

Expressive power of the languages we use would ideally be tailored to the problems we solve.

Expressive power makes checking computationally expensive.

FOL occupies a "goldilocks" area in the design space of symbolic languages, being expressive while admitting effective reasoning, i.e. complete proof systems and useful decidable fragments.

Still, a sometimes glaring limitation of FOL, especially when using it to reason about programs, is that it cannot use recursion to define concepts.

Examples: Reachability in an unbounded number of steps, axiomatizing recursive data structures like lists and trees...

Example: Modeling finite linked lists

Attempt 1

Constant nil and next pointer field

modeled by a unary function symbol next.

Data held at each cell of the list
modeled by a unary function symbol data.

So what should count as a list in
our model? Let list be a unary predicate.

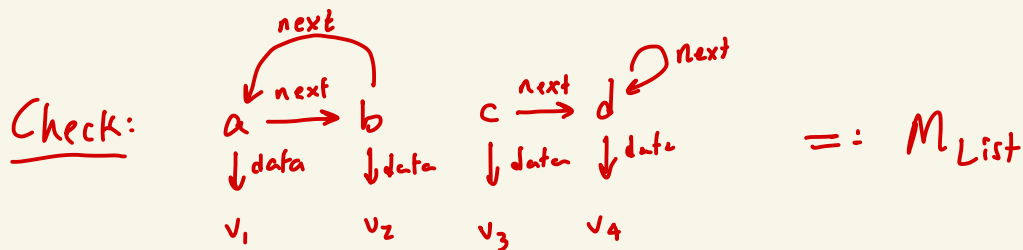
Intended meaning of list:

"For any element of the domain which
reaches nil within a finite number of
next "steps", list should hold for that element."

$$\varphi_{\text{Axiom}} := \forall x. \text{list}(x) \leftrightarrow (x = \text{nil} \vee \exists y. y = \text{next}(x) \wedge \text{list}(y))$$

Did we capture lists as intended?

$$\varphi_{\text{Axiom}} := \forall x. \text{list}(x) \leftrightarrow (x = \text{nil} \vee \exists y. y = \text{next}(x) \wedge \text{list}(y))$$



$$\text{And, } \text{list}^{M_{\text{List}}} = \{a, b, c, d\}$$

$$\text{nil}^{M_{\text{List}}} = d$$

Claim: $M_{\text{List}}, \{x \mapsto a\} \models \text{list}(x) \wedge \varphi_{\text{Axiom}}$

The issue is that φ_{Axiom} doesn't actually capture Intended meaning from last page.

This is not a failure of our ingenuity. The intended meaning involves "unbounded reachability" and turns out to be a concept which cannot be expressed in FOL, much like the exercise proving FOL cannot express connectivity of graphs.

How could we strengthen FOL for the current purpose?

FOL + recursive definitions (FOL + LFP)

(see Libkin's Elements of Finite Model Theory, Ch. 10)

Extend FOL with a construct for
defining recursive relations:

New syntax by example:

$\varphi ::= \dots \mid [\text{Ifp}_{R, \bar{x}} \varphi(R, \bar{x})] (\bar{t}) \mid \dots$

Example: $[\text{Ifp}_{\text{list}, x} (x = \text{nil} \vee \exists y. \text{next}(x) = y \wedge \text{list}(y))] (a)$

"user-friendly":

$\text{let list}(x) :=_{\text{Ifp}} x = \text{nil} \vee \exists y. \text{next}(x) = y \wedge \text{list}(y)$

in φ

Semantics next ...

$$\text{list}(x) :=_{\text{lfp}} \underbrace{x = \text{nil} \vee \exists y. \text{next}(x) = y \wedge \text{list}(y)}_{\varphi(\text{list}, x)}$$

Fix a structure M .

$M, \{x \mapsto a\} \models \text{list}(x)$? What is the semantics?

$$M, \gamma \models \text{list}(x) \quad \text{iff} \quad \gamma(x) \in \underset{\substack{| \\ \text{"least fixed point" }}}{\text{lfp}}(F_{\varphi(\text{list}, x)})$$

Each recursive definition defines an operator on relations over any fixed structure's domain.

$$F_{\varphi(\text{list}, x)}(U) = \left\{ a \in D(M) : M, \{x \mapsto a, \text{list} \mapsto U\} \models \varphi(\text{list}, x) \right\}$$

Concerns ? Does a least fixed point exist ?

