DueList: A theory of lists with combinators for SMT solvers

Pierre Goutagny*, Aymeric Fromherz, Raphaël Monat
pierre.goutagny@inria.fr Inria

List constraint from French housing benefits

```
let child_custody =
  List.fold (+) 0
    (List.map (fun child -> match child.custody with
         | Sole -> 1
         | Shared -> 0.5
         | None -> 0)
        children)
in (child_custody > child_custody_threshold)
&& List.length children > child_count_threshold
```

Position among related work

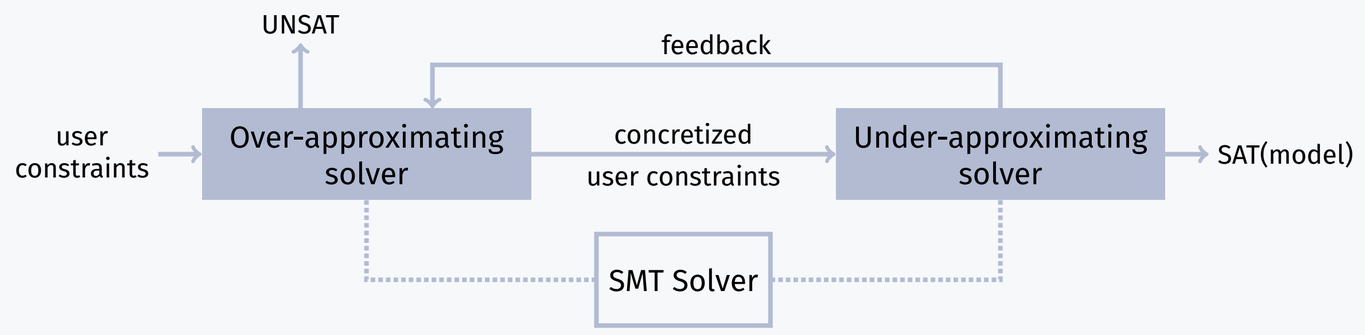
Solver	combinators	fast UNSAT	domain
SMT-LIB Lists [1]	recursive	✗	∞
SMT-LIB Seq [2]	implem. dep.	✗	∞
SMT-LIB String [4]	structural	✓	finite
SMT-LIB Array [6]	index-based	✓	∞
SyGuS [7]	recursive	✗	∞
Rosette [8]	recursive	N/A	∞
Our goal	native support	✓	∞

Combinator language

SMT-LIB syntax + list combinators:

- Supported: `fold`, `map`, `length`
- Unsupported: `cons`, `concat`, `init`
- Possible extension: `filter`

Solver architecture



Over-approximating solver

Over-approx(user constraints ∪ feedback):

- Infer necessary constraints that over-approximate the problem
- Query the SMT solver:

If UNSAT:
return UNSAT

If SAT with a model:
- use it to concretize some constraints, and
- send to the Under-approximating solver

Under-approximating solver

Under-approx(constraints):

- Encode (partially concretized) list constraints in SMT solver
- Call the solver:
  - If SAT with a model, return SAT with a list model
  - If UNSAT, send feedback to refine the over-approximation

Length Solver

- Infer list length constraints from input, including combinators
- If user constraints are SAT, then length constraints are SAT
- If length constraints are SAT, the SMT solver returns a model for *possible* list lengths (concretization)

- If $C \equiv \texttt{len}\ (\texttt{map}\ f\ l) > 2$, then InferLen($C$) $\equiv |l| > 2$, and a concrete length model can be $|l| \mapsto 3$
- InferLen(running example) $\equiv |children| > child\_count\_threshold$

Element-wise encoding

Lists without other abstractions are encoded as $|l|$ SMT variables.

If $|l| \mapsto 3$,

- l is encoded as l_1, l_2, l_3
- $\texttt{fold}\ f\ i\ l = f\ (f\ (f\ i\ l_1)\ l_2)\ l_3$
- $\texttt{map}\ f\ l$ is encoded with m_1, m_2, m_3 using an additional constraint:

$$(m_1 = f\ l_1) \wedge (m_2 = f\ l_2) \wedge (m_3 = f\ l_3)$$

Unreachable value

Return UNSAT early if a function cannot reach its expected result.

