Recap – Situation Calculus

- Situation Calculus
  - allows for reasoning about change and actions
  - a dialect of the second-order logic
  - uses the concept of situations
  - allows for proving properties
  - solves the frame problem

- Basic Action Theory
  - implements the situation calculus
  - Foundation Axioms & Action Precondition & Successor State Axioms & Unique Name Assumption

Golog

- Situation Calculus is yet only a theoretical construct
- Golog (aGol for Logic) is based on the Situation Calculus
- it is a program language for dynamic systems
- it allows a balance between reasoning/planning and imperative programming (i.e., planning is expensive)
- it allows complex actions (so for only primitive actions)
- Golog can be implemented using logic programming
Golog
- A Golog program $\delta$ is based on SC
- It uses the macro $Do(\delta, s, s')$
- $Do(\delta, s, s')$ will be macro-expanded to a SC formula
- The formula $Do$ states that $s'$ is reachable from $s$ by executing the program $\delta$
- The syntax supports primitive and complex actions

Golog Syntax (1)
- Primitive Action: $\alpha$
  - Has a precondition $Poss$ and the effects are modeled in the successor state axioms
  - $walk(R, L)$
- Test action: $\phi$?
  - Tests if $\phi$ holds in a situation, does not change the situation
  - $(\neg(\exists x, y).nextTo(x, y))?$
- Sequence: $\delta_1; \delta_2$
  - Executes $\delta_1$ and $\delta_2$ one after each other
  - $walk(R, L); pickup(R, K)$

Golog Syntax (2)
- Non-deterministic Choice of Actions: $\delta_1 | \delta_2$
  - (Randomly) action $\delta_1$ or $\delta_2$ will be executed
  - $walk(R, A) | walk(R, B)$
- Non-deterministic Choice of Arguments: $(\pi x).\delta(x)$
  - (Randomly) choose an argument $x$ for the action $\delta$
  - $(\pi x).pickup(R, x)$
- Non-deterministic Iteration: $\delta^*$
  - Executes $\delta$ for a not defined number of times ($n \geq 0$)
  - $(pickup(R, L); drop(R, L))^*$

Golog Syntax (3)
- Conditionals: if $\Phi$ then $\delta_1$ else $\delta_2$ endif
  - Based on the truth value of $\Phi$ either $\delta_1$ or $\delta_2$ is executed
  - if $low\_battery(R)$ then $recharge(R)$ else $walk(R, Party)$ endif
- Loops: while $\Phi$ do $\delta$ endwhile
  - As long as $\Phi$ is true repeat action $\delta$
  - while $\neg low\_battery(R)$ do $pickup(R, L); drop(R, L)$ endwhile
- Procedures: proc P(x) $\delta$ endproc
  - Defines the procedure $P$ with the parameters $x$
  - proc d(n) (n=0)? | d(n-1) endproc
On the Semantics of Golog

Golog programs are **macro-expanded** to Situation Calculus formulas using the macro $Do(\delta, s, s')$

- **what is the meaning of Golog and a program $\delta$**
  - $D(\delta, s, s')$, $Do(s, s')$
  - the Basic Action Theory entails if a given program $\delta$ lead to the situation $s$ starting from $S_0$

- **drawback**: the macros are **less expressive**
- a program trace can be obtained by a constructive proof of the above sentence
- **some properties are provable**: e.g., termination

Semantics of Golog Parts (1)

- **Primitive Action: $a$**
  - $Do(a, s, s') = \text{Poss}(a[s], s) \land s' = do(a[s], s)$
  - $a[s]$ denotes restoring of all situation arguments in the functional fluents mentioned in $a$
  - $a = \text{goTo(location(Sam))}$, $a[s] = \text{goTo(location(Sam, s))}$

- **Test action: $\phi$**
  - $Do(\phi?, s, s') = \phi[s] \land s = s'$
  - $\phi[s]$ denotes restoring of all situation arguments in the fluents mentioned in $\phi$
  - $\phi(\forall x).\text{onTable}(x)$, $\phi[s](\forall x).\text{onTable}(x, s)$

- **Sequence: $\delta_1; \delta_2$**
  - $Do(\delta_1; \delta_2, s, s') = (\exists s''). Do(\delta_1, s, s'') \land Do(\delta_2, s'', s')$

