Justifying Proofs using Memo Tables

Abhik Roychoudhury

C.R. Ramakrishnan

I.V. Ramakrishnan

SUNY Stony Brook

PPDP '00, Montréal, Canada

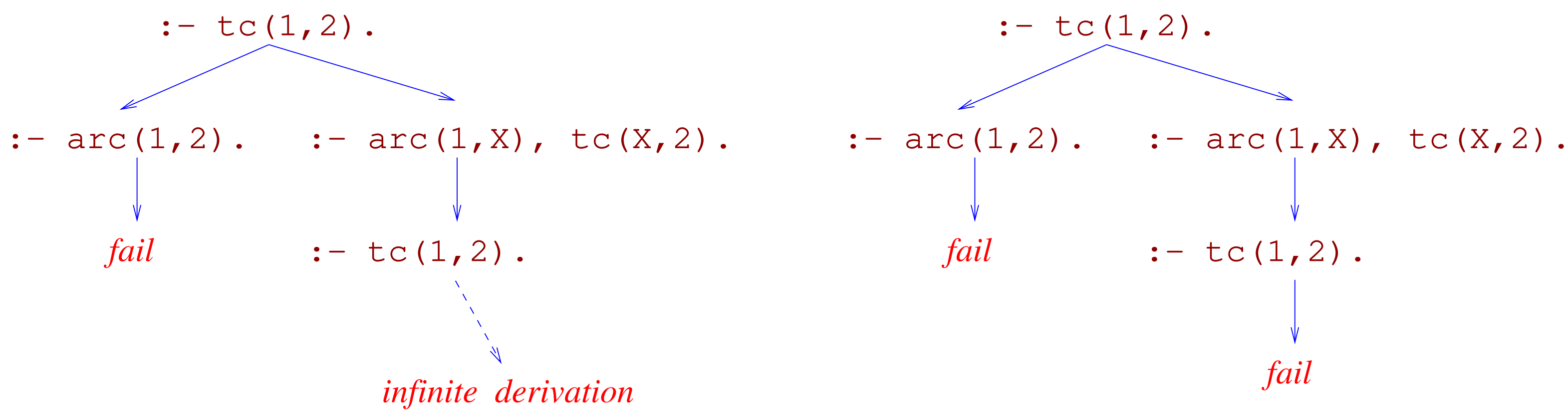
Overview

- Encoding: Tableau based proof systems can be directly encoded as a logic program. :-)
- Evaluation: Memoized evaluation of a proof obligation amounts to searching for the existence of a tableau for the proof obligation. :-)
- Evidence Generation: Proof tree constructed by LP engine is not useful as evidence. :-(
- Application: Evidence to explain results of *model checking* concurrent systems.

Use the memo tables a-posteriori *justify* the results of LP evaluation.

Memoized LP evaluation

```
tc(A,B) :- arc(A,B).
tc(A,B) :- arc(A,C), tc(C,B).
arc(1,1).
```



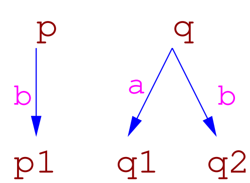
- Remembers the calls encountered and the answers produced in memo tables.
- Gives the power to compute *fixed points*.

Simple Example: Bisimulation Checking

$$(1) \qquad \frac{p' \not\approx q'_1, \ldots, p' \not\approx q'_n}{p \not\approx q} \qquad \exists a\ p \xrightarrow{a} p' \wedge \{q'_1, \ldots, q'_n\} = \{q' \mid q \xrightarrow{a} q'\}$$

