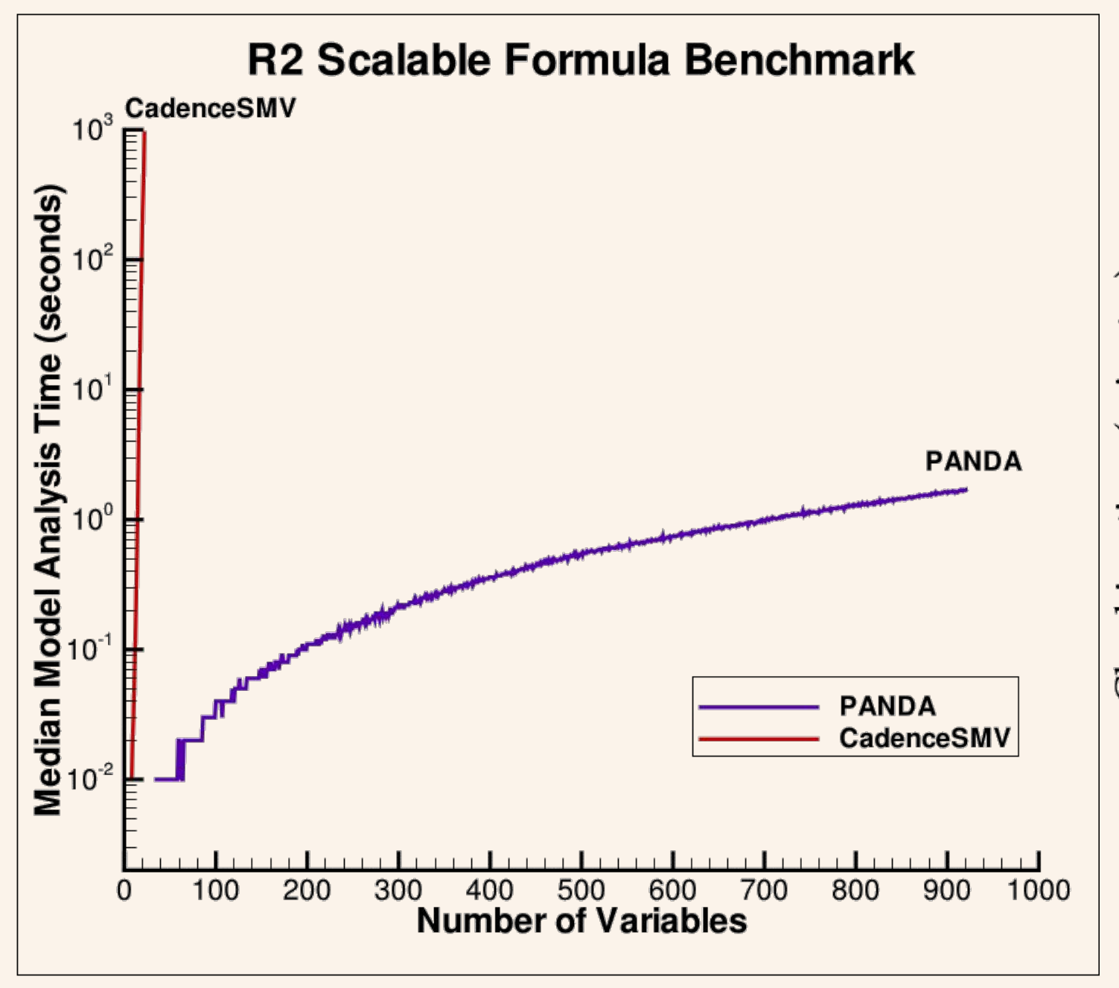
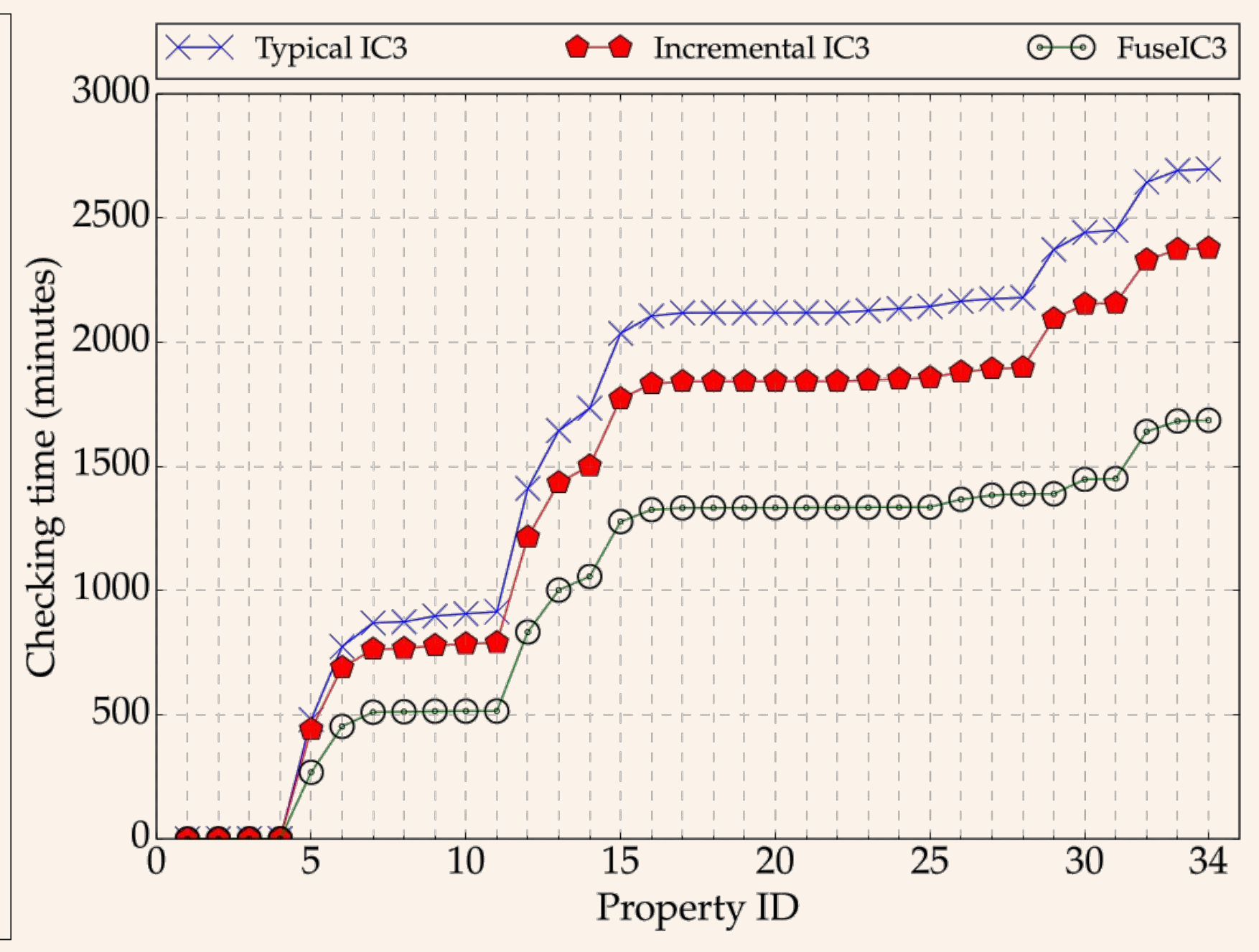
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Investigators: Kristin Yvonne Rozier, Natarajan Shankar, Cesare Tinelli, Moshe Vardi

