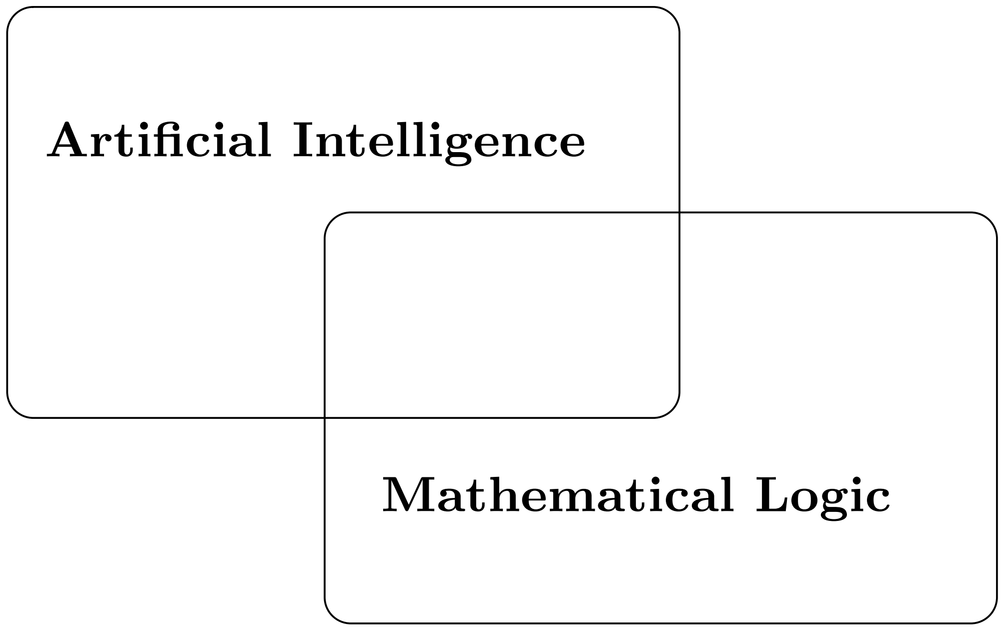
An Approach in Logic-based Artificial Intelligence

Joohyung Lee (joolee@asu.edu)

AI lab
Computer Science and Engineering Department
Arizona State University

My Research Interests



- Knowledge representation and reasoning
- Commonsense reasoning
- Logic programming (answer set programming)
- Nonmonotonic logics

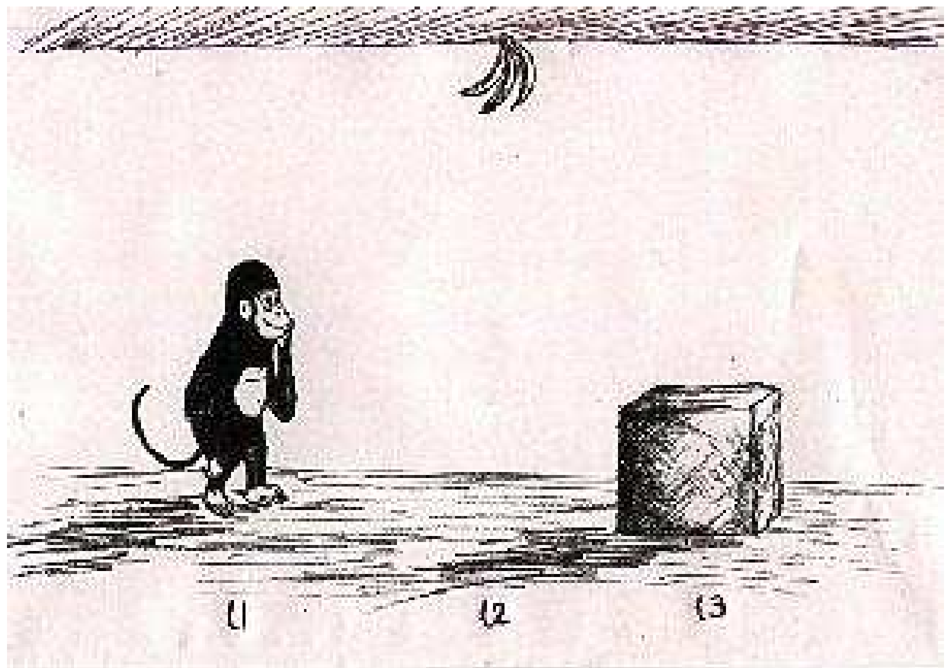
Alternative View

Or may be viewed as developing and applying declarative languages.