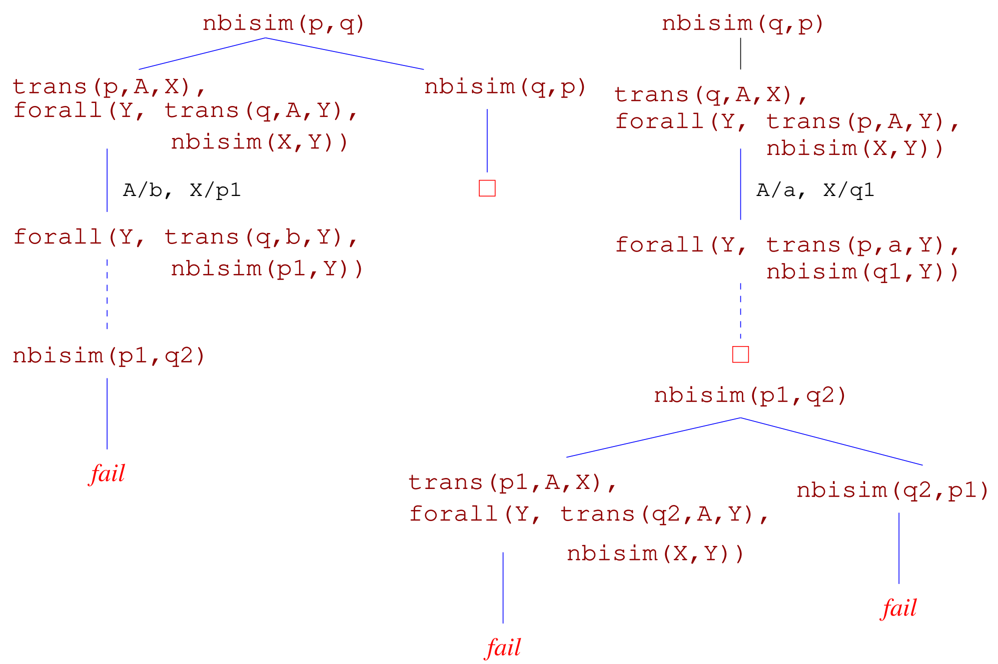
$$(2) \qquad \frac{q \not\approx p}{p \not\approx q}$$

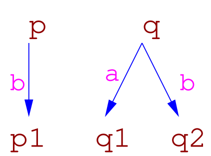
```
:- table nbisim/2.
nbisim(P,Q) :- trans(P,A,P1),
               forall(Q1, trans(Q,A,Q1), nbisim(P1,Q1)).
nbisim(P,Q) :- nbisim(Q,P).
```



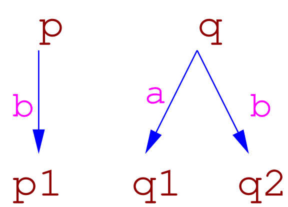

```
?- nbisim(p,q).
```

Evidence Generation : What we get





Evidence Generation : What we want



$$
\begin{array}{c}
p \not\approx q \\
\downarrow \\
q \not\approx p \\
\downarrow \; q \xrightarrow{a} q1, \quad \{p' \mid p \xrightarrow{a} p'\} = \{\} \\
\color{red}{fact}
\end{array}
$$

Justifying Successful Queries

Query | `?- p.`

```
p :- costly, q.
p :- r.
q :- p.
r.
costly :- ...
```

Program

p	*true*
q	*true*
r	*true*
costly	*true*
...	...

Tables

p

↓

r

↓

fact

Justification

Justifying Failed Queries

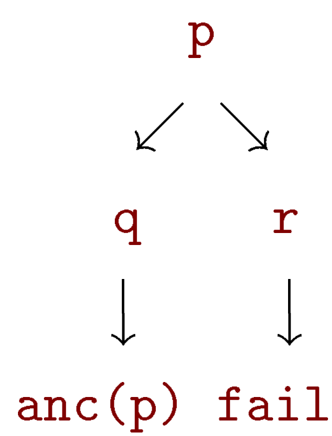
Query | `?- p.`

```
p :- costly, q.
p :- r.
q :- p.
costly :- ...
```

Program

`p`	*false*
`q`	*false*
`r`	*false*
`costly`	*true*
...	...

