Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification

HIROSHI UNNO, University of Tsukuba, Japan and RIKEN AIP, Japan
TACHIO TERAUCHI, Waseda University, Japan
YU GU, University of Tsukuba, Japan
ERIC KOSKINEN, Stevens Institute of Technology, USA

We present a novel approach to deciding the validity of formulas in first-order fixpoint logic with background theories and arbitrarily nested inductive and co-inductive predicates defining least and greatest fixpoints. Our approach is constraint-based, and reduces the validity checking problem of the given first-order-fixpoint logic formula (formally, an instance in a language called μCLP) to a constraint satisfaction problem for a recently introduced predicate constraint language.

Coupled with an existing sound-and-relatively-complete solver for the constraint language, this novel reduction alone already gives a sound and relatively complete method for deciding μCLP validity, but we further improve it to a novel *modular primal-dual* method. The key observations are (1) μCLP is closed under complement such that each (co-)inductive predicate in the original *primal* instance has a corresponding (co-)inductive predicate representing its complement in the *dual* instance obtained by taking the standard De Morgan's dual of the primal instance, and (2) *partial solutions* for (co-)inductive predicates synthesized during the constraint solving process of the primal side can be used as sound upper-bounds of the corresponding (co-)inductive predicates in the dual side, and vice versa. By solving the primal and dual problems in parallel and exchanging each others' partial solutions as sound bounds, the two processes mutually reduce each others' solution spaces, thus enabling rapid convergence. The approach is also *modular* in that the bounds are synthesized and exchanged at granularity of individual (co-)inductive predicates.

We demonstrate the utility of our novel fixpoint logic solving by encoding a wide variety of temporal verification problems in μCLP, including termination/non-termination, LTL, CTL, and even the full modal μ-calculus model checking of infinite state programs. The encodings exploit the modularity in both the program and the property by expressing each loops and (recursive) functions in the program and sub-formulas of the property as individual (possibly nested) (co-)inductive predicates. Together with our novel modular primal-dual μCLP solving, we obtain a novel approach to efficiently solving a wide range of temporal verification problems.

CCS Concepts: • **Theory of computation → Program verification**; **Logic and verification**; **Constraint and logic programming**; **Automated reasoning**.

Additional Key Words and Phrases: temporal verification, fixpoint logics, constraint logic programming

ACM Reference Format:
Hiroshi Unno, Tachio Terauchi, Yu Gu, and Eric Koskinen. 2023. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification. *Proc. ACM Program. Lang.* 7, POPL, Article 72 (January 2023), 34 pages. https://doi.org/10.1145/3571265

Authors' addresses: Hiroshi Unno, uhiro@cs.tsukuba.ac.jp, University of Tsukuba, Japan and RIKEN AIP, Japan; Tachio Terauchi, terauchi@waseda.jp, Waseda University, Japan; Yu Gu, kou@logic.cs.tsukuba.ac.jp, University of Tsukuba, Japan; Eric Koskinen, eric.koskinen@stevens.edu, Stevens Institute of Technology, USA.

2475-1421/2023/1-ART72
https://doi.org/10.1145/3571265

1 INTRODUCTION

One approach to temporal property verification of programs is to reduce the problem to that of deciding the validity of a formula in a fixpoint logic [Delzanno and Podelski 2001; Fioravanti et al. 2013; Fribourg 1999; Jaffar and Maher 1994; Kobayashi et al. 2019, 2018; Nanjo et al. 2018; Nilsson and Lübcke 2000]. In such an approach, both the program and the property to be verified are encoded together as a fixpoint logic formula so that the logic serves as a *common intermediate language* for uniformly expressing various verification problems, separating the concern of reducing verification problems to logical validity problems from that of solving the validity problems. The verification community has historically embraced such a separation of concerns. For instance, constrained Horn clauses (CHCs) [Bjørner et al. 2015] is a popular formal system for expressing various verification problems, and it facilitated the rapid development of methods for problem reduction and those for solving the reduced problems [Champion et al. 2018; Gurfinkel et al. 2015; Hojjat and Rümmer 2018; Kahsai et al. 2016; Komuravelli et al. 2014; Unno and Kobayashi 2009]. In comparison to predicate constraint systems such as CHCs, an advantage of a fixpoint logic is that it lends itself naturally to modular encoding of verification problems by representing both finite (i.e., terminating) and infinite (i.e., diverging) behavior of program components as least and greatest fixpoints [Kobayashi et al. 2019; Nanjo et al. 2018; Unno et al. 2017a].

In this paper, we present a novel approach to solving the validity checking problem for a first-order fixpoint logic with background theories and arbitrarily nested inductive and co-inductive predicates defining least and greatest fixpoints (formalized as a language called μCLP). Our approach is constraint-based, and reduces the validity checking problem of the given μCLP instance to a constraint satisfaction problem for a predicate constraint language called pfwCSP, a generalization of CHCs that was introduced in the work of Unno et al. [2021]. Coupled with an existing sound-and-relatively-complete solver for pfwCSP from the same work, our novel reduction alone already gives a sound and relatively complete method for deciding μCLP validity, but we further improve the method by extending it to a novel *modular primal-dual* method[1]. The key observations are (1) μCLP is closed under complement where each inductive (resp. co-inductive) predicate in the original *primal* instance has a corresponding co-inductive (resp. inductive) predicate representing its complement in the *dual* instance obtained by taking the De Morgan's dual of the primal, and (2) *partial solutions* (described below) for inductive (resp. co-inductive) predicates synthesized during the constraint solving process of the primal side can be used as sound upper-bounds of the corresponding co-inductive (resp. inductive) predicates in the dual side, and vice versa.

Based on the observations, our modular primal-dual approach translates both the primal μCLP instance and its dual to their corresponding pfwCSP constraint sets, and solves both sets of constraints in parallel. Partial solutions are assignments to predicate variables that satisfy some but not necessarily all clauses of the given constraint set[2]. They are obtained as by-products of the constraint solving process, and while generally insufficient for solving the given set of constraints, they are guaranteed to be lower-bounds of the fixpoints corresponding to the (co-)inductive predicate defined by the clauses that they satisfy. Thus, they can be used as sound upper-bounds of the corresponding fixpoints in the dual instance.

For example, in termination verification, an inductive predicate $X(\widetilde{x})$ in the primal problem may represent the set of states from which some loop in the program always terminates, and the corresponding co-inductive predicate in the dual problem represents the set of states from

[1] Here, the terminology *relative completeness* is in the sense of Jhala and McMillan [2006]; Terauchi and Unno [2015]. That is, the solver is guaranteed to terminate relative to the assumption that there is a solution syntactically expressible as predicates of the background theory.

[2] Technically, we also allow partial solutions to only satisfy sound *under-approximations* of the clauses (cf. Section 5.2 for details).

which the same loop may diverge. Then, a partial solution to the co-inductive predicate in the dual problem, say φ, represents a lower-bound on the set of states from which the loop may diverge. Therefore, its complement, $\neg\varphi$, is an upper-bound on the set of states from which the loop always terminates and can be used as a sound upper-bound of the inductive predicate in the primal problem. The bound is sound in the sense that asserting it does not rule out any actual solutions. We assert the bound by adding the clause $\underline{X}(\widetilde{x}) \Rightarrow \neg\varphi$ to the primal's constraint set where $\underline{X}$ is the predicate variable corresponding to X (cf. Section 5.1 for details). By solving the primal and dual problems in parallel and exchanging each others' partial solutions as sound bounds in this way, the two processes mutually reduce each others' solution spaces, thus enabling rapid convergence. An analogy can be made to how clause-learning DPLL SAT solvers reduce the solution space by asserting learned clauses. The approach is also *modular* in that the bounds are synthesized and exchanged at granularity of individual (co-)inductive predicates. The approach synergizes well with the aforementioned modularity present in fixpoint logic encodings of verification problems as it lends itself to bounds synthesized and exchanged at granularity of individual program components, as seen in the example above, as well as at the granularity of sub-properties of the properties to be verified (e.g., sub-formulas of a temporal logic formula expressing the property to be verified).

We have implemented the approach in a tool called MuVal and applied it to a diverse collection of temporal verification problems including termination/non-termination, LTL, CTL, and even the full modal μ-calculus model checking of infinite-state programs. For encoding the problems, we adopt the approaches of Kobayashi et al. [2019]; Nanjo et al. [2018]; Unno et al. [2017a] to exploit the modularity present in both the program and the property by expressing individual loops and (recursive) functions in the program and sub-formulas of the property as individual (possibly nested) (co-)inductive predicates. Thanks to our novel modular primal-dual solving method, MuVal is able to go beyond the capabilities of existing related tools. Namely, MuVal beats AProVE [Giesl et al. 2017], the winner of 2021 Termination Competition, on termination benchmarks in number of solved instances. Additionally, MuVal was able to solve 4 challenging benchmarks from Dietsch et al. [2015] that cannot be solved by their UltimateLTLAutomizer, a state-of-the-art tool for LTL model checking of infinite state programs[3]. We summarize the paper's contributions below.

- The novel modular primal-dual approach to deciding the validity of a first-order fixpoint logic with background theories by a reduction to parallel pfwCSP constraint solving.
- The implementation of the approach in the tool MuVal and experimental validation on a diverse collection of temporal verification problems.

The rest of the paper is organized as follows. Section 2 gives a brief overview with a running example. Section 3 defines μCLP, a first-order fixpoint logic with background theories. Section 4 reviews pfwCSP and its solver PCSat. Section 5 formalizes the reduction of μCLP to pfwCSP and the novel modular primal-dual solving method. Section 6 reports on the implementation and experimental evaluation of our method. We discuss related work in Section 7 and conclude the paper in Section 8. We note that pfwCSP and its solver PCSat reviewed in Section 4 are from Unno et al. [2021] and are not new contributions of this paper. They are described there for the purpose of making the paper self-contained (and also with the intent to later refer to the description of PCSat to explain the modifications needed for our work in Section 5). As also remarked above, the main novel contributions of this paper are the primal-dual solving method described in detail in Section 5 and its implementation presented in Section 6. This includes the reduction from μCLP validity to pfwCSP satisfiability described in detail in Section 5.1 which has only appeared previously in an unrefereed arXiv paper [Unno et al. 2020].

