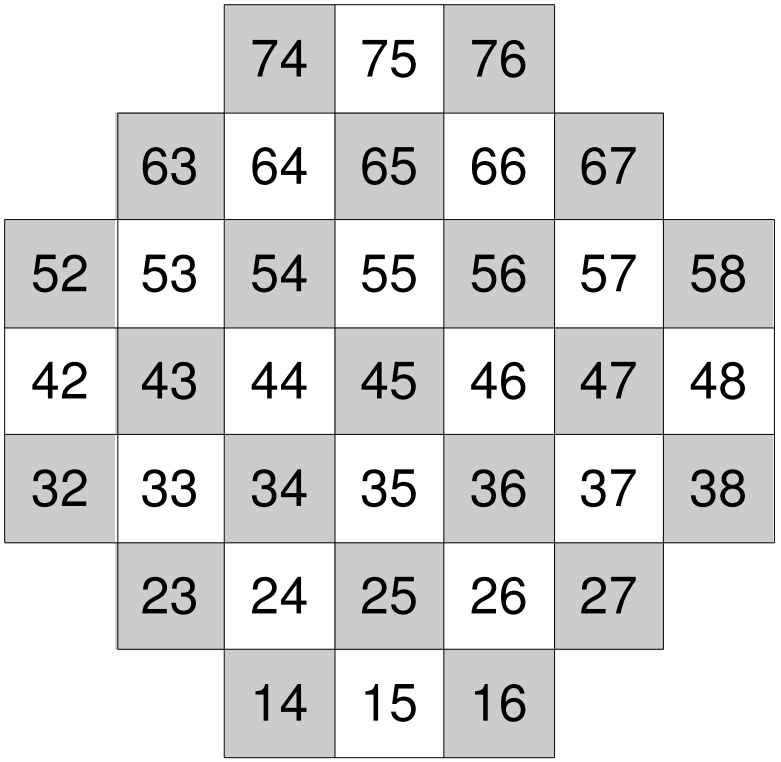
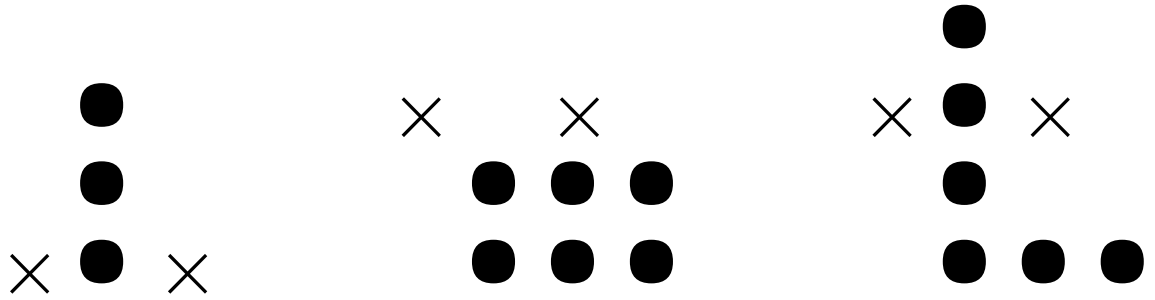
Declarative Action Programs

- Agent Logic Programs
- Semantics
- Fast State Update in Prolog
- Wumpus, Sensing, and Constraint Logic Programming

Example: Peg Solitaire



An Agent Logic Program (ALP)



3-pattern 6-pattern L-pattern

```
is_pattern([C,X1,X2,X3]) :- ?(peg(C)),
                            X1 = C+1, X2 = X1+10, X3 = X1+20,
                            ?(peg(X1) and peg(X2) and peg(X3)).
is_pattern([C,X1,X2,X3]) :- ?(peg(C)),
                            X1 = C-1, X2 = X1+10, X3 = X1+20,
                            ?(peg(X1) and peg(X2) and peg(X3)).
...
```