Claim: $F_{\varphi(\text{list}, x)} : \mathcal{P}(D(M)) \rightarrow \mathcal{P}(D(M))$ is monotone

w.r.t. $\subseteq$, i.e. if $U \subseteq V$ then $F_{\varphi(\text{list}, x)}(U) \subseteq F_{\varphi(\text{list}, x)}(V)$

If we can guarantee our recursive definitions give monotone operators, then the least fixed point exists and the semantics is well defined, because:

Theorem: (Knaster-Tarski Fixed Point Theorem)

Let $L = (D, \leq)$ be a complete lattice and let

$f: D \rightarrow D$ be a monotone function

with respect to $\leq$, i.e.

if $a \leq b$ then $f(a) \leq f(b)$.

Then the set of fixed points of f forms a complete lattice under $\leq$.

And in particular,

f has a least fixed point.

Claim: Any recursive definition in which the defined symbol (e.g. list) appears "positively" only has a monotone operator.

Example: $\text{list}(x) := \text{ifp } x = \text{nil} \vee \exists y. \text{next}(x) = y \wedge \underline{\text{list}(y)}$
positive occurrence

$R(x) := \text{ifp } \underline{\neg R(x)}$
negative occurrence

Idea: Restrict the syntax to positive recursive definitions, a conservative requirement that guarantees semantics is well defined on any structure.

Note: Undecidable to check whether a given definition yields a monotone operator over every structure.

FOL + LFP is quite expressive

Expressing "safety" properties in FOL + LFP

$$\varphi_{\text{reach}} \rightarrow \neg \varphi_{\text{bad}}$$

$C \models \varphi_{\text{reach}}$ iff configuration C can be reached in some finite number of steps

$C \models \varphi_{\text{bad}}$ iff configuration C belongs to the set of configurations we want the machine to avoid

$$\varphi_{\text{reach}}(\bar{x}) := \text{ifp } \varphi_{\text{pre}}(\bar{x})$$

$$\vee \exists \bar{x}'. \varphi_{\text{reach}}(\bar{x}') \wedge \varphi_{\text{trans}}(\bar{x}', \bar{x})$$

Example: Program avoids division by 0.

@pre: $x > 0$

while true:	l_1	
$y := y/x$	l_2	Avoid dividing by 0
$x := x+1$	l_3	

$$\varphi_{pre}(x, y, pc) := x > 0 \wedge pc = l_1$$

$$\varphi_{bad}(x, y, pc) := x = 0 \wedge pc = l_2$$

primed variables for "post"

$$\varphi_{trans}(x', y', pc', x, y, pc) :=$$

$$pc' = l_1 \rightarrow x = x' \wedge y = y' \wedge pc = l_2 \wedge$$

$$pc' = l_2 \rightarrow x = x' \wedge y = y'/x' \wedge pc = l_3 \wedge$$

$$pc' = l_3 \rightarrow x = x'+1 \wedge y = y' \wedge pc = l_1$$

$$(\mathbb{R}, 0, 1, +, x, <) \models \forall x, y, pc. (\varphi_{reach}(x, y, pc) \rightarrow \neg \varphi_{bad}(x, y, pc))$$

" $a = b/c$ " shorthand for:
" $a \cdot c = b \wedge c \neq 0$ "

Theorem: There is no ^{sound +} complete proof system for FOL+LFP.

Suppose there is. Then the validity problem is r.e. (really? Assumes proofs are finite and can be enumerated and checked).

But, FOL+LFP can recursively axiomatize standard theory of arithmetic on natural numbers.

$\text{nat}(x) := \text{iff } x = 0 \vee \exists y. x = s(y) \wedge \text{nat}(y)$, etc.

But $\text{Th}(\mathbb{N}, 0, 1, +, \times, <)$ would be decidable

if FOL+LFP had a sound + complete

reasoning procedure. In fact, $\text{Th}(\mathbb{N}, 0, 1, +, \times, <)$

is not even r.e. by Gödel's Incompleteness Thm.

Best we can do is sound heuristics.

Program logics, e.g. Hoare logic, automate parts of verification and isolate "hard" part:

coming up with loop invariants.