CCRI: Developing an Open-Source, State-of-the-Art Symbolic Model-Checking Framework for the Model-Checking Research Community

Lack of Open-Source Model-Checking Platform





Left [1]: Graph depicting model analysis time between encodings used by the industrial tool CadenceSMV and proposed PANDA tool. Right [2]: Graph depicting model checking time for variations of the Ic3 algorithm, including the proposed FuseIC3 version.

The problems facing the model-checking research community are made more challenging by the lack of any openly available model-checking platform:

- Research in symbolic model checking previously showed that significant results (up to exponential performance improvement) are possible; but the state-of-the-art model checkers are all closed-source.
- Comparing advances in model-checking algorithms requires re-implementation of existing algorithms.
- Model checkers accept different high-level model languages; we cannot compare model-checking back-ends over different language models.

Language Design

The design of the intermediate language is intended to:

- Serve as an intermediate target language for model checkers
- Support a variety of user-facing modeling languages
- Directly supported by tools or compiled to lower level languages
- Leverage SAT/SMT technology

The language extends the SMT-LIB2 [3] standard by adding `define-system` and `check-system` commands for defining and checking a model (i.e., Kripke Structure) of First-Order Linear Temporal Logic (FO-LTL).

```
(set-logic QF_LIA)

(define-system Counter :input ((in Int)) :output ((out Int))
   :init (= out 0)
   :trans (= out' (+ out in))
)

(check-system Counter :input ((i Int)) :output ((o Int))
   :assumption (a (= i 2))
   :reachable (rch (= o 10))
   :query (q (a rch))
)
```

Project Abstract

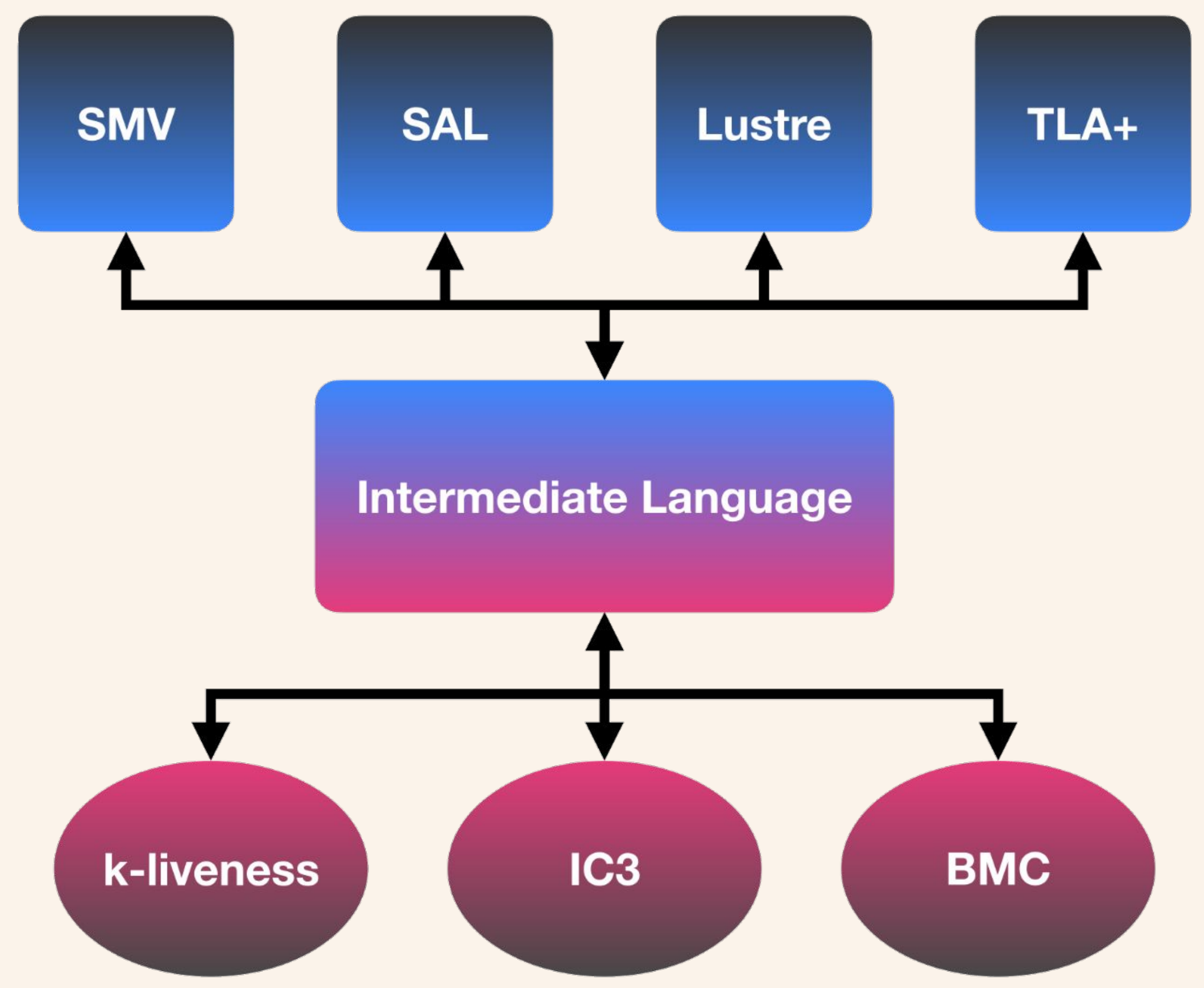
Safety-critical and security-critical systems are entering our lives at an increasingly rapid pace. These are the systems that help fly our planes, drive our cars, deliver our packages, ensure our electricity, or even automate our homes. Especially when humans cannot perform a task in person, e.g., due to a dangerous working environment, we depend on such systems. Before any safety-critical system launches into the human environment, we need to be sure it is really safe. Model checking is a popular and appealing way to rigorously check for safety: given a system, or an accurate model of the system, and a safety requirement, model checking is a "push button" technique to produce either a proof that the system always operates safely, or a counterexample detailing a system execution that violates the safety requirement.