ALP for Peg Solitaire (Cont'd)

```
board_solved :- ?(peg(45) and forall(X, X=45 or not peg(X))).

pattern_solved(P) :- member(Y,P), do(jump(U,V,W)),
                     pattern_solved(P).

pattern_solved([Catalyst|P]) :- ?(peg(Catalyst)), empty(P).

empty([]).
empty([X|L]) :- ?(not peg(X)), empty(L).

strategy :- board_solved.
strategy :- is_pattern(P), pattern_solved(P), strategy.
```

ALP Syntax

- Terms of sort FLUENT and ACTION
- If `p?` is an *n*-ary relation and $\texttt{t}_1, \ldots, \texttt{t}_n$ are terms, then $\texttt{p}(\texttt{t}_1, \ldots, \texttt{t}_n)$ is a program atom
- If *a* is an ACTION term, then `do`(*a*) is a program atom
- If ϕ is a state property composed of FLUENT terms, then `?`(ϕ) is a program atom
- If $\texttt{H}, \texttt{B}_1, \ldots, \texttt{B}_n$ are program atoms, then $\texttt{H} \texttt{:-} \texttt{B}_1, \ldots, \texttt{B}_n$ is an ALP clause
- An ALP is a finite set of ALP clauses
- An ALP query is a finite sequence of program atoms

Declarative Semantics

Clause and Query Expansion

For a clause `H:-B1,...,Bn` let s_1, ..., s_{n+1} be a sequence of TIME variables.

- For i = 1, ..., n

  - If B_i is of the form `p(t1,...,tm)`, it is expanded to $p(t_1, ..., t_m, s_i, s_{i+1})$;

  - If B_i is of the form `do`(a), it is expanded to $Poss(a, s_i, s_{i+1})$;

  - If B_i is of the form ?(ϕ), it is expanded to $\phi[s_i]$ and $s_{i+1} = s_i$.

- The head `H = p(t1,...,tm)` is expanded to $p(t_1, ..., t_m, s_1, s_{n+1})$.

- "`:-`" is replaced by "$\subset$" and "`,`" is replaced by "$\wedge$".

- A query is expanded like the body of a clause but with s_1 set to the initial TIME constant.

Example

```
strategy :- board_solved.
strategy :- is_pattern(P), pattern_solved(P), strategy.
```

is expanded to

$Strategy(s_1, s_2) \subset BoardSolved(s_1, s_2)$

$Strategy(s_1, s_4) \subset IsPattern(p, s_1, s_2) \wedge PatternSolved(p, s_2, s_3) \wedge Strategy(s_3, s_4)$

```
board_solved :- ?(peg(45) and forall(X, X=45 or not peg(X))).
```

is expanded to

$BoardSolved(s, s) \subset Holds(Peg(45), s) \wedge (\forall x)\,(x = 45 \vee \neg Holds(Peg(x), s))$

Importance of Atom Ordering

```
p :- ?(f), do(a).
```

is expanded to

$P(s_1, s_2) \subset Holds(F, s_1) \wedge Poss(A, s_1, s_2)$

```
p :- do(a), ?(f).
```

is expanded to

$P(s_1, s_2) \subset Poss(A, s_1, s_2) \wedge Holds(F, s_2)$

Requirements for Time Structure (1)

Action execution in ALPs is strictly sequential.

Therefore the time structure must satisfy

$$Poss(a, s, t) \supset s < t$$

$$Poss(a, s, t) \wedge Poss(a', s', t') \supset (t < t' \supset t \leq s') \wedge (t = t' \supset a = a' \wedge s = s')$$

Note: This is always true when situations are used.

Requirements for Time Structure (2)

There are no “gaps” in between action executions.

Therefore the time structure must satisfy

$$(\exists t)\ Poss(a, s, t) \wedge (\forall f)\ (Holds(f, s) \equiv Holds(f, s')) \supset (\exists t')\ Poss(a, s', t')$$

Note: This is always true when situations are used.

