

Macro-actions in the Situation Calculus



Yilan Gu

Department of Computer Science

University of Toronto

August 10, 2003



Outline

- ❖ Situation calculus
- ❖ Introducing macro-actions
- ❖ Developing the knowledge base for the macro-actions
- ❖ Reuse of the macro-actions
- ❖ Conclusion and future work



Situation Calculus

- ❖ The basic elements of the situation calculus
 - ❖ Actions *kick(x): kick object x*
 - ❖ Situations *S₀, do(a,s)*
 - ❖ Objects *mary, boxA*
- ❖ The basic action theory
 - ❖ Action precondition axioms
 - ❖ Successor state axioms
 - ❖ Initial database
- ❖ Complex actions
 - α;β, p?, α|β, (πx)α(x), if-then-else, while loop, procedure*
- ❖ Golog program – a bunch of procedures



Situation Calculus (Cont.)

- ❖ The regression operator \mathcal{R}

Example: Successor state axiom for fluent *ontable*(*t*,*s*)

$$\text{ontable}(t, \text{do}(a, s)) \equiv a = \text{putonTable}(t) \vee \text{ontable}(t, s) \wedge \neg a = \text{pickup}(t)$$

$$\mathcal{R}[\text{ontable}(x, \text{do}(a, S_0))]$$

$$= a = \text{putonTable}(x) \vee \text{ontable}(x, S_0) \wedge \neg a = \text{pickup}(x)$$

- ❖ The regression theorem

For any regressable formula *W*, $\mathcal{D} \models W$ iff $\mathcal{D}_{S_0} \models \mathcal{R}[W]$

- ❖ Uncertainty system

- ❖ Stochastic actions

Example: $\text{choice}(\text{pickup}(x), a) \equiv a = \text{pickupS}(x) \vee a = \text{pickupF}(x)$

- ❖ Probability

Example: $\text{prob}(\text{pickupS}(x), \text{pickup}(x), s) = 0.9$

- ❖ Modified Golog interpreter – stGolog

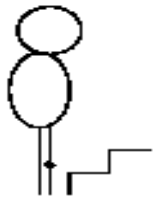


Motivation of Introducing Macro-actions

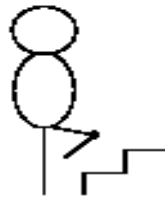
- ❖ Create intelligent autonomous agents to help human on particular topics
- ❖ Agents often meet same local situations and asked to achieve same tasks and perform same strategies
- ❖ Human beings often act “without thinking”
- ❖ Purpose: save time on re-computation
- ❖ Idea:
 - ❖ Consider certain kind of complex actions as a whole
 - ❖ Compute and keep intermediate knowledge
 - ❖ Reuse the existing intermediate knowledge



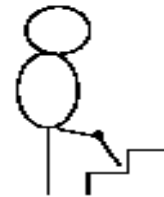
Example of Robot climbing Stairs



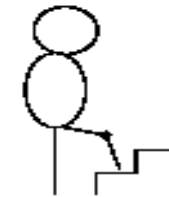
(0) ready



(1) liftUpperLeg(h)



(2) forwLowLeg



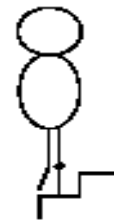
(3) stepDown(main)



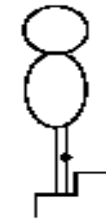
(4) moveBarycenter(main)



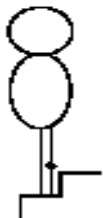
(5) straightMain



(6) forwSupLeg



(7) stepDown(supporting)



(8) moveBarycenter(supporting)
(ready again)



Robot Climbing Stairs (Cont.)

