Integrating Dynamic Aspects into Deductive Object-Oriented Databases

Wolfgang May

Christian Schlepphorst

Georg Lausen

Institut für Informatik

Universität Freiburg

Germany

June 26, 1997

Overview

- The Roles of States in Modeling
- Theoretical issues:
  Declarative and Operational semantics
- Implementation in F-Logic

Active + Deductive Database Systems

Deductive Rules: Derive and express knowledge *inside* a state.

Active Rules: Derive and express actions (additional to the user's interaction) leading from one state to another.

Representation of the dynamic aspect:

Snapshot Database: Represents one state at a time.

History Database: Contains knowledge about past states and actions.

- (complex) event detection
- object reuse
- garbage collection

Object-Oriented Model

- is-a atoms: o:c
  relational encoding: isa(o,c)
- subclass atoms: c::d
  relational encoding: subcl(o,c)
- Method applications to objects:
  o[m→v] (scalar)
  o[m↠v] (multivalued)
  analogous with arguments: o[m@(x_1,...,x_n)→v].
  inheritable:
  o[m•→v]
  o[m•↠v]

  relational encoding:
  method_appl_sc(o,m,v), method_appl_mvd(o,m,v)
- path expressions: $o.m \equiv o' : o[m{\rightarrow}o']$
- Inheritance
- Transitivity of subclass hierarchy

Representation of States

Relational Model

Reification: $r(x_1, \ldots, x_n) \rightsquigarrow r(s, x_1, \ldots, x_n)$.

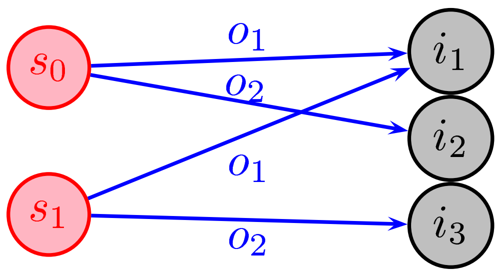
Other Approaches in OODB's

Versioning of objects.

The Roles of States

States as objects

Focus: computation sequence.
Every state s is an object. Abstract objects o act on them as methods, addressing the instance i corresponding to object o in state s.

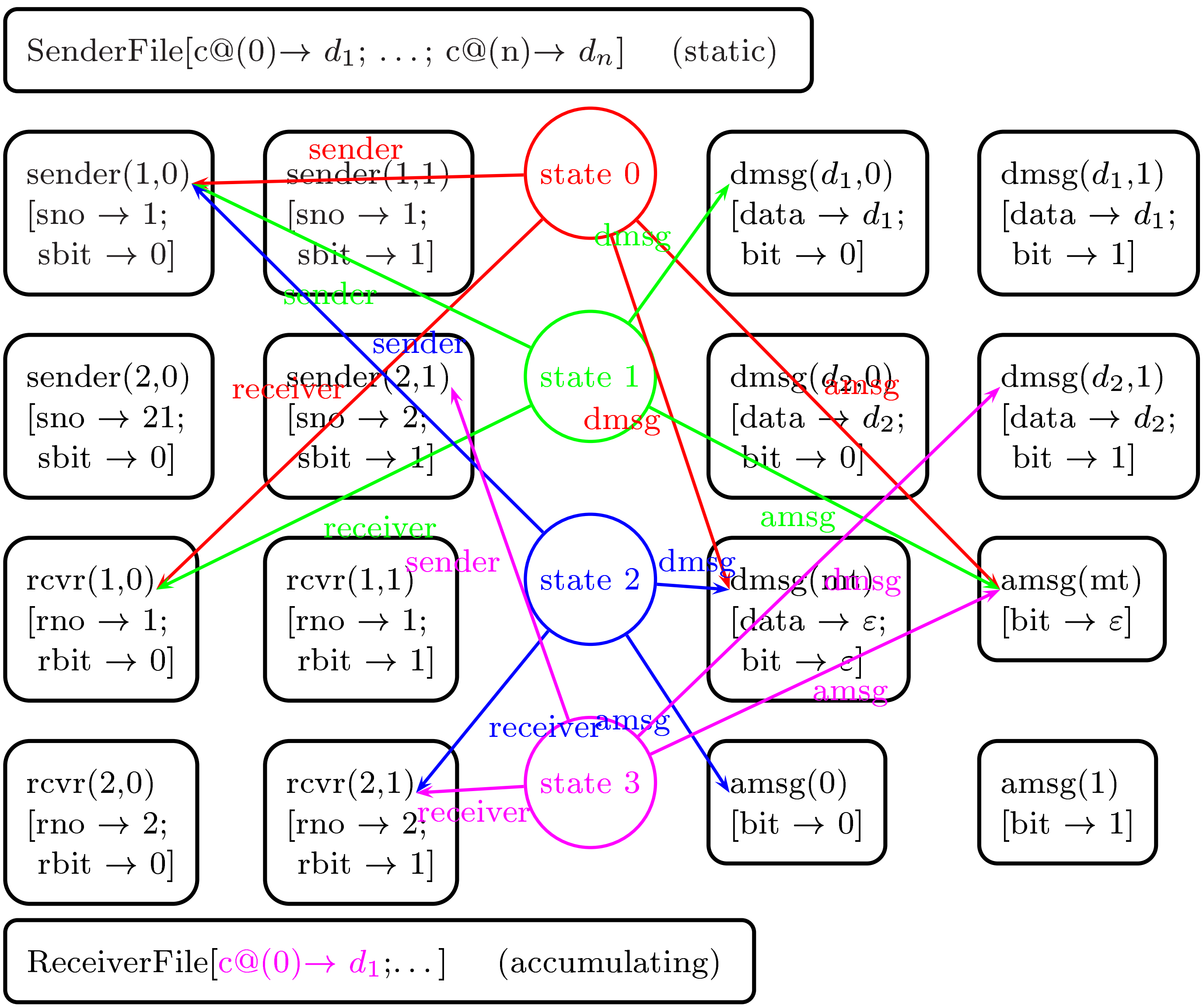


Syntax: s.o[m $\rightarrow$ v]

- Allows comparison of states (Deep Equality) to detect cycles.
- Application: Subordinate internal computations of the database system.