Semantics of Golog Parts (2)

- **Non-deterministic Choice of Actions: $\delta_1|\delta_2$**
  - $Do(\delta_1|\delta_2, s, s') = Do(\delta_1, s, s') \lor Do(\delta_2, s, s')$

- **Non-deterministic Choice of Arguments: $(\forall x)\delta(x)$**
  - $Do((\forall x)\delta(x), s, s') = (\exists x) Do(\delta(x), s, s')$

- **Non-deterministic Iteration: $\delta^*$**
  - $Do(\delta^*, s, s') = (\forall P). (\exists s_0. P(s_0, s_0) \land s_0, s_0, s_0 \models P(s_0, s_0) \land Do(\delta^*, s_0, s_0) \rightarrow P(s, s))$

Semantics of Golog Parts (3)

- **Conditionals: if $\phi$ then $\delta_1$ else $\delta_2$ endif**
  - expressed by the previous constructs
  - $Do(\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2 \text{ endif}) = Do(\delta_1 \text{ if } \phi \land \neg \phi \land \delta_2)$

- **Loops: while $\phi$ do $\delta$ endwhile**
  - expressed by the previous constructs
  - $Do(\text{while } \phi \text{ do } \delta \text{ endwhile}, s, s') = Do((\phi? \land \neg \phi? \land \phi?) \land Do(\delta, s, s') | s, s')$
Semantics of Golog Parts (4)

- A Program:
  \[ \text{proc } P_1(x_1) \delta_1 \text{ endproc}; \ldots; \text{proc } P_n(x_n) \delta_n \text{ endproc}; \delta_0 \]
  a sequence of procedure declarations plus a main program \( \delta_0 \)

- we define \( \text{Do}(P(t_1, \ldots, t_n), s, s') = P(t_1[s], \ldots, t_n[s], s, s') \)
  defines a procedure call
  \( t_i[s] \) denotes the evaluation of \( t_i \) in situation \( s \) before passing to \( P \)
  represents a call by value

\[ \begin{align*}
  \text{Do}( & \text{proc } P_1(x_1) \delta_1 \text{ endproc}; \ldots; \text{proc } P_n(x_n) \delta_n \text{ endproc}; \delta_0, S_0, s, s') \\
  = & (\forall P_1, \ldots, P_n) \left( \bigwedge (\forall x_i, s, s'), \text{Do}(P_i(x_i, s), s, s') \right) \\
  & \rightarrow \text{Do}(\delta_0, s, s')
\end{align*} \]

A Simple Example

- \( \delta = \{ x \} \{ A(x) ; ? \} ; \{ B(x) \} \{ C(x) ; ? \} \)\n- \( \text{Do}(\delta, S_0, s) ? \)

\[ \begin{align*}
  (\exists x). \left( (\exists s_1). \text{Poss}(A(x), S_0) \land s_1 = \text{do}(A(x), S_0) \land (\exists s_2). s_1 = s_2 \land [\text{Poss}(B(x), s_2) \land s = \text{do}(B(x), s_2) \lor (\exists s_3). \text{Poss}(C(x), s_2) \land s_2 = s_3 \land \text{do}(C(x), s_3) \land s = \text{do}(C(x), s_3)] \right)
\end{align*} \]

- two possible traces: \( \text{do}(B(x), \text{do}(A(x), S_0)) \land \text{do}(C(x), \text{do}(A(x), S_0)) \) assuming \( ? \) and \( ?' \) holds in the related situations

Situation Calculus vs. Golog Programs

- Situation Calculus and Golog Programs are related
  macro expansion uses the basic action theory (e.g., preconditions and successor state axioms)
- Golog is not a classical program language
  there are no side effects or states
- program traces come from theorem proving
  axioms \( \{ (\exists s). \text{Do}(\delta, s, s) \} \)
  allows for proving of properties of the program

Executing a Golog Program
Planning versus Programming

```
proc toh()
  while ¬solved do
    (πo) (πd)
    move(o,d)
  endwhile
endproc
Do(toh(),s0,s)
```

```
proc toh(n,o,d,h)
  toh(n-1,o,h,d)
  move(o,d)
  toh(n-1,h,d,o)
endproc
Do(toh(3,a,b,c),s0,s)
```