- High level language for knowledge representation: automated reasoning about actions.
- Low level language for logic programming: answer set programming.

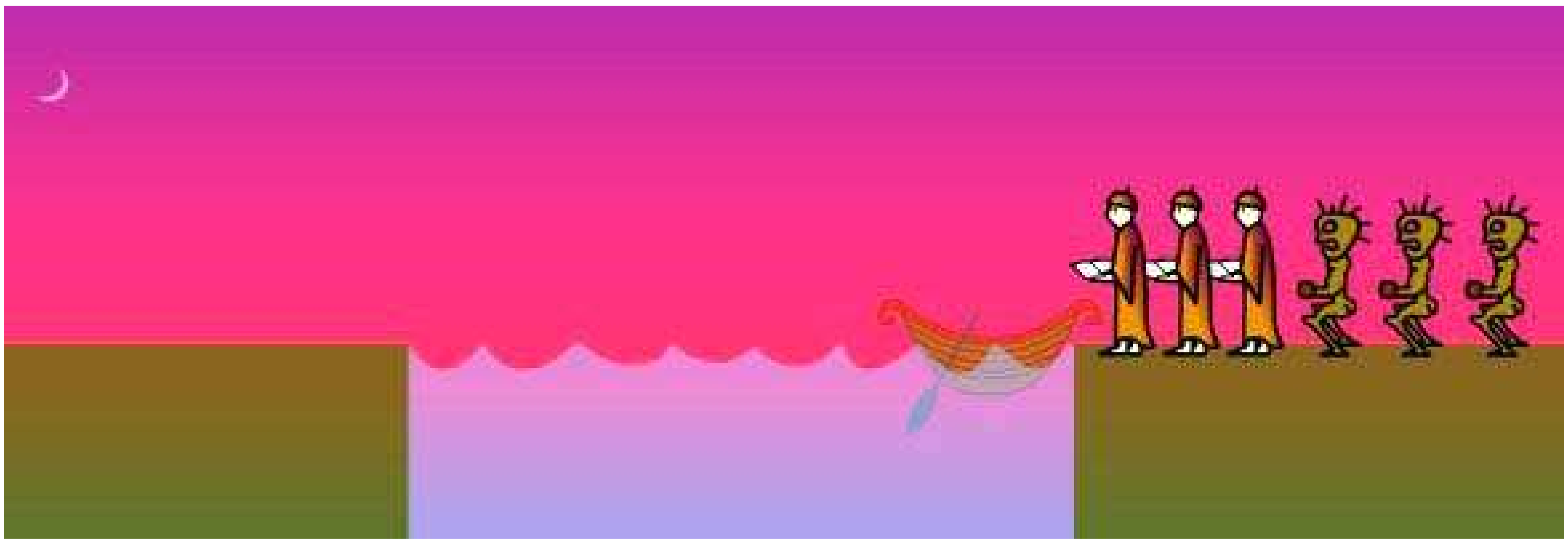
Automated Reasoning About Actions

Monkey and Bananas Problem



How can the monkey grasp the bananas?

Missionaries and Cannibals Puzzle



(From `http://www.plastelina.net/games/game2.html`)

How can they cross the river?

History

Advice taker
(McCarthy, "Programs with Commonsense," 1959)

Nonmonotonic logics appeared (1980)

Satisfiability planning (1992)

McCain-Turner causal logic and CCALC 1 (1997)
Action language $\mathcal{C}$ (1998)

Action Language $\mathcal{C}+$ and CCALC 2 (2004)