- ❖ Nature's choices: (example)

$$\text{choice}(\text{liftUpperLeg}(h), a) \equiv a = \text{liftTill}(h) \vee a = \text{malfunc}(h)$$

- ❖ Fluents: *Barycenter(l,s), footOnGround(l,s), overNewStair(l,s), ..., etc*

- ❖ Precondition axioms: (example)

$$\text{Poss}(\text{liftTill}(h), s) \equiv \text{barycenter}(\text{supporting}, s)$$

- ❖ Successor State Axioms: (example)

$$\text{overNewStair}(l, \text{do}(a, s)) \equiv a = \text{forwSupLegS} \wedge l = \text{supporting} \vee$$

$$a = \text{forwLowLegS} \wedge l = \text{main} \vee$$

$$\text{overNewStair}(l, s) \wedge \neg a = \text{stepDownS}(l)$$

- ❖ Probabilities: (examples)

$$\text{prob}_0(\text{liftTill}(h), \text{liftUpperLeg}(h), s) = h/(h+100).$$

$$\text{prob}_0(\text{malfunc}(h), \text{liftUpperLeg}(h), s) = h/(h+100).$$



Robot Climbing Stairs (Cont.)

❖ Procedure of climbing a stair of height h

```
proc climbing( $h$ )
```

```
  legalStair( $h$ )?; liftUpperLeg( $h$ ); forwLowLeg; stepDown(main);
```

```
  moveBarycenter(main); straightLeg; forwSupLeg;
```

```
  stepDown(supporting); moveBarycenter(supporting)
```

```
Endproc
```

The robot need to climb thousands of stairs, by using former stGolog, the agent need to do regression for each primitive action which includes lots of repetition information. We can treat climbing one stair as a whole, pre-compute and reuse its information.

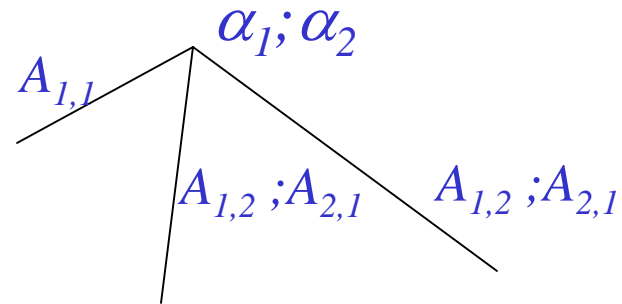
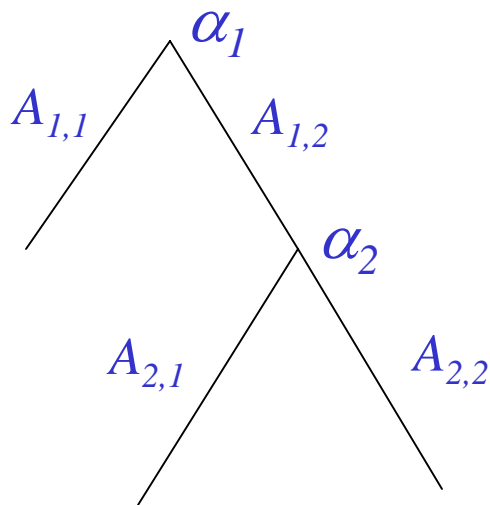


Introducing macro-actions

- ❖ Discard uncertain logic actions α/β and $(\pi x)\alpha(x)$
- ❖ Not consider $p?$, *if-then-else* or *while loop* as a part of macro-actions
- ❖ Small sequence of actions is appropriate
 - ❖ Uniform syntax
 - ❖ Easy to find its deterministic choices and information of corresponding probabilities
 - ❖ Feasible to give extended precondition axioms and successor state axiom

Introducing macro-actions (Cont.)

- ❖ Example: sequence of stochastic actions $\alpha_1; \alpha_2$ executed in current situation



Extend precondition axioms and successor state axioms and compute probabilities for sequence of deterministic actions rather than for primitive actions, therefore regression operator can be extended to sequential actions as well.



