Lecture 5:
Automated Reasoning

- Unification
- Linear Resolution
- Linear Resolution with Negation

Legal Play Requires Automated Reasoning

Given:

```
true(cell(1,1,b))
   ⋮
true(cell(3,3,b))
true(control(xplayer))
legal(P,mark(X,Y)) <= true(cell(X,Y,b)) ∧ true(control(P))
legal(xplayer,noop) <= true(cell(X,Y,b)) ∧ true(control(oplayer))
legal(oplayer,noop) <= true(cell(X,Y,b)) ∧ true(control(xplayer))
```

What can be derived for a query like e.g. `legal(P,M)`?

Warning: Prolog may be Incorrect

Given:

```
role(red)
role(green)
role(blue)
true(protected(red))
legal(P,capture(Q)) <= ¬true(protected(Q)) ∧
                       role(P) ∧ role(Q) ∧ distinct(P,Q)
```

Prolog:

```
?- legal(red,M)
no
```

Substitutions

A *substitution* is a finite set of pairs of variables and terms, called *replacements*.

```
{X↣a, Y↣f(b), V↣W}
```

The result of applying a substitution σ to an expression φ is the expression $\varphi\sigma$ obtained from φ by replacing every occurrence of every variable in the substitution by its replacement.

```
p(X,X,Y,Z){X↣a,Y↣f(b),V↣W}=p(a,a,f(b),Z)
```

Unification

A substitution σ is a *unifier* for an expression φ and an expression ψ if and only if $\varphi\sigma=\psi\sigma$.

`move(X,Y){X↦a,Y↦b,V↦b}=move(a,b)`
`move(a,V){X↦a,Y↦b,V↦b}=move(a,b)`

If two expressions have a unifier, they are said to be *unifiable*.

`move(X,X)` and `move(a,b)` not unifiable

Non-Uniqueness of Unification

Unifier 1:

```
move(X,Y){X↦a,Y↦b,V↦b}=move(a,b)
move(a,V){X↦a,Y↦b,V↦b}=move(a,b)
```

Unifier 2:

```
move(X,Y){X↦a,Y↦f(W),V↦f(W)}=move(a,f(W))
move(a,V){X↦a,Y↦f(W),V↦f(W)}=move(a,f(W))
```

Unifier 3:

```
move(X,Y){X↦a,Y↦V}=move(a,V)
move(a,V){X↦a,Y↦V}=move(a,V)
```

Most General Unifier

A substitution σ is *more general* than a substitution θ if and only if there is a substitution τ such that $\sigma \circ \tau = \theta$.

A substitution σ is a *most general unifier (mgu)* of two expressions if and only if it is more general than any other unifier.

Theorem: If two expressions are unifiable, then they have an mgu that is unique up to variable permutation.

$$move(X,Y)\{X \rightarrowtail a, Y \rightarrowtail V\} = move(a,V)$$
$$move(a,V)\{X \rightarrowtail a, Y \rightarrowtail V\} = move(a,V)$$

$$move(X,Y)\{X \rightarrowtail a, V \rightarrowtail Y\} = move(a,Y)$$
$$move(a,V)\{X \rightarrowtail a, V \rightarrowtail Y\} = move(a,Y)$$

Unification (1)

Terms encoded as lists: `[f,[g,x],c]` means `f(g(x),c)`

function *mgu(x,y,σ)*
begin
    if *x=y* **then return** *σ*;
    if *is_variable(x)* **then return** *mguvar(x,y,σ);*
    if *is_variable(y)* **then return** *mguvar(y,x,σ);*
    if not *is_list(x)* **then**
        if *is_list(y)* **or** *x≠y* **then return** *fail*
            else return *σ;*
    if not *is_list(y)* **then return** *fail;*
    σ := *mgu(head(x),head(y),σ);*
    if *σ=fail* **then return** *fail*
        else return *mgu(tail(x),tail(y),σ)*
end

Unification (2 – Straightforward but Incorrect)

function *mguvar(x,y,σ)*

begin

    if $x \rightarrowtail t \in \sigma$ **then return** *mgu(t,y,σ)*

    else return $\{x \rightarrowtail y\} \cup \sigma$

end

Problem: The Occur Check

`trajectory(X,X) <= ...`

Is this clause applicable to `trajectory(Y,neighbor-of(Y))` ?

Before assigning a variable to an expression, first check that the variable does not occur within that expression.