[3] Our experiments with MuVal even discovered that the benchmark files contained incorrect classifications, claiming "safe" when they actually are not.

2 OVERVIEW

This section highlights the contributions of our work using a running example.

2.1 Modeling Verification Problems in First-Order Fixpoint Logic

Let us consider the termination verification problem of the following program taken from the benchmark set of the FuncTion tool [Urban 2013; Urban and Miné 2014] slightly modified to illuminate the salient aspects of our approach:

```
assume (x2 <= 3);
while (x1 >= 0 && x2 >= 0) {
  if (nondet()) { while (x2 != 3 && nondet()) { x2 = x2 + 1; }
                  x1 = x1 - 1; } x2 = x2 - 1; }
```

Here, `nondet()` returns a non-deterministic Boolean value. This program is *always terminating* for any external integer inputs x1,x2 and any internal Boolean non-deterministic choices. Note that the termination is witnessed by, for example, the lexicographic order of x1,x2.

As we shall show formally in Section 3.2, we systematically encode the termination verification problem to a validity checking problem for a first-order fixpoint logic (formally, as a μCLP). Our encoding adopts the approaches given in Kobayashi et al. [2019]; Nanjo et al. [2018]; Unno et al. [2017a] to *modularly* encode the verification problem using *both least and greatest fixpoints*. Roughly, the idea is to use least (resp. greatest) fixpoints to describe the terminating (resp. non-terminating) behavior of program components. For the running example, we obtain the following μCLP $\mathcal{P}_{\text{term}}$:

$$\forall x_1, x_2 : \texttt{int}.\ x_2 > 3 \lor I(x_1, x_2) \text{ where}$$

$$\begin{array}{rcl} I(x_1,x_2) & =_\mu & \neg(x_1 \geq 0 \land x_2 \geq 0) \lor \left(\begin{array}{l} I(x_1, x_2 - 1) \land \mathcal{J}(x_2) \land \\ \forall x_2' : \texttt{int}.\ \mathit{NP}(x_2, x_2') \lor I(x_1 - 1, x_2' - 1) \end{array} \right) \\ \mathcal{J}(x_2) & =_\mu & \neg(x_2 \neq 3) \lor \mathcal{J}(x_2 + 1) \\ \mathit{NP}(x_2, x_2') & =_\nu & \neg\left(\ x_2' = x_2 \lor x_2 \neq 3 \land \neg \mathit{NP}(x_2 + 1, x_2')\ \right) \end{array}$$

Here, $\mathcal{J}$ is an *inductive predicate* defined as the *least fixpoint* (indicated by $=_\mu$) of the function $\mathcal{F}(\mathcal{J})(x_2) \triangleq \neg(x_2 \neq 3) \lor \mathcal{J}(x_2 + 1)$ over predicates. I is also an inductive predicate and is defined as a least fixpoint. By contrast, NP is a *co-inductive predicate* defined as the *greatest fixpoint* (indicated by $=_\nu$) of the function $\mathcal{G}(\mathit{NP})(x_2, x_2') \triangleq \neg(x_2' = x_2 \lor x_2 \neq 3 \land \neg \mathit{NP}(x_2 + 1, x_2'))$. Roughly, $I(x_1, x_2)$ and $\mathcal{J}(x_2)$ characterize the weakest pre-conditions for the termination of the outer and the inner loops, respectively. Note that the inner loop terminates for all non-deterministic choices if and only if $\neg(x_2 \neq 3)$ eventually holds after a *finite* number of iterations of incrementing x_2, which is here enforced by the *least-fixpoint* definition of $\mathcal{J}$. $\mathit{NP}(x_2, x_2')$ denotes the *complement* of the following inductive predicate $P(x_2, x_2')$ which characterizes the transition relation of the inner loop: $P(x_2, x_2') =_\mu x_2' = x_2\ \lor x_2 \neq 3 \land P(x_2 + 1, x_2')$. In the definition of I, $\mathit{NP}(x_2, x_2')$ is used to bind x_2' to a possible value of the program variable x2 upon the termination of the inner loop, encapsulating the internal behavior of the inner loop. Thus, the query formula is valid if and only if the program terminates for all initial integer valuations of x1 and x2 and for all internal non-deterministic choices. Though we could encode the termination verification problem using only least fixpoints by regarding the given program as a single monolithic transition system, this example demonstrates an important property of first-order fixpoint logics such as μCLP: *the verification problem can be encoded modularly so that each program component (e.g., the inner and outer loop in the running example) can be represented by separate (co-)inductive predicates.*

The other important property of μCLP that we shall exploit in our approach is that it is *closed under complement.* Namely, taking the De Morgan dual of a μCLP instance yields a complement μCLP instance that contains inductive (resp. co-inductive) predicate representing the dual of the each co-inductive (resp. inductive) predicate in the primal instance. As the dual of the running example $\mathcal{P}_{\text{term}}$, we obtain the μCLP $\mathcal{P}_{\text{nterm}}$ shown below:

$$\exists x_1, x_2 \colon \texttt{int}.\ x_2 \leq 3 \land \mathit{NI}(x_1, x_2) \text{ where}$$

$$\begin{array}{rl} \mathit{NI}(x_1, x_2) & =_\nu \ \ x_1 \geq 0 \land x_2 \geq 0 \land \left(\begin{array}{l} \mathit{NI}(x_1, x_2 - 1) \lor \mathit{N\mathcal{J}}(x_2) \lor \\ \exists x_2' \colon \texttt{int}.\ P(x_2, x_2') \land \mathit{NI}(x_1 - 1, x_2' - 1) \end{array} \right) \\ \mathit{N\mathcal{J}}(x_2) & =_\nu \ x_2 \neq 3 \land \mathit{N\mathcal{J}}(x_2 + 1) \\ P(x_2, x_2') & =_\mu \ \ x_2' = x_2 \lor x_2 \neq 3 \land P(x_2 + 1, x_2') \end{array}$$

Intuitively, $\mathit{NI}(x_1, x_2)$ and $\mathit{N\mathcal{J}}(x_2)$ respectively characterize the weakest pre-conditions for the *non-termination* of the outer and the inner loops, generalizing the notion of recurrent sets [Gupta et al. 2008] for *modular verification*: the non-emptiness of $\mathit{NI}(x_1, x_2)$ and $\mathit{N\mathcal{J}}(x_2)$ respectively implies the non-termination of the inner and outer loops. Recall that $P(x_2, x_2')$ characterizes the strongest post-condition of the inner loop. Note that each inductive predicate in the primal instance $\mathcal{P}_{\text{term}}$ (i.e., I and $\mathcal{J}$) has a corresponding co-inductive predicate in the dual $\mathcal{P}_{\text{nterm}}$ (i.e., NI and $\mathit{N\mathcal{J}}$), and the co-inductive predicate in the primal (i.e., NP) has a corresponding inductive predicate in the dual (i.e., P). Note that the encoding is modular in the sense that the terminating and non-terminating behavior of each program components are encoded as separate (co-)inductive predicates, namely by I and NI for the inner loop and by $\mathcal{J}$ and $\mathit{N\mathcal{J}}$ for the outer loop. As we shall see next, MuVal tries to prove the validity of the primal $\mathcal{P}_{\text{term}}$ and the dual $\mathcal{P}_{\text{nterm}}$ simultaneously in parallel by sharing learned information about each other's (co-)inductive predicates.

2.2 Reduction to pfwCSP and Modular Primal-Dual Solving

Our main contribution is a novel primal-dual approach to deciding the validity of μCLP. The approach works by (1) soundly and completely reducing both primal and dual μCLP instances to satisfiability problems of the predicate constraint language pfwCSP, and (2) solving the two constraint satisfiability problems in parallel while trading learned information. pfwCSP was introduced in recent work by Unno et al. [2021] and generalizes CHCs [Bjørner et al. 2015] to arbitrary (i.e., possibly non-Horn) clauses, functionality constraints, and well-foundedness constraints over predicate variables.

We overview the reduction to pfwCSP on the running example. The reduction is inspired by the deduction technique in Nanjo et al. [2018] that eliminates least and greatest fixpoints by over- and under-approximations via (co-)inductive invariants and well-founded relations, and eliminates quantifiers by Skolemization. For the termination verification problem $\mathcal{P}_{\text{term}}$, our reduction gives the following pfwCSP C_{term} whose term variables are implicitly universally quantified:

$$\begin{array}{rl} (1) & x_2 > 3 \lor \underline{I}(x_1, x_2) \\ (2) & \underline{I}(x_1, x_2) \Rightarrow \ \neg(x_1 \geq 0 \land x_2 \geq 0) \lor \\ & \qquad \left(\begin{array}{l} \underline{I}(x_1, x_2 - 1) \land I_{\Downarrow}(x_1, x_2, x_1, x_2 - 1) \land \underline{\mathcal{J}}(x_2) \land \\ (\underline{\mathit{NP}}(x_2, x_2') \lor \underline{I}(x_1 - 1, x_2' - 1) \land I_{\Downarrow}(x_1, x_2, x_1 - 1, x_2' - 1)) \end{array} \right) \\ (3) & \underline{\mathcal{J}}(x_2) \Rightarrow \neg(x_2 \neq 3) \lor \underline{\mathcal{J}}(x_2 + 1) \land \mathcal{J}_{\Downarrow}(x_2, x_2 + 1) \\ (4) & \underline{\mathit{NP}}(x_2, x_2') \Rightarrow \neg \left(\ x_2' = x_2 \lor x_2 \neq 3 \land \neg \underline{\mathit{NP}}(x_2 + 1, x_2')\ \right) \end{array}$$

$\underline{I}$, $\underline{\mathcal{J}}$, and $\underline{\mathit{NP}}$ are *predicate variables* that respectively represent *under-approximations* of (co-)inductive predicates I, $\mathcal{J}$, and NP of $\mathcal{P}_{\text{term}}$. $I_{\Downarrow}$ and $\mathcal{J}_{\Downarrow}$ are *well-founded* predicate variables that represent well-founded relations and used to enforce *bounded* unfoldings of inductive predicates I and $\mathcal{J}$, respectively[4]. Namely, $\mathcal{J}_{\Downarrow}(x_2, x_2 + 1)$ constrains the formal argument x_2 of J and the actual argument

[4] Here, we use the terminology *bounded unfolding* in a rather liberal sense. Namely, the unfolding need not be finite as its *bound* may be witnessed by a (non-finitary) well-founded relation.