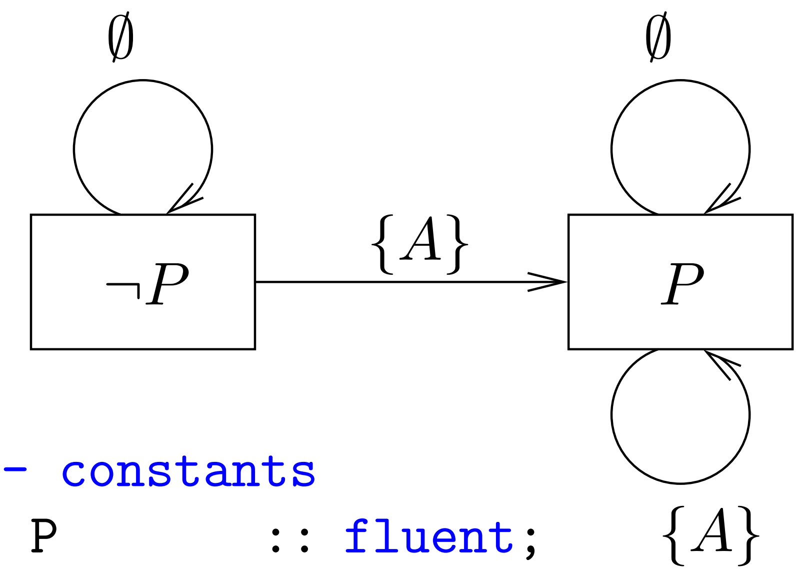
My Dissertation

- Identify essential limitations of $\mathcal{C}$ and overcome them — action language $\mathcal{C}+$.
- Extend CCalc in accordance with the theoretical extension and make it more convenient to use — CCalc 2.
- Apply CCalc to more difficult examples of commonsense reasoning.

Action Languages

- Formal models of parts of natural language that are used for describing the effects of actions.
- Define "transition systems"—directed graphs whose vertices correspond to states and whose edges are labeled by actions.
- – STRIPS [Fikes and Nilsson, 1971]
  – ADL [Pednault, 1994]
  – $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998]
  – $\mathcal{C}+$ [Giunchiglia, Lee, Lifschitz, McCain and Turner, 2004]

Transition Systems and Language $\mathcal{C}+$




```
:- constants
  P         :: fluent;
  A         :: action.

inertial P.
exogenous A.
A causes P.
```

Fluent: anything that depends on the state of the world.

Monkey and Bananas in CCALC — A Part

```
walk(L) causes at(monkey,L).
nonexecutable walk(L) if at(monkey,L).
nonexecutable walk(L) if onBox.

inertial loc(monkey).

caused loc(bananas)=L if hasBananas & loc(monkey)=L.
```

How to Get the Bananas?

```
% Calling mChaff spelt3...  done.
% Reading output file(s) from SAT solver...  done.
% Solution time:  0.01 seconds

0: loc(monkey)=loc1  loc(bananas)=loc2  loc(box)=loc3

ACTIONS: walk(loc3)

1: loc(monkey)=loc3  loc(bananas)=loc2  loc(box)=loc3

ACTIONS: pushBox(loc2)

2: loc(monkey)=loc2  loc(bananas)=loc2  loc(box)=loc2

ACTIONS: climbOn

3: onBox  loc(monkey)=loc2  loc(bananas)=loc2  loc(box)=loc2

ACTIONS: graspBananas

4: hasBananas  onBox  loc(monkey)=loc2  loc(bananas)=loc2
   loc(box)=loc2
```

Syntax of $\mathcal{C}+$

⟨Causal Law⟩ ::= caused ⟨Formula⟩ if ⟨Formula⟩ [after ⟨Formula⟩]

```
walk(L) causes loc(monkey)=L.
nonexecutable walk(L) if onBox.
inertial loc(monkey).
```

stand for

```
caused loc(monkey)=L if true after walk(L).
caused false if true after onBox & walk(L).
caused loc(monkey)=L if loc(monkey)=L after loc(monkey)=L.
```

Elaboration Tolerance

- "A formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances.

  The simplest kind of elaboration is the addition of new formulas." [McCarthy, *Elaboration Tolerance*, 1999].

- – Only one missionary and one cannibal can row.
  – Three missionaries alone with a cannibal can convert him into a missionary.
  – One of the missionaries is Jesus Christ (He can walk on water).
  – There are four missionaries and four cannibals.

The Causal Calculator (CCALC) Version 2

- Implementation of $\mathcal{C}+$.
- Provides solutions for the frame and the ramification problems.
- Can represent actions with conditional effects, nondeterministic actions, and concurrently executed actions.
- Multi-valued fluents.
- Definitions of new fluents. $\surd$
- Defeasible causal laws. $\surd$
- Attributes. $\surd$
- Additive fluents.
- Rigid constants.
- Action dynamic laws. $\surd$
- Verification of invariants.