The “famous” Elevator Example

- **Elevator**
  - several floors
  - each floor has a door
  - each floor has a call button

- **Possible Actions**
  - up(n)
  - down(n)
  - trunoff(n)
  - open
  - close

- **Task**
  - serve all requests

Elevator Part 1

- **Fluents**
  - currentFloor(n,s), elevator is on floor n
  - on(n,s), call button is on at floor n

- **Primitive Action Preconditions**
  - Poss(up(n),s) ≡ ∃m.currentFloor(m,s) ∧ m<n
  - Poss(down(n),s) ≡ ∃m.currentFloor(m,s) ∧ m>n
  - Poss(open,s) = true
  - Poss(close,s) = true
  - Poss(trunoff(n),s) = on(n,s)

Elevator Part 2

- **Successor State Axioms**
  - currentFloor(m,do(a,s)) = a=up(m) ∨ a=down(m) ∨ currentFloor(m,s) ∧ ¬(∃n).[(a=up(n) ∨ a=down(n)) ∧ m≠n]
  - on(m,do(a,s)) = on(m,s) ∧ a=trunoff(m)

- **Initial Situation**
  - currentFloor(4,S₀), on(3,S₀), on(5,S₀)
Elevator Part 3

• Golog Procedures
  
  • proc serve(n)
    goFloor(n); turnoff(n); open; close endproc
  
  • proc goFloor(n)
    (currentFloor(n))? up(n); down(n) endproc
  
  • proc serveAFloor
    if serve(n) endproc
  
  • proc control
    [while (3) ? on(n) do serveAFloor endwhile]; park endproc
  
  • proc park
    if currentFloor(0) then open else down(0); open endproc

Elevator Part 4

• Running the Program
  
  • prove the following entailment: Axioms \( \exists s. Do(\exists S_0. s) \)
  
  • action sequence: down(3), turnoff(3), open, close, up(5), turnoff(5), open
  
  • there are multiple solutions

Golog - Summary

• Golog is logic program language
• Golog is based on Situation Calculus axioms
• its interpreter is a general purpose theorem prover
• Golog programs are executed for their side effects
• Golog programs are executed off-line
Prolog as Implementation of Logic

- Prolog is a logic program language
- Prolog allows for declarative programming (i.e., concentrates on the problem formulation)
- Prolog can be used to implement a logical theory
- If some properties hold there is mapping between Prolog and the logical theory behind
  - Use a definitional theory
  - Use a proper Prolog interpreter
  - Theorem of Clark

Definitional Theory

- Definition axioms have the following form
  \((\forall x_1, \ldots, x_n). P(x_1, \ldots, x_n) \equiv \Phi\), \(P\) is a predicate other the equality, \(\Phi\) is a FOL sentence can also be written as \((\forall x_1, \ldots, x_n). \Phi \rightarrow P(x_1, \ldots, x_n)\)
- A set of definition axioms form a definitional theory
- An atom \(A\) is a formula of the form \(P(x_1, \ldots, x_n)\)
- A literal \(L\) is an atom or its negation
- A clause \(C\) has the form \(L_1 \land \ldots \land L_m \rightarrow A\), \(m \geq 0\)

Prolog Interpreter

- Prolog represents a clause as \(A \leftarrow L_1, \ldots, L_m\).
- Prolog represents a goal as \(L_1, \ldots, L_k\).
- A proper Prolog interpreter has the following properties
  - It interprets the negation not \(A\) as negation as failure
  - It does so only if \(A\) is a ground literal
  - If \(A\) is not ground the interpreter suspend the evaluation until \(A\) become ground or it may abort the computation
- The Clark’s Theorem guarantees the right “return” value of a Prolog program

A Golog Interpreter

- It is based on Prolog for theorem proving
- The Situation Calculus is based on second-order logic
- Prolog does not support the whole power of second-order logic
- The interpreter use some assumptions
  - There are only relational fluents
  - The initial database is closed
  - Special assumptions for of action preconditions and relational fluents
Clark’s Theorem

- Suppose $T$ is a set of definitions for every predicate except equality
- If the following properties hold:
  - For two distinct function symbols: $f(x) \neq g(y)$
  - For a function symbol: $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \land \ldots \land x_n = y_n$
  - For every term $t[x]$ that mentions the variable $x$ $t[x] \neq x$

  Then when a proper Prolog interpreter succeeds on a goal $G$ then $T \vDash (\forall)G$.