$x_2 + 1$ of the recursive call to $\mathcal{J}$ to be related by a well-founded relation, and $I_{\Downarrow}(x_1, x_2, x_1, x_2 - 1)$ and $I_{\Downarrow}(x_1, x_2, x_1 - 1, x_2' - 1)$ constrain the formal arguments (x_1, x_2) and the actual arguments of the two recursive calls to be related by a well-founded relation. The transformation is inspired by an analogous transformation done in *binary reachability analysis* [Cook et al. 2006; Grebenshchikov et al. 2012; Kuwahara et al. 2014] that reduces termination verification problems to safety verification problems.

Coupled with the sound-and-relatively-complete pfwCSP solver PCSat [Unno et al. 2021], our novel reduction alone already gives a sound and relatively complete method for deciding μCLP validity. As remarked before, we make a further innovation by extending the method to a modular primal-dual method in which the dual μCLP problem is also reduced to a pfwCSP satisfiability problem by applying the same reduction process. For the running example, we obtain from the dual problem $\mathcal{P}_{\text{nterm}}$ the following pfwCSP C_{nterm}:

$$\begin{array}{ll}
(5) & S_\lambda(x_1) \land T_\lambda(x_2) \Rightarrow x_2 \le 3 \land \underline{NI}(x_1, x_2) \\
(6) & \underline{NI}(x_1, x_2) \Rightarrow \;\; x_1 \ge 0 \land x_2 \ge 0 \land \\
 & \qquad\qquad\qquad \left(\begin{array}{l} \underline{NI}(x_1, x_2 - 1) \lor \underline{N\mathcal{J}}(x_2) \lor \\ U_\lambda(x_1, x_2) \Rightarrow \left(\; \underline{P}(x_2, x_2') \land \underline{NI}(x_1 - 1, x_2' - 1) \; \right) \end{array} \right) \\
(7) & \underline{N\mathcal{J}}(x_2) \Rightarrow x_2 \ne 3 \land \underline{N\mathcal{J}}(x_2 + 1) \\
(8) & \underline{P}(x_2, x_2') \Rightarrow \;\; x_2' = x_2 \lor x_2 \ne 3 \land \underline{P}(x_2 + 1, x_2') \land P_{\Downarrow}(x_2, x_2', x_2 + 1, x_2')
\end{array}$$

As above, $\underline{NI}$, $\underline{N\mathcal{J}}$, and $\underline{P}$ represent under-approximations of (co-)inductive predicates NI, $N\mathcal{J}$, and P of $\mathcal{P}_{\text{nterm}}$, and $P_{\Downarrow}$ is a well-founded predicate variable used to enforce a bounded unfolding of P. S_λ, T_λ, and U_λ are *functional* predicate variables that characterize total functions to be synthesized and used to Skolemize the existential quantification of the term variables x_1, x_2 in clause (5) and x_2' in clause (6), respectively.

Our modular primal-dual solver MuVal now tries to decide the satisfiability of both C_{term} and C_{nterm} by running two PCSat processes in parallel while synthesizing and exchanging sound bounds to reduce each others' solution spaces. As remarked before, the bounds are synthesized from *partial solutions*. For example, $\rho_t \triangleq \{\underline{\mathcal{J}} \mapsto \lambda x_2.x_2 = 3, \mathcal{J}_{\Downarrow} \mapsto \_\}$ is a partial solution for $\underline{\mathcal{J}}$ in C_{term} as it satisfies the clauses that define $\underline{\mathcal{J}}$ (i.e., clause (3)). While the partial solution is not an actual solution of C_{term}, from this information, we know that $\rho_t(\underline{\mathcal{J}})$ is a *lower-bound* of the inductive predicate J of $\mathcal{P}_{\text{term}}$ and therefore its complement is an *upper-bound* of the dual co-inductive predicate NJ of $\mathcal{P}_{\text{nterm}}$. We assert the upper-bound in the dual pfwCSP C_{nterm} by adding the clause $\underline{NJ}(x_2) \Rightarrow x_2 \ne 3$. Roughly, the exchange says: the primal proved that the inner loop always terminates when x2 = 3 and communicated the information to the dual so that it may limit its search for states from which the inner loop may diverge to only those that satisfy x2 ≠ 3.

The communicated bound is still insufficient for the dual process to prove its satisfiability (inevitably because the program actually always terminates and therefore C_{nterm} is unsatisfiable). But it may lead to a synthesis and exchange of another bound, this time from dual to primal: $\underline{\mathcal{J}}(x_2) \Rightarrow x_2 \le 3$, an upper-bound on the set of states from which the inner loop always terminate. With such bounds, MuVal arrives at the following actual solution for C_{term}, thus proving the validity of $\mathcal{P}_{\text{term}}$ (and hence the termination of the program):

$$\begin{array}{rcl}
\underline{I}(x_1, x_2) & \mapsto & 3 \ge x_2, \\
\underline{\mathcal{J}}(x_2) & \mapsto & x_2 \ge 0 \land x_2 \le 3, \\
\underline{NP}(x_2, x_2') & \mapsto & x_2 \ge 0 \land x_2 \le 3 \land x_2' \ge 4
\end{array}$$

The assignments to the well-founded predicate variables $I_{\Downarrow}$ and $\mathcal{J}_{\Downarrow}$ are deferred to Appendix A.

Note that our primal-dual solving method takes advantage of the modularity in the μCLP encoding by synthesizing partial solutions and exchanging the bounds derived from them at the granularity individual (co-)inductive predicates. Namely, in the example above, the information about the set of states from which the inner loop terminates (i.e., the information about the co-inductive predicate $N\mathcal{J}$) was synthesized by the primal side and communicated to the dual side, and the the information about the set of states from which the inner loop does not terminate (i.e., the information about the inductive predicate $\mathcal{J}$) was synthesized by the dual side and communicated to the primal side. These information are then asserted as clauses that upper-bound the possible solutions for the predicate variables corresponding to the (co-)inductive predicates, thus reducing the solution spaces of the constraint solver processes.

In the remainder of this section, we informally describe how partial solutions such as the ones in the running example above are synthesized. We defer the formal details to Sections 4.1 and 5.2.

PCSat adopts the counterexample-guided inductive synthesis (CEGIS) approach to semi-decide the satisfiability of the given pfwCSP instance. As standard for a CEGIS-based approach, it consists of two sub-phases, a *synthesis phase* and a *validation phase*, and maintains a set of *counterexamples*. A counterexample is a ground clause obtained by instantiating the term variables of a clause in the input pfwCSP instance. In each iteration, the synthesis phase attempts to synthesize a *candidate solution* that satisfies the current set of counterexamples, and then the validation phase checks if the candidate solution is a *genuine solution*, i.e., one that actually satisfies the input pfwCSP instance. If the validation phase detects that the candidate solution is not genuine, then it adds to the set of counterexamples ground clauses that are unsatisfied by the candidate solution[5]. The semi-algorithm terminates with success when the validation phase detects that the latest candidate solution is genuine, and terminates with failure when the synthesis phase detects the current set of counterexamples unsatisfiable (i.e., the input pfwCSP instance is actually unsatisfiable).

As described above, in an ordinary CEGIS-based approach, non-genuine candidate solutions are only used to obtain additional counterexamples. Our novel modular primal-and-dual approach adds an additional phase that follows the synthesis phase in which the non-genuine candidate solution is analyzed to see if it is a *partial solution*. As discussed in the example above and in Section 1, roughly, a candidate solution ρ is detected to be a partial solution for an ordinary predicate variable X if ρ satisfies the clauses that define X, that is, the clauses in which X appears negatively and transitively the clauses that define the ordinary predicate variables appearing positively in such clauses. For instance, in the example above, only $\underline{\mathcal{J}}$ itself appears as a positively occurring ordinary predicate variable in the clause defining $\underline{\mathcal{J}}$ (i.e., clause (3)), and so a candidate solution that satisfies clause (3) (but may fail to satisfy the other clauses) is a partial solution for $\underline{\mathcal{J}}$.

As we shall describe in more detail in Section 5.2, the formal definition of partial solutions extends the above by also allowing a partial solution to only satisfy *under-approximations* of the clauses obtained by replacing some of the positively-occurring predicate variables by their lower-bounds, where the lower-bounds are computed from (previously discovered) partial solutions themselves. As we shall show in Section 5.2, the extension allows discovering more partial solutions, especially for pfwCSP instances obtained from encoding methods that generates (co-)inductive predicate definitions that mutually depend on each others.

3 μCLP: A FIRST-ORDER FIXPOINT LOGIC WITH BACKGROUND THEORIES

This section defines μCLP, a first-order fixpoint logic with background theories. Its name is derived from constraint logic programming (CLP) [Jaffar and Maher 1994] because it can be seen as an extension of CLP with arbitrarily-nested inductive and co-inductive predicate definitions and quantifiers.

[5] Such ground clauses are obtained as by-products of checking the genuineness of the candidate solution (cf. Section 4.1).

Let $\mathcal{T}$ be a (possibly many-sorted) first-order theory with the signature Σ. The syntax of $\mathcal{T}$-formulas ϕ and $\mathcal{T}$-terms t is:

$$\begin{aligned}\phi &::= X(t_1,\dots,t_{\mathrm{ar}(X)}) \mid p(t_1,\dots,t_{\mathrm{ar}(p)}) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \forall x\colon s.\phi \\ t &::= x \mid f(t_1,\dots,t_{\mathrm{ar}(f)})\end{aligned}$$

Here, the meta-variables X and x respectively range over predicate and term variables. The meta-variables p and f respectively denote predicate and function symbols of the signature Σ. We use s as a meta-variable ranging over sorts of the signature Σ. We write $\bullet$ for the sort of propositions and $s_1 \longrightarrow s_2$ for the sort of functions from s_1 to s_2. The return sort of a predicate variable is $\bullet$. We write $\mathrm{ar}(o)$ and $\mathrm{sort}(o)$ respectively for the arity and the sort of a syntactic element o. A function f represents a constant if $\mathrm{ar}(f) = 0$. We write $\mathit{ftv}(\phi)$ and $\mathit{fpv}(\phi)$ respectively for the set of free term and predicate variables of ϕ. We write $\widetilde{x}$ for a sequence of term variables, $|\widetilde{x}|$ for the length of $\widetilde{x}$, and ϵ for the empty sequence. We assume the usual derived forms $\phi_1 \wedge \phi_2$, $\phi_1 \Rightarrow \phi_2$, and $\exists x\colon s.\phi$.