Specifying macro-actions

- ❖ Reason:

- ❖ Distinguish from ordinary sequential actions
- ❖ Need to let the autonomous agent know

- ❖ Syntax:

macro name $\alpha_1; \alpha_2; \dots; \alpha_n$ endmacro



Example of macro-actions

❖ Example 1

```
macro stepMain(h)
```

```
    liftUpperLeg(h); forwLowLeg; stepDown(main);
```

```
    moveBarycenter(main); straightLeg
```

```
endmacro
```

```
macro stepSupp
```

```
    forwSupLeg; stepDown(supporting); moveBarycenter(supporting)
```

```
endmacro
```

```
proc climbing(h)
```

```
    legalStair(h)?; stepMain(h); stepSupp
```

```
endproc
```



Example of macro-actions (Cont.)

❖ Example 2

```
macro climbStair(h)  
    liftUpperLeg(h); forwLowLeg; stepDown(main);  
    moveBarycenter(main); straightLeg; forwSupLeg;  
    stepDown(supporting); moveBarycenter(supporting)  
endmacro  
  
proc climbing(h)  
    legalStair(h)?; climbStair(h)  
endproc
```



The knowledge base for macro-actions

- ❖ What intermediate information do we want to keep?
 - ❖ Definition of macro-actions
 - ❖ The extended successor state axioms
 - ❖ The extended precondition axioms of nature's choices of macro-actions that are not equivalent to false
 - ❖ The Extended probabilities of nature's choices of macro-actions
- ❖ By using a simple algorithm, we can develop knowledge base for macro-actions
- ❖ Implemented in Prolog



Extended regression operator

- ❖ Reason: reuse the existing info. in the knowledge base
- ❖ *s*-regressable formula
 - ❖ Situation terms of from $do([\alpha_1, \dots, \alpha_n], s)$
 - ❖ Regressable formula is same as S_0 -regressable
- ❖ The extended regression operator \mathcal{R}^* for *s*-regressable formula W
 - ❖ Boarder than original regression operator
 - ❖ Save computational steps
$$\mathcal{R}^*[F(x, do(a_1; a_2; a_3, S_0))] = \mathcal{R}[F(x, do([a_1, a_2, a_3], S_0))], \text{ but has different computational steps}$$
 - ❖ Now having $\mathcal{R}^*[F(x, do(a_1; a_2; a_3, s))]$, but there was no $\mathcal{R}[F(x, do([a_1, a_2, a_3], s))]$;



The regression theorems for \mathcal{R}^\star

- ❖ Given *s*-regressable sentence W and a basic action theory \mathcal{D} , $\mathcal{R}^\star[W]$ is a sentence uniform in s and

$$\mathcal{D} \models W \equiv \mathcal{R}^\star[W]$$

- ❖ Given regressable sentence W and a basic action theory \mathcal{D} , then $\mathcal{D} \models \mathcal{R}[W] \equiv \mathcal{R}^\star[W]$

$$\mathcal{D} \models W \text{ iff } \mathcal{D}_{S_0} \models \mathcal{R}^\star[W]$$

- ❖ Given S_0 -regressable sentence W_1 , W_2 be the corresponding sentence by changing $do(a_1; a_2; \dots; a_n, s)$ to be $do([a_1, a_2, \dots, a_n], s)$, and a basic action theory \mathcal{D} , then

$$\mathcal{D} \models W_2 \text{ iff } \mathcal{D}_{S_0} \models \mathcal{R}^\star[W_1]$$



Reuse of macro-actions

- ❖ After developing the knowledge base of macro-actions given by the user, we want to use and reuse the remembered information
- ❖ A primitive application – macGolog
 - ❖ Modified from stGolog
 - ❖ Program now allows to include macro-actions
 - ❖ Use extended regression operator
 - ❖ Adding the checking that if an action is a macro-action, then use the existing extended axioms in the knowledge base for calculating; otherwise, treat as before



