Breaking out of the Dark Ages of Computer Programming: The REASON Family of Logic-Based Programming Languages

Selmer Bringsjord & Konstantine Arkoudas

9.27.06

$\mathcal{R}$

Breaking out of the Dark Ages of Computer Programming: The REASON Family of Logic-Based Programming Languages

Selmer Bringsjord & Konstantine Arkoudas

9.27.06

Warning:

Warning:

nascent

Warning:

nascent

short talk

Warning:

nascent

short talk

attacks the *status quo*

Warning:

nascent

short talk

attacks the *status quo*

focus on *teaching* programming

Warning:

nascent

short talk

attacks the *status quo*

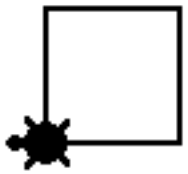
focus on *teaching* programming

no discussion of Slate-ish workbench interface

The LOGO-MIT Mistake

http://el.media.mit.edu/Logo-foundation/logo/turtle.html

```
forward 50
```



```
to square
repeat 4 [forward 50 right 90]
end
```

Constructivism is the exact opposite of how best to teach kids how to think, reason, problem-solve.

Constructivism is the exact opposite of how best to teach kids how to think, reason, problem-solve.

Deep, context-independent reasoning capacity is the thing to teach — and the thing an abstract, symbol-driven marketplace rewards.

More generally, this capacity is the essence of (academic) intelligence, as psychometrics shows.

We do not focus on writing programs to simulate dumb animals and draw pictures.

We do not focus on writing programs to simulate dumb animals and draw pictures.

We focus on writing programs to solve problems that call for first-rate, distinctively human thinking:

Mystery 1

Here are your two clues:

If Billy is a monkey, Jane stole the cookie.

Jane stole the cookie!

Do you know whether Billy is a monkey?

Mystery 2

Here are your two clues:

If Billy is a monkey, Jane stole the cookie.

Jane did not steal the cookie!

Do you know whether Billy is a monkey?

Mystery 3

All the Frenchmen in the restaurant are gourmets.

Some of the gourmets are wine drinkers.

Does it follow that some of the Frenchmen are wine drinkers?

Mystery 4

The power set of the null set is the singleton set whose member is the null set.

True? False? Prove that you are correct.

Mystery 5

Assume the following two propositions are true:

(1) Everyone loves anyone who loves someone.

(2) Alvin loves Bill.

Now, given (1) and (2), does it follow that:

(3) Everyone loves Bill.
(4) Katherine loves Bill.
(5) Katherine loves Dave.

Students should learn how to program by working collaboratively with REASON to solve such mysteries.

Students should learn how to program by working collaboratively with REASON to solve such mysteries.

Such students would be better prepared to succeed in algebra (where the US math debacle begins) because they would be better prepared to grasp the underlying structure of surface-level problems, which is what counts:

Monkeys and cookies and wine drinking and loving — these things have nothing to do with the mysteries we just looked at!

Three Programming Paradigms

Three Programming Paradigms

- Procedural/Imperative
- Functional
- Logic Programming/Declarative

Three Programming Paradigms

- Procedural/Imperative

K-12, historical accident, and a bad thing.

- Functional
- Logic Programming/Declarative

Turing's Mind-Numbing Procedural Approach

Turing Machines

http://ironphoenix.org/tril/tm

Procedural Languages in Computability

Subtracting 1 from a variable: `V <- V - 1`

Adding 1 to a variable: `V <- V + 1`

Moving by conditional to a line labeled *L* in a program:

`If V != 0 GOTO L`

```
Y <- X1
Z <- X2
[B] IF Z != 0 GOTO A
GOTO E
[A] Z <- Z - 1
Y <- Y + 1
GOTO B
```

Procedural Languages in Computability

Subtracting 1 from a variable: `V <- V - 1`

Adding 1 to a variable: `V <- V + 1`

Moving by conditional to a line labeled *L* in a program:

`If V != 0 GOTO L`

```
Y <- X1
Z <- X2
[B] IF Z != 0 GOTO A
GOTO E
[A] Z <- Z - 1
Y <- Y + 1
GOTO B
```

The underlying picture is a Register machine: very turtle-like.

Functional Paradigm

Not going to discuss the lambda calculus, ML, Scheme, "purely functional" Common Lisp, etc.

Will simply say:

Better (and here MIT doesn't fall flat: Abelson), but code that computes a mechanical function is still, by my lights, disturbingly disconnected from human-level thinking, and the approach is plagued by the defects of the procedural one.