Example for “Gaps”

`p :- do(a), do(b)` is expanded to

$P(s_1, s_3) \subset Poss(A, s_1, s_2) \wedge Poss(B, s_2, s_3)$

With the precondition axioms

$$Poss(A, s, t) \equiv s = 0 \wedge t = 1$$

$$Poss(B, s, t) \equiv s = 2 \wedge t = 3$$

there is no derivation for query `p`

Procedural Semantics

Derivation Rules (1)

- Standard atoms:

$$\frac{Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n}{\left(B_1 \wedge \ldots \wedge B_m \wedge Q_2 \wedge \ldots \wedge Q_n\right)\theta}$$

where Q_1 is a literal defined in the ALP and $H \subset B_1 \wedge \ldots \wedge B_m$ is the expansion of a clause in the ALP such that $Q_1\theta = H\theta$, for some substitution θ

Derivation Rules (2)

- Actions:

$$\frac{Poss(a, s, t) \wedge Q_2 \wedge ... \wedge Q_n}{(Q_2 \wedge ... \wedge Q_n)\theta}$$

where $\Sigma \models Poss(a, s, t)\theta$ with substitution θ on the variables in $Poss(a, s, t)$

- Tests:

$$\frac{\phi[s] \wedge Q_2 \wedge ... \wedge Q_n}{(Q_2 \wedge ... \wedge Q_n)\theta}$$

where $\Sigma \models \phi[s]\theta$ with substitution θ on the variables in ϕ

Example

```
office(alice, 101).
office(bob,   102).
deliver :- ?(has_package_for(P)), office(P,R), do(go(R)).
```

The expansion is

$Office(Alice, 101, s, s)$

$Office(Bob, 102, s, s)$

$Deliver(s_1, s_3) \subset Holds(HasPackageFor(p), s_1) \wedge Office(p, r, s_1, s_2) \wedge$
$Poss(Go(r), s_2, s_3)$

A Successful Derivation

$$\frac{\dfrac{\dfrac{\dfrac{Deliver(S_0, s)}{Holds(HasPackageFor(p), S_0) \wedge Office(p, r, S_0, s_2) \wedge Poss(Go(r), s_2, s)}}{Office(Bob, r, S_0, s_2) \wedge Poss(Go(r), s_2, s)}}{Poss(Go(102), S_0, s)}}{\square}$$

where we assume the precondition axiom $Poss(Go(r), s, t) \equiv t = Do(Go(r), s)$

Soundness

Theorem

If there exists a successful derivation for Q with computed answer θ then the domain axiomatization and the expanded program together entail $Q\theta$, provided the reasoner for the domain axiomatization is correct.

Incompleteness

```
p :- ?(f).
p :- ?(not f).
```

The expansion

$P(s, s) \subset Holds(F, s)$

$P(s, s) \subset \neg Holds(F, s)$

entails $(\exists s)\ P(S_0, s)$ but there is no derivation for query `p`.

ALPs in Prolog:

Fast State Update

Piece-Wise Update vs. State Update

A problem with the direct use of successor state axioms in GOLOG – or GDL effect axioms – is that update is computed piecewise for each state property.

```
next(peg(W)) :- does(P,jump(U,V,W)).
next(peg(X)) :- true(peg(X)),
                does(P,jump(U,V,W)),
                distinct(X,U), distinct(X,V).
```

It is often more efficient to compute entire state updates.

```
   [peg(14),...,peg(43),peg(44),peg(46),...]
→ [peg(14),...,peg(45),peg(46),...]
```

A State-Based Description Language

A triple (F, $\circ$, $\emptyset$) is a fluent calculus state signature if

- F finite, non-empty set of function symbols into sort FLUENT
- $\circ$: STATE $\times$ STATE $\mapsto$ STATE
- $\emptyset$: STATE

A fluent is a term of sort FLUENT.

A state is a term of sort STATE, with FLUENT being a sub-sort of STATE.

