PROGRAM SYNTHESIS THROUGH GENTZEN CALCULUS OF MODAL LOGICS

16TH LATIN AMERICAN WORKSHOP ON NEW METHODS OF REASONING

LANMR 2024, SEPTEMBER 7TH

RICARDO LÓPEZ | EVERARDO BÁRCENAS

INTRODUCTION: WHAT IS PROGRAM SYNTHESIS?

A longstanding problem in computer science, program synthesis is the task of generating a program that satisfies a given specification.

INTRODUCTION: WHAT IS PROGRAM SYNTHESIS?

The problem has been investigated since late 1950's. Some regard it as the "Church Problem" since Alonzo Church posed one of the earliest instances of the problem as a question: *How to build a circuit from a mathematical formula?*

WHAT DO WE KNOW?

There is a very well known system based on LLM (Large Language Models) solving the problem for a great amount of consults. However, it is known to be error-prone.

Besides machine learning, formal methods have been a strongly visited approach to the problem.

THE FORMAL APPROACH

Diverse authors thought of the problem as a question concerning automatic theorem proving systems.

THE PROBLEM WITH THE FORMAL APPROACH

Eventually, it was found that the ATP techniques were unable to express recursion.

Some authors were inclined to use proving methods based on recursion, sacrificing other desired properties.

THE WALDINGER-MANNA APPROACH TO PROGRAM SYNTHESIS

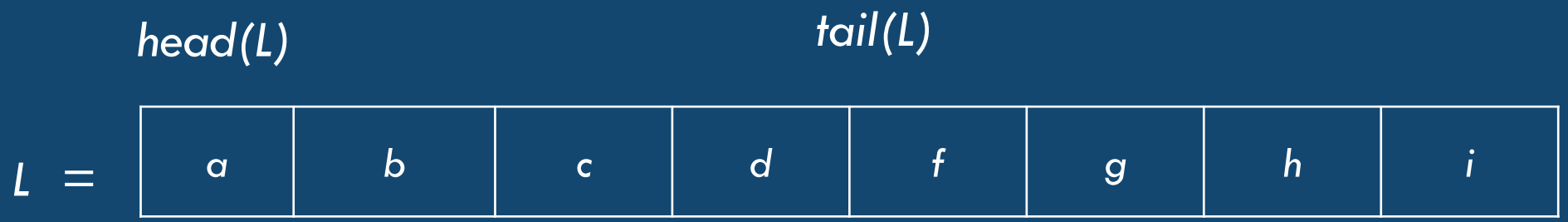
It was the seminal paper *A deductive approach to Program Synthesis* of Richard Waldinger and Zohar Manna that introduces a formal framework without giving up recursion in proofs.

The formal framework they present is a Gentzen-style deduction system.

THE WALDINGER-MANNA APPROACH TO PROGRAM SYNTHESIS

Suppose we wish to develop a program such that, for a certain input x, the output is $f(x)$.

E.g. our input is a nonempty list L and we want to find the maximum element, $max(L)$.



$head(L) > max(tail(L))$?

Let us take the following formulas:

$$A_1(x) \equiv islist(x) \qquad A_2(x) \equiv \neg(\, x = [\,]\,)$$

We will call them our *assertions*. For an x satisfying both properties, we want to find its maximum element. Take the formula

$$v = max(L)$$

In order to find what the maximum of our input list is, first find a derivation of

$$\forall x\ A_1(x), \forall x\ A_2(x) \vdash \exists v\ v = max(L)$$

For any v: If $v = max(x)$, then $v \in x$, which is equivalent to say that

$$v = head(x) \quad \text{or} \quad v \in tail(x)$$

If $v = head(x)$, $\neg(\, tail(x) = [\,]\,)$, and $head(x) > max(tail(x))$, then $v = max(x)$.

If $\neg(\, tail(x) = [\,]\,)$, $head(x) \leq max(tail(x))$ and $v = max(tail(x))$, $v = max(x)$.

Therefore, $v = max(x)$ if and only if

$$v = head(x) \text{ and } \neg(\, tail(x) = [\,]\,) \text{ and } head(x) > max(tail(x))$$
$$\text{or}$$
$$v = max(tail(x)) \text{ and } \neg(\, tail(x) = [\,]\,) \text{ and } head(x) \leq max(tail(x))$$

Fix the *goals*

$$G_1(x) \equiv \neg(\ tail(x) = [\ ]\ ) \text{ and } head(x) > max(tail(x))$$

$$G_2(x) \equiv \neg(\ tail(x) = [\ ]\ ) \text{ and } head(x) \leq max(tail(x))$$

and add the assertion

$$A_3(x) \equiv \quad \exists v \;\; v = max(x)$$

Thus, the sequent we want to derive becomes

$$\forall x\ A_1(x),\ \forall x\ A_2(x),\ \forall x\ A_3(x) \vdash \exists x\ G_1(x),\ \exists x\ G_2(x)$$

The bottom of a proof-tree would look something like the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}}{islist(L),\ \neg(L=[\,]),\ \exists v\ v=max(L) \vdash \neg(tail(L)=[\,]) \wedge (\ head(L)>max(tail(L)) \vee head(L)\leq max(tail(L))\ )}}{A_1(L),\ A_2(L),\ A_3(L) \vdash G_1(L),\ G_2(L)}}{A_1(L),\ A_2(L),\ A_3(L) \vdash \exists x\ G_1(x),\ \exists x\ G_2(x)}\ \exists\ Right}{\forall x\ A_1(x),\ \forall x\ A_2(x),\ \forall x\ A_3(x) \vdash \exists x\ G_1(x),\ \exists x\ G_2(x)}\ \forall\ Left$$