McCarthy's Elaboration No 4

"The boat can carry three. Four [pairs] can cross but not five."

```
% File 'basic'
...
constraint capacity(boat)=2 unless ab5.
```

```
% File 'jmc4'
:- include 'basic'.

caused ab5.
constraint capacity(boat)=3 unless ab6.
```

Formalizing Medium-size Action Domains

Formalized the Traffic World proposed as part of the Logic Modeling Workshop.

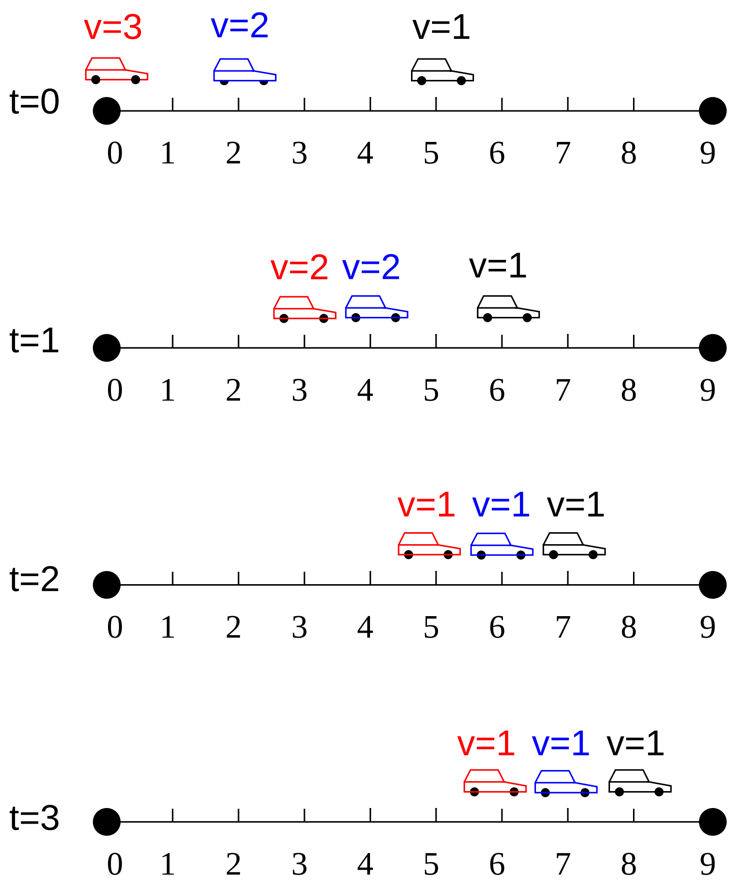
> The Traffic World includes cars traveling on road segments at the maximum velocity as allowed while respecting restrictions such as speed limits and the surrounding traffic condition (a car is not allowed to get too close to the car in front of it).

Formalizing the Traffic World

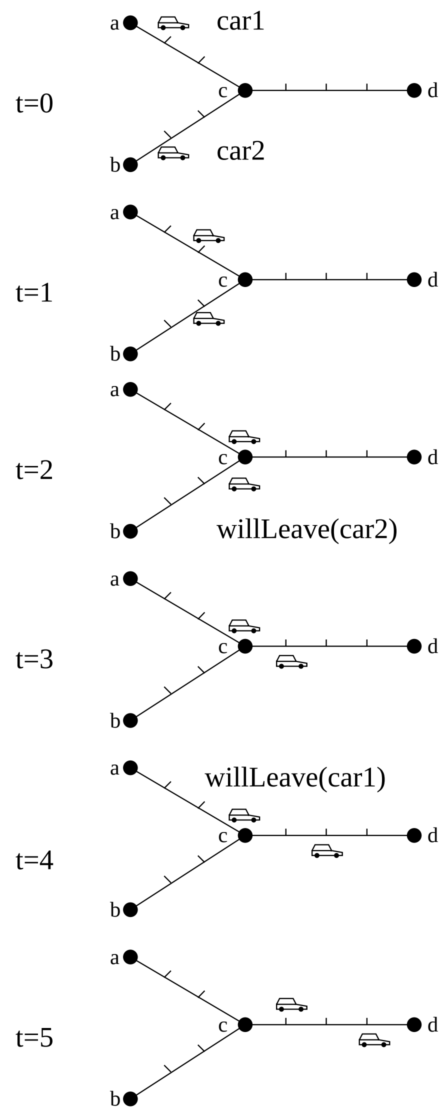
```
% The distance covered by a car which remained on the same
%  segment
caused distance(C)=Ds+Sp
   after distance(C)=Ds & speed(C)=Sp.

% No two cars on the same segment and having the same
%  orientation can be closer than varsigma (lmw)
constraint position(C,Sg,Ds) & position(C1,Sg,Ds1)
      ->> orientation(C)\=orientation(C1)
    where C\=C1 & abs(Ds1-Ds)<varsigma.
```

CCALC can predict what happens if cars behind are moving faster.