Unification (2 – Corrected Version)

function *mguvar(x,y,σ)*

begin

if $x \rightarrowtail t \in \sigma$ **then return** *mgu(t,y,σ);*

if *mgucheck(x,y,σ)* **then return** *fail*

else return $\{x \rightarrowtail y\} \cup \sigma$

end

Unification (3)

function *mgucheck(x,y,σ)*

begin
    if *x=y* **then return** *true;*
    if *is_variable(y)* **then**
        if *y$\rightarrowtail$ t$\in\sigma$* **then return** *mgucheck(x,t,σ)*
                else return *false;*
    if not *is_list(y) or y=*[ ] **then return** *false;*
    if *mgucheck(x,head(y),σ)* **then return** *true*
                else return *mgucheck(x,tail(y),σ)*

end

Example (1)

Call: *mgu*([move,X,b],[move,a,Y],{})

Call: *mgu*(move,move,{})
Exit: {}

Call: *mgu*(X,a,{})
Exit: {X↣a}

Call: *mgu*(b,Y,{X↣a})
Exit: {Y↣b,X↣a}

Exit: {Y↣b,X↣a}

Example (2)

Call: *mgu*(`[move,X,X],[move,a,b],{}`)

Call: *mgu*(`move,move,{}`)
Exit: `{}`

Call: *mgu*(`X,a,{}`)
Exit: `{X↣a}`

Call: *mgu*(`X,b,{X↣a}`)
Call: *mgu*(`a,b,{X↣a}`)
Exit: `fail`
Exit: `fail`

Exit: `fail`

Example (3)

Call: *mgu*([move,[f,X],[f,X]],[move,Y,[f,a]],{})
- **Call**: *mgu*(move,move,{})
- **Exit**: {}
- **Call**: *mgu*([f,X],Y,{})
- **Exit**: {Y↣[f,X]}
- **Call**: *mgu*([f,X],[f,a],{Y↣[f,X]})
  - **Call**: *mgu*(f,f,{Y↣[f,X]})
  - **Exit**: {Y↣[f,X]}
  - **Call**: *mgu*(X,a,{Y↣[f,X]}
  - **Exit**: {X↣a,Y↣[f,X]}
- **Exit**: {X↣a,Y↣[f,X]}

(Note that *mgu* does not return a true unifier but a succinct representation thereof!)

Example (4)

Call: *mgu*(`[move,X,X],[move,Y,[f,Y]],{}`)

Call: *mgu*(`move,move,{}`)

Exit: `{}`

Call: *mgu*(`X,Y,{}`)

Exit: `{X↣Y}`

Call: *mgu*(`X,[f,Y],{X↣Y}`)

Call: *mgu*(`Y,[f,Y],{X↣Y}`)

Call: *mgucheck*(`Y,[f,Y],{X↣Y}`)

Exit: `true`

Exit: `fail`

Exit: `fail`

Exit: `fail`

Linear Resolution Step

Given:

Query $L_1 \wedge L_2 \wedge ... \wedge L_m$ (without negation)

Clauses (without negation)

Let:

$A <= B_1 \wedge ... \wedge B_n$ "fresh" variant of a clause

σ mgu of L_1 and A

Then $L_1 \wedge L_2 \wedge ... \wedge L_m \rightarrow (B_1 \wedge ... \wedge B_n \wedge L_2 \wedge ... \wedge L_m)\sigma$

is a linear resolution step.

Example

Clauses:

```
true(cell(1,1,b))
   ⋮
true(cell(3,3,b))
true(control(xplayer))
legal(P,mark(X,Y)) <= true(cell(X,Y,b)) ∧ true(control(P))
legal(xplayer,noop) <= true(cell(X,Y,b)) ∧ true(control(oplayer))
legal(oplayer,noop) <= true(cell(X,Y,b)) ∧ true(control(xplayer))
```

Query `legal(P,M)` has answer substitutions

```
{P ↣ xplayer, M ↣ mark(1,1)}, ..., {P ↣ xplayer, M ↣ mark(3,3)}
{P ↣ oplayer, M ↣ noop}
```

Derivation

A sequence of resolution steps is called a derivation.

A successful derivation ends with the empty query.

The answer substitution (inferred by a successful derivation) is obtained by composing the mgu's $\sigma_1 \circ ... \circ \sigma_n$ of each step (and restricting the result to the variables in the original query).

A failed derivation ends with a query to which no clause applies.

Specific Predicate `distinct`(X,Y)

Given:

Query `distinct`$(t_1,t_2) \wedge L_2 \wedge \ldots \wedge L_m$ (without negation)

Clauses (without negation)

Let:

t_1 and t_2 variable-free and syntactically different

Then `distinct`$(t_1,t_2) \wedge L_2 \wedge \ldots \wedge L_m \rightarrow (L_2 \wedge \ldots \wedge L_m)\sigma$ is a linear resolution step.