Tables



Justification

Justification: Algorithm

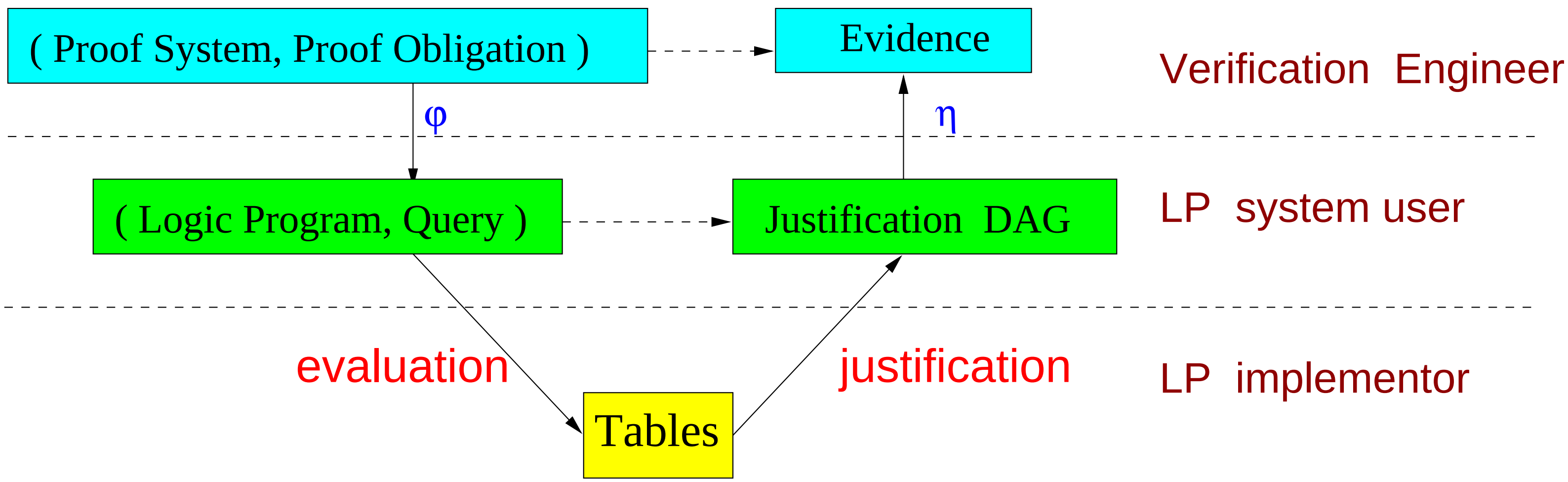
Explain an atom A with:

- True : Any unifying clause C of A s.t.
  - C has true body
  - C does not produce cycle.
- False:
  - $\emptyset$ if A leads to a cycle.
  - A false literal in every unifying clause, otherwise

Complexity : $O(|T|^2)$, $|T|$ = Table size

Extended to explaining well-founded model computation for normal LP.

Justification : Software Architecture



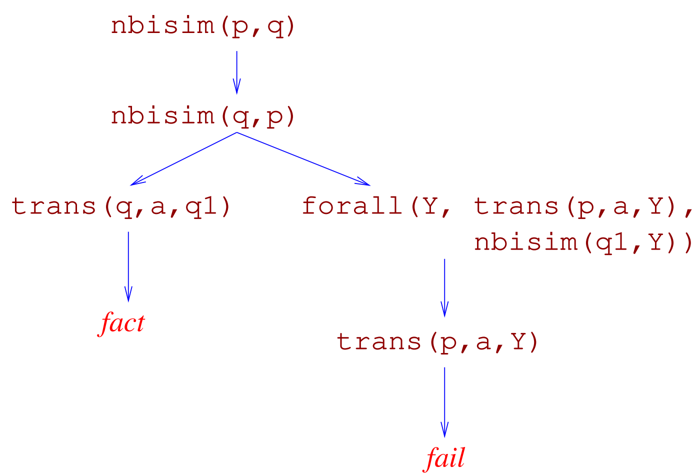
Extracting high-level Evidence

$$\frac{p' \not\approx q'_1, \ldots, p' \not\approx q'_n}{p \not\approx q}$$

$$\frac{q \not\approx p}{p \not\approx q}$$

$\xrightarrow{\varphi}$

```
nbisim(P,Q) :- trans(P,A,Q), ...
nbisim(P,Q) :- nbisim(Q,P).
```



