Alda: Integrating Logic Rules with Everything Else, Seamlessly (System Demonstration)*

Yanhong A. Liu Scott D. Stoller Yi Tong Bo Lin
Computer Science Department, Stony Brook University, Stony Brook, NY
{liu,stoller,yittong}@cs.stonybrook.edu

Sets and rules have been used for easier programming since the late 1960s. While sets are central to database programming with SQL and are also supported as built-ins in high-level languages like Python, logic rules have been supported as libraries or in rule-based languages with limited extensions for other features. However, rules are central to deductive database and knowledge base programming, and better support is needed.

This system demonstration highlights the design of a powerful language, Alda [16, 14], that supports logic rules together with sets, functions, updates, and objects, all as seamlessly integrated built-ins, including concurrent and distributed processes. The key idea is to allow sets of rules to be defined in any scope, support predicates in rules as set-valued variables that can be used and updated directly, and support queries using rules as either explicit or implicit automatic calls to an inference function.

Alda has a formal semantics [15] and is implemented by building on an object-oriented language (DistAlgo [13, 3] extending Python [18]) and an efficient logic rule system (XSB [19, 20]). It has been used successfully on benchmarks and problems from a wide variety of application domains—including those in OpenRuleBench [6], role-based access control (RBAC) [1, 4], and program analysis—with generally good performance [17]. Our implementation and benchmarks are publicly available [23].

This system demonstration shows how Alda is used for OpenRuleBench benchmarks, ANSI standard for role-based access control, and program analysis for large Python programs, including with persistence support for large datasets, all programmed seamlessly without boiler-plate code. For comparisons with related work on rule languages and benchmarking, see [16, 14, 17].

An example. Figure 1 shows an example program in Alda. It is for a small portion of the ANSI standard for role-based access control (RBAC) [1, 4]. It shows the uses (with line numbers in parentheses) of

- classes (1-8, 9-21) with inheritance (9, 11), and object creation (22) with setup (2-3, 10-12);
- sets, including relations (3, 12);
- methods, including procedures (5-6, 13-14) and functions (7-8, 18-19, 20-21), and calls (23, 24);
- updates, including initialization (3, 12) and membership changes (6, 14); and
- queries, including set queries (8, 19 after union "+", 21) and queries using rules (19 before "+");

where the rules are defined in a rule set (15-17), explained in the next part.

Note that queries using set comprehensions (e.g., on lines 8, 19, 21) can also be expressed by using rules and inference, even though comprehensions are more widely used. However, only some queries using rules and inference can be expressed by using comprehensions; queries using recursive rules (e.g., on lines 16-17) cannot be expressed using comprehensions.

*This work was supported in part by NSF under grants CCF-1954837, CCF-1414078, and IIS-1447549 and ONR under grants N00014-21-1-2719, N00014-20-1-2751, and N00014-15-1-2208.

Names (Pedro Cabalar and Theresa Swift); Event name (ICLP)
EPTCS ??, 2024, pp. 1–4, doi:10.4204/EPTCS.??.??

```
class CoreRBAC:                    # class for Core RBAC component/object
  def setup():                     # method to set up the object, with no arguments
    self.USERS, self.ROLES, self.UR := {},{},{}
                                   # set users, roles, user-role pairs to empty sets
  def AddRole(role):               # method to add a role
    ROLES.add(role)                # add the role to ROLES
  def AssignedUsers(role):         # method to return assigned users of a role
    return {u: u in USERS | (u,role) in UR}  # return set of users having the role
  ...
class HierRBAC extends CoreRBAC: # Hierarchical RBAC extending Core RBAC
  def setup():
    super().setup()                # call setup of CoreRBAC, to set sets as in there
    self.RH := {}                  # set ascendant-descendant role pairs to empty set
  def AddInheritance(a,d):         # to add inherit. of an ascendant by a descendant
    RH.add((a,d))                  # add pair (a,d) to RH
  rules trans_rs:                  # rule set defining transitive closure
    path(x,y) if edge(x,y)         # path holds for (x,y) if edge holds for (x,y)
    path(x,y) if edge(x,z), path(z,y)  # ... if edge holds for (x,z) and for (z,y)
  def transRH():                   # to return transitive RH and reflexive role pairs
    return infer(path, edge=RH, rules=trans_rs) + {(r,r): r in ROLES}
  def AuthorizedUsers(role):       # to return users having a role transitively
    return {u: u in USERS, r in ROLES | (u,r) in UR and (r,role) in transRH()}
  ...
h = new(HierRBAC, [])              # create HierRBAC object h, with no args to setup
h.AddRole('chair')                 # call AddRole of h with role 'chair'
...
h.AuthorizedUsers('chair')         # call AuthorizedUsers of h with role 'chair'
...
```

Figure 1: An example program in Alda, for Role-Based Access Control (RBAC). In rules `trans_rs`, the first rule says there is a path from `x` to `y` if there is an edge from `x` to `y`, and the second rule says there is a path from `x` to `y` if there is an edge from `x` to `z` and there is an edge from `z` to `y`. The call to infer queries and returns the set of pairs for which `path` holds given that `edge` holds for exactly the pairs in set `RH`, by doing inference using rules in `trans_rs`.

