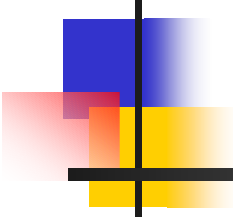


*Handling Uncertainty System in
the Situation Calculus with
Macro-actions*



Yilan Gu

Department of Computer Science

University of Toronto

November 28, 2002



Outline

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



Review

- ❖ 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
 - $\alpha; \beta$, $p?$, α / β , $(\pi x)\alpha(x)$, if-then-else, while loop*
 - proc name body endproc*
- ❖ Golog program – a bunch of procedures followed by complex actions



Review(Cont.)

- ❖ The regression operator

- ❖ Regressable formulas

example: $Poss(pickup(x), S_0) \wedge ontable(x, do(putTable(x), S_0))$

- ❖ The regression operator \mathcal{R}

examples: Suppose $Poss(pickup(t), s) \equiv handfree(s)$,

$ontable(t, do(a, s)) \equiv a = putTable(t) \vee ontable(t) \wedge \neg a = pickup(t)$

then $\mathcal{R}[Poss(pickup(x), S_0) \wedge ontable(x, do(putTable(x), S_0))]$

$$= \mathcal{R}[Poss(pickup(x), S_0)] \wedge \mathcal{R}[ontable(x, do(putTable(x), S_0))]$$

$$= \mathcal{R}[handfree(S_0)] \wedge \mathcal{R}[true]$$

$$= handfree(S_0)$$

- ❖ The regression theorem

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



Review(cont.)

- ❖ Uncertainty system

- ❖ Stochastic actions

Examples: $choice(pickup(x), a) \equiv a = pickupS(x) \vee a = pickupF(x)$

- ❖ Probability

$$prob(a, \beta, s) = p \equiv choice(\beta, a) \wedge Poss(a, s) \wedge p = prob_0(a, \beta, s) \vee \\ \neg [choice(\beta, a) \wedge Poss(a, s)] \wedge p = 0$$

example: if $Poss(pickupS(x), S_0), \neg Poss(pickupS(x), S')$,

$prob_0(pickupS(x), pickup(x), s) = 0.9$, then

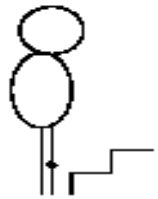
$prob(pickupS(x), pickup(x), S_0) = 0.9$,

$prob(pickupS(x), pickup(x), S') = 0$.

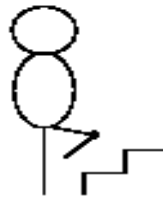
- ❖ Modified Golog interpreter – stGolog



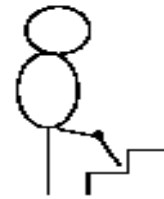
Example of Robot climbing Stairs



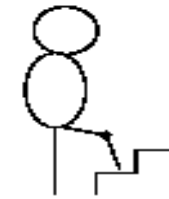
(0) ready



(1) liftUpperLeg(h)



(2) forwLowLeg



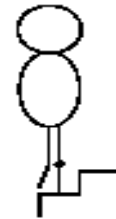
(3) stepDown(main)



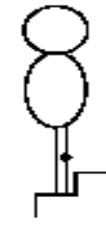
(4) moveBarycenter(main)



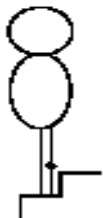
(5) straightMain



(6) forwSupLeg



(7) stepDown(supporting)



(8) moveBarycenter(supporting)
(ready again)



Robot Climbing Stairs (Cont.)

❖ Nature's choices

$choice(liftUpperLeg(h), a) \equiv a = liftTill(h) \vee a = malfunc(h).$

$choice(forwLowLeg, a) \equiv a = forwLowLegS \vee a = forwLowLegF.$

$choice(stepDown(l), a) \equiv a = stepDownS(l) \vee a = stepDownF(l).$

$choice(moveBarycenter(l), a) \equiv$

$a = moveBarycenterS(l) \vee a = moveBarycenterF(l).$

$choice(straightLeg, a) \equiv a = straightLeg.$

$choice(forwSupLeg, a) \equiv a = forwSupLegS \vee a = forwSupLegF.$

❖ Precondition axioms: (example)

$Poss(straightLeg, s) \equiv$

$straightMain(s) \wedge footOnGround(main, s) \wedge barycenter(main, s).$



Robot Climbing Stairs (Cont.)