As model checking becomes more integrated into the standard design and verification process for safety-critical systems, the platforms for model checking research have become more limited. Previous options have become closed-source or industry tools; current research platforms don't have support for expressive specification languages needed for verifying real systems.

Project Goal

Fill the current gap in model checking research platforms by

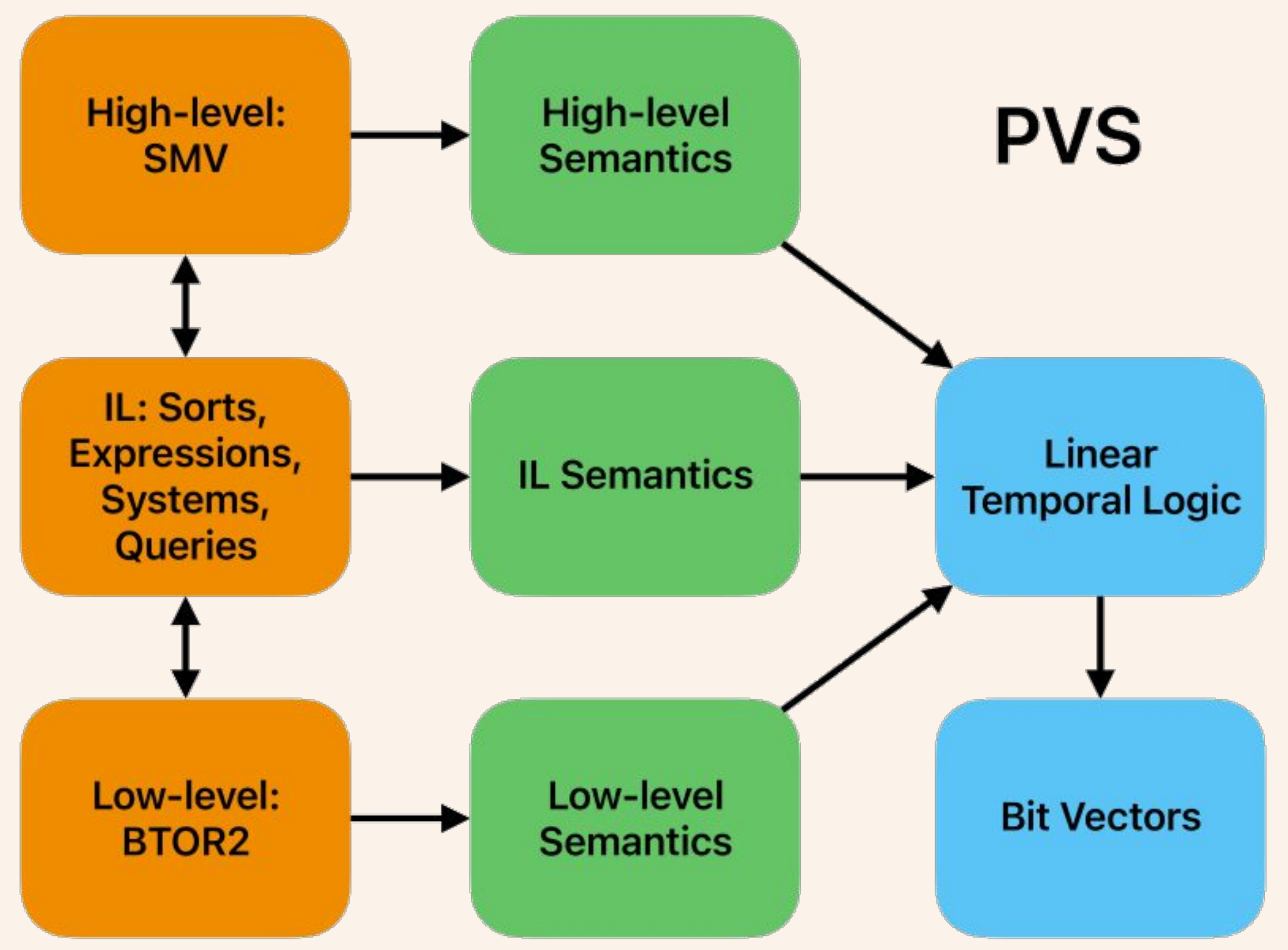
- Building a freely-available, open-source, scalable model checking infrastructure that
- Accepts expressive models,
- Efficiently interfaces with state-of-the-art back-end algorithms,
- Provides an extensible research and verification toolset.



References

[1] K.Y.Rozier and M.Y.Vardi, ``A Multi-Encoding Approach for LTL Symbolic Satisfiability Checking,'' FM 2011.
[2] Rohit Dureja and Kristin Yvonne Rozier. ``FuseIC3: An Algorithm for Checking Large Design Spaces.'' In Formal Methods in Computer-Aided Design (FMCAD), IEEE/ACM, Vienna, Austria, October 2-6, 2017.
[3] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard - Version 2.0. Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England), 2010.
[4] Adrien Champion, Alain Mebsout, Christoph Sticksel, Cesare Tinelli. The Kind 2 Model-Checker. In Proceedings of 28th International Conference on Computer Aided Verification (CAV), 2016.
[5] F´erey, G., Sh, N.: Code Generation using a formal model of reference counting, NFM 2016.