A μCLP $\mathcal{P}$ over the theory $\mathcal{T}$ is a sequence of mutually (co-)recursive equations of the form: $(X_1(\widetilde{x}_1) =_{\alpha_1} \phi_1);\ \dots;\ (X_m(\widetilde{x}_m) =_{\alpha_m} \phi_m)$. Here, $\alpha_i \in \{\mu, \nu\}$ and for any $i, j \in \{1,\dots,m\}$, X_i may occur only positively in ϕ_j. Informally, $X(\widetilde{x}) =_\mu \phi$ (resp. $X(\widetilde{x}) =_\nu \phi$) represents the inductive (resp. co-inductive) predicate $\mu X(\widetilde{x}).\ \phi$ (resp. $\nu X(\widetilde{x}).\ \phi$) defined as the least (resp. greatest) fixpoint of the function $\mathcal{F}(X) = \lambda\widetilde{x}.\phi$. Note here that $\mathcal{F}$ is monotonic because the bound predicate variable X occurs only positively in the body ϕ.

The order of the equations is important. They express nestings of quantifiers μ and ν in ordinary fixpoint logics. For example, a μCLP $(X =_\nu X \wedge Y);\ (Y =_\mu X \vee Y)$ corresponds to $\nu X.X \wedge (\mu Y.X \vee Y)$ whereas its order reversed, that is, $(Y =_\mu X \vee Y);\ (X =_\nu X \wedge Y)$,corresponds to $\mu Y.(\nu X.X \wedge Y) \vee Y$.

We define $\mathrm{dom}(\mathcal{P}) = \{X_1,\dots,X_m\}$. We say that $\mathcal{P}$ is *closed* if $\mathit{ftv}(\phi_i) \subseteq \{\widetilde{x}_i\}$, $\mathit{fpv}(\phi_i) \subseteq \mathrm{dom}(\mathcal{P})$ for each equation $X_i(\widetilde{x}_i) =_{\alpha_i} \phi_i$ of $\mathcal{P}$. A *query* for a μCLP $\mathcal{P}$ is defined as a $\mathcal{T}$-formula ϕ. We say that the query is *closed* with respect to $\mathcal{P}$ if $\mathit{ftv}(\phi) = \emptyset$ and $\mathit{fpv}(\phi) \subseteq \mathrm{dom}(\mathcal{P})$. For a query ϕ and an interpretation ρ of $\mathit{ftv}(\phi) \cup \mathit{fpv}(\phi)$, we write $\rho \models \phi$ if ϕ is true under ρ. For a μCLP $\mathcal{P}$, the interpretation $[\![\mathcal{P}]\!](\emptyset)$ maps $\mathrm{dom}(\mathcal{P})$ to their fixpoints (cf. Section 3.1 for the formal definition). A μCLP *validity problem instance* is pair $(\phi, \mathcal{P})$ of a closed μCLP $\mathcal{P}$ and a query ϕ closed with respect to $\mathcal{P}$. We say that $(\phi, \mathcal{P})$ is *valid*, written $\models (\phi, \mathcal{P})$, if $[\![\mathcal{P}]\!](\emptyset) \models \phi$.

We define the *De Morgan dual* (or simply *dual*) of a closed μCLP $\mathcal{P} = (X_i(\widetilde{x}_i) =_\alpha \phi_i)_{i=1}^m$ to be the μCLP $\neg\mathcal{P}$ defined by $(X_i^\neg(\widetilde{x}_i) =_{\neg\alpha} \sigma(\neg\phi_i))_{i=1}^m$ where $\sigma \triangleq \{X_1 \mapsto \neg X_1^\neg, \dots X_m \mapsto \neg X_m^\neg\}$, $\neg\mu \triangleq \nu$, and $\neg\nu \triangleq \mu$. The *dual* of a μCLP validity problem instance $(\phi, \mathcal{P})$ is the μCLP validity problem instance $(\sigma(\neg\phi), \neg\mathcal{P})$ where σ is the substitution used in the definition of $\neg\mathcal{P}$ as shown above. We say that each predicate variable X_i in the primal μCLP $\mathcal{P}$ *corresponds* to the predicate variable $\sigma(X_i) = X_i^\neg$ in its dual $\neg\mathcal{P}$, and vice versa.

μCLP generalizes CLP and existing first-order fixpoint logics with background theories such as Mu-Arithmetic [Bradfield 1999; Kobayashi et al. 2019; Lubarsky 1993; Nanjo et al. 2018], and can naturally encode diverse classes of temporal program verification problems:

- Termination and non-termination verification for imperative programs (Section 3.2).
- Model checking of labeled transition systems (LTSs) and recursive programs where specifications are given as non-deterministic Büchi word automata, which strictly subsume LTL. The encodings for LTSs and recursive programs are explained in Section 3.3 and Kobayashi et al. [2019], respectively.
- Modal-μ-calculus model checking of imperative programs. The encoding is given in Kobayashi et al. [2019].
- Value-dependent temporal properties verification of higher-order effectful functional programs. The encoding is given in Nanjo et al. [2018].

As concrete examples of encodings, we present in Section 3.2 a modular encoding of termination and non-termination verification for imperative programs, and in Section 3.3, a modular encoding of linear temporal property verification for LTSs. As we shall show, the former is modular in the program side, that is, the μCLP encoding for a statement of a program is built from the predicates defined in the μCLP encodings for its sub-statements. By contrast, the latter is modular in the property side, that is, a temporal property is given by a Büchi automaton whereby the encoding defines inductive and co-inductive predicates for each state of the automaton whose definitions are built from the (co-)inductive predicates defined for the states one-step reachable from that state. Section 3.1 presents the formal semantics of μCLP. Readers acquainted with first-order fixpoint logics and their semantics may skip the section.

3.1 Semantics

We formalize the semantics of μCLP. Let $\mathcal{A} = (\mathcal{D}, \Sigma, I)$ be the structure of the background first-order theory $\mathcal{T}$. Here, $\mathcal{D}$ is the universe, Σ is the signature, and I is the interpretation function for the predicate and function symbols in Σ. We write $\mathcal{D}_s$ for the set of values in $\mathcal{D}$ of the sort s. In particular, we define $\mathcal{D}_\bullet \triangleq \{\top, \bot\}$ for the sort $\bullet$ of propositions. For a sequence $\widetilde{s} = s_1, \dots, s_m$ of sorts with $m \geq 0$, we write $\mathcal{D}_{\widetilde{s}}$ for the sequence $\mathcal{D}_{s_1}, \dots, \mathcal{D}_{s_m}$. We define $\mathcal{D}_{(\widetilde{s} \to s)} \triangleq \mathcal{D}_{\widetilde{s}} \to \mathcal{D}_s$. We assume that $I(p) \in \mathcal{D}_{\widetilde{s}} \to \mathcal{D}_\bullet$ if $\mathrm{sort}(p) = \widetilde{s} \to \bullet$, and $I(f) \in \mathcal{D}_{\widetilde{s}} \to \mathcal{D}_s$ if $\mathrm{sort}(f) = \widetilde{s} \to s$. We introduce the partially ordered sets $(\mathcal{D}_{(\widetilde{s} \to \bullet)}, \sqsubseteq_{(\widetilde{s} \to \bullet)})$ by defining

$$\sqsubseteq_\bullet \triangleq \{(\top,\top), (\bot,\top), (\bot,\bot)\}, \qquad \sqsubseteq_{(\widetilde{s}\to\bullet)} \triangleq \{(f,g) \mid \forall \widetilde{v} \in \mathcal{D}_{\widetilde{s}}. f(\widetilde{v}) \sqsubseteq_\bullet g(\widetilde{v})\}.$$

Note that the least upper bound $\sqcup_{(\widetilde{s}\to\bullet)}$ and the greatest lower bound $\sqcap_{(\widetilde{s}\to\bullet)}$ operators with respect to $\sqsubseteq_{(\widetilde{s}\to\bullet)}$ satisfy

$$\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl@{\qquad}lcl}
\top \sqcap_\bullet \top & = & \top, & \top \sqcap_\bullet \bot & = & \bot, & \bot \sqcap_\bullet \top & = & \bot, & \bot \sqcap_\bullet \bot & = & \bot, \\
\top \sqcup_\bullet \top & = & \top, & \top \sqcup_\bullet \bot & = & \top, & \bot \sqcup_\bullet \top & = & \top, & \bot \sqcup_\bullet \bot & = & \bot
\end{array}$$

$$f \sqcap_{(\widetilde{s}\to\bullet)} g = \lambda \widetilde{v} \in \mathcal{D}_{\widetilde{s}}.\, f(\widetilde{v}) \sqcap_\bullet g(\widetilde{v}), \qquad f \sqcup_{(\widetilde{s}\to\bullet)} g = \lambda \widetilde{v} \in \mathcal{D}_{\widetilde{s}}.\, f(\widetilde{v}) \sqcup_\bullet g(\widetilde{v})$$

Note that $(\mathcal{D}_{\widetilde{s}\to\bullet}, \sqsubseteq_{\widetilde{s}\to\bullet})$ is a complete lattice. The least (resp. greatest) element of $\mathcal{D}_{\widetilde{s}\to\bullet}$ is $\lambda\widetilde{x}.\ \bot$ (resp. $\lambda\widetilde{x}.\ \top$). The negation operator $\neg_{(\widetilde{s}\to\bullet)}$ is defined by

$$\neg_\bullet \top = \bot, \qquad \neg_\bullet \bot = \top, \qquad \neg_{(\widetilde{s}\to\bullet)} f = \lambda \widetilde{v} \in \mathcal{D}_{\widetilde{s}}.\, \neg_\bullet f(\widetilde{v})$$