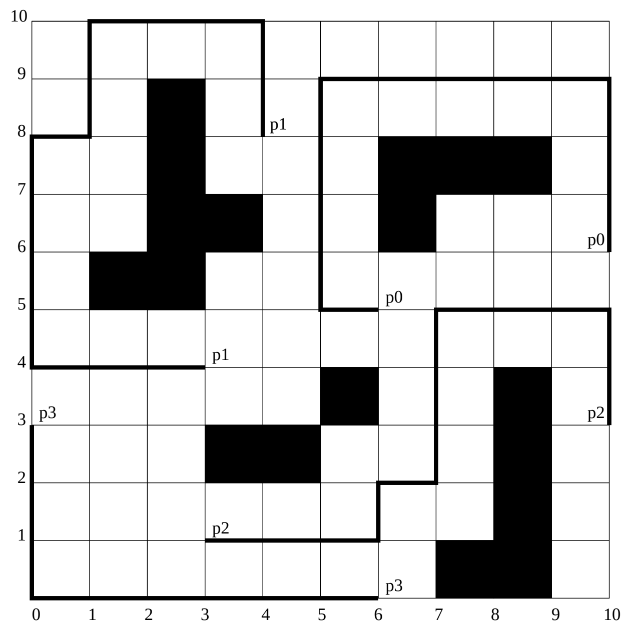


What happens if two cars are arriving at a junction at the same time?



Wire Routing

Wire routing is the problem of determining the physical locations of all wires interconnecting the circuit components on a chip [Erdem, Lifschitz, Wong, CL-2000].



The effect of moving right:

```
move(N,right) causes at_x(N,X+1) if at_x(N,X).
```

The points visited by a robot includes its current position and all points it had visited by the previous time instant:

```
caused occupied(N,X,Y) if at(N,X,Y).
caused occupied(N,X,Y) after occupied(N,X,Y).
```

The paths of different robots don't intersect:

```
constraint occupied(N,X,Y) ->> -occupied(N1,X,Y)
      where N \= N1.
```

A robot never visits the same point twice:

```
caused false if at(N,X,Y)
      after occupied(N,X,Y) & -at(N,X,Y).
```

Verification of Security Protocols

Consider a simple one-way authentication protocol:

$$(1)\ Alice \rightarrow Bob : Alice, \{N\}K\textsf{\textit{ab}}$$
$$(2)\ Bob \rightarrow Alice : \{f(N)\}K\textsf{\textit{ab}}$$

Verification of Security Protocols

Consider a simple one-way authentication protocol:

$$(1)\ Alice \rightarrow Bob : Alice, \{N\}K\mathsf{ab}$$
$$(2)\ Bob \rightarrow Alice : \{f(N)\}K\mathsf{ab}$$

$$(1.1)\ Alice \rightarrow Ivory : Alice, \{N\}K\mathsf{ab}$$
$$(2.1)\ Ivory \rightarrow Alice : Bob, \{N\}K\mathsf{ab}$$
$$(2.2)\ Alice \rightarrow Ivory : \{f(N)\}K\mathsf{ab}$$
$$(1.2)\ Ivory \rightarrow Alice : \{f(N)\}K\mathsf{ab}$$

Uses of CCalc

CCalc has been used to formalize

- McCarthy's elaborations of the Missionaries and Cannibals Puzzle
- Medium-size action domains proposed by Erik Sandewall
- Wire routing in VLSI domain
- Verification of Security Protocols (preliminary work)

CCalc was also used by other researchers outside Austin:

- *Specifying Electronic Societies with the Causal Calculator.*
- *An Executable Specification of an Argumentation Protocol.*
- *Nonmonotonic commitment machines.*
- *Specification of Workflow Process Using the Action Description Language C*

Extension of $\mathcal{C}+$

- $(\mathcal{C}/\mathcal{C}+)^{++}$ [Sergot, *An Action Language for Representing Norms and Institutions*, 2004]
- $P\mathcal{C}+$ [Eiter and Lukasiewicz, *Probabilistic Reasoning about Actions in Nonmonotonic Causal Theories*, 2003]
- $G\mathcal{C}+$ [Finzi and Lukasiewicz, *Game-theoretic Reasoning about Actions in Nonmonotonic Causal Theories*, 2005]

Answer Set Programming

Answer Set Programming

A form of declarative programming.