Implementation of macGolog

- ❖ Implemented in Polog
- ❖ Has same functions as stGolog
- ❖ Benefit: save computing steps in the searching trees, therefore save computational time
- ❖ Example:

The robot climbing stairs, comparing running the procedures of climbing a stair without or with macro-actions, we have following different searching trees

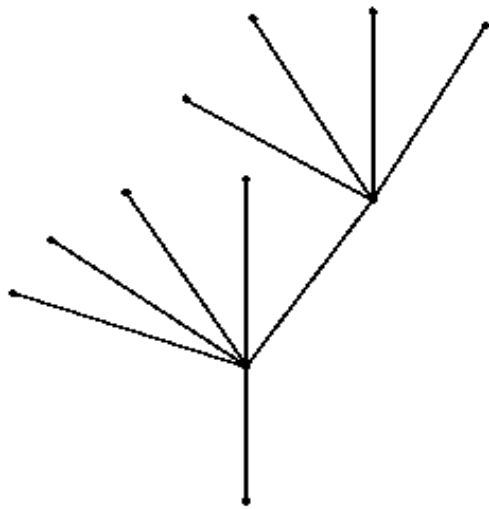


Experiments of climbing stairs

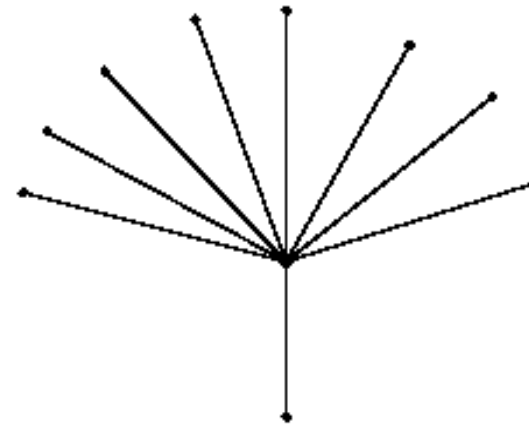
- ❖ Test on 3 different definitions of climbing stair procedure
 - ❖ No macro-actions (using stGolog)
 - ❖ Define 2 short macro-actions: *stepMain(h)*, *stepSupp*
 - ❖ Define 1 long macro-action: *climbStair(h)*
- ❖ 4 tests – simulating different environments
 - ❖ Test 1 – all stairs are of the same height
 - ❖ Test 2 – at most 10 different heights
 - ❖ Test 3 – at most 50 different heights
 - ❖ Test 4 – at most 800 different heights

Searching Trees (Figure 2)

- ❖ The searching tree of climbing a stair for using two macro-actions
- ❖ The searching tree of climbing a stair for using one macro-action