May Need to Reorder Query Atoms

Clauses:

```
true(cell(1,1,x))
true(cell(1,2,b))
⋮
true(cell(3,3,b))
next(cell(M,N,W)) <= distinct(W,b) ∧ true(cell(M,N,W))
```

Query `next(F)` has answer substitution

```
{F ↦ cell(1,1,x)}
```

Linear Resolution with Negation

Given:

Query $L_1 \wedge L_2 \wedge ... \wedge L_m$

Clauses

- If L_1 is an atom, proceed as before
- If L_1 is of the form $\neg A$ and A is variable-free then
  - if all derivations for A fail then
    $L_1 \wedge L_2 \wedge ... \wedge L_m \rightarrow L_2 \wedge ... \wedge L_m$
  - if there is a successful derivation for A then
    $L_1 \wedge L_2 \wedge ... \wedge L_m \rightarrow$ fail

May Need to Reorder Query Atoms

Clauses:

```
role(red)
role(green)
role(blue)
true(protected(red))
legal(P,capture(Q)) <= ¬true(protected(Q)) ∧ role(P) ∧
                       role(Q) ∧ distinct(P,Q)
```

Query `legal(red,M)` has answer substitutions

`{M ↦ capture(green)}, {M ↦ capture(blue)}`

Example

Clauses:

```
role(red)
role(blue)
role(green)
true(freecell(blue))
trapped(P) <= role(P) ∧ ¬true(freecell(P))
goal(P,100) <= role(P) ∧ ¬trapped(P)
```

Query `goal(P,100)` has the only computed answer

`{P ↣ blue}`

Disjunction

A clause with a disjunction

$$A <= B \wedge (C_1 \vee C_2) \wedge D$$

is logically equivalent to the conjunction of the clauses

$$A <= B \wedge C_1 \wedge D$$
$$A <= B \wedge C_2 \wedge D$$

Example

Clauses:

```
true(p)
does(robot,b)
next(p) <=  does(robot,a) ∧ ¬true(p)
          ∨ does(robot,b) ∧ true(q)
next(q) <=  does(robot,a) ∧ true(q)
          ∨ does(robot,b) ∧ true(p)
```

Query `next(Fluent)` has the only answer substitution

`{Fluent ↣ q}`

Computing Answers

function *prove* (*query*,*clauses*)

/* return *true* if *query* has a successful derivation wrt. *clauses*
else return *false* */

function *findone* (*x*,*query*,*clauses*)

/* return computed answer for expression *x* of a successful derivation for *query* wrt. *clauses* */

function *findall* (*x*,*query*,*clauses*)

/* return the list of all answers for expression *x* for *query* wrt. *clauses* */

Example

Clauses:

```
true(cell(1,1,b))
   ⋮
true(cell(3,3,b))
true(control(xplayer))
legal(P,mark(X,Y)) <= true(cell(X,Y,b)) ∧ true(control(P))
legal(xplayer,noop) <= true(cell(X,Y,b)) ∧ true(control(oplayer))
legal(oplayer,noop) <= true(cell(X,Y,b)) ∧ true(control(xplayer))
```

findall (`M,legal(xplayer,M)`,*Clauses*) returns

`[mark(1,1), ..., mark(3,3)]`

Play this Game!

```
succ(1,2)  succ(2,3)  ...  succ(7,8)

init(cell(1,1))
init(cell(Y,1)) <= succ(X,Y) ∧ init(cell(X,1))
init(step(1))

next(step(Y)) <= true(step(X)) ∧ succ(X,Y)
next(cell(X,0)) <= does(you,jump(X,Y))
next(cell(Y,2)) <= does(you(jump(X,Y))
next(cell(X,C)) <= does(you,jump(Y,Z)) ∧
                   true(cell(X,C)) ∧
                   distinct(X,Y) ∧ distinct(X,Z)

terminal <= ¬continuable
continuable <= legal(you,M)
goal(you,100) <= true(step(5))
goal(you,0) <= true(cell(X,1))
```

```
legal(you,jump(X,Y)) <=
   true(cell(X,1)) ∧ true(cell(Y,1)) ∧
   ( obama(X,Y) ∨ obama(Y,X) )

clinton(X,Y) <= succ(X,Y)
clinton(X,Y) <= succ(X,Z) ∧ true(cell(Z,0))
                  ∧ clinton(Z,Y)
bush(X,Y) <= succ(X,Z) ∧ true(cell(Z,0))
                  ∧ bush(Z,Y)
bush(X,Y) <= succ(X,Z) ∧ true(cell(Z,1))
                  ∧ clinton(Z,Y)
obama(X,Y) <= succ(X,Z) ∧ true(cell(Z,0))
                  ∧ obama(Z,Y)
obama(X,Y) <= succ(X,Z) ∧ true(cell(Z,1))
                  ∧ bush(Z,Y)
obama(X,Y) <= succ(X,Z) ∧ true(cell(Z,2))
                  ∧ clinton(Z,Y)
```

Schedule

Lectures		Tutorials
Oct, 13th, 2008	Introduction	Oct, 20th, 2008
Oct, 27th, 2008	Game Description Language	Nov, 3rd, 2008
Nov, 10th, 2008	State-Space Search and Planning	Nov, 17th, 2008
Nov, 24th, 2008	Incomplete Information	Dec, 1st, 2008
Dec, 8th, 2008	Automated Reasoning	Dec, 15th, 2008
Jan, 5th, 2009	Metagaming	Jan, 12th, 2009
Jan, 26th, 2009	Game Theory	Feb, 2nd, 2009
End of March	Competition	