Represents a given combinatorial search problem by a logic program whose "answer sets" correspond to solutions, and then use an answer set solver to find an answer set for this program.

Brief History of Answer Set Programming

1988: Answer set semantics for Prolog programs.

1996: SMODELS: first answer set solver.

1999: ASP identified as a new programming paradigm.

2005: Answer set semantics for propositional formulas.

Efficient solvers are available: SMODELS, DLV, ASSAT, CMODELS, ASPPS, ...

WASP (Working Group on Answer Set Programming) : 17 European universities in 8 countries. Funded by EU.

Example: Estimating Schur numbers

A set S of integers is called *sum-free* if there are no numbers x, y in S such that $x + y$ is in S. For instance, $\{1, 3, 5\}$ is sum free; $\{2, 3, 5\}$ and $\{2, 4\}$ are not.

The Schur number $S(k)$ is the largest integer n for which the interval $\{1, \ldots, n\}$ can be partitioned into k sum-free sets.

Is it possible to partition $\{1, \ldots, n\}$ into 3 sum-free subsets?

Solution for $n = 13$:

$$\{2, 3, 11, 12\}, \{5, 6, 7, 8, 9\}, \{1, 4, 10, 13\}.$$

The Schur number $S(3)$ is 13.

Input:

```
subset(1..k).
number(1..n).
#domain number(X;Y).

1{s(X,I) : subset(I)}1.
:- s(X,I), s(Y,I), s(X+Y,I), subset(I), X+Y<=n.
```

```
> lparse -c k=3 -c n=13 -d none schur | smodels
```

Output:

```
Stable Model: s(1,3) s(2,1) s(3,1) s(4,3) s(5,2) s(6,2)
s(7,2) s(8,2) s(9,2) s(10,3) s(11,1) s(12,1) s(13,3)
```

which represents a partition of $\{1, \ldots, 13\}$ into 3 sum-free sets:

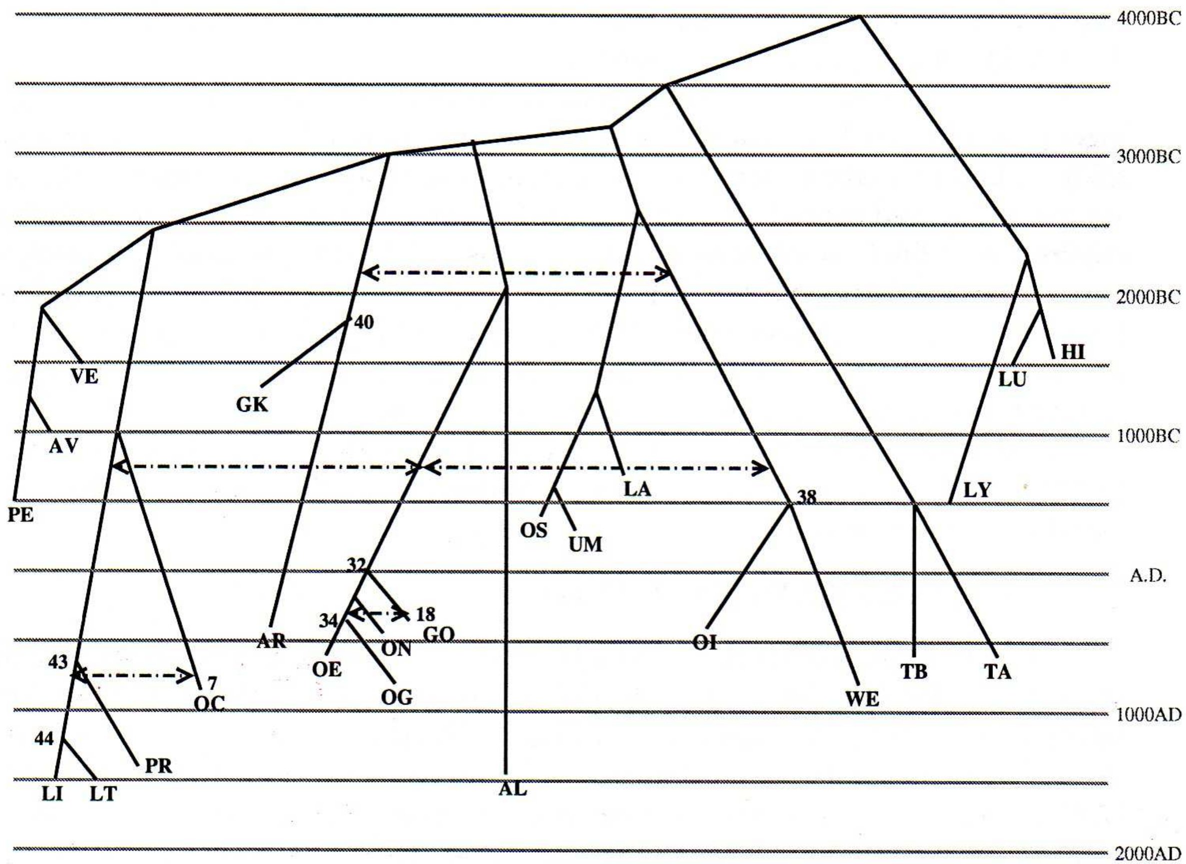
$$\{2, 3, 11, 12\} \cup \{5, 6, 7, 8, 9\} \cup \{1, 4, 10, 13\}.$$