Given a $\mathcal{T}$-formula ϕ and an interpretation ρ of free term and predicate variables in ϕ, we write $[\![\phi]\!](\rho)$ for the truth value of ϕ which is defined as follows:

$$\begin{array}{rcl@{\qquad}rcl}
[\![X(\widetilde{t})]\!](\rho) & \triangleq & \rho(X)([\![\widetilde{t}]\!](\rho)), & [\![p(\widetilde{t})]\!](\rho) & \triangleq & I(p)([\![\widetilde{t}]\!](\rho)), \\
[\![\phi_1 \vee \phi_2]\!](\rho) & \triangleq & [\![\phi_1]\!](\rho) \sqcup_\bullet [\![\phi_2]\!](\rho), & [\![\phi_1 \wedge \phi_2]\!](\rho) & \triangleq & [\![\phi_1]\!](\rho) \sqcap_\bullet [\![\phi_2]\!](\rho), \\
[\![\exists x \colon s.\phi]\!](\rho) & \triangleq & \bigsqcup_\bullet \{\, [\![\phi]\!](\rho\,\{x \mapsto v\}) \mid v \in \mathcal{D}_s \,\}, & [\![\neg\phi]\!](\rho) & \triangleq & \neg_\bullet [\![\phi]\!](\rho) \\
[\![\forall x \colon s.\phi]\!](\rho) & \triangleq & \bigsqcap_\bullet \{\, [\![\phi]\!](\rho\,\{x \mapsto v\}) \mid v \in \mathcal{D}_s \,\}, & & & \\
[\![x]\!](\rho) & \triangleq & \rho(x), & [\![f(\widetilde{t})]\!](\rho) & \triangleq & I(f)([\![\widetilde{t}]\!](\rho))
\end{array}$$

Here, we assume that $\rho(x) \in \mathcal{D}_{\mathrm{sort}(x)}$ and $\rho(X) \in \mathcal{D}_{\widetilde{s}} \to \mathcal{D}_\bullet$ if $\mathrm{sort}(X) = \widetilde{s} \to \bullet$. We write $\rho \models \phi$ if and only if $[\![\phi]\!](\rho') = \top$ holds for any extension ρ' of the interpretation ρ for the term and predicate variables in $(\mathit{ftv}(\phi) \cup \mathit{fpv}(\phi)) \setminus \mathrm{dom}(\rho)$, where $\mathrm{dom}(\rho)$ represents the domain of ρ. We say the given formula ϕ is *valid* and write $\models \phi$ if and only if $\emptyset \models \phi$ holds.

Given a μCLP $\mathcal{P}$ and an interpretation ρ of free term and predicate variables in $\mathcal{P}$, we write $[[\mathcal{P}]](\rho)$ for the predicate interpretation for $\mathrm{dom}(\mathcal{P})$ induced by $\mathcal{P}$ which is defined by

$$\begin{aligned}
[\![\epsilon]\!](\rho) &\triangleq \emptyset,\\
[\![(X(\widetilde{x}) =_\alpha \phi);\ \mathcal{P}]\!](\rho) &\triangleq [\![X(\widetilde{x}) =_\alpha \phi]\!]_{\mathcal{P}}(\rho) \uplus [[\mathcal{P}]](\rho \uplus [[X(\widetilde{x}) =_\alpha \phi]]_{\mathcal{P}}(\rho)),\\
[\![X(\widetilde{x}) =_\alpha \phi]\!]_{\mathcal{P}}(\rho) &\triangleq \left\{X \mapsto \mathbf{FP}_\alpha^{\mathrm{sort}(X)}\left(\lambda q.\lambda\widetilde{v}.[\![\phi]\!]\left(\rho\{X \mapsto q, \widetilde{x} \mapsto \widetilde{v}\} \uplus [\![\mathcal{P}]\!]\left(\rho\{X \mapsto q\}\right)\right)\right)\right\}
\end{aligned}$$

where $\mathrm{dom}(\rho) \cap \mathrm{dom}(\mathcal{P}) = \emptyset$ and the fixpoint operator $\mathbf{FP}_\alpha^{\widetilde{s}\to\bullet}(\bullet)$ is defined by:

$$\begin{aligned}
\mathbf{FP}_\mu^{\widetilde{s}\to\bullet}(F) &\triangleq \textstyle\bigsqcap_{(\widetilde{s}\to\bullet)}\{\, q \in \mathcal{D}_{\widetilde{s}\to\bullet} \mid F(q) \sqsubseteq_{(\widetilde{s}\to\bullet)} q \,\},\\
\mathbf{FP}_\nu^{\widetilde{s}\to\bullet}(F) &\triangleq \textstyle\bigsqcup_{(\widetilde{s}\to\bullet)}\{\, q \in \mathcal{D}_{\widetilde{s}\to\bullet} \mid q \sqsubseteq_{(\widetilde{s}\to\bullet)} F(q) \,\}
\end{aligned}$$

We note that the domain over which fixpoints are taken is $\mathcal{D}_{\widetilde{s}\to\bullet}$, with the semantic ordering $\sqsubseteq_{\widetilde{s}\to\bullet}$.

Example 3.1. Let us consider μCLP $\mathcal{P}_{\nu\mu} \triangleq (X =_\nu X \wedge Y);\ (Y =_\mu X \vee Y)$ and μCLP $\mathcal{P}_{\mu\nu} \triangleq (Y =_\mu X \vee Y);\ (X =_\nu X \wedge Y)$. Note that the semantics of $\mathcal{P}_{\nu\mu}$ and $\mathcal{P}_{\mu\nu}$ are different as shown below, though the definition of $\mathcal{P}_{\nu\mu}$ and $\mathcal{P}_{\mu\nu}$ only differ in the order of the equations:

$$\begin{aligned}
[\![\mathcal{P}_{\nu\mu}]\!](\emptyset) &= \rho^{\nu\mu} \uplus [\![Y =_\mu X \vee Y]\!]_\epsilon(\rho^{\nu\mu}) = \{X \mapsto \top, Y \mapsto \top\}\\
[\![\mathcal{P}_{\mu\nu}]\!](\emptyset) &= \rho^{\mu\nu} \uplus [\![X =_\nu X \wedge Y]\!]_\epsilon(\rho^{\mu\nu}) = \{X \mapsto \bot, Y \mapsto \bot\}
\end{aligned}$$

where

$$\begin{aligned}
\rho^{\nu\mu} &= [\![X =_\nu X \wedge Y]\!]_{(Y=_\mu X\vee Y)}(\emptyset) = \left\{X \mapsto \mathbf{FP}_\nu^\bullet\left(\lambda q.[\![X \wedge Y]\!](\{X \mapsto q\} \uplus \rho_q^\mu)\right)\right\}\\
&= \left\{X \mapsto \textstyle\bigsqcup_\bullet\{\, q \mid q \sqsubseteq_\bullet [\![X \wedge Y]\!](\{X \mapsto q, Y \mapsto q\}) \,\}\right\} = \{X \mapsto \top\}\\
\rho_q^\mu &= [\![Y =_\mu X \vee Y]\!](\{X \mapsto q\}) = \left\{Y \mapsto \mathbf{FP}_\mu^\bullet\left(\lambda q'.[\![X \vee Y]\!](\{X \mapsto q, Y \mapsto q'\})\right)\right\} = \{Y \mapsto q\}\\
\rho^{\mu\nu} &= [\![Y =_\mu X \vee Y]\!]_{(X=_\nu X\wedge Y)}(\emptyset) = \left\{Y \mapsto \mathbf{FP}_\mu^\bullet\left(\lambda q.[\![X \vee Y]\!](\{Y \mapsto q\} \uplus \rho_q^\nu)\right)\right\}\\
&= \left\{Y \mapsto \textstyle\bigsqcap_\bullet\{\, q \mid [\![X \vee Y]\!](\{X \mapsto q, Y \mapsto q\}) \sqsubseteq_\bullet q \,\}\right\} = \{Y \mapsto \bot\}\\
\rho_q^\nu &= [\![X =_\nu X \wedge Y]\!](\{Y \mapsto q\}) = \left\{X \mapsto \mathbf{FP}_\nu^\bullet\left(\lambda q'.[\![X \wedge Y]\!](\{X \mapsto q', Y \mapsto q\})\right)\right\} = \{X \mapsto q\}
\end{aligned}$$

■

Remark 1. *Note that quantifiers over recursively enumerable (r.e.) domains (e.g., integers) can be eliminated in* μCLP; *We can encode* $\exists x\colon \mathtt{int}.\phi$ *and* $\forall x\colon \mathtt{int}.\phi$ *respectively as* $E(0)$ *and* $A(0)$ *using the following inductive and co-inductive predicates* E *and* A*:*

$$\begin{aligned}
E(x) &=_\mu \phi \vee [-x/x]\phi \vee E(x+1)\\
A(x) &=_\nu \phi \wedge [-x/x]\phi \wedge A(x+1)
\end{aligned}$$

Intuitively, E *and* A *are required to hold for some and for all integer* x*, respectively. This encoding strategy, however, cannot apply to non r.e. domains like real numbers and is not useful in practice even for r.e. domains like rational numbers that have no simple way to enumerate all its elements. This is the reason why our reduction given in Section 5 instead applies Skolemization with functional predicate variables to handle quantifiers.*

3.2 Encoding Termination and Non-Termination Verification for Imperative Programs

We consider a simple imperative language whose syntax of statements is given below.