REASON's Roots

Logic Programming

Horn formulas only:

$$\frac{}{\neg\phi_1 \vee \ldots \vee \neg\phi_n \vee \phi} \text{ where } n \in N, \phi_i, \phi \text{ atomic}$$

$$\frac{}{\neg\phi_0 \vee \ldots \vee \neg\phi_n} \text{ where } n \in N, \phi_i \text{ atomic}$$

$$\frac{\phi, \psi}{(\phi \wedge \psi)} \qquad \frac{\phi}{\exists x \phi} \qquad \frac{\phi}{\forall x \phi}$$

Logic Programming

Assert Φ

Logic Programming

Assert Φ

Query: $\Phi \vdash \exists x_1 \ldots \exists x_m (\phi_0 \wedge \ldots \wedge \phi_l)$?

Logic Programming

Assert Φ

Query: $\Phi \vdash \exists x_1 \ldots \exists x_m (\phi_0 \wedge \ldots \wedge \phi_l)$?

$GI(\Phi)$ and $(\neg\phi_0 \vee \ldots \vee \neg\phi_l)(\frac{m}{x}|\frac{m}{t})$

$\downarrow$

$\emptyset$

Logic Programming

Assert Φ

Query: $\Phi \vdash \exists x_1 \ldots \exists x_m (\phi_0 \wedge \ldots \wedge \phi_l)$?

$GI(\Phi)$ and $(\neg\phi_0 \vee \ldots \vee \neg\phi_l)(\frac{m}{x}|\frac{m}{t})$

$\downarrow$

$\emptyset$

based on *m* particular terms

$(\neg\phi_0 \vee \ldots \vee \neg\phi_l)(\frac{m}{x}|\frac{m}{t})$

Logic Programming

Assert Φ

Query: $\Phi \vdash \exists x_1 \ldots \exists x_m (\phi_0 \wedge \ldots \wedge \phi_l)$?

$GI(\Phi)$ and $(\neg\phi_0 \vee \ldots \vee \neg\phi_l)(\frac{m}{x}|\frac{m}{t})$

$\downarrow$

$\emptyset$

based on *m* particular terms $(\neg\phi_0 \vee \ldots \vee \neg\phi_l)(\frac{m}{x}|\frac{m}{t})$

So you get the terms back as an answer.

Problems Infecting Logic Programming

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...
- LP is based on a fragment of full first-order logic: it's inexpressive.

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...
- LP is based on a fragment of full first-order logic: it's inexpressive.
- LP is lazy.

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...
- LP is based on a fragment of full first-order logic: it's inexpressive.
- LP is lazy.
  - What do you do? Do you create any proofs, e.g.? Do you create any models? Etc.

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...
- LP is based on a fragment of full first-order logic: it's inexpressive.
- LP is lazy.
  - What do you do? Do you create any proofs, e.g.? Do you create any models? Etc.
- You can't get back a proof.

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...
- LP is based on a fragment of full first-order logic: it's inexpressive.
- LP is lazy.
  - What do you do? Do you create any proofs, e.g.? Do you create any models? Etc.
- You can't get back a proof.
- You can't get back a model/countermodel.

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...
- LP is based on a fragment of full first-order logic: it's inexpressive.
- LP is lazy.
  - What do you do? Do you create any proofs, e.g.? Do you create any models? Etc.
- You can't get back a proof.
- You can't get back a model/countermodel.
- You're locked into an obnoxious mode of deductive reasoning: resolution.

Problems Infecting Logic Programming

- Some of the basic scheme is right, as REASON shows (as shall see momentarily), but...
- LP is based on a fragment of full first-order logic: it's inexpressive.
- LP is lazy.
  - What do you do? Do you create any proofs, e.g.? Do you create any models? Etc.
- You can't get back a proof.
- You can't get back a model/countermodel.
- You're locked into an obnoxious mode of deductive reasoning: resolution.
- ...

Denotational Proof Languages

- Further down the road toward REASON.
- E.g., NDL (type-alpha) and Athena (type-omega), both in serious use at RPI and a number of other institutions.
- Invented by Konstantine Arkoudas.
- Emphasis is on the human engineering a proof, and calling an oracle to do low-level reasoning.
- Methods can also be created: take input and produce a proof.
- And Athena can be used as a functional programming language.

Denotational Proof Languages

- Further down the road toward REASON.
- E.g., NDL (type-alpha) and Athena (type-omega), both in serious use at RPI and a number of other institutions.
- Invented by Konstantine Arkoudas.
- Emphasis is on the human engineering a proof, and calling an oracle to do low-level reasoning.
- Methods can also be created: take input and produce a proof.
- And Athena can be used as a functional programming language.

Denotational Proof Languages

- Further down the road toward REASON.
- E.g., NDL (type-alpha) and Athena (type-omega), both in serious use at RPI and a number of other institutions.
- Invented by Konstantine Arkoudas.
- Emphasis is on the human engineering a proof, and calling an oracle to do low-level reasoning.
- Methods can also be created: take input and produce a proof.
- And Athena can be used as a functional programming language.