Foundational Axioms

$$Holds(f : \text{FLUENT}, z : \text{STATE}) \stackrel{\text{def}}{=} (\exists z' : \text{STATE})\ z = f \circ z'$$

The foundational axioms Σ are

1. Associativity and commutativity

$$(z_1 \circ z_2) \circ z_3 = z_1 \circ (z_2 \circ z_3) \qquad z_1 \circ z_2 = z_2 \circ z_1$$

2. Empty state axiom $\neg Holds(f, \emptyset)$

3. Irreducibility $Holds(f_1, f) \supset f_1 = f$

4. Decomposition $Holds(f, z_1 \circ z_2) \supset Holds(f, z_1) \vee Holds(f, z_2)$

5. State Equality $(\forall f)(Holds(f, z_1) \equiv Holds(f, z_2)) \supset z_1 = z_2$

6. State Existence $(\forall P)(\exists z)(\forall f)(Holds(f, z) \equiv P(f))$

Finite States

A finite state τ is a term $f_1 \circ ... \circ f_n$ such that each $f_i (1 \leq i \leq n)$ is a fluent ($n \geq 0$).

If $n = 0$, then τ is $\emptyset$. A ground state is a finite state without variables.

Proposition

Let $\tau = f_1 \circ ... \circ f_n$ be a finite state ($n \geq 0$), then

$$\sum \models Holds(f, \tau) \equiv \bigvee_{i=1}^{n} (f_i = f)$$

Fluent Addition and Removal

- $z_1 + f = z_2 \stackrel{\text{def}}{=} z_2 = z_1 \circ f$
- $z_1 - f = z_2 \stackrel{\text{def}}{=} (z_2 = z_1 \vee z_2 + f = z_1) \wedge \neg \mathit{Holds}(f, z_2)$

$z = \text{Peg}(14) \circ \ldots \circ \text{Peg}(43) \circ \text{Peg}(44) \circ \text{Peg}(46) \circ \ldots - \text{Peg}(43) - \text{Peg}(44) + \text{Peg}(45)$

$\rightarrow z = \text{Peg}(14) \circ \ldots \circ \text{Peg}(45) \circ \text{Peg}(46) \circ \ldots$

Fluent Calculus Signatures

A tuple S $\cup$ ⟨A, S_0, *Do*, *State*, *Poss*⟩ is a fluent calculus signature if

- S state signature
- A finite, non-empty set of function symbols into sort ACTION

- S_0 : SIT
- *Do* : ACTION $\times$ SIT $\mapsto$ SIT
- *State* : SIT $\mapsto$ STATE
- *Poss* : ACTION $\times$ STATE

State and Situation Formulae

$Holds(f,s) \stackrel{\text{def}}{=} Holds(f,State(s))$

Definition

A state formula $\Delta(z)$ is a first-order formula with free state variable z and without any occurrence of states other than in expressions of the form $Holds(f, z)$, and without actions or situations.

If $\Delta(z)$ is a state formula and s a situation variable, then $\Delta(z)\{z/State(s)\}$ is a situation formula, written $\Delta(s)$.

Precondition Axioms

A precondition axiom for an action A is a formula

$$Poss(A(\vec{x}),z) \equiv \Pi(z)$$

where $\Pi(z)$ is a state formula with free variables among $\vec{x}$, z.

$$Poss(a,s) \stackrel{\text{def}}{=} Poss(a,State(s))$$

$$Poss(Jump(u,v,w),z) \equiv Holds(Peg(u),z) \wedge Holds(Peg(v),z) \wedge \neg Holds(Peg(w),z) \wedge IsBoardCell(w)$$