$$s \ ::= \ \mathtt{skip} \mid x := e \mid x := * \mid s_1;\ s_2 \mid \mathtt{if}\ e\ \mathtt{then}\ s_1\ \mathtt{else}\ s_2 \mid \mathtt{while}\ e\ \mathtt{do}\ s$$

$$\mathit{encTerm}(\texttt{skip}) \triangleq (W, T, W(\widetilde{x}) =_\mu \top, T(\widetilde{x}, \widetilde{x}') =_\mu \textstyle\bigwedge_{x \in \widetilde{x}} x = x')$$

$$\mathit{encTerm}(x \texttt{ := } e) \triangleq (W, T, (W(\widetilde{x}) =_\mu \top);\ (T(\widetilde{x}, \widetilde{x}') =_\mu x' = e \land \textstyle\bigwedge_{y \in \widetilde{x}, y \neq x} y' = y)\})$$

$$\mathit{encTerm}(x \texttt{ := } *) \triangleq (W, T, (W(\widetilde{x}) =_\mu \top);\ (T(\widetilde{x}, \widetilde{x}') =_\mu \textstyle\bigwedge_{y \in \widetilde{x}, y \neq x} y' = y))$$

$$\frac{\mathit{encTerm}(s_1) = (W_1, T_1, \mathcal{P}_1) \quad \mathit{encTerm}(s_2) = (W_2, T_2, \mathcal{P}_2) \quad (\_, T_1^\neg, \mathcal{P}_1^\neg) = \neg(W_1, T_1, \mathcal{P}_1)}{\begin{array}{l} \mathit{encTerm}(s_1;\ s_2) \triangleq (W, T, W(\widetilde{x}) =_\mu W_1(\widetilde{x}) \land \forall \widetilde{x}'. T_1^\neg(\widetilde{x}, \widetilde{x}') \lor W_2(\widetilde{x'}); \\ \qquad\qquad T(\widetilde{x}, \widetilde{x}') =_\mu \exists \widetilde{x}''. T_1(\widetilde{x}, \widetilde{x}'') \land T_2(\widetilde{x}'', \widetilde{x}');\ \mathcal{P}_1;\ \mathcal{P}_1^\neg;\ \mathcal{P}_2) \end{array}}$$

$$\frac{\mathit{encTerm}(s_1) = (W_1, T_1, \mathcal{P}_1) \quad \mathit{encTerm}(s_2) = (W_2, T_2, \mathcal{P}_2)}{\begin{array}{r} \mathit{encTerm}(\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2) \triangleq (W, T, W(\widetilde{x}) =_\mu b \land W_1(\widetilde{x}) \lor \neg b \land W_2(\widetilde{x}); \\ T(\widetilde{x}, \widetilde{x}') =_\mu b \land T_1(\widetilde{x}, \widetilde{x}') \lor \neg b \land T_2(\widetilde{x}, \widetilde{x}');\ \mathcal{P}_1;\ \mathcal{P}_2) \end{array}}$$

$$\frac{\mathit{encTerm}(s_0) = (W_0, T_0, \mathcal{P}_0) \quad (\_, T_0^\neg, \mathcal{P}_0^\neg) = \neg(W_0, T_0, \mathcal{P}_0)}{\begin{array}{l} \mathit{encTerm}(\texttt{while}\ b\ \texttt{do}\ s_0) \triangleq (W, T, (W(\widetilde{x}) =_\mu \neg b \lor b \land \forall \widetilde{x}'. T_0^\neg(\widetilde{x}, \widetilde{x}') \lor W(\widetilde{x}'); \\ \qquad\qquad T(\widetilde{x}, \widetilde{x}') =_\mu \neg b \land \textstyle\bigwedge_{s \in \widetilde{x}} x = x' \lor b \land \exists \widetilde{x}''. T_0(\widetilde{x}, \widetilde{x}'') \land T(\widetilde{x}'', \widetilde{x}');\ \mathcal{P}_0;\ \mathcal{P}_0^\neg) \end{array}}$$

Fig. 1. Modular encoding of termination/non-termination verification problems.

Here e and b respectively range over integer type expressions and boolean expressions. For (integer-typed) variables of the language, we purposefully overload the term variables of μCLP and use x, y, z, etc. to range over them. The statement $x \texttt{ := } *$ denotes an assignment of a non-deterministic integer value to the variable x.

The semantics of the language is standard. A *state* of the program s is a sequence of integers $\widetilde{v}$ such that $|\widetilde{v}| = |\widetilde{x}|$ where $\widetilde{x}$ are the variables that appear in s. Roughly, a state represents the values of the variables. We write $\langle \widetilde{v}, s \rangle \Downarrow \widetilde{v}'$ to mean that there is a terminating execution of s from the initial state $\widetilde{v}$ to the final state $\widetilde{v}'$, and write $\langle \widetilde{v}, s \rangle \Uparrow$ to mean that there is a non-terminating execution of s from the initial state $\widetilde{v}$. We say that a program s *always terminates* (or, simply *terminates*) if not $\langle \widetilde{v}, s \rangle \Uparrow$ for any $\widetilde{v}$, and that s *may non-terminate* (or, simply *non-terminates*) if $\langle \widetilde{v}, s \rangle \Uparrow$ for some $\widetilde{v}$. That is, a program is said to be terminating if every execution of it is terminating, and otherwise it is said to be non-terminating. The *termination verification problem* seeks to decide if the given program is terminating or not. We reduce the problem soundly and completely to a μCLP validity problem. Recall that μCLP is parameterized by a background first-order theory $\mathcal{T}$. For the encoding described in this section, we assume that the background theory is the theory of integers[6].

Let $\widetilde{x}$ be the variables of the given program. We define the encoding procedure $\mathit{encTerm}(s)$ which returns a triple $(W, T, \mathcal{P})$ where W and T are predicate variables defined in the μCLP $\mathcal{P}$ such that $\mathrm{ar}(W) = |\widetilde{x}|$ and $\mathrm{ar}(T) = 2 \cdot |\widetilde{x}|$. As we shall show, W encodes the weakest precondition for the termination of s and T encodes the transition relation of s. The encoding is defined inductively on the syntax of s as shown in Figure 1. Here, we assume that W and T in the rules are fresh predicate variables, and we write $\neg(W, T, \mathcal{P})$ for the De Morgan dual of $(W, T, \mathcal{P})$ defined by: $\neg(W, T, \mathcal{P}) = (W^\neg, T^\neg, \mathcal{P}^\neg)$ where $\mathcal{P}^\neg$ is the De Morgan dual of the μCLP $\mathcal{P}$, and $W^\neg$ and $T^\neg$ are the predicate variables defined in $\mathcal{P}^\neg$ that correspond to W and T, respectively.

The encoding ensures the following property.

[6] More precisely, we assume that the background theory is one that can interpret the integer and boolean expressions of the programming language.

Lemma 3.2. *Let* $(W, T, \mathcal{P}) = encTerm(s)$*. Then,* $[\![\mathcal{P}]\!](\emptyset) \models W(\widetilde{v})$ *iff not* $\langle \widetilde{v}, s\rangle \Uparrow$*, and* $[\![\mathcal{P}]\!](\emptyset) \models T(\widetilde{v}, \widetilde{v}')$ *iff* $\langle \widetilde{v}, s\rangle \Downarrow \widetilde{v}'$*.*

That is, the μCLP $\mathcal{P}$ obtained by the encoding $(W, T, \mathcal{P}) = encTerm(s)$ precisely encodes the weakest precondition for termination of s as the inductive predicate W, and the transition relation of s as the inductive predicate T. Therefore, to verify (resp. refute) the termination of s, it suffices to check the validity of the μCLP validity problem instance $(\forall \widetilde{x}.W(\widetilde{x}), \mathcal{P})$ where $(W, T, \mathcal{P}) = encTerm(s)$. More formally, we state the correctness of the encoding in the theorem below.

Theorem 3.3. *Let* $(W, T, \mathcal{P}) = encTerm(s)$*. Then, s terminates iff* $\models (\forall \widetilde{x}.W(\widetilde{x}), \mathcal{P})$*.*

Note that the encoding contains nested (co-)inductive predicates. Namely, while the predicates W and T appearing in the conclusion of each rule is inductive, the complementation done in the rules for sequences and loops introduce co-inductive predicate definitions. Also, note that the encoding is *modular*. That is, the encoding for a statement s contains the (co-)inductive predicates defined in the encodings for the sub-statements of s as sub-formulas. Our encoding is inspired by similar encodings for temporal property verification of functional programs given in Nanjo et al. [2018]; Unno et al. [2017a].

Example 3.4. Recall the program from Section 2.1. Applying *encTerm* to the inner while loop yields the triple $(\mathcal{J}, P, \mathcal{P}_{\text{in}})$ where $\mathcal{P}_{\text{in}}$ consists of the following recursive equations defining the inductive predicates $\mathcal{J}$ and P (after some simplification, and by modeling `nondet` by a non-deterministic assignment):

$$\mathcal{J}(x_2) =_\mu \neg(x_2 \neq 3) \lor \mathcal{J}(x_2 + 1)$$
$$P(x_2, x_2') =_\mu x_2' = x_2 \lor x_2 \neq 3 \land P(x_2 + 1, x_2')$$

Note that the equation defining $\mathcal{J}$ is the same as the one in $\mathcal{P}_{\text{term}}$ from Section 2.1, and the other equation defining P is the same as the one in $\mathcal{P}_{\text{nterm}}$. The order of the equations does not matter in this case because the equations do not refer to each others' predicates[7]. It can be readily seen that $\mathcal{J}$ and P respectively express the weakest precondition for the termination and the transition relation of the inner while loop. Then, using this encoding result for the inner while loop, the encoding procedure yields for the outer while loop the triple $(I, Q, \mathcal{P}_{\text{out}})$ where $\mathcal{P}_{\text{out}}$ consists of the equations in $\mathcal{P}_{\text{in}}$ and the equations defining I and NP from $\mathcal{P}_{\text{term}}$ and the following equation defining Q:

$$\begin{aligned}Q(x_1, x_2, x_1', x_2') =_\mu \;& \neg(x_1 \geq 0 \land x_2 \geq 0) \land x_2' = x_2 \land x_1' = x_1 \lor \\ & x_1 \geq 0 \land x_2 \geq 0 \land \\ & \quad x_2' = x_2 - 1 \land (x_1' = x_1 \lor \exists x_2''.P(x_2, x_2'') \land x_2' = x_2'' - 1 \land x_1' = x_1 - 1)\end{aligned}$$

The equations in $\mathcal{P}_{\text{out}}$ are ordered appropriately so that an equation come before those that define predicate variables occurring in the equation. Note that the equation defining I contains an occurrence of the co-inductive predicate NP whose definition is obtained by taking the dual of the equation defining P. Then, by modeling the `assume` with a conditional statement, for the whole program, the encoding procedure returns $(K, R, \mathcal{P}_{\text{prog}})$ where $\mathcal{P}_{\text{prog}}$ consists of the equations in $\mathcal{P}_{\text{out}}$ and the following equations (ordered appropriately as remarked above):

$$K(x_1, x_2) =_\mu \neg(x_2 \leq 3) \lor I(x_1, x_2)$$
$$R(x_1, x_2, x_1', x_2') =_\mu \neg(x_2 \leq 3) \land x_1' = x_1 \land x_2' = x_2 \lor x_2 \leq 3 \land Q(x_1, x_2, x_1', x_2')$$

Finally, by simplifying $\mathcal{P}_{\text{prog}}$ by removing predicate definitions on which K does not depend (i.e., the equations defining Q and R) and moving the (non-recursive) equation defining K to the query part, we obtain the μCLP validity problem instance $(\forall x_1, x_2.x_2 > 3 \lor I(x_1, x_2), \mathcal{P}_{\text{term}})$ from Section 2.1.