Some Applications

- *Answer Set Programming and Plan Generation.*
- *A Logic Programming Approach to Knowledge-state Planning.*
- *Reconstructing the Evolutionary History of Indo-European Languages Using Answer Set Programming.*
- *Character-based Cladistics and Answer Set Programming.*
- *Rectilinear Steiner Tree Construction Using Answer Set Programming.*

- *An A-Prolog Decision Support System for the Space Shuttle* (RCS/USA-Advisor : decision support system for shuttle controllers).
- *Developing a Declarative Rule Language for Applications in Product Configuration* (variantum : spin-off).
- *Bounded LTL Model Checking with Stable Models.*
- *Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets.*
- *Data Integration: A Challenging ASP Application.*

Reconstructing the Evolutionary History of Indo-European Languages Using Answer Set Programming, (Erdem, Lifschitz, Nakhleh, Ringe, PADL-03)



How to Turn Logic Programs into Propositional Logic?

Theorem on loop formulas The answer sets of a program are exactly the models of the program and loop formulas.

Π_1	Π_1	$\cup$ $LF(\Pi_1)$
$p \leftarrow q$	$q \supset p$	$p \supset q$
$q \leftarrow p$	$p \supset q$	$q \supset p$
$r \leftarrow \textit{not}\ p$	$\neg p \supset r$	$r \supset \neg p$
		$p \vee q \supset \bot$

Study on Loop Formulas

- Loop formulas for normal logic programs
  [Lin and Zhao, AAAI-02].
- Loop formulas for disjunctive logic programs
  [Lee and Lifschitz, ICLP'03]
- Loop formulas for McCain–Turner causal logic
  [Lee, LPNMR'04]
- Loop formulas for circumscription
  [Lee and Lin, AAAI'04]
- A model-theoretical account of loop formulas
  [Lee, IJCAI'05]
- Elementary loops
  [Gebser and Schaub, LPNMR'05]

Why ASP is a Good Formalism for Knowledge Representation?

Any equivalent translation from logic programs to propositional formulas involves a significant increase in size assuming a plausible conjecture ($P \not\subseteq NC^1/poly$) [Lifschitz and Razborov, "Why are there so many loop formulas?", ACM TOCL, to appear].

> *How succinctly can the formalism express the set of models that it can? . . . [W]e consider formalism A to be stronger than formalism B if and only if any knowledge base in B has an equivalent knowledge base in A that is only polynomially longer, while there is a knowledge base in A that can be translated to B only with an exponential blowup.* [Gogic, Kautz, Papadimitriou, Selman, IJCAI-95]

Spring 2006 course

CSE 591G Answer Set Programming : an Approach to Declarative Problem Solving

MW 4:40 - 5:55 (BYAC260)