State Update Axioms

A state update axiom for an action A is a formula

$$Poss(A(\vec{x}),s) \supset (\exists \vec{y}_1)(\phi_1(s) \wedge State(Do(A(\vec{x}),s)) = State(s) - \vartheta_1^- + \vartheta_1^+)$$
$$\vee \ ... \ \vee$$
$$(\exists \vec{y}_n)(\phi_n(s) \wedge State(Do(A(\vec{x}),s)) = State(s) - \vartheta_n^- + \vartheta_n^+)$$

where

- $n \geq 1$
- for $i = 1, ..., n$,

  - $\phi_i(s)$ is situation formula with free variables among $\vec{x}, \vec{y}_i, s$

  - $\vartheta_i^+, \vartheta_i^-$ are finite states with variables among $\vec{x}, \vec{y}_i$

The terms $\vartheta_i^+, \vartheta_i^-$ are called, respectively, positive and negative effects of $A(\vec{x})$ under condition $\phi_i(s)$. Empty positive or negative effects are omitted.

State Update Axioms and the Frame Problem

Proposition

Under the foundational axioms it follows that

$$State(t) = State(s) - g_1 - ... - g_m + f_1 + ... + f_n$$

implies

$$Holds(f,t) \equiv [f = f_1 \vee ... \vee f = f_n] \vee [Holds(f,s) \wedge f \neq g_1 \wedge ... \wedge f \neq g_m]$$

Example: Peg Jump

$$Poss(Jump(u,v,w),s) \supset$$

$$State(Do(Jump(u,v,w),s) = State(s) - Peg(u) - Peg(v) + Peg(w)$$

State Update in Prolog

```
holds(F, Z) :- member(F,Z).

holds(F, [F|Z], Z).
holds(F, Z, [F1|Z2]) :- Z=[F1|Z1], F\==F1, holds(F, Z1, Z2).

minus(Z, [], Z).
minus(Z, [F|Fs], Z2) :- holds(F,Z,Z1), minus(Z1,Fs,Z2).
minus(Z, [F|Fs], Z2) :- not member(F,Z), minus(Z,Fs,Z2).

plus(Z, [], Z).
plus(Z, [F|Fs], Z2) :- not member(F, Z), plus([F|Z],Fs,Z2).
plus(Z, [F|Fs], Z2) :- member(F, Z), plus(Z,Fs,Z2).

update(Z1,ThetaP,ThetaN,Z2) :- minus(Z1,ThetaN,Z),
                               plus(Z,ThetaP,Z2).
```

Example

```
init(Z0) :- Z0 = [peg(14),...,peg(43),peg(44),peg(46),...].

state_update(Z1,jump(U,V,W),Z2) :-
      update(Z1,[peg(W)],[peg(U),peg(V)],Z2).


?- init(Z0), do(jump(43,44,45),Z0,Z1).

Z2 = [peg(45,peg(14),...,peg(46),...]
```

Online / Offline Execution

- Online execution of an ALP means that the agent actually performs each action and updates (progresses) its state

```
do(A,Z1,Z2) :- perform(A), state_update(Z1,A,Z2).
```

- Offline execution of an ALP means that the agent seraches for a successful sequence of actions (a situation) while progressing its state

```
do(Z1,S1,A,Z2,S2) :- state_update(Z1,A,Z2),
                     S2 = do(A,S1).
```

Example ALP (Online Execution; cf.Slide 4)

```
board_solved(Z,Z) :- holds(peg(45) and
                           forall(X, X=45 or not peg(X)),Z).

pattern_solved(P,Z1,Z4) :- member(Y,P,Z1,Z2),
                           do(jump(U,V,W),Z2,Z3),
                           pattern_solved(P,Z3,Z4).

pattern_solved([Catalyst|P],Z1,Z2) :- holds(peg(Catalyst),Z1),
                                      empty(P,Z1,Z2).

empty([],Z,Z).

empty([X|L],Z1,Z2) :- holds(not peg(X),Z1), empty(L,Z1,Z2).

strategy(Z1,Z2) :- board_solved(Z1,Z2).

strategy(Z1,Z4) :- is_pattern(P,Z1,Z2),
                   pattern_solved(P,Z2,Z3),
                   strategy(Z3,Z4).
```