Since the domain of $(\texttt{max}\ 0)$ is positive integers, $(\texttt{fold max}\ 0\ l = -1)$ must be UNSAT.

Rewriting rules

To reduce the number of SMT variables, list constraints can be rewritten using the compositional properties of combinators.

- $\texttt{map}\ f\ (\texttt{map}\ g\ l) \rightarrow \texttt{map}\ (f \circ g)\ l$
- $\texttt{fold}\ f\ i\ (\texttt{map}\ g\ l) \rightarrow \texttt{fold}\ (\lambda a x.\ f\ a\ (g\ x))\ i\ l$

List folding abstraction

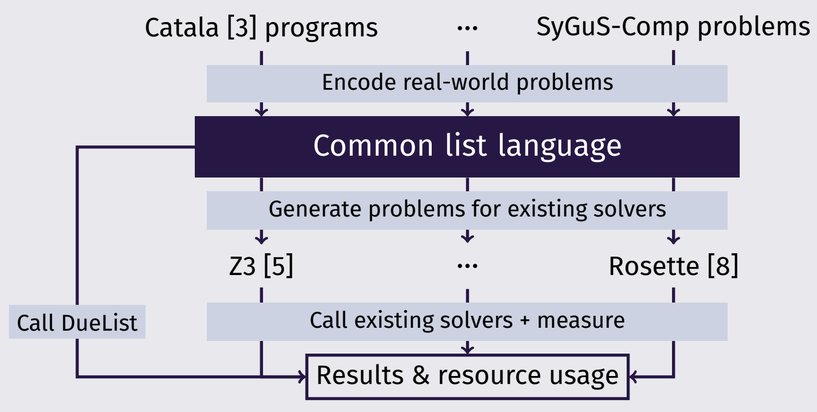
Given $\texttt{fold}\ f\ i\ l$, if there is a neutral padding element e satisfying

$$\forall x, \exists hd : (f\ i\ hd = x) \wedge (f\ x\ e = x),$$

then l has the shape $[hd, e, \ldots, e]$, and $\texttt{fold}\ f\ i\ l = f\ i\ hd$.

$\texttt{fold}\ (+)\ 0\ l = \texttt{fold}\ (+)\ 0\ [\texttt{sum}\ l, 0, \ldots, 0]$ so l has shape $[s, 0, \ldots, 0]$ such that $\texttt{sum}\ l = s$.

Evaluation plan



[1] Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard: Version 2.7 (2025), URL SMT-LIB.org
[2] Bjørner, N., Ganesh, V., Michel, R., Veanes, M.: An SMT-LIB Format for Sequences and Regular Expressions (2012)
[3] Merigoux, D., Chataing, N., Protzenko, J.: Catala: A programming language for the law (2021), DOI: 10.1145/3473582
[4] Mora, F., Berzish, M., Kulczynski, M., Nowotka, D., Ganesh, V.: Z3str4: A Multi-armed String Solver. In: Formal Methods (2021), DOI: 10.1007/978-3-030-90870-6_21
[5] de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS (2008), DOI: 10.1007/978-3-540-78800-3_24
[6] de Moura, L., Bjørner, N.: Generalized, efficient array decision procedures. In: FMCAD (2009), DOI: 10.1109/FMCAD.2009.5351142
[7] Raghothaman, M., Reynolds, A., Udupa, A.: The SyGuS Language Standard Version 2.0 (2021), URL https://sygus-org.github.io/assets/pdf/SyGuS-IF_2.1.pdf
[8] Torlak, E., Bodik, R.: A lightweight symbolic virtual machine for solver-aided host languages. In: PLDI (2014), DOI: 10.1145/2594291.2594340

*Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, 59000 Lille, France