Example: Alternating Bit Protocol

- Sender and receiver file
- Sender and receiver unit
- Data Messages: data + control bit
- Ack Messages: control bit

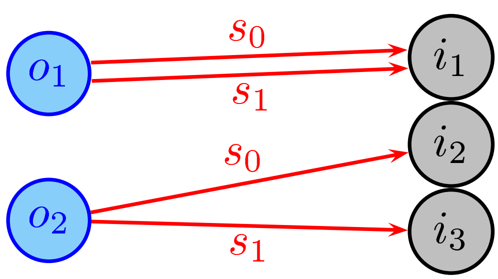


A State

S[sender → sender(sn,sb)[sno → sn; sbit → sb];
receiver → rcvr(rn,rb)[rno → rn; rbit → rb];
dmsg → dmsg(d,b)[data → d; bit → b];
amsg → amsg(b')[bit → b']]

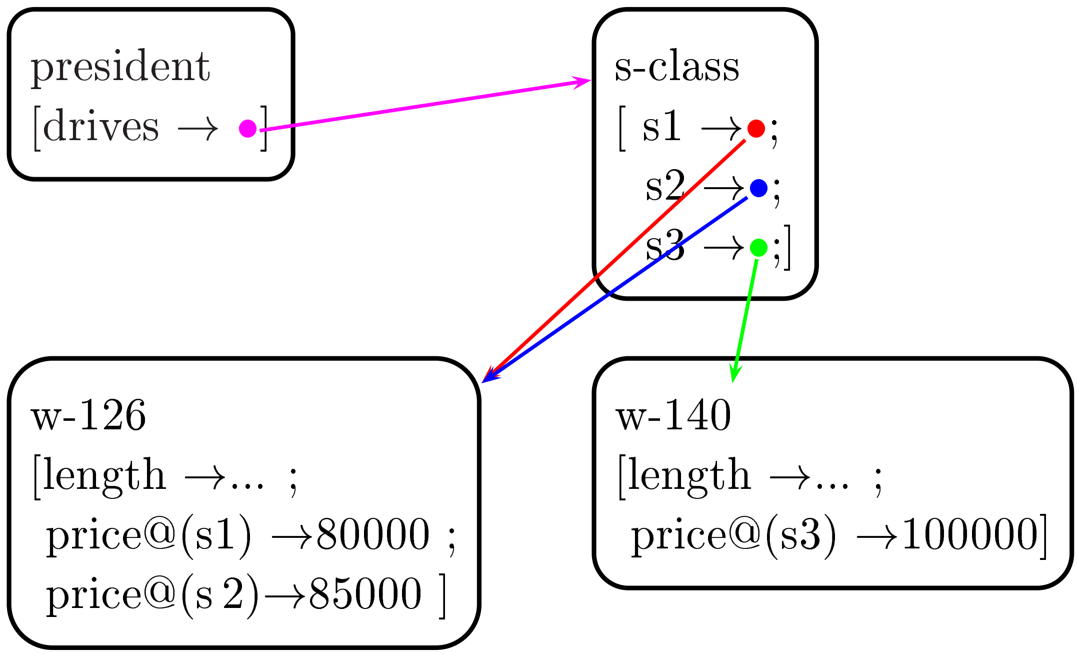
Dynamic objects

Objects changing their behavior only from time to time. For an abstract object o, a state s is a method, giving the instance of o corresponding to state s.



Syntax: o.s[m $\rightarrow$ v]

Example: Enterprise's car fleet

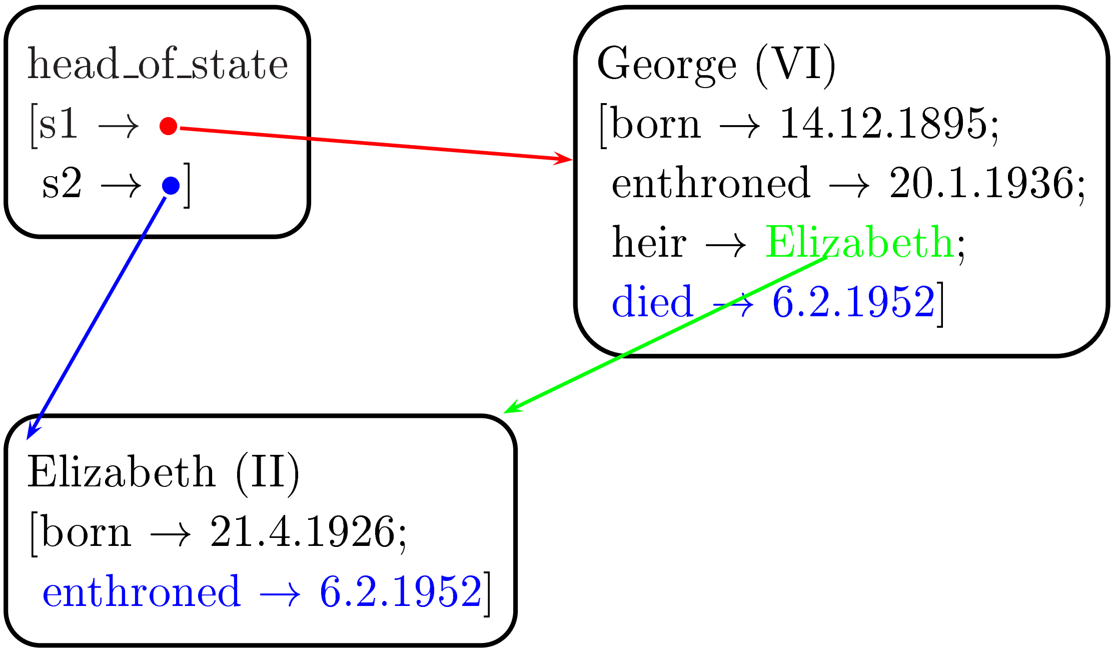


Query:

new_car_for_president_costs(S,Z) ←

president[drives → X], X[S → Y], Y[price@(S) → Z].

Example: English Administration



Event: dies(person) in state s.

$$\mathsf{X[dies \rightarrow D] \leftarrow dies(X,S),\ date(S) = D.}$$

Event creates a new state:

$$\mathsf{state(S{+}1) \leftarrow dies(X,S).}$$

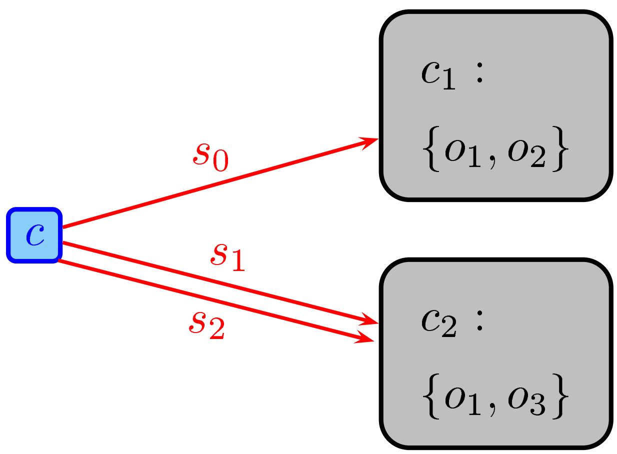
Frame rules:

$$\mathsf{head\_of\_state[S{+}1 \rightarrow Y],\ Y[enthroned \rightarrow D] \leftarrow}$$
$$\mathsf{head\_of\_state[S \rightarrow X],\ date(S) = D,\ X[heir \rightarrow Y],\ dies(X,S).}$$

$$\mathsf{head\_of\_state[S{+}1 \rightarrow X] \leftarrow}$$
$$\mathsf{head\_of\_state[S \rightarrow X],\ \neg\ dies(X,S),\ state(S{+}1).}$$

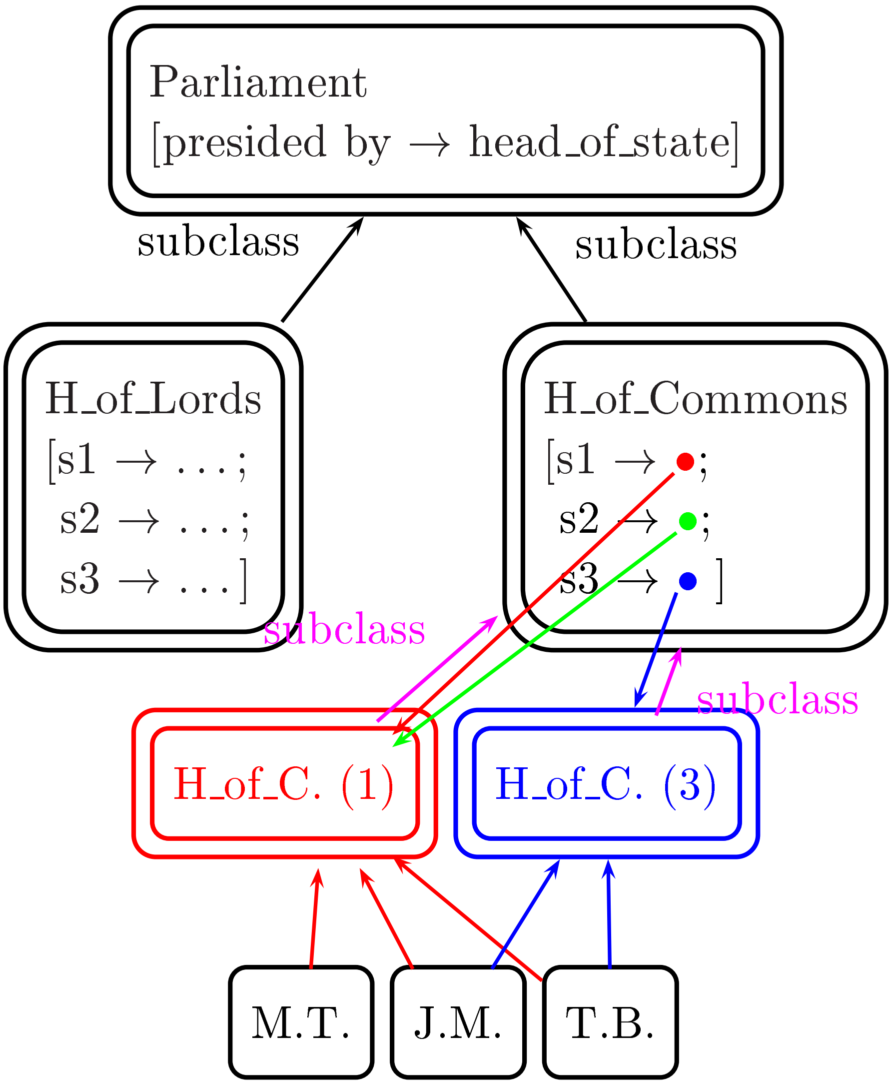
Dynamic classes

Classes which change their extension or some inheritable properties.
Closely related with dynamic objects: For an abstract class c, a state s is a method, giving the instance c_s of the class c in this state.