[7] Also, the order of equations does not matter when the equations are all of the same fixpoint modality (i.e., all μ or all ν).

By construction, $\models (\forall x_1, x_2.K(x_1, x_2), \mathcal{P}_{\text{prog}})$ iff $\models (\forall x_1, x_2.x_2 > 3 \lor I(x_1, x_2), \mathcal{P}_{\text{term}})$, and thus by Theorem 3.3, we have obtained a sound-and-complete encoding of the termination verification problem for the program as a μCLP validity problem. Taking the dual of the problem instance yields the dual μCLP validity problem instance $(\exists x_1, x_2.x_2 \leq 3 \land NI(x_1, x_2)), \mathcal{P}_{\text{nterm}})$ from Section 2.1 that soundly-and-completely encodes the non-termination verification problem for the same program.

3.3 Encoding Linear Temporal Verification for Labeled Transition Systems

A *labeled transition system* (LTS) is a triple $(\mathbb{Z}^n, \{\psi_\ell\}_{\ell \in L}, L)$ where $\mathbb{Z}^n$ is the the set of *states* where each state is a n-tuple of integers, L is the finite set of *labels*, and $\{\psi_\ell\}_{\ell \in L}$ is an L-indexed set of *transition relations* where $\models \psi_\ell(\widetilde{v}, \widetilde{v}')$ for $\widetilde{v}, \widetilde{v}' \in \mathbb{Z}^n$ represents that the LTS may transition from the state $\widetilde{v}$ to the state $\widetilde{v}'$ with the label ℓ. Note that our LTS is an infinite-state system[8].

We review the notion of a *Büchi automaton*. A non-deterministic Büchi automaton A is a tuple $(Q, L, \delta, q_{init}, F)$ where Q is the finite set of states (unrelated to the states of the LTS), $\delta \subseteq Q \times L \times Q$ is the transition relation, $q_{init} \in Q$ is the starting state, and $F \subseteq Q$ is the set of final states. For $q \in Q$ and $\ell \in L$, we write $\delta(q, \ell)$ for the set $\{q' \in Q \mid (q, \ell, q') \in \delta\}$. An infinite word $\ell_0\ell_1 \cdots \in L^\omega$ is accepted by A if and only if there exists an infinite sequence of states $q_0, q_1, \ldots$ such that $q_0 = q_{init}$, $q_{i+1} \in \delta(q_i, \ell)$ for all $i \geq 0$, and some state in F occurs infinitely often.

We consider the temporal property verification problem in which we are given an LTS $M = (\mathbb{Z}^n, \{\psi_\ell\}_{\ell \in L}, L)$, a predicate $\phi_{init}(\widetilde{x})$ on states of M, and a Büchi automaton A such that the label set of A is L. The goal of the verification is to decide if for any (infinite) execution of M from a state satisfying $\phi_{init}(\widetilde{x})$, the infinite sequence of labels of the execution is accepted by A. That is, the goal is to verify whether the given LTS satisfies the linear temporal property specified by the given Büchi automaton. We give an encoding of the verification problem in μCLP. Our encoding is inspired by a similar encoding of temporal verification problems for recursive programs given in [Kobayashi et al. 2019]. Let $\mathcal{P}_{M,A}$ be the μCLP comprising, for each $q \in Q$ and $\alpha \in \{\mu, \nu\}$, the following equations defining mutually-recursive (co-)inductive predicates $\text{LV}_{q,\alpha}(\widetilde{x})$:

$$\text{LV}_{q,\alpha}(\widetilde{x}) =_\alpha \bigwedge_{\ell \in L} \forall \widetilde{y}.\psi_\ell(\widetilde{x}, \widetilde{y}) \Rightarrow \bigvee_{q' \in \delta(q,\ell)} \text{LV}_{q',\alpha(q')}(\widetilde{y})$$

where $\alpha(q) = \nu$ if $q \in F$ and $\alpha(q) = \mu$ otherwise. The equations are ordered so that the co-inductive equations come before the inductive equations. The correctness of the encoding is stated below.

Theorem 3.5. *M satisfies the temporal property given by ϕ_{init} and A iff the μCLP validity problem $(\forall \widetilde{x}.\phi_{init}(\widetilde{x}) \Rightarrow LV_{q_{init},\nu}(\widetilde{x}), \mathcal{P}_{M,A})$ is valid.*

The theorem follows from the fact that $\text{LV}_{q,\_}(\widetilde{x})$ represents the set of states from which the labels along the execution of the LTS is accepted by A when A is run from the state q. Note that the occurrence of a predicate in the body of the recursive definition becomes the $\text{LV}_{\_,\mu}$ variant when no state in F is visited in the corresponding execution step. This ensures that there must be a path in which a state from F is visited infinitely often.

4 PFWCSP

We review pfwCSP [Unno et al. 2021] that generalizes constrained Horn clauses (CHCs) and serves as an intermediary in our modular primal-dual method for automatically deciding μCLP validity problems. We use a meta-variable φ to range over $\mathcal{T}$-formulas without quantifiers and predicate variables. First, a pCSP [Satake et al. 2020] C is a finite set of clauses of the form $\varphi \lor \bigvee_{i=1}^{\ell} X_i(\widetilde{t_i}) \lor \bigvee_{i=\ell+1}^{m} \neg X_i(\widetilde{t_i})$ where $0 \leq \ell \leq m$. We define $ftv(c)$, $ftv(C)$ and $fpv(C)$ in the obvious manner. We regard that the variables in $ftv(C)$ are implicitly universally quantified. A pCSP is CHCs if

[8] For concreteness we explain the encoding for LTSs over integers. However, our encoding method can be easily extended to LTSs over any domain in the background theory of μCLP.

$\ell \le 1$ for all clauses, and co-CHCs if $m \le \ell + 1$ for all clauses. A *predicate substitution* σ is a map from predicate variables X to closed predicates of the form $\lambda x_1, \ldots, x_{\mathrm{ar}(X)}.\varphi$. We write $\sigma(C)$ for the application of σ to C and $\mathrm{dom}(\sigma)$ for the domain of σ. We call σ a *syntactic solution* for C if $fpv(C) \subseteq \mathrm{dom}(\sigma)$ and $\models \bigwedge \sigma(C)$. Similarly, we call a predicate interpretation ρ a *semantic solution* for C if $fpv(C) \subseteq \mathrm{dom}(\rho)$ and $\rho \models \bigwedge C$.

We next extend pCSP to pfwCSP by adding functionality and well-foundedness constraints. A pfwCSP $(C, \mathcal{K})$ consists of

- a finite set C of pCSP clauses, and
- a kinding map $\mathcal{K}$ that maps each $X \in fpv(C)$ to its kind: any one of $\bullet$, $\Downarrow$, or λ which respectively represent *ordinary*, *well-founded*, and *functional* predicate variable.

We write $\rho \models WF(X)$ if the interpretation $\rho(X)$ of the predicate variable X is *well-founded*, that is, $\mathrm{sort}(X) = (\widetilde{s}, \widetilde{s}) \to \bullet$ for some sequence $\widetilde{s}$ of sorts and there is no infinite sequence $\widetilde{v}_1, \widetilde{v}_2, \ldots$ of sequences $\widetilde{v}_i$ of values of the sorts $\widetilde{s}$ such that $(\widetilde{v}_i, \widetilde{v}_{i+1}) \in \rho(X)$ for all $i \ge 1$. We write $\rho \models FN(X)$ if $\mathrm{sort}(X) = (\widetilde{s}, s) \to \bullet$ and $\rho \models \forall \widetilde{x} : \widetilde{s}.(\exists y : s.X(\widetilde{x}, y)) \land \forall y_1, y_2 : s.(X(\widetilde{x}, y_1) \land X(\widetilde{x}, y_2) \Rightarrow y_1 = y_2)$ holds. We call a predicate interpretation ρ a *semantic solution* for $(C, \mathcal{K})$, written $\rho \models (C, \mathcal{K})$, if ρ is a semantic solution of C, $\rho \models WF(X)$ for all X such that $\mathcal{K}(X) = \Downarrow$, and $\rho \models FN(X)$ for all X such that $\mathcal{K}(X) = \lambda$. The notion of syntactic solution can be similarly generalized to pfwCSP.

Definition 4.1 (Satisfiability of pfwCSP*).* The predicate satisfiability problem of a pfwCSP $(C, \mathcal{K})$ is that of deciding whether it has a semantic solution.

It is known that the satisfiability of CHCs and the validity of CLP are inter-reducible (see e.g., Unno et al. [2017b]). Unno et al. [2021] presents a semi-algorithm called PCSat for semi-deciding the satisfiability of pfwCSP. PCSat implements a counterexample-guide inductive synthesis (CEGIS) based semi-algorithm that is sound and (relatively) complete that we will review in the next subsection. In Section 5, we will show a sound and complete reduction from the μCLP validity problem to the pfwCSP satisfiability problem.

4.1 CEGIS-based pfwCSP Solving

We review the CEGIS-based pfwCSP satisfiability checking semi-algorithm PCSat introduced in Unno et al. [2021]. As informally described in Section 2.2, PCSat consists of two phases, the *synthesis phase* and the *validation phase*, that are iteratively executed until convergence (or up to some time limit).