3 steps of regression



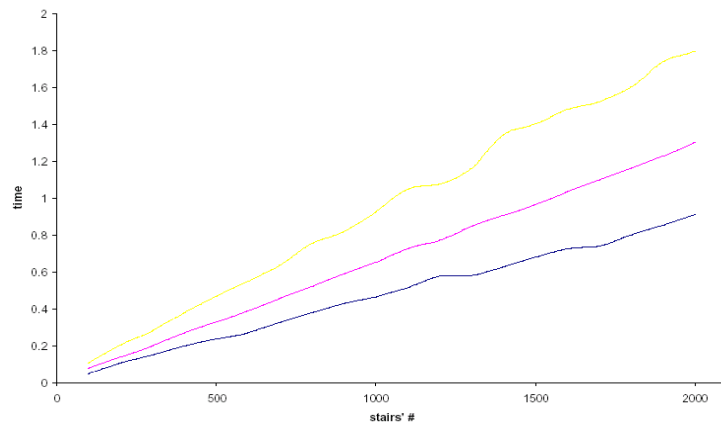
2 steps of regression

Comparing Results

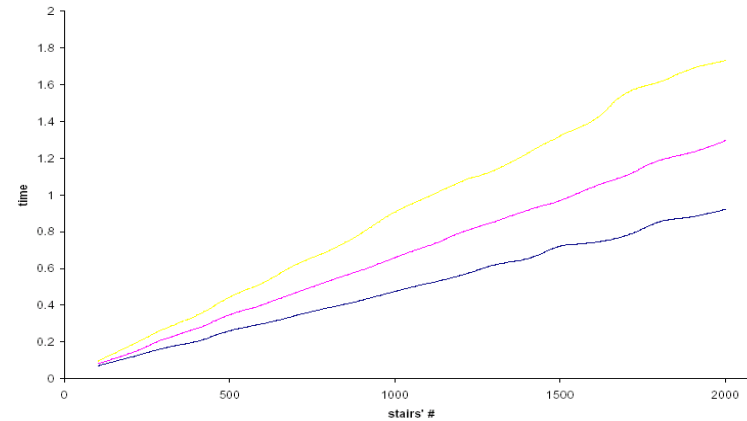
- 2 macro-actions
- 1 macro-action
- no macro-action

X axis: number of stairs
Y axis: CPU time

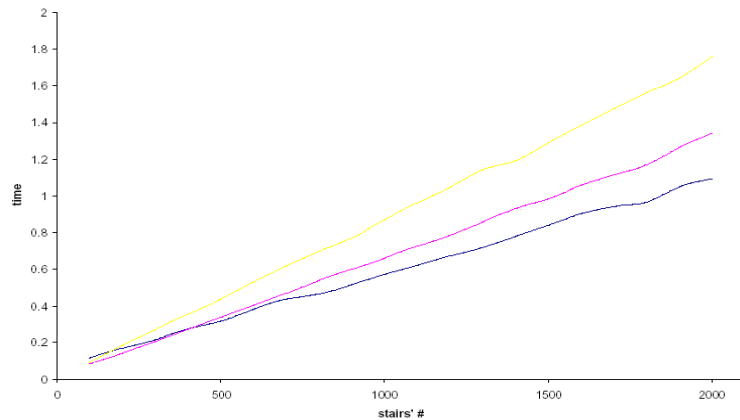
Test 1



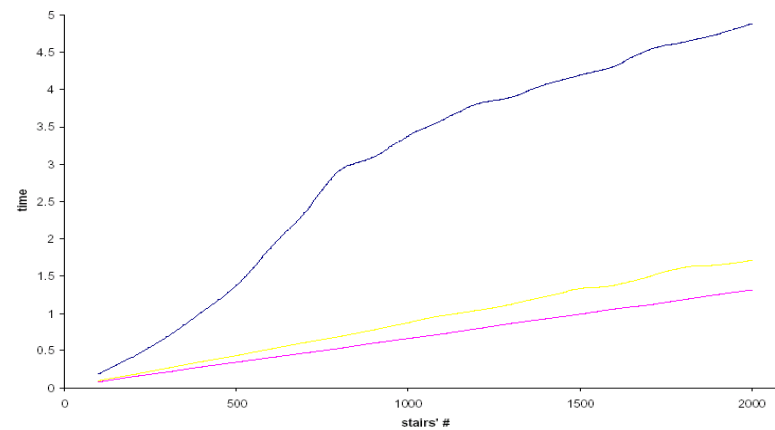
test 2



test 3



Test 4





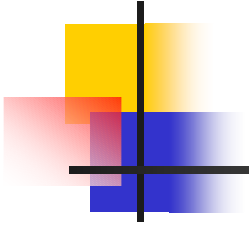
Discussion: benefits and limits

- ❖ Benefits:
 - ❖ Saving computational time
 - ❖ Agents have “memories” in some sense and keep the “experiences”
- ❖ Limits, especially under current implementation:
 - ❖ User’s duty to choose suitable macro-actions
 - ❖ The agents can not be aware of similar local situations themselves yet