Syntax: o:c.s

Example: English Administration Revisited



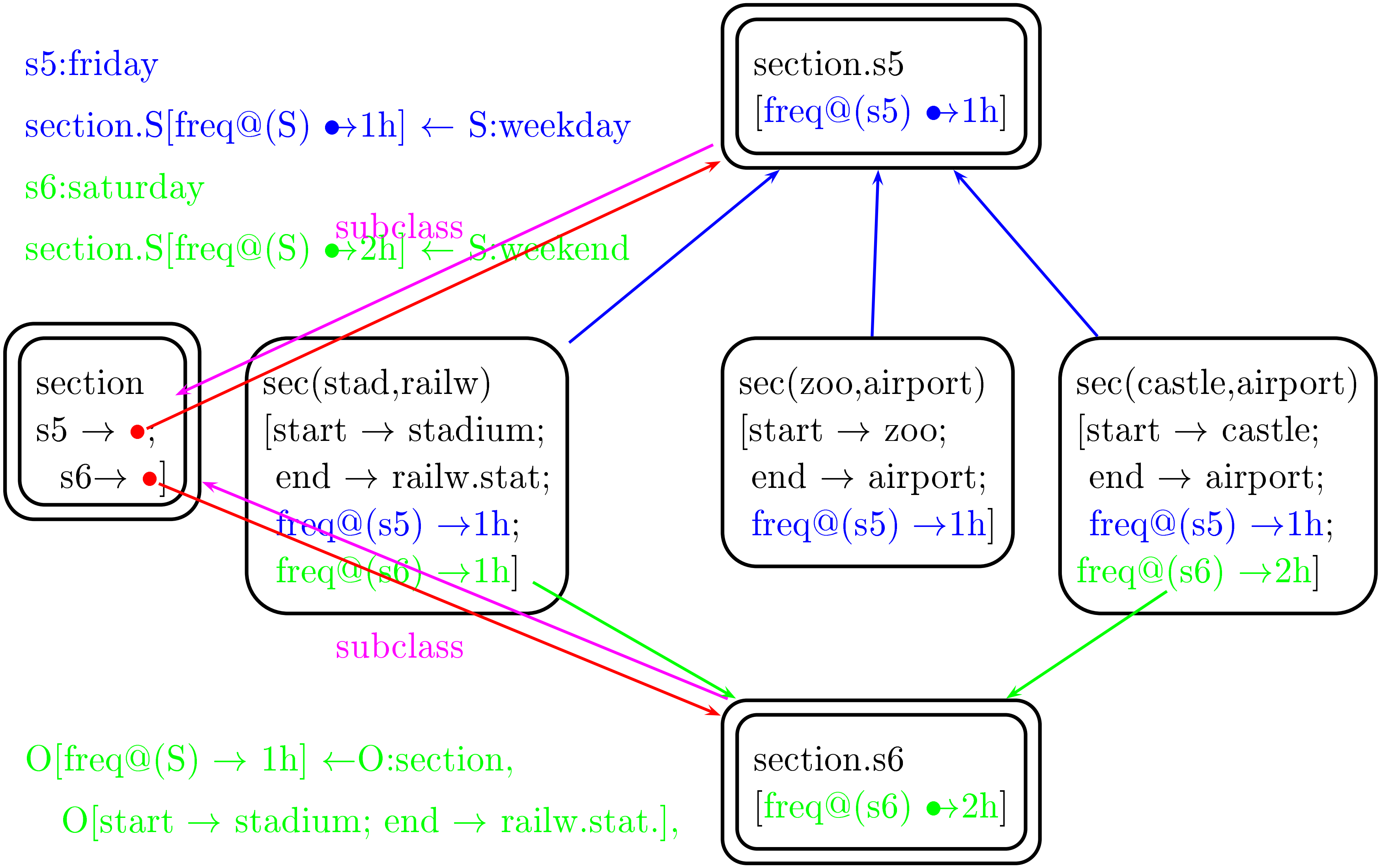
Example: Tram net

s5:friday

section.S[freq@(S) •→1h] ← S:weekday

s6:saturday

section.S[freq@(S) •→2h] ← S:weekend



O[freq@(S) → 1h] ←O:section,

O[start → stadium; end → railw.stat.],

S:saturday

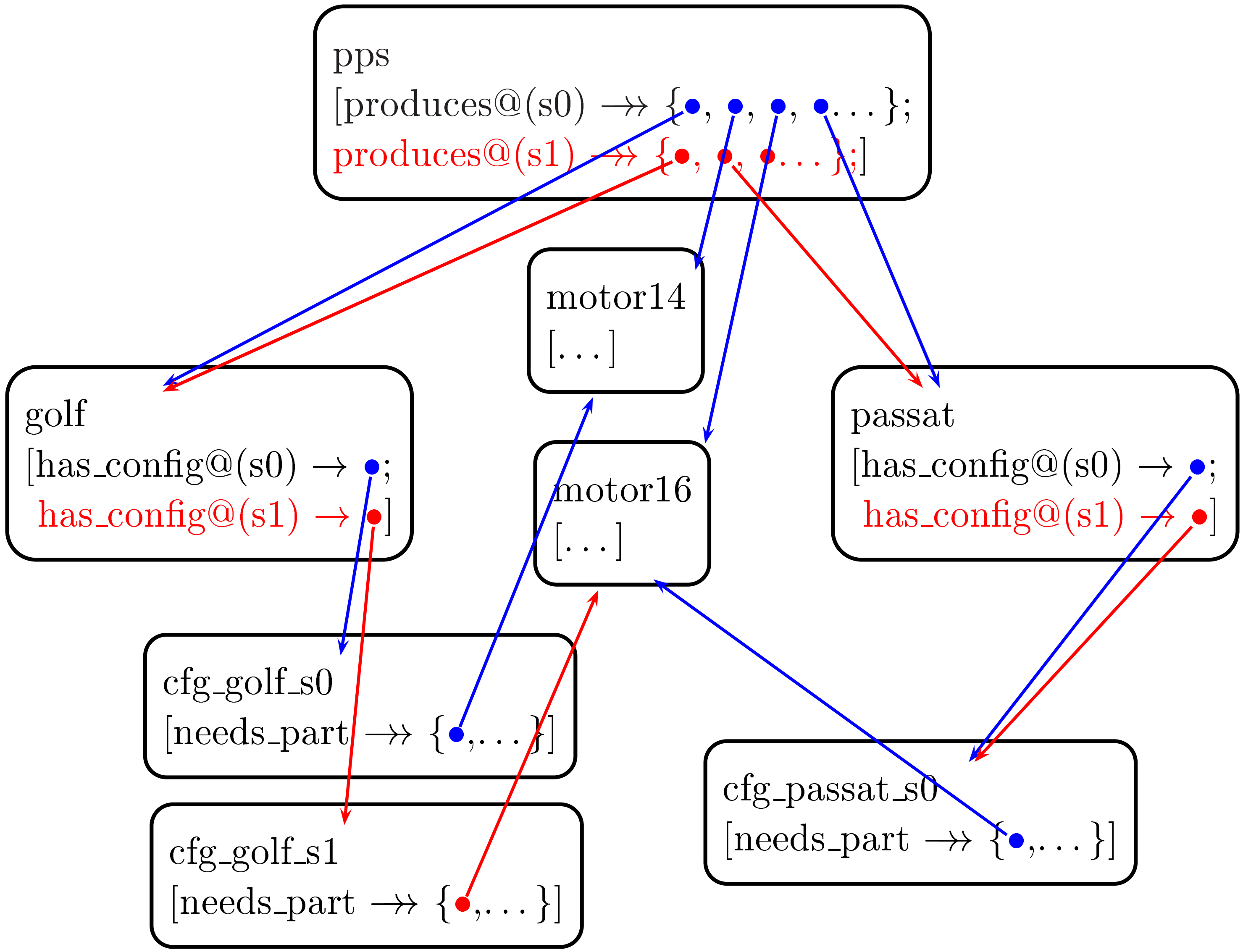
Dynamic methods

Object where only parts of its behavior are changing. For an object o, a state s is an additional argument of a method m, giving the value of the method in this state.

$$
\boxed{\begin{array}{l}
o\,[\,m_1 \rightarrow x, \\
\quad m_2@(s_1) \rightarrow y, \\
\quad m_2@(s_2) \rightarrow y, \\
\quad m_2@(s_3) \rightarrow z\,]
\end{array}}
$$

Syntax: o[m@(s,...) → v]

Example: Production Planning System



Requirements

- entities act simultaneously as objects, classes, and methods.