If $head(L)\leq max(tail(L))$, in the upper sequent that formula is replaced by

$$\neg(tail(tail(L))=[\,]) \wedge (\ head(tail(L))>max(tail(tail(L))) \vee head(tail(L))\leq max(tail(tail(L)))\ )$$

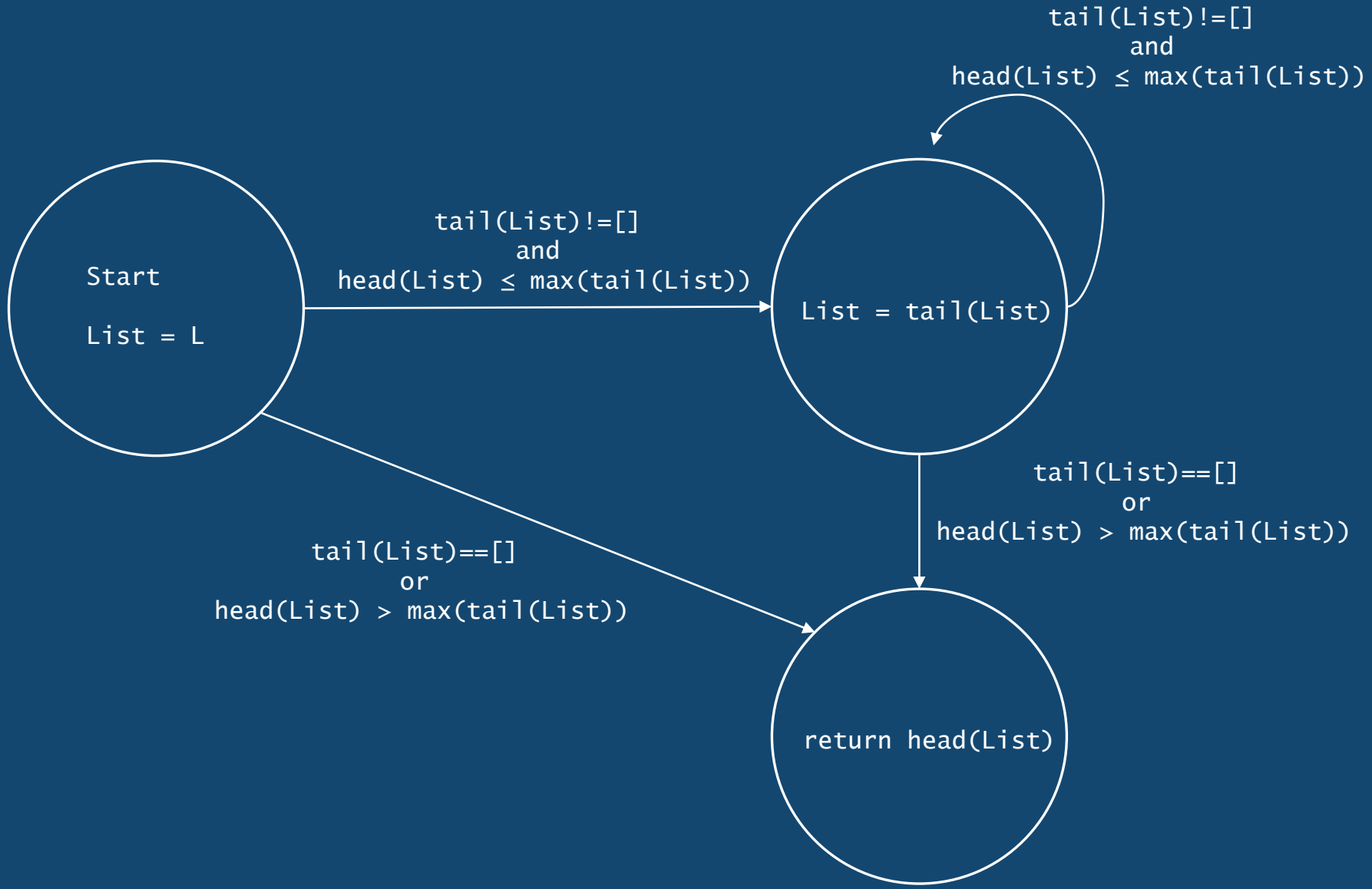
which is obtained by writing "$tail(L)$" instead of "L".

- Only one of formulas $G_1(L)$ and $G_2(L)$ can be true, and it depends on what are the elements of the input list L. This will determine what term will be written instead of v in the assertion

$$\exists v \;\; v = max(L)$$

- We keep track of the candidate to be the maximum of the list.
- Notice that it replicates the following algorithm on a nonempty list L:

```
Max(L):
if tail(L) != [ ]:
    if head(L) > Max(tail(L)):
        return head(L)
    else
        return Max(tail(L))
else
    return head(L)
```


Start
List = L
tail(List)!=[]
and
head(List) ≤ max(tail(List))
List = tail(List)
tail(List)!=[]
and
head(List) ≤ max(tail(List))
tail(List)==[]
or
head(List) > max(tail(List))
tail(List)==[]
or
head(List) > max(tail(List))
return head(List)

CONCLUSION: RESEARCH PROPOSAL

- How can we use modal logic in this methodology?
- Is it possible to describe the process of a computation with modal logic?
- What kind of specifications are possible to express with modal logic formulas?

THANK YOU!

rlv@ciencias.unam.mx