Conclusions and future work

- ❖ What we did:
 - ❖ Introduce concept of macro-action
 - ❖ Define extended regression operation
 - ❖ Develop knowledge base for macro-actions
 - ❖ macGolog – primary implementation of reuse macro-actions
- ❖ What we can do in the future:
 - ❖ Improve implementation of reuse macro-actions
 - ❖ Possible to formalize the way of resetting the local situation, make agent forgettable
 - ❖ Try to make agents aware of similar local situations
 - ❖ Consider if we can adopt macro-action into the problem of solving MDP by using the situation calculus



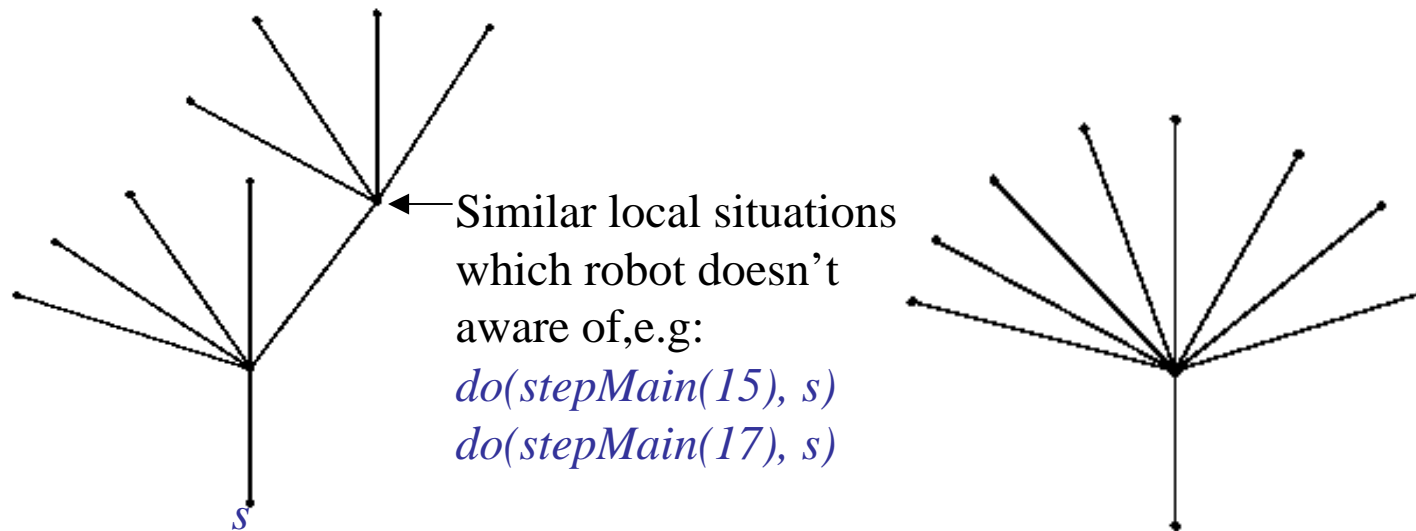
The End

Thank you !

Why 2 macro-actions perform bad when height h changes often

❖ The searching tree of climbing a stair for using two macro-actions

❖ The searching tree of climbing a stair for using one macro-action





Why 2 macro-actions perform better than 1 macro when height h seldom changes

- ❖ For 2 macro-action example, the longest nature's choices of macro-actions are of length 5, therefore we only need to keep extended Successor State Axioms $F(x, do(a_1; a_2, s))$, $F(x, do(a_1; a_2; a_3, s))$, ..., $F(x, do(a_1; a_2; \dots; a_5, s))$ for each fluent F ; while 1 macro-action example, the longest nature's choices of macro-action are of length 8, therefore we need to keep extended Successor State Axioms $F(x, do(a_1; a_2, s))$, $F(x, do(a_1; a_2; a_3, s))$, ..., $F(x, do(a_1; a_2; \dots; a_8, s))$ for each fluent F , which makes the knowledge base larger. And, searching knowledge base costs time.