- variables occur at arbitrary positions of rules, standing for arbitrary entities.
- "states as objects", "dynamic objects", and "dynamic classes" require variables to appear at method positions.
  "states as objects": objects are methods to states. Variables at object positions become variables at method positions.
  "dynamic objects" and "dynamic classes": states appear as methods, thus state variables appear as variables at method positions.
- object creation, anonymous objects, and anonymous classes.

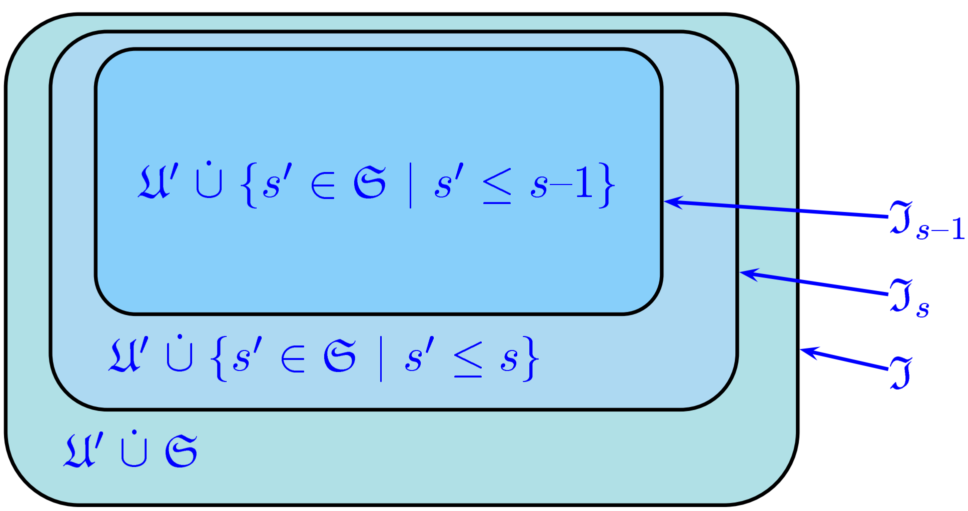
State Space

Given a single-state framework: $\mathfrak{X}$ PL1, F-Logic

Definition 1 (State-$\mathfrak{X}$-Structure)
A *State-$\mathfrak{X}$-structure* is an $\mathfrak{X}$-structure with universe
$\mathfrak{U} = \mathfrak{U}' \dot{\cup} \mathfrak{S}$;
$\mathfrak{U}'$ a classical universe, $\mathfrak{S}$ the state space.