Rules with sets, functions, updates, and objects. In Alda, rules are defined in rule sets, each with a name and optional declarations for the predicates in the rules.

$$\begin{aligned} \textit{ruleset} &::= \texttt{rules}\ \textit{name}\ (\textit{declarations})\texttt{:}\ \textit{rule}+ \\ \textit{rule} &::= p(\textit{arg}_1, \ldots, \textit{arg}_a)\ \texttt{if}\ p_1(\textit{arg}_{11}, \ldots, \textit{arg}_{1a_1}), \ldots, p_k(\textit{arg}_{k1}, \ldots, \textit{arg}_{ka_k}) \end{aligned}$$

In the rule form, p, p_1, ..., p_k denote predicates, $p(arg_1, ..., arg_a)$ denotes that p holds for its tuple of arguments, and if denotes that its left-side conclusion holds if its right-side conditions all hold. In a rule set, predicates not in any conclusion are called base predicates; the other predicates are called derived predicates.

The key ideas of seamless integration of rules with sets, functions, updates, and objects are:

1. a predicate is a set-valued variable that holds the set of tuples for which the predicate is true;
2. queries using rules are calls to an inference function, infer, that computes desired values of derived predicates using given values of base predicates;
3. values of base predicates can be updated directly as for other variables, whereas values of derived predicates can only be updated by infer; and
4. predicates and rule sets can be object attributes as well as global and local names, just as variables and functions can.

Declarative semantics of rules are ensured by automatically maintaining values of derived predicates when values of base predicates are updated, by appropriate implicit calls to infer.

For example, in Figure 1, one could use an object field `transRH` in place of calls to `transRH()` in `AuthorizedUsers(role)`, use the following rule set instead of `trans_rs`, and remove `transRH()`.

```
rules transRH_rs:                # no need to call infer explicitly
  transRH(x,y) if RH(x,y)
  transRH(x,y) if RH(x,z), transRH(z,y)
  transRH(x,x) if ROLES(x)
```

Field `transRH` is automatically maintained at updates to `RH` and `ROLES` by implicit calls to infer.

Higher-order, patterns, distributed programming, and more. Note that predicates in rules as set-valued variables, e.g., `edge`, and calling infer to take or return values of set variables, e.g., `RH` in `edge=RH`, avoids the need of high-order predicates or other sophisticated features, e.g., [2], to reuse rules for different predicates in logic languages.

Alda also supports tuple patterns for set elements in set queries (as in DistAlgo [13]) and in queries using rules, e.g., `(1,=x,y) in p` matches any triple in set `p` whose first element is 1 and whose second element equals the value of `x`, and binds `y` to the third element if such a triple exists.

Of course, through DistAlgo, Alda also supports distributed programming, e.g., for distributed RBAC [7, 9], also called trust management [5], in decentralized systems.

Declarations in rules could specify predicate types and scopes, but are designed more importantly for specifying assumptions about predicates being certain, complete, closed, or not [10, 11, 12]. This is to give respective desired semantics for rules with unrestricted negation and aggregation.

Note that the examples discussed use an ideal syntax, while the Alda implementation supports the Python syntax. For example, `x := {}` is written as `x = set()` in Python syntax.

The Alda implementation compiles rule sets in rules and queries using infer to XSB rules and queries, and compiles the rest to Python, which calls XSB to do the inference. The current implementation supports primarily Datalog rules, but also handles unrestricted negation by using XSB's computation of the well-founded semantics [24]. More general forms of rules and queries can be compiled to rules and queries in XSB or other rule systems using the same approach. In general, any efficient inference algorithm and implementation method can be used to compute the semantics of rules and infer.

Future work includes (1) support for easy use of different desired semantics, especially with modular use of rules, similar to knowledge units in DA-logic [11]; and (2) efficient implementation with complexity guarantees [8, 21, 22] for computing different desired semantics.

References

[1] ANSI INCITS (2004): *Role-Based Access Control*. ANSI INCITS 359-2004, American National Standards Institute, International Committee for Information Technology Standards.

[2] Weidong Chen, Michael Kifer & David S. Warren (1993): *HiLog: A Foundation for Higher-Order Logic Programming*. *Journal of Logic Programming* 15(3), pp. 187–230, doi:10.1016/0743-1066(93)90039-J.

[3] (2024): *DistAlgo*. `http://distalgo.cs.stonybrook.edu`. Accessed July 8, 2024.

[4] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn & Ramaswamy Chandramouli (2001): *Proposed NIST Standard for Role-Based Access Control*. *ACM Transactions on Information and Systems Security* 4(3), pp. 224–274, doi:10.1145/501978.501980.

[5] Ninghui Li, John C. Mitchell & William H. Winsborough (2002): *Design of a Role-Based Trust-Management Framework*. In: *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 114–130, doi:10.1109/SECPRI.2002.1004366.

