Knowledge of Uncertain Worlds: Programming with Logical Constraints: An Overview

Yanhong A. Liu Scott D. Stoller
Computer Science Department, Stony Brook University
{liu,stoller}@cs.stonybrook.edu

Programming with logic has allowed many design and analysis problems to be expressed more easily and clearly at a high level. Examples include problems in program analysis, network management, security frameworks, and decision support. However, when sophisticated problems require reasoning with negation and recursion, possibly causing contradiction in cyclic reasoning, programming with logic has been a challenge. Many languages and semantics have been proposed, but they have different underlying assumptions that are conflicting and subtle, and each is suitable for only certain kinds of problems.

Liu and Stoller [3] describes a unified language, DA logic, which stands for Design and Analysis logic, for programming with logic using logical constraints. It supports logic rules with unrestricted negation in recursion, as well as unrestricted universal and existential quantification. It is based on the unifying *founded semantics* and *constraint semantics* [1, 2], which give a single three-valued model and a set of two-valued models, respectively, and it supports the power and ease of programming with different intended semantics without causing contradictions in cyclic reasoning.

1. The language provides *meta-constraints* on predicates. These meta-constraints capture the different underlying assumptions of different logic language semantics. They indicate whether: (1) a predicate is certain (assertions of P are two-valued: true or false) or uncertain (assertions of P are three-valued: true, false, or undefined); (2) the set of rules specifying a predicate is complete (hence negative facts $\neg P$ can be inferred using the negations of the hypotheses of those rules) or not; (3) a predicate is closed (specified assertions of the predicate are considered false if inferring any of them to be true requires assuming that some of them are true) or not.

2. The language supports the use of uncertain information in the results of different semantics, in the form of either undefined values or possible combinations of values. The assertions for which P is true, false, and undefined in founded semantics are captured using three automatically derived predicates, P.`T`, P`.F`, and P.`U`, respectively. The constraint semantics of a set of rules, facts, and meta-constraints is captured using an automatically derived predicate `CS`. For each model `m` in `CS`, the assertion `CS(m)` holds, and `m.`P captures the truth values of predicate P in `m`. All of these predicates can be used explicitly and directly for further reasoning, unlike with the truth values in well-founded semantics, stable model semantics, founded semantics, and constraint semantics.

3. The language further supports the use of *knowledge units* that can be instantiated by any new predicates, including predicates with additional arguments. A knowledge unit, abbreviated as *kunit*, is a set of rules, facts, and meta-constraints. A kunit K can be used in another kunit with an instantiation of the form `use` K $(P_1 = Q_1(Y_{1,1},...,Y_{1,b_i}),\ ...,\ P_n = Q_n(Y_{n,1},...,Y_{n,b_n}))$, which replaces each occurrence P_i in K with Q_i and passes $Y_{i,1},...,Y_{i,b_i}$ as additional arguments to Q. This powerful form of instantiation allows knowledge in a kunit to be re-used in any contexts.

Together, the language allows complex problems to be expressed clearly and easily, where different assumptions can be easily used, combined, and compared for expressing and solving a problem modu-

To appear in EPTCS.

larly, unit by unit. We discuss one example below. The paper presents additional examples for different games that show the power and ease of programming with DA logic.

Example: Unique undefined positions. In an uncertain world, among the most critical information is assertions that have a unique true or false value in all possible ways of satisfying given constraints but cannot be determined to be true by just following founded reasoning. Having both founded semantics and constraint semantics at the same time allows one to find such information.

Consider the following kunits. With default meta-constraints, `win`, `prolog`, and `asp` are complete, `move` is certain in `win_unit`, and `unique` is certain in `cmp_unit`. First, `win_unit` defines `win`—`x` is a winning position if there is a move from `x` to `y` and `y` is not a winning position. Then, `pa_unit` defines `prolog`, `asp`, and `move` and uses `win_unit`. Finally, `cmp_unit` uses `pa_unit` and defines `unique(x)` to be true if (1) `win(x)` is undefined in founded semantics, (2) a constraint model of `pa_unit` exists, and (3) `win(x)` is true in all models in the constraint semantics.

```
kunit win_unit:
  win(x) ← move(x,y) ∧ ¬ win(y)
kunit pa_unit:
  prolog ← ¬ asp
  asp ← ¬ prolog
  move(1,0) ← prolog
  move(1,0) ← asp
  closed(move)
  use win_unit ()
kunit cmp_unit:
  use pa_unit ()
  unique(x) ← win.U(x) ∧ ∃ m ∈ pa_unit.CS ∧ ∀ m ∈ pa_unit.CS | m.win(x)
```

In `pa_unit`, founded semantics gives `move.U(1,0)` (because `prolog` and `asp` are undefined), `win.F(0)` (because there is no move from `0`), and `win.U(1)` (because `win(1)` cannot be true or false). Constraint semantics `pa_unit.CS` has two models: {`prolog, move(1,0), win(1)`} and {`asp, move(1,0), win(1)`}. We see that `win(1)` is true in all two models. So `win.U(1)` from founded semantics is imprecise.

In `cmp_unit`, by definition, `unique(1)` is true. That is, `win(1)` is undefined in founded semantics, the constraint semantics is not empty, and `win(1)` is true in all models of the constraint semantics. ■

Overall, DA logic is essential for general knowledge representation and reasoning, because not only rules but also different assumptions must be captured, and these rules and different inference results must be used modularly for scaled up applications.

Acknowledgments. This work was supported in part by ONR under grant N00014-20-1-2751 and NSF under grants CCF-1954837, CCF-1414078, CNS-1421893, and IIS-1447549.

References

[1] Yanhong A. Liu & Scott D. Stoller (2018): *Founded Semantics and Constraint Semantics of Logic Rules*. In: *Proceedings of the International Symposium on Logical Foundations of Computer Science (LFCS 2018)*, *Lecture Notes in Computer Science* 10703, Springer, pp. 221–241, doi:10.1007/978-3-319-72056-2_14.

[2] Yanhong A. Liu & Scott D. Stoller (2020): *Founded Semantics and Constraint Semantics of Logic Rules*. *Journal of Logic and Computation* 30(8). To appear. Preprint available at `https://arxiv.org/abs/1606.06269`.

[3] Yanhong A. Liu & Scott D. Stoller (2020): *Knowledge of Uncertain Worlds: Programming with Logical Constraints*. In: *Proceedings of the International Symposium on Logical Foundations of Computer Science (LFCS 2020)*, *Lecture Notes in Computer Science* 11972, Springer, pp. 111–127, doi:10.1007/978-3-030-36755-8_8. Also `https://arxiv.org/abs/1910.10346`.