❖ Fluents

❖ Relational Fluents:

*straightMain(s), Barycenter(l,s), footOnGround(l,s),
overNewStair(l,s)*

❖ Functional Fluents: *mainToCurr(s)*

❖ Successor State Axioms: (example)

overNewStair(l,do(a,s)) \equiv

a = forwSupLegS \wedge *l = supporting* \vee

a = forwLowLegS \wedge *l = main* \vee

overNewStair(l,s) $\wedge \neg a = stepDownS(l)$.



Robot Climbing Stairs (Cont.)

❖ Probabilities: (examples)

$prob_0(\text{liftTill}(h), \text{liftUpperLeg}(h), s) = h/(h+100).$

$prob_0(\text{malfunc}(h), \text{liftUpperLeg}(h), s) = h/(h+100).$

❖ Initial Database:

$\text{straightMain}(S_0), \text{mainToCurr}(0, S_0), \neg \text{overNewStair}(l, S_0),$

$\text{barycenter}(\text{supporting}, S_0), \text{legalStair}(h) \equiv \text{number}(h) \wedge 0 < h < 20.$

$\text{footOnGround}(l, S_0) \equiv l = \text{main} \vee l = \text{supporting}.$

❖ 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



Motivation

- ❖ 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



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.)

- ❖ Sequence of stochastic actions $\alpha = \alpha_1; \alpha_2; \dots; \alpha_n$ and a sequence of deterministic actions $A = A_1; A_2; \dots; A_m$

- ❖ Nature's choice of α

$choiceMac(\alpha, a) \equiv$

$$a \in \{A_1; A_2; \dots; A_m \mid m \in \mathcal{N} \wedge 1 \leq m \leq n \wedge (\bigwedge_{i=1}^m choice(\alpha_i, A_i))\}$$

- ❖ Extended precondition axiom of A

$$Poss(A, s) \equiv \bigwedge_{i=2}^m Poss(A_i, do([A_1; A_2; \dots; A_{i-1}], s)) \wedge Poss(A, s)$$

$$\equiv \bigwedge_{i=1}^m \pi_{A_i}(t_i, s)$$

- ❖ Extended successor state axiom of $a = a_1; a_2; \dots; a_m$ (a_i variables of sort deterministic actions)

$$m=1, F(x, do(a, s)) \equiv \phi_F(x, a, s)$$

$$m>1, F(x, do(a, s)) \equiv \phi_F(x, a_m, do(a_1; a_2; \dots; a_{m-1}, s))$$

$$\equiv \Psi_F(x, a_1, a_2, \dots, a_m, s)$$



Introducing macro-actions (Cont.)

❖ Extended probabilities

$$probMac(A, \alpha, s) = p \equiv choiceMac(\alpha, A) \wedge Poss(A, s)$$

$$\wedge p = prob_0(A_1, \alpha_1, s) * \dots * prob_0(A_m, \alpha_m, do([A_1, \dots, A_{m-1}], s)) \vee$$

$$\neg(choiceMac(\alpha, A) \wedge Poss(A, s)) \wedge p=0.$$

Properties

1. $(\forall a, s). 0 \leq probMac(a, \alpha, s) \leq 1.$
2. $(\forall a, s). \neg choiceMac(\alpha, a) \supset probMac(a, \alpha, s)=0.$
3. $(\forall s). Poss(A, s) \equiv probMac(A, \alpha, s) > 0.$
4. $\forall (A = A_1, \dots, A_n \wedge choiceMac(\alpha, A) \wedge Poss(A, s)) \supset$
 $\sum_{A \in maxPoss(\alpha, s)} probMac(A, \alpha, s) = 1, \text{ where}$
 $maxPoss(\alpha, s) \equiv \{ A = A_1, \dots, A_m \mid choiceMac(\alpha, A) \wedge Poss(A, s) \}$
 $\wedge (m = n \vee m < n \wedge ((\forall a). Choice(\alpha_{m+1}, a) \supset \neg Poss(A; a, s))) \}.$



Specifying macro-actions

- ❖ Reason:

- ❖ Should be distinguish from ordinary sequential actions
- ❖ Need to let the autonomous agent know

- ❖ Syntax:

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

- ❖ Extended definition:

choiceMac(name, a) \equiv choiceMac($\alpha_1; \alpha_2; \dots; \alpha_n, a$),
probMac(A, name, s)=p \equiv probMac(A, $\alpha_1; \alpha_2; \dots; \alpha_n, s$)=p,
seqLength(name) = seqLength($\alpha_1; \alpha_2; \dots; \alpha_n$) = n,
maxPoss(name, s)=l \equiv maxPoss($\alpha_1; \alpha_2; \dots; \alpha_n, s$)=l.



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

- ❖ What intermediate information do we want to keep?

Static part:

- ❖ Definition of macro-actions
- ❖ Maximal length of current existing 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

Dynamic part:

- ❖ Facts : $\text{maxPossBase}(\text{list}, \alpha, S) \equiv \text{list} = \text{maxPoss}(\alpha, S)$ for particular situation instance S .



Extended regression operator

- ❖ Reason: reuse the existing information 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
 - ❖ Example
 - $\mathcal{R}^* [F(x, do(a_1; a_2; a_3, s))]$, no $\mathcal{R}[F(x, do([a_1, a_2, a_3], s))]$;
 - $\mathcal{R}^* [F(x, do(a_1; a_2; a_3, S_0))]$ = $\mathcal{R}[F(x, do([a_1, a_2, a_3], S_0))]$, but has different computational steps



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

- ❖ \mathcal{L}_{sc} , and \mathcal{L}_{sc}' (allow notation $do(a_1; a_2; \dots; a_n, s)$)
- ❖ Given s -regressable sentence W in \mathcal{L}_{sc}' and a basic action theory \mathcal{D} , $\mathcal{R}^\star[W]$ is a sentence uniform in s and

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

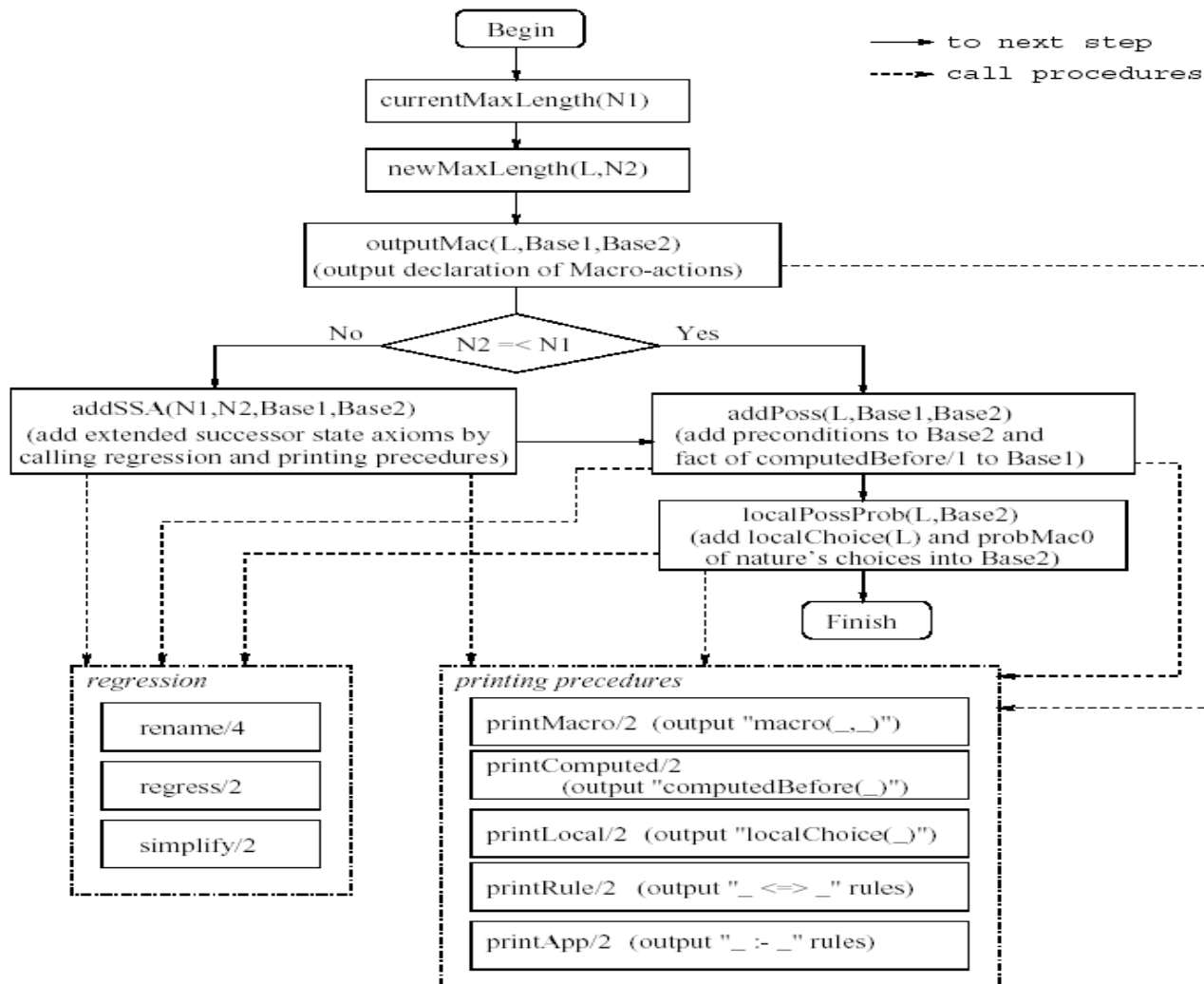
- ❖ Given regressable sentence W in \mathcal{L}_{sc} and a basic action theory \mathcal{D} , then $\mathcal{D} \models \mathcal{R}[W] \equiv \mathcal{R}^\star[W]$