[6] Senlin Liang, Paul Fodor, Hui Wan & Michael Kifer (2009): *OpenRuleBench: An Analysis of the Performance of Rule Engines*. In: *Proceedings of the 18th International Conference on World Wide Web*, ACM Press, pp. 601–610, doi:10.1145/1526709.1526790.

[7] Yanhong A. Liu (2018): *Role-Based Access Control as a Programming Challenge*. In: *LPOP: Challenges and Advances in Logic and Practice of Programming*, https://arxiv.org/abs/2008.07901, pp. 14–17, doi:10.48550/arXiv.2008.07901.

[8] Yanhong A. Liu & Scott D. Stoller (2009): *From Datalog Rules to Efficient Programs with Time and Space Guarantees*. *ACM Transactions on Programming Languages and Systems* 31(6), pp. 1–38, doi:10.1145/1552309.1552311.

[9] Yanhong A. Liu & Scott D. Stoller (2018): *Easier Rules and Constraints for Programming*. In: *LPOP: Challenges and Advances in Logic and Practice of Programming*, https://arxiv.org/abs/2008.07901, pp. 52–60, doi:10.48550/arXiv.2008.07901.

[10] Yanhong A. Liu & Scott D. Stoller (2020): *Founded Semantics and Constraint Semantics of Logic Rules*. *Journal of Logic and Computation* 30(8), pp. 1609–1638, doi:10.1093/logcom/exaa056. Also http://arxiv.org/abs/1606.06269.

[11] Yanhong A. Liu & Scott D. Stoller (2021): *Knowledge of Uncertain Worlds: Programming with Logical Constraints*. *Journal of Logic and Computation* 31(1), pp. 193–212, doi:10.1093/logcom/exaa077. Also https://arxiv.org/abs/1910.10346.

[12] Yanhong A. Liu & Scott D. Stoller (2022): *Recursive Rules with Aggregation: A Simple Unified Semantics*. *Journal of Logic and Computation* 32(8), pp. 1659–1693, doi:10.1093/logcom/exac072. Also http://arxiv.org/abs/2007.13053.

[13] Yanhong A. Liu, Scott D. Stoller & Bo Lin (2017): *From Clarity to Efficiency for Distributed Algorithms*. *ACM Transactions on Programming Languages and Systems* 39(3), pp. 12:1–12:41, doi:10.1145/2994595.

[14] Yanhong A. Liu, Scott D. Stoller, Yi Tong & Bo Lin (2023): *Integrating logic rules with everything else, seamlessly*. *Theory and Practice of Logic Programming* 23(4), pp. 678–695, doi:10.1017/S1471068423000108.

[15] Yanhong A. Liu, Scott D. Stoller, Yi Tong & Bo Lin (2023): *Integrating logic rules with everything else, seamlessly*. *Computing Research Repository* arXiv:2305.19202 [cs.PL], doi:10.48550/arXiv.2305.19202.

[16] Yanhong A. Liu, Scott D. Stoller, Yi Tong, Bo Lin & K. Tuncay Tekle (2022): *Programming with Rules and Everything Else, Seamlessly*. *Computing Research Repository* arXiv:2205.15204 [cs.PL], doi:10.48550/arXiv.2205.15204.

[17] Yanhong A. Liu, Scott D. Stoller, Yi Tong & K. Tuncay Tekle (2023): *Benchmarking for Integrating Logic Rules with Everything Else*. In: *Proceedings of the 39th International Conference on Logic Programming (Technical Communications)*, Open Publishing Association, pp. 12–26, doi:10.4204/EPTCS.385.3.

[18] Python Software Foundation (2024): *Python*. http://python.org/.

[19] Terrance Swift & David S Warren (2012): *XSB: Extending Prolog with tabled logic programming*. *Theory and Practice of Logic Programming* 12(1-2), pp. 157–187, doi:10.1017/S1471068411000500.

[20] Theresa Swift, David S. Warren, Konstantinos Sagonas, Juliana Freire, Prasad Rao, Baoqiu Cui, Ernie Johnson, Luis de Castro, Rui F. Marques, Diptikalyan Saha, Steve Dawson & Michael Kifer (2022): *The XSB System Version 5.0,x*. http://xsb.sourceforge.net. Latest release May 12, 2022.

[21] K. Tuncay Tekle & Yanhong A. Liu (2010): *Precise Complexity Analysis for Efficient Datalog Queries*. In: *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, pp. 35–44, doi:10.1145/1836089.1836094.

[22] K. Tuncay Tekle & Yanhong A. Liu (2011): *More Efficient Datalog Queries: Subsumptive Tabling Beats Magic Sets*. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pp. 661–672, doi:10.1145/1989323.1989393.

[23] Yi Tong, Bo Lin, Yanhong A. Liu & Scott D. Stoller (2018 (Latest update May, 2024)): *Alda*. http://github.com/DistAlgo/alda.

[24] Allen Van Gelder, Kenneth Ross & John S. Schlipf (1991): *The Well-Founded Semantics for General Logic Programs*. *Journal of the ACM* 38(3), pp. 620–650, doi:10.1145/116825.116838.