GOLOG

Situation Calculus

- *S0* – initial situation
- *s* – situation, *a* – action, do(*a,s*) is the situation that results from execution of *a* in *s*
- Axioms describing *S0*
- Action precondition axioms, for each primitive action *a*, characterizing *Poss(a,s)*
- Successor state axioms, one for each fluent *F*(**x**)**,** stating under what conditions *F*(**x**) holds after executing *a*, i.e. defining holds(*F*(**x**),do(*a*,s)) [**x** denotes the parameters of *F*]

Planning

Given a domain *D,* a goal formula ψ(s), the planning task is to find a sequence of actions $a_1,\ldots,a_n$ such that

$D \models Legal([a_1,\ldots,a_n], S0) \wedge holds(\psi, do([a_1,\ldots,a_n], S0))$

where $do([a_1,\ldots,a_n], s)$ stands for $do(a_n,\ldots,do(a_1,s))$ and

$Legal([a_1,\ldots,a_n], s)$ stands for

$Poss(a_1,s) \wedge Poss(a_2,do(a_1,s)) \wedge \ldots \wedge$
$Poss(a_n, do([a_1,\ldots,a_{n-1}], s))$

GOLOG

- Programs in GOLOG is define recursively as follows:
  - a, primitive action
  - ψ?, wait for a condition
  - $(\delta_1;\delta_2)$, sequence
  - $(\delta_1|\delta_2)$, nondeterministic choice between actions
  - πx.δ, nondeterministic choice between arguments
  - δ*, nondeterministic iteration
  - [**proc** $P_1(v_1)$ δ_1 **end**;… **proc** $P_n(v_n)\delta_n$ **end**;δ] procedure

Current situation - *now*

now – representing the current situation can be used in the construction of GOLOG program such as:

proc(removeABlock,

[πb.[OnTable(b,*now*)?;pickup(b);putAway(b)]

end;

removeABlock*; $\neg$ $\exists$b. OnTable(b,*now*)?

- *a*[s] – action formula obtained by substituting the situation s with all occurrence of *now* appearing in *a*
- ψ[s] – formula obtained by substituting the situation s with all occurrence of *now* appearing in ψ

Simplification – Programs without '*now*'

proc(removeABlock,

[πb.[OnTable(b)?;pickup(b);putAway(b)]

end;

removeABlock*; $\neg$ $\exists$b. OnTable(b)?

means 'OnTable(b, now)'

NOTE: the two 'now' represent different time moments

Program Execution – Evaluation Semantics

- Given a domain *D* and a program δ, the execution task is to find a sequence of actions $\alpha=[a_1,\ldots,a_n]$ such that *D* |= Do(δ,S0,do(α,S0) where Do(δ,s,s') is defined as follows:
  - Do(a,s,s') :≡ Poss(a[s], s) $\wedge$ s'=do(a[s],s)
  - Do(ψ?,s,s') :≡ ψ[s] $\wedge$ s'=s
  - Do(δ_1;δ_2,s,s') :≡ $\exists$s''. Do(δ_1,s,s'') $\wedge$ Do(δ_2,s'',s')
  - Do(δ_1|δ_2,s,s') :≡ Do(δ_1,s,s') $\vee$ Do(δ_2,s,s')
  - Do(πx.δ,s,s') :≡ $\exists$x Do(δ(x),s,s')
  - Do(δ*,s,s') :≡ $\forall$P.[$\forall$s1.P(s1,s1) $\wedge$
    $\forall$s1,s2,s3.[P(s1,s2) $\wedge$ Do(δ,s2,s3) $\supset$P(s1,s3)]] $\supset$ P(s,s')

GOLOG interpreter (Reiter)

- Fluents
- Actions
- Successor state axioms written in the form:

  F(x, do(A,s)) instead of holds(F, do(A,s))
- Need to specify 'restoreSitArg(F,S,G)' for each fluent

Transition Semantics

- Easier to deal with concurrency
- Based on '*single step*' of computation
- Defined by two predicates:
  - *Trans*(δ, s, δ', s'): executing δ in s will result in s' and δ' is what remains from δ (that needs to be executed).
  - *Final*(δ, s): δ can legally finish in s (no program remains to be executed)
- (δ,s): a *configuration*

Defining *Trans* and *Final*

- *nil* – the *empty program*
- *Trans*(δ, s, δ', s') is true iff there is a transition from the configuration (δ, s) to (δ', s')
- *Final*(δ, s) is true iff δ can legally finish at s
- Axioms: see De Giacomo et al.

Trans and *Final*

- *Trans*(nil,s,δ',s') ≡ *False*
- *Trans*(a,s,δ',s') ≡ *Poss*(a[s], s) ∧ s'=do(a[s],s) ∧ δ'=nil
- *Trans*(ψ?,s,δ',s') ≡ ψ[s] ∧ s'=s ∧ δ'=nil
- *Trans*(δ_1;δ_2,s,δ',s') ≡ ∃γ.δ'=(γ;δ_2) ∧ *Trans*(δ_1,s,γ,s') ∨ *Final*(δ_1,s) ∧ *Trans*(δ_2,s,δ',s')
- *Trans*(δ_1|δ_2,s,δ',s') ≡ *Trans*(δ_1,s,δ',s') ∨ *Trans*(δ_2,s,δ',s')
- *Trans*(πv.δ,s, δ', s') ≡ ∃x.*Trans*(δ(x/v),s,δ',s')
- *Trans*(δ*,s,δ',s')≡ ∃γ.δ'=(γ;δ*) ∧ *Trans*(δ_1,s,γ,s')

Trans and *Final*

- *Final*(nil,s) ≡ *True*
- *Final*(a,s) ≡ *False*
- *Final*(ψ?,s) ≡ *False*
- *Final*(δ_1;δ_2,s) ≡ *Final*(δ_1,s) ∧ *Final*(δ_2,s)
- *Final*(δ_1|δ_2,s) ≡ *Final*(δ_1,s) ∨ *Final*(δ_2,s)
- *Final*(πv.δ,s) ≡ ∃x.*Final*(δ(x/v),s)
- *Final*(δ*,s)≡ *True*

Relationship between *Do* and *Trans* and *Final*

$Tran^*(\delta,s,\delta',s') = \forall T.[$

$True \supset T(\delta_1,s,\delta,s) \wedge$

$Tran(\delta,s,\delta'',s'') \wedge T(\delta'',s'',\delta',s') \supset T(\delta,s,\delta',s')$

$\supset T(\delta,s,\delta',s')]$

$Do(\delta,s,s') \equiv \exists\delta'\ Trans^*(\delta,s,\delta',s') \wedge Final(\delta',s')$

Example

- Moving around in bi-directed graph
- Elevator