- acyclic ordering on $\mathfrak{S}$, $(\mathbb{N}, <)$
- notions of "next" state(s), "earlier", and "later" expressed by atoms $S > T$ or $S = T{+}n$.

Definition 2
$\mathfrak{I}$ a State-$\mathfrak{X}$-structure with universe $\mathfrak{U}' \cup \mathfrak{S}$, $s \in \mathfrak{S}$.
The part which is *known in state* s, denoted by $\mathfrak{I}_{\leq s}$ is obtained by restricting $\mathfrak{I}$ to the universe
$\mathfrak{U}' \cup \{s' \in \mathfrak{S} \mid s' \leq s\}$



Database Evolution

- Initial database D,
- Sequence $E_0, E_1, \ldots$ of sets of events, represented by ground $\mathfrak{X}$-atoms.

Example:
event *move x to y* occurring in state $s \rightsquigarrow$ x[moveTo@(s)→y].

Definition 3 A State-$\mathfrak{X}$-structure $\mathfrak{I}$ is a model of P, D, and $E_0, E_1, \ldots, E_n$ (as above) if

$$\mathfrak{I} \models P \cup D \cup E_0 \cup \ldots \cup E_n \quad .$$

Declarative semantics:

$$\mathfrak{D}_{\mathfrak{X}}(P \cup D \cup E_0 \cup \ldots \cup E_n)$$

inflationary, stratified, well-founded

Operational Semantics

Computing states *successively*:

$$\begin{aligned} D_0 &:= \mathfrak{D}(P \cup D), \\ D_1 &:= \mathfrak{D}(P \cup D_0 \cup E_0), \\ D_2 &:= \mathfrak{D}(P \cup D_1 \cup E_1), \\ D_3 &:= \mathfrak{D}(P \cup D_2 \cup E_2), \\ &\vdots \end{aligned}$$

With

$$\mathfrak{I} := \mathfrak{D}(P \cup D \cup E_0 \cup \ldots \cup E_n)$$

$$\begin{aligned} D_0 &\sim \mathfrak{I}_{\leq 0} \\ D_1 &\sim \mathfrak{I}_{\leq 1} \\ D_2 &\sim \mathfrak{I}_{\leq 2} \\ &\vdots \end{aligned}$$

Definition 4 A State-$\mathfrak{X}$-program P is *incremental* if for every D, $E_0, \ldots, E_n$, with

$$\mathfrak{I} := \mathfrak{D}(P \cup D \cup E_0 \cup \ldots \cup E_n) \quad ,$$

for every $s \in \mathbb{N}$, the following holds:

$$\mathfrak{I}_{\leq s+1} = \mathfrak{D}(P \cup \mathfrak{I}_{\leq s} \cup E_s) \ .$$

Rules

Rules are e.g. of the form

$$head(S) \leftarrow \; body(S,T), \; \mathsf{S} = \mathsf{T} + \mathsf{1} \quad .$$

Definition 5

State-ground instance $\beta(r)$ of a State-$\mathfrak{X}$-rule r: replace all terms denoting states by some elements of $\mathfrak{S}$

$$\beta := \{s_1/n_1, \ldots, s_k/n_k\}$$

State-ground model: state-ground instance $\beta(r)$ which satisfies the requirements imposed by the rule for states/natural numbers.

Example: For a rule

$$\mathsf{h(t)} \leftarrow \ldots, \mathsf{s{:}state}, \mathsf{t{:}state}, \mathsf{t{>}s}, \ldots$$

every β: $(s,t) \rightarrow \mathbb{N}^2$ is a state-ground instance, but only those β: $(s,t) \rightarrow \{(n,m) \in \mathbb{N}^2 \mid n < m\}$ are state-ground models.

Types of Rules (informally)

An State-$\mathfrak{X}$ rule $\mathsf{r} = \mathsf{h} \leftarrow \mathsf{b}$ is

- *global* if there occurs no state term in it.
- *local* if there is at least one state term S occurring in h, and for every state-ground model β of $h \wedge b$ and all other state terms T_i occurring in r, $\beta(T_i) = \beta(S)$.
- *progressive* if for every state-ground model β of $h \wedge b$, there is a state term S occurring in h s.t. $\beta(S) \geq \beta(T_i)$ for all other state terms T_i occurring in r.
- *collective* if h contains no state term, but b contains one or more state terms.
- *backwards* if there is a state-ground model β and a state term S occurring in b such that for every state term T occuring in h, $\beta(T) < \beta(S)$.

Example 1

1-progressive:

Frame rules for methods of dynamic objects, i.e., objects o which have an individual instance $o.s$ for every state s.

O.T[M→X] ← S:state, T:state, O.S[M→X], T= S + 1, not O.change@(S,M)[ ].

O.T[M→Q] ← S:state, T:state, T= S + 1, O[change@(S,M)→Q].

collective:

P[hasTalkedTo↠X] ← P[talksWith@(S)→X], S:state.

Theorem 1 *Every program P containing only progressive rules and not deriving any facts about a state s+1 if there are no events in state s is incremental.*

⇝ Controlling state-generation

Meta-Knowledge

Collective Rules:

P[hasTalkedTo$\twoheadrightarrow$X] $\leftarrow$ P[talksWith@(S)$\rightarrow$X], S:state.