$\xrightarrow{\eta}$

$$\begin{tikzcd}
p \not\approx q \arrow[d] \\
q \not\approx p \arrow[d, "q \xrightarrow{a} q1, \quad \{p' \mid p \xrightarrow{a} p'\} = \{\}"] \\
fact
\end{tikzcd}$$

Application: Model Checking

$$\mathtt{s} \overset{?}{\models} \mathtt{f}$$

- Encoding: For a process language and temporal logic, encode the $\models$ relation as a logic program predicate `models` s.t.

$$\forall \mathtt{s} \forall \mathtt{f}\ \mathtt{models}(\mathtt{s},\mathtt{f}) \equiv \mathtt{s} \models \mathtt{f}$$

- Evaluation: For a given system state `s` and property `f`, perform model checking by evaluating the query `models(s,f)`.
- Evidence Generation: Explain the truth or falsehood of $\mathtt{s} \models \mathtt{f}$ by justifying the query `models(s,f)`

XMC model checker

- Built at SUNY Stony Brook in 1997, on top of XSB tabled LP system.
- Encoding of `models` obtained directly from semantic rules of:
  - process language in which systems are specified (similar to value-passing CCS)
  - formulae are specified in the branching time temporal logic (alternation free modal μ calculus)
- Time performance comparable to highly optimized model checkers like Spin and Murφ.

First release: Summer 2000
Software available: `http://www.cs.sunysb.edu/~lmc`

Justifying Model Checking results

- Counterexample Generation : Existing model checkers generate a counterexample trace(tree) if the property is not satisfied.
- Justification : Generates evidence of truth/falsehood of a query.

Thus :

- Given the LP encoding, justification explains both truth/falsehood of a temporal property.
- Not restricted to specific temporal logics *e.g.* linear/branching time.

XMC Justifier

- A justifier for tabled LP hooked up to XMC model checker.
- Distributed with the first release of XMC.
- Used to explain truth/falsehood of temporal properties in:
  - GNU UUCP i-protocol
  - Sun Microsystems Java Metalocking Algorithm

System	Formula	Run-time Trace	Justification
Metalock(2,1)	Mutex	6400	890
Metalock(2,2)	Mutex	103K	14K
i-protocol(1)	Livelock	83K	2K
i-protocol(2)	Livelock	602K	13K

Related Work: LP, DDB, Deduction

- Algorithmic Debugging: *Tracing* SLD resolution proof search.
- Declarative Debugging: Finding a false program clause in intended model (*given by user*).
- Assertion based debugging: Static and Dynamic checking of *user provided* assertions.
- Deductive Databases: Providing explanations at different levels of detail from the *execution trace*.
- Elf meta-language: Search automatically constructs proofs *during* query evaluation.

Salient Features of Justification

- Constructed after query evaluation.
- Imposes no additional overhead to proof search.
- Requires no creative input from user.
- Can provide explanations at different levels of abstraction.
- Much more succinct than entire proof search.
- Generated evidence can be navigated on demand.

Application in Programming: Building Parse Trees

- Tabled LP allows for declarative encoding and evaluation of CFGs. [Warren:99]. Worst case complexity : $O(n^3)$, n = length of string.

```
S -> Bc.
B -> BB.
B -> a.
```

```
:- auto_table.
s --> b,[c].
b --> b,b.
b --> [a].
```

- Building parse trees by simply adding a parameter deteriorates complexity *e.g.* the string a^n.

```
:- auto_table.
s(s1(B,c)) --> b(B),[c].
b(b1(B1,B2)) --> b(B1),b(B2).
b(b2(a)) --> [a].
```

Justification : A Programming Abstraction

- *Search* and *Construct*.
  1. Encode the grammar directly.
  2. Query Evaluation to parse a given string.
  3. Justification to construct the parse tree.
- Complexity is still $O(n^3)$.
- Avoids complex encoding tricks.

Justification: The three *AVATARS*

- Evidence Generation: Generate evidence for absence/presence of tableaus for a proof system and a proof obligation.
- Debugging: Generic technique for debugging tabled LP.
- Programming Abstraction: Search and Construct.