}
\]

The update rule removes the entire sub-tree which is incompatible with the newly added formula. If, for example, we have \( A \land B \land C \land E \land B \land C \land F \) and we add \( B \land D \) the entire sub-tree is removed:

\[
\begin{array}{c}
T \\
A \\
B \\
E \\
F \\
\end{array}
\quad \xrightarrow{B \land D}
\quad \begin{array}{c}
T \\
A \\
B \\
D \\
\end{array}
\]

Fig. 13: Adding \( B \land D \) when starting with \( A \land B \land C \).

Fig. 14: Adding \( B \land D \) when starting with \( A \land B \land C \land E \land B \land C \land F \).

Both these properties are false when updating exclusion logics as we shall now see.
Updating the Database

Define \( \leq \) on databases so that \( X \leq Y \) iff \( X \) contains at least all the information of \( Y \). Now define add \( a : D \to D \) as the curried version of \( \text{add}_y \) specialized to a particular formula \( a \). With traditional database updatey we have:

\[
\text{add}_a \leq x \text{add}_a \cdot \text{add}_b = \text{add}_b \cdot \text{add}_a
\]

Both these properties are false when updating exclusion logicy as we shall now see.

6.3 Update Rules

What makes Exclusion Logic update nonzmonotonic is this: when adding a formula \( f \) of Exclusion Logic to a database \( D \) we first remove all information from \( D \) which is incompatible with \( f \). For example, if our database is the LRT for \( A \land B \) \( m \) \( C \) and then we add \( B \) \( m \) \( D \):

![Diagram showing update process]

The update rule removes the entire sub-tree which is incompatible with the newly added formula. If, for example, we have \( A \land B \) \( m \) \( C \cdot E \) \( \land \) \( B \) \( m \) \( C \cdot F \) and we add \( B \) \( m \) \( D \) the entire sub-tree is removed:

![Diagram showing update process]
Using Exclusion Logic as a KRL
An Object is a Sub-Tree

- brown.sex!male
- brown.class!upper
- brown.in!dining_room
- brown.relationship.lucy.evaluation.attractive!40
- brown.relationship.lucy.evaluation.humour!20
An Object is a Sub-Tree

• Specify the life-time of a piece of data by placing it in the right part of the tree

• brown.relationship.lucy.evaluation.attractive!40
• process.whist.data.whose_move!brown
Garbage Collection

- An FSM has two states **a** and **b**
- State **a** has two bits of data: **x** and **y**
- We are in state **a**:
  - `fsm.state!a.x ∨ fsm.state!a.y`
- Now insert `fsm.state!b`
- The data `(a.x ∨ a.y)` is removed automatically
Simpler Postconditions

action move(A, X, Y)
preconditions
   at(A, X)
postconditions
   add at(A, Y)
   remove at(A, X)

Herewith we need to explicitly state that when A moves from X to Y, A is no longer at X. It might seem obvious to us that if A is now at Y, he is no longer at X, but we need to explicitly tell the system this. This is unnecessary and cumbersome. In Exclusion Logic, by contrast, the exclusion operator means we do not need to specify the facts that are no longer true:

action move(A, X, Y)
preconditions
   A.at!X
postconditions
   add A.at!Y

The "m" operator makes it clear that something can only be at one place at a time and the nonmonotonic update rule automatically removes the old invalid location data.

7.4 The Exclusion Operator Helps the Author Specify Her Intent

The semantics of the exclusion operator remove various error-prone bookkeeping tasks from the implementer. But perhaps the exclusion operator's main benefit is that it allows the simulation author to specify her intent more precisely. When we specify that:

A(agent).sex!G(gender)

We are saying that an agent has exactly one gender. This exclusion information is available to the typechecker which can rule out at initialisation time any piece of code which assumes that an agent could have multiple genders.

Some modern logics are starting to add the ability to specify uniqueness properties of predicates [6]. But they treat uniqueness properties as metalinguistic predicates. EL is the first language to treat exclusion as a first-class element of the language.
Simpler Postconditions

action move(A, X, Y)
preconditions
  A.at!X
postconditions
  add A.at!Y
Simpler Queries

\[ \text{Married}(\text{Bride}, \text{Groom}, \text{Place}, \text{Time}, \text{Official}) \]

Who is Jill married to?

\[ (\exists g, p, t, f) \text{ Married}(\text{Jill}, g, p, t, f) \]

\[ \text{Married. Jill} \]
Exclusion is Typing Information

- A(agent).sex!G(gender)
- brown.sex.male
- Bad typing in brown.sex.male in line 65
- The first problem appears to be with “male”
Improvements to Praxis
Exclusion Logic

\[ E ::= S \mid S.E \mid S!E \]

\[ C ::= E \mid \neg C \mid C \land C \]
Extended Exclusion Logic

\[ E ::= T \mid T.E \mid T!E \mid E \land E \]

\[ A.(B \land C) = A.B \land A.C \]

\[ A!(B \land C) \neq A!B \land A!C \]
Extended Exclusion Logic

\[ A. (B \land C) \models A. (C \land B) \]

\[ A. (B.D \land C) \models A. (B \land C) \]

\[ A. (B \land C) \models A. B \land A. C \]

\[ A!(B \land C) \models A!(C \land B) \]

\[ A!(B.D \land C) \models A!(B \land C) \]

\[ A!(B \land C) \models \neg A!B \land \neg A!C \]
Improving Praxis

• Data abstraction
• Hindley-Milner type system
Compiling Praxis

• Warren Abstract Machine?
• Or Mercury-style compilation?
  – Explicit mode declarations for predicates
    • append(in, in, out)
    • append(out, out, in)
  – Separate procedures generated for each mode declaration