  Then when a proper Prolog interpreter fails on a goal $G$ then $T \nvDash (\forall)G$.

Negation as Failure

- Prolog implements negation as “negation as failure.”
- For $\neg p$ Prolog tries to prove $p$ if $p$ can be proved return false otherwise return true.
- For $P(x) = x = A, Q(x) = x = B$
  - $\neg P(B) \land Q(B) \Rightarrow YES$
  - $p(a), q(b)$.
  - $\neg p(x), q(X)$
  - $q(X), \neg p(X)$
- Prolog usually does no NAF on non-ground atoms.

Lloyd-Topor Transformation

- Problem:
  - $(\forall x,y). subset(x,y)$
  - $(\forall x,y). member(z,x) \rightarrow member(z,y)$
  - $(\forall x,y). (\forall z)[member(z,x) \rightarrow member(z,y)] \rightarrow subset(x,y)$
- The LT Transformation transforms a sentence $W \rightarrow A$ in a clause suitable for Prolog.
- There are 12 (syntactic) rules (e.g., replace $\forall \phi(W, \forall W) \rightarrow A$ with $\forall \phi(W, \forall W) \land \theta \rightarrow A$, $\forall \phi(W, \forall W) \land \theta \rightarrow A$.
- $\neg p(x,y) \rightarrow subset(x,y)$.
  - $member(z,x) \land \neg member(z,y) \rightarrow p(x,y)$

Closed Initial Database

- An initial database $D_{\theta_{1}}$ is closed iff
  - For every relation fluent $F$ there is only one sentence in the form $F(x_1, \ldots, x_n, S_0) = \phi(x_1, \ldots, x_n, S_0)$
  - For every non-fluent predicate $P$ there is only one sentence in the form $P(x_1, \ldots, x_n) = \phi(x_1, \ldots, x_n)$ where $\phi$ is situation independent.
  - For two distinct function symbols: $f(x) = g(y)$
  - For a function symbol: $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \land \ldots \land x_n = y_n$
  - For every term $t[x]$ that mentions the variable $x$ $t[x] \neq x$ (objects)
  - For every term $t[a]$ that mentions the variable $a$ $t[a] \neq a$ (actions)

- A restriction on the expressiveness, not allowed:
  - $F(A, S_0) \lor F(B, S_0)$
  - $(\exists x) F(x, S_0)$
  - $\text{murderer} (\text{Caesar}) = \text{Brutus}$
Implementation Theorem

- Let \( D \) be a basic action theory and \( P \) a Prolog program obtained by Lloyd-Topor rules
  - for each non-fluent predicate of \( D_{\text{nc}} \) of form
    \[ P(x_1, \ldots, x_n) \iff \text{\( P(x_1, \ldots, x_n) \rightarrow P(x_1, \ldots, x_n) \)}} \]
  - for each relational fluent predicate of \( D_{\text{rf}} \) of form
    \[ F(x_1, \ldots, x_n, S_0) \iff \text{\( F(x_1, \ldots, x_n, S_0) \rightarrow F(x_1, \ldots, x_n, S_0) \)}} \]
  - for each action precondition axiom of \( D_{\text{ap}} \) of form
    \[ \text{Poss}(A(x_1, \ldots, x_n, s)) \iff \text{\( \text{Poss}(A(x_1, \ldots, x_n, s)) \rightarrow \text{Poss}(A(x_1, \ldots, x_n, s)) \)}} \]
  - for each successor state axiom of \( D_{\text{ss}} \) of form
    \[ F(x_1, \ldots, x_n, \text{do}(a, s)) \iff \text{\( F(x_1, \ldots, x_n, \text{do}(a, s)) \rightarrow F(x_1, \ldots, x_n, \text{do}(a, s)) \)}} \]

- then \( P \) provides a correct Prolog implementation of \( D \) (assuming a closed database)

Golog Summary

- Golog is a program language for dynamic systems
- Golog used the principles of the Situation Calculus
- execution of a Golog program is related to theorem proving
- there exists Golog interpreter using logic programming
- if some assumptions hold the interpreter correctly executes the program
- some limitations for real world systems

Questions?