Consider the following, theorem, so-called Theorem 89 (Th 89) from Patrick Suppes' textbook *Axiomatic Set Theory*, penned in 1960 to *introduce* set theory to its readers. Th 89 says simply that the power set of the null set is the set composed of just the null set (our Mystery 4).

$$\mathcal{P}(\emptyset) = \{\emptyset\}$$

The greatest automated theorem prover (ATP) on the planet as of 2006, Vampire, cannot discover a proof of Th 89.

This was brought to my attention by Konstantine. When he first told me, I was, frankly, very skeptical: After all, Th 89 is so simple that it's reminiscent of 1956-level "triumphs" in AI. I decided to first prove Th 89 from scratch, *qua* human, not machine (easy). My proof, which appears on the following two slides, has a lot of "human-natural" structure.

Theorem 89. $\mathcal{P}(\emptyset) = \{\emptyset\}$.

Proof. To show $A = B$, where A and B are sets, it suffices to show that

1. $A \subseteq B$
2. $B \subseteq A$

This is expressed in Theorem 4, which is

$$\forall A \forall B[(A \subseteq B \wedge B \subseteq A) \rightarrow A = B]$$

Put in terms of the present challenge, we need to show

1^* $\mathcal{P}(\emptyset) \subseteq \{\emptyset\}$
2^* $\{\emptyset\} \subseteq \mathcal{P}(\emptyset)$

We prove these separately. For 2^*, we know that $\emptyset \in \{\emptyset\}$, and need to show that $\emptyset \in \mathcal{P}(\emptyset)$. But we know that every set is a subset of itself, so $\emptyset \subseteq \emptyset$, and by the definition of power set,

$$\mathcal{P}(A) = \{B : B \subseteq A\}$$

we are done.

For 1^*, assume that $A \in \mathcal{P}(\emptyset)$, for arbitrary A. Theorem 86 is:

$$\forall A \forall B[B \in \mathcal{P}(A) \leftrightarrow B \subseteq A],$$

so we can infer that $A \subseteq \emptyset$. Suppose now for contradiction that $A \notin \{\emptyset\}$. It follows immediately that $A \neq \emptyset$. But we know that $\emptyset \subseteq A$; this is vacuously true. Hence we infer that $A = \emptyset$, and we have our contradiction. By *reductio*, we are done. QED

For 1^*, assume that $A \in \mathcal{P}(\emptyset)$, for arbitrary A. Theorem 86 is:

$$\forall A \forall B[B \in \mathcal{P}(A) \leftrightarrow B \subseteq A],$$

so we can infer that $A \subseteq \emptyset$. Suppose now for contradiction that $A \notin \{\emptyset\}$. It follows immediately that $A \neq \emptyset$. But we know that $\emptyset \subseteq A$; this is vacuously true. Hence we infer that $A = \emptyset$, and we have our contradiction. By *reductio*, we are done. QED

Let's take a look at Vampire's failure, and human-powered success, via Athena...

REASON v DPLs

- REASON takes various inputs from the user: proofs, partial proofs, inference rules, etc.
- REASON allows the human to collaboratively engineer not just proofs, but models/countermodels, and therefore meta-proofs.
- REASON revolves around answers returned in logic programming style: verdicts, proofs, partial proofs, terms (as answers), models.
- REASON paramaterizes the intelligence of the computer.
- REASON allows abductive and inductive reasoning.

Mystery 1, Solved

Here are your two clues:

If Billy is a monkey, Jane stole the cookie.

Jane stole the cookie!

Assuming that model construction is oracular at the level of atomic formulas, first build a model in which the conditional is false, and then employ REASON. E.g., set *B* = FALSE, and ask REASON to verify 'If *B* then *S*' = TRUE. Now ask REASON to verify that *B* cannot be proved from the two premises.

Mystery 2, Solved

Here are your two clues:

If Billy is a monkey, Jane stole the cookie.

Jane did not steal the cookie!

Do you know whether Billy is a monkey?

Assuming that REASON is oracular with respect to *modus ponens*, assume that Billy is a monkey. Now set up a partial proof in which, given the contradiction of *S* and ~*S*, ~*B*. Ask REASON to execute. Certification comes; problem solved.

Mystery 3, Solved

All the Frenchmen in the restaurant are gourmets.

Some of the gourmets are wine drinkers.

Does it follow that some of the Frenchmen are wine drinkers?

We are now programming over standard model theory. Assume that model finding is oracular at the level of I sentences. Declare that lone object *a* is a *G*, and a *W*, and not an *F*. Ask REASON to verify that the following formula is TRUE:

(forall x (if (F x) (G x)))

REASON does so. Now ask REASON to verify that

(exists x (and (F x) (W x)))

can't be proved from the two premises in question. REASON certifies the answer (and can provide the full countermodel).

Challenges

- Proof identity.
- Parameterized natural deduction-style ATPs.
- Parameterized model building.
- Formalisms for non-deductive reasoning in the family.
- ...