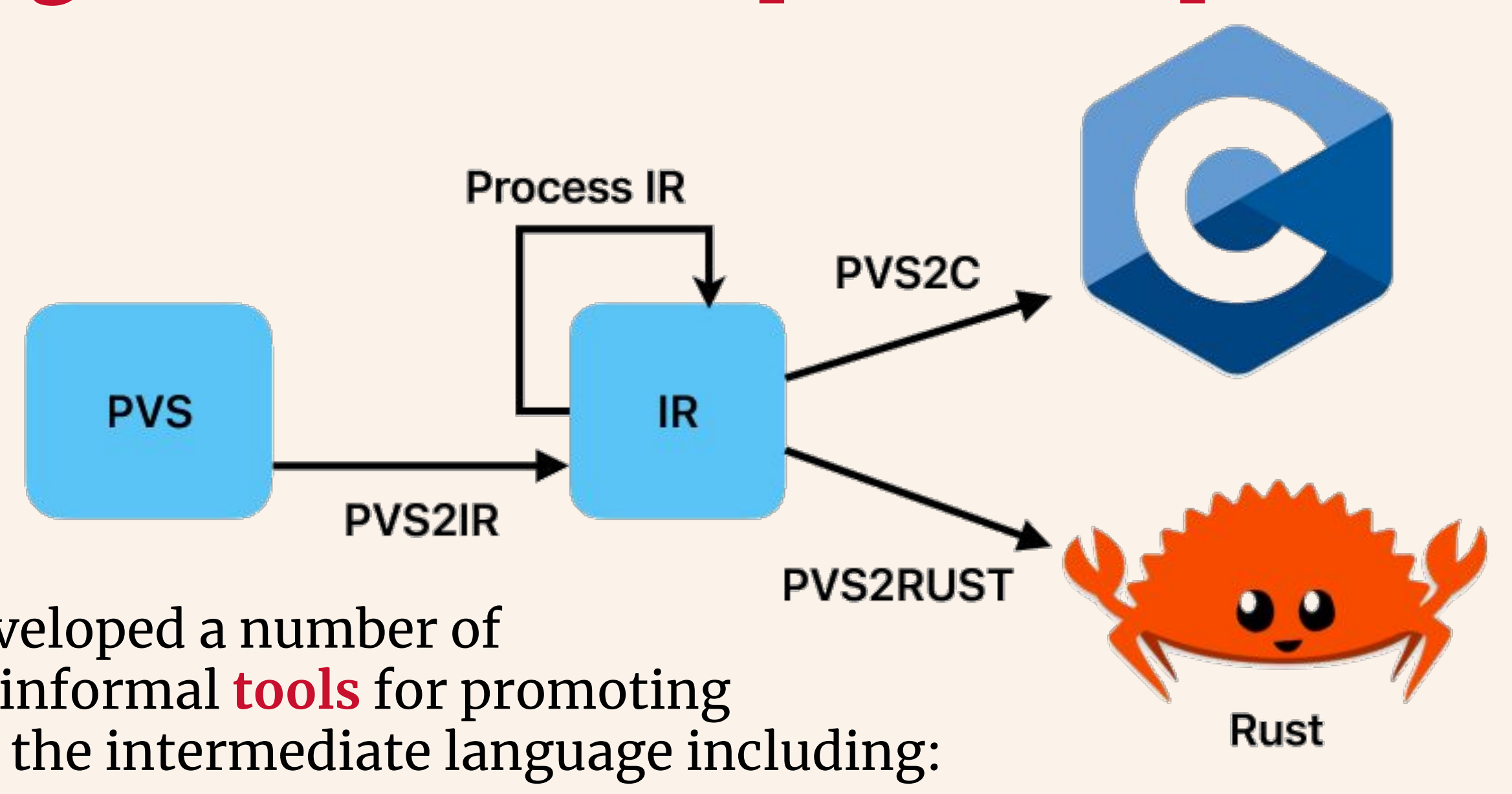
Formal Embedding for a Provably Correct Translation Pipeline



PVS is an interactive proof assistant based on higher-order logic developed at SRI over the last three decades. We use PVS to:

- Create a syntactic and semantic bridge between high-level and low-level languages, mapping problems and witnesses.
- Define a deep embedding of the sort, expression, system, and query syntax and semantics.

Providing Tools for Widespread Adoption



We have developed a number of formal and informal tools for promoting adoption of the intermediate language including:

- Translation from SMV to IL to BTOR2 in Python
- Translation of BTOR2 witnesses to IL witnesses in Python
- Interface with KIND2 model checker [4]

We leverage PVS2C [5] to generate safe, efficient, standalone, executable C code for the embedded language written in PVS, to obtain a provably correct executable of the SMV-IL-BTOR2 pipeline.

Website
modelchecker.github.io