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

- ❖ Given S_0 -regressable sentence W_1 in \mathcal{L}_{sc}' , 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

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

Developing knowledge base for macro-action





Reuse of macro-actions

- ❖ After developing the knowledge base of macro-actions given by the user, we want to use the remembered information
- ❖ A simple application – macGolog
 - ❖ Modified stGolog
 - ❖ Program now allows to include macro-actions
 - ❖ Adding the test checking that if an action is a macro-action
 - ❖ Checking if there exists fact *maxPossBase* for macro-action at current situations (i.e., the dynamic part of the knowledge base), if exists, choose the maximal nature's choices from it, otherwise, compute the fact, insert it to base and use the computed information



Implementation of macGolog

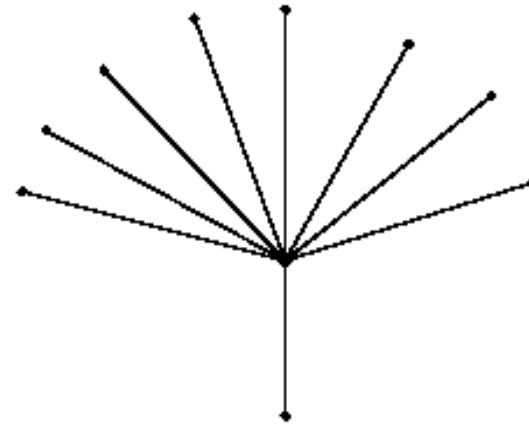
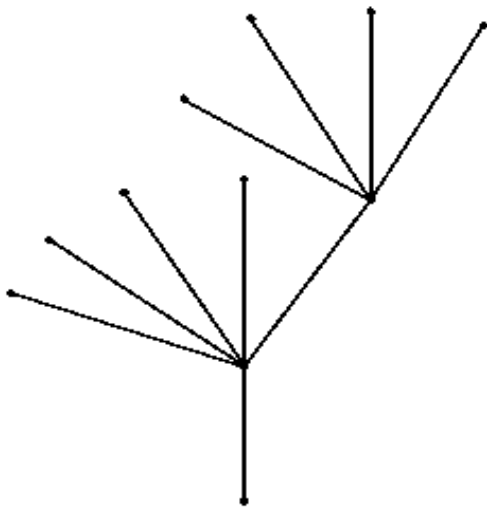
- ❖ Implemented in Polog
- ❖ Has same functions as stGolog
- ❖ Benefit: same computing steps in the searching trees, therefore save computational time
- ❖ Example:

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



Searching Trees (Fig. 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



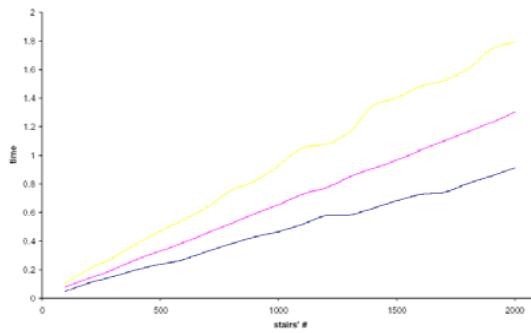


Experiments of climbing stairs

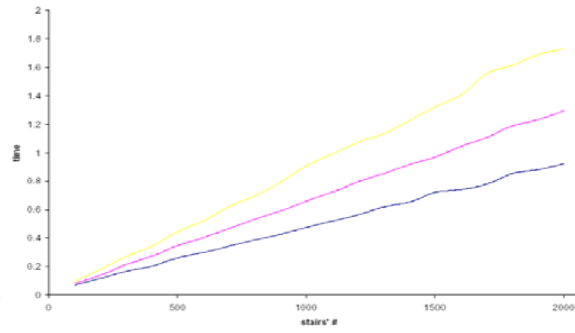
- ❖ Pre-assumption
 - ❖ Test on 3 different definitions of climbing stair procedure
 - ❖ No macro-actions (using stGolog)
 - ❖ Define 2 short macro-actions: *stepMain(h)*, *stepSupp*
 - ❖ Define 1 long macro-action: *climbStair(h)*
 - ❖ 5 tests – simulating different environments
 - ❖ Test 1 – all stairs 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
 - ❖ Test 5 – all stairs are of different height

Comparing Results

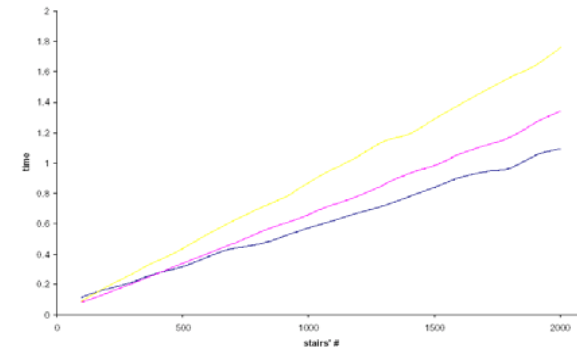
Test 1



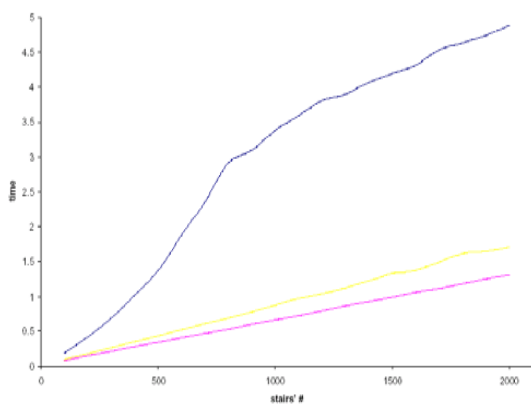
test 2



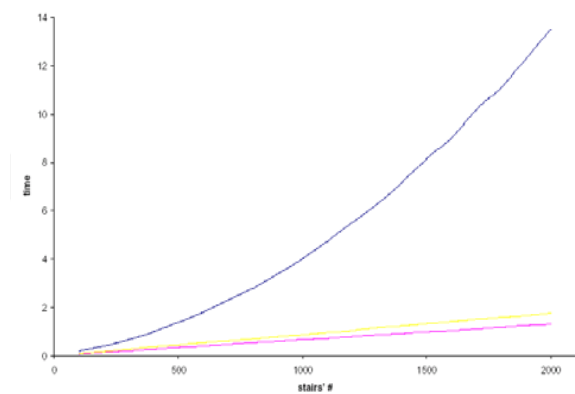
test 3



Test 4



Test 5



-- 2 macro-actions

-- 1 macro-action

-- no macro-action



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



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