Definition 6 A State-$\mathfrak{X}$-program P is *incremental modulo a set $\mathfrak{M}$ of ground atoms* if for every D, $E_0, \ldots, E_n$ as above, for

$$\mathfrak{I} := \mathfrak{D}(P \cup D \cup E'_0 \cup \ldots \cup E'_n) \ ,$$

for every $s \in \mathbb{N}$, the following holds:

$$\begin{aligned} \mathfrak{I}_{\leq s+1} \setminus \mathfrak{M} &= (\mathfrak{D}(P \cup \mathfrak{I}_{\leq s} \cup E'_s)) \setminus \mathfrak{M} \\ &= (\mathfrak{D}(P \cup \mathfrak{I}_{\leq s} \setminus \mathfrak{M} \cup E'_s)) \setminus \mathfrak{M} \ . \end{aligned}$$

Theorem 2 *Let P be a State-$\mathfrak{X}$-program containing only global, progressive, and collective rules and $\mathfrak{M}$ a set of ground atoms. Then, P is incremental modulo $\mathfrak{M}$ if $\mathfrak{M}$ contains all ground atoms unifying with heads of collective rules and no atom from $\mathfrak{M}$ is used to derive any state-dependent information.*

Definition 7 For a State-$\mathfrak{X}$-program P which is incremental modulo a set $\mathfrak{M}$ of ground atoms, a database D, sets $E_0, E_2, \ldots$ of events, and

$$\mathfrak{I} := \mathfrak{D}(P \cup D \cup E_0 \cup E_1 \cup \ldots) \quad ,$$

the *operational semantics* is defined as the sequence

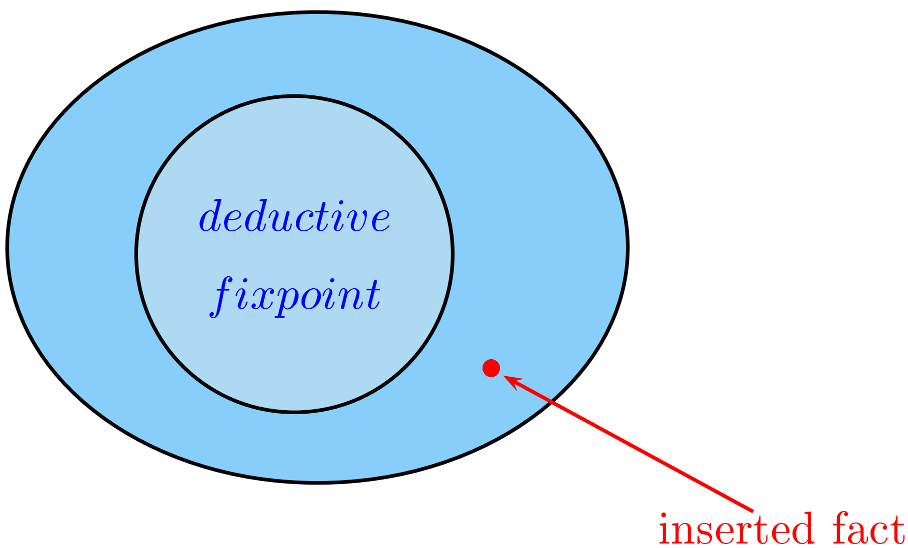
$$\mathfrak{I}_{\leq 0} \setminus \mathfrak{M},\ \mathfrak{I}_{\leq 1} \setminus \mathfrak{M},\ \ \ldots\ .$$

In this case, the database in state $s{+}1$ can be computed from the database D_s and a set of events E_s as

$$D_{s+1} = \mathfrak{D}(P \cup D_s \cup E_s)$$

Programming Explicit States in F-Logic

- Inflationary semantics,
- user-defined stratification (fixed number of predefined strata),
- Trigger mechanism: Insert atoms into the database after reaching a deductive fixpoint (used for nonmonotonic inheritance).



The State Sequence

Every state passes through several stages, e.g.

- Computing the EDB
- Computing the IDB
- User Interaction
- Computing necessary changes

Sequence of deductive fixpoint computations via inheritable methods:

(A) inheritable methods:

stage1::state[ready_edb•→true].

stage2::state[ready_idb•→true].

stage3::state.

stage4::state[ready_changes•→true].

0:stage1.

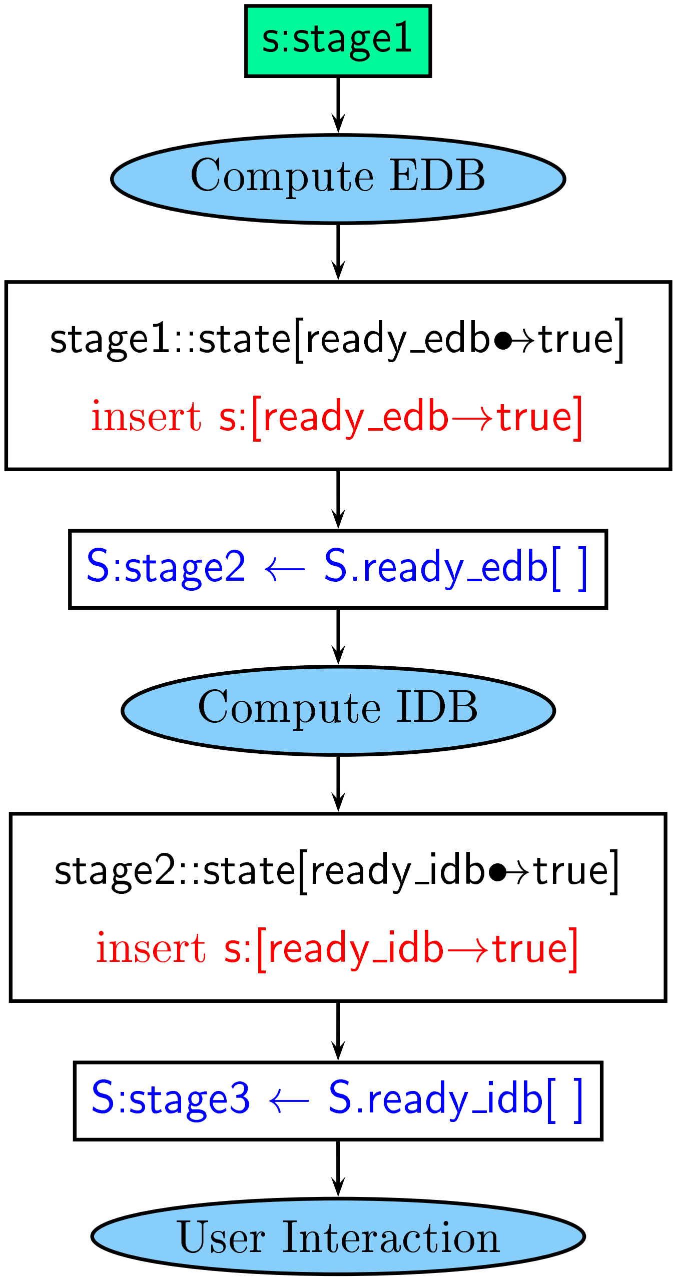
(B) the stage sequence:

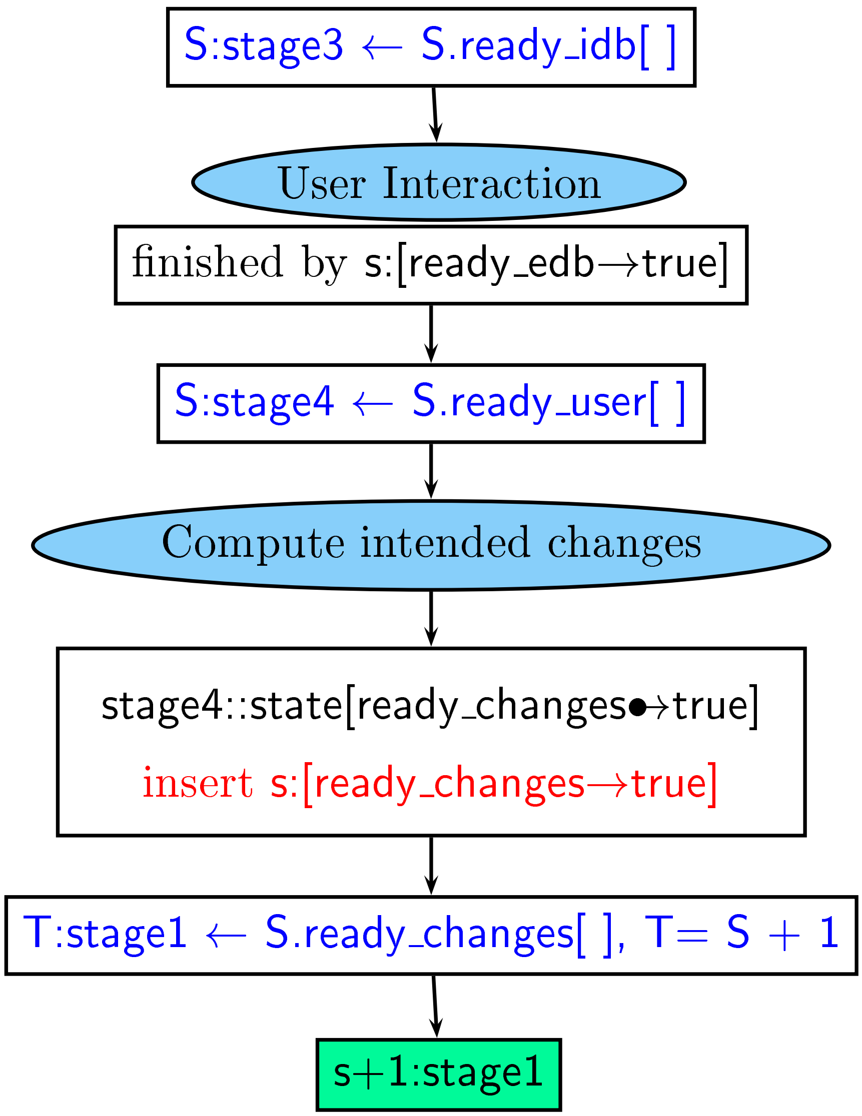
S:stage2 ← S.ready_edb[ ].

S:stage3 ← S.ready_idb[ ].

S:stage4 ← S:stage3, S.ready_user[ ].

T:stage1 ← S.ready_changes[ ], T= S + 1.


s:stage1
Compute EDB
stage1::state[ready_edb•→true]
insert s:[ready_edb→true]
S:stage2 ← S.ready_edb[ ]
Compute IDB
stage2::state[ready_idb•→true]
insert s:[ready_idb→true]
S:stage3 ← S.ready_idb[ ]
User Interaction


S:stage3 ← S.ready_idb[ ]
User Interaction
finished by s:[ready_edb→true]
S:stage4 ← S.ready_user[ ]
Compute intended changes
stage4::state[ready_changes•→true]
insert s:[ready_changes→true]
T:stage1 ← S.ready_changes[ ], T= S + 1
s+1:stage1

Conclusion

- OO: flexibility in modeling
- F-Logic: flexible syntax
- generic frame rules
- declarative + operational semantics
- specification = implementation
- meta-reasoning *about* database behavior
- in the same language