Algorithm 1 shows the pseudo-code of PCSat. PCSat takes as input a pfwCSP instance $(C, \mathcal{K})$. It first initializes the set of counterexamples E to be an empty set (line 2)[9]. Then, it iterates the CEGIS iteration loop (lines 3–17) until convergence (or up to some time limit). In each iteration, we call the sub-routine SynthesizeCandSol to synthesize a *candidate solution* that satisfies the current set of counterexamples (line 4), that is, a substitution ρ such that $\rho \models (E, \mathcal{K})$. If there is no such a substitution, then SynthesizeCandSol returns UNSAT, indicating that $(E, \mathcal{K})$ is actually unsatisfiable. Since this implies that the input pfwCSP instance $(C, \mathcal{K})$ is also unsatisfiable, the CEGIS loop terminates by returning UNSAT (line 6). Otherwise, we obtain the substitution ρ as a candidate solution to the input pfwCSP instance, and we call the sub-routine ValidateCandSol to check if the candidate solution also satisfies the input instance $(C, \mathcal{K})$ (line 9). If so, then the candidate solution is a genuine solution to the input instance and we terminate the CEGIS loop by returning SAT (line 11). Otherwise, we obtain additional counterexamples E' that witness the failure of ρ to be a genuine solution. Formally, E' is a set of ground clauses obtained by instantiating the

[9] E can also be initialize with any set of ground clauses obtained by instantiating the term variables of the input clauses.

Algorithm 1 PCSat

```
function PCSat($(C, \mathcal{K})$)
    $E \leftarrow \emptyset$
    while true do
        $res_{syn} \leftarrow$ SynthesizeCandSol($E$)
        if $res_{syn}$ = UNSAT then
            return UNSAT
        else
            let $\rho = res_{syn}$ in
            $res_{val} \leftarrow$ ValidateCandSol($\rho$, $(C, \mathcal{K})$)
            if $res_{val}$ = SAT then
                return SAT
            else
                let $E' = res_{val}$ in
                $E \leftarrow E \cup E'$
            end if
        end if
    end while
end function
```

term variables of the clauses of C such that $\rho \not\models (E', \mathcal{K})$. The new counterexamples are added to the set of counterexamples E and we repeat the CEGIS iteration.

We note that the SynthesizeCandSol always terminates, and returns UNSAT only when it finds the given set of counterexamples semantically unsatisfiable (since a counterexample is a ground clause, there always exists a candidate solution conforming to a finite and semantically-consistent set of counterexamples). Also, it only returns candidate solutions that satisfy functionality and well-foundedness constraints by construction. The termination of ValidateCandSol depends on the background theory (more precisely, whether the backend SMT solver terminates or not on the background theory), and we assume that it terminates on theories with decidable satisfiability such as LIA and NRA. We defer further details of the candidate solution synthesis and candidate solution validation processes to [Unno et al. 2021]. PCSat is sound and complete, in the following sense.

Theorem 4.2 (Soundness and Completeness of PCSat [Unno et al. 2021]). PCSat($(C, \mathcal{K})$) *returns* SAT *only if* $(C, \mathcal{K})$ *is satisfiable, and returns* UNSAT *only if* $(C, \mathcal{K})$ *is unsatisfiable.*

As remarked before, PCSat is a semi-algorithm and is not guaranteed to always terminate. It is necessarily so for most background theories including the theory of linear integer arithmetic, because the satisfiability problem even for CHCs which is a subclass of pfwCSP as remarked before, is already undecidable for such theories. However, PCSat is *relatively complete* in the sense that if a syntactic solution exists and ValidateCandSol terminates on the background theory then it is guaranteed to eventually converge. We defer the details of relative completeness to Unno et al. [2021].

5 MODULAR PRIMAL-DUAL SOLVING

We now formally present the main contribution of this paper: MuVal, the novel modular primal-dual approach to semi-deciding μCLP validity. Given a μCLP validity checking problem instance $(\phi_p, \mathcal{P}_p)$, MuVal first constructs its De Morgan dual $(\phi_d, \mathcal{P}_d)$ as described in Section 3. Next, both the primal instance $(\phi_p, \mathcal{P}_p)$ and the dual instance $(\phi_d, \mathcal{P}_d)$ are each reduced to corresponding pfwCSP constraint sets $(C_p, \mathcal{K}_p)$ and $(C_d, \mathcal{K}_d)$ by the process described in Section 5.1. Finally, MuVal solves the two constraint sets in parallel. Each constraint solving process runs the PCSat constraint solving semi-algorithm of Unno et al. [2021] except that with the modification to infer *partial solutions* (cf. Section 5.2) so that whenever a partial solution for some predicate variable is inferred by one process, a corresponding upper-bound is synthesized and passed to the other process to reduce their search space. The partial solutions are also used to synthesize lower-bounds of predicate variables for the process that inferred it. When one of the processes returns *satisfiable* (resp. *unsatisfiable*), we conclude by returning *valid* (resp. *invalid*) if it is the primal process that returned, or *invalid* (resp. *valid*) if it is the dual process.

The following subsections describe the details of the sub-processes. Section 5.1 describes the reduction of μCLP to pfwCSP and Section 5.2 describes the synthesis and exchange of sound bounds. Section 5.3 gives a detailed description of the entire MuVal semi-algorithm.

5.1 Reduction of μCLP to pfwCSP

We now define the reduction of the given μCLP validity problem $(\phi, \mathcal{P})$ to a pfwCSP satisfiability problem $(C, \mathcal{K})$. We assume without loss of generality that predicates $X \in \mathrm{dom}(\mathcal{P})$ occur only positively in the query ϕ: we can always transform the given query into this form by replacing each negative occurrence of X in ϕ with $\neg X^{\neg}$ where the predicate $X^{\neg}$ is defined by the dual $\neg\mathcal{P}$.

Our reduction consists of three steps: The first step, $\mathbf{elim}_{\exists}$, Skolemizes positive occurrences of existential quantifiers and negative occurrences of universal quantifiers by introducing fresh *functional* predicate variables. The second step, $\mathbf{elim}_{\mu}$, replaces inductive predicates defined by μ-equations with co-inductive predicates defined by ν-equations with guards (i.e., *well-foundedness* constraints) for co-recursion added to preserve the semantics. The third step, $\mathbf{elim}_{\nu}$, further eliminates co-inductive predicates by replacing them with uninterpreted predicates represented as fresh predicate variables. The reduction is defined as

$$\mathbf{red}(\phi, \mathcal{P}) \triangleq \begin{array}{l} \mathbf{let}\ (\phi_\mu, \mathcal{P}_\mu, \mathcal{K}_\mu) = \mathbf{elim}_{\exists}(\phi, \mathcal{P}, \emptyset)\ \mathbf{in} \\ \mathbf{let}\ (\phi_\nu, \mathcal{P}_\nu, \mathcal{K}_\nu) = \mathbf{elim}_{\mu}(\phi_\mu, \mathcal{P}_\mu, \mathcal{K}_\mu)\ \mathbf{in}\ \mathbf{elim}_{\nu}(\phi_\nu, \mathcal{P}_\nu, \mathcal{K}_\nu) \end{array}$$

Here, the μCLP $(\phi_\mu, \mathcal{P}_\mu)$ with free predicate variables $\mathcal{K}_\mu$ is obtained from $(\phi, \mathcal{P})$ by eliminating existential quantifiers with fresh functional predicate variables as stated above. The definition of $\mathbf{elim}_{\nu}(\phi, \mathcal{P}, \mathcal{K})$ is given as:

$$\begin{array}{l} \mathbf{elim}_{\nu}(\phi, \epsilon, \mathcal{K}) \triangleq (\mathbf{cnf}(\underline{\phi}), \mathcal{K}) \\ \mathbf{elim}_{\nu}(\phi, \mathcal{P};\ (X(\widetilde{x}) =_{\nu} \phi'), \mathcal{K}) \triangleq (C \cup \mathbf{cnf}(\underline{X}(\widetilde{x}) \Rightarrow \underline{\phi'}), \mathcal{K}' \cup \{\underline{X} \mapsto \bullet\}) \\ \hspace{12em} \text{where } (C, \mathcal{K}') = \mathbf{elim}_{\nu}(\phi, \mathcal{P}, \mathcal{K}) \end{array}$$

Here, $\underline{\phi}$ is the formula obtained from ϕ by replacing each occurrence of a predicate $X \in \mathrm{dom}(\mathcal{P})$ with the predicate variable $\underline{X}$ that represents an under-approximation of X. $\mathbf{cnf}(\phi)$ converts ϕ into its prenex and conjunctive normal form $\forall\widetilde{x}. \bigwedge C$ and returns the set C of clauses. The trickiest part of the algorithm, namely, $\mathbf{elim}_{\mu}(\phi, \mathcal{P}, \mathcal{K})$, is defined by:

$$\begin{array}{l} \mathbf{elim}_{\mu}(\phi, \mathcal{P}, \mathcal{K}) \triangleq (\phi, \mathcal{P}, \mathcal{K})\ \text{ where } \mathcal{P} \text{ is } =_{\mu} \text{ free.} \\ \mathbf{elim}_{\mu}(\phi, \mathcal{P};\ (X(\widetilde{x}) =_{\mu} \phi');\ (X_i(\widetilde{x_i}) =_{\nu} \phi_i)_{i=1}^{m}, \mathcal{K}) \triangleq \\ \hspace{6em} \mathbf{elim}_{\mu}(\sigma_0(\phi), \sigma_0(\mathcal{P});\ \mathcal{P}', \mathcal{K} \cup \{X_{\Downarrow} \mapsto \Downarrow\}) \text{ where} \end{array}$$

$$\begin{array}{rcl} \mathcal{P}' & = & (X(\widetilde{x}) =_{\nu} \sigma_X(\phi'));\ (X_i(b_i, \widetilde{x}, \widetilde{x_i}) =_{\nu} \sigma_i(\phi_i))_{i=1}^{m} \\ \sigma_0 & = & \{X_i \mapsto \lambda\widetilde{y}.X_i(\bot, \widetilde{v}, \widetilde{y}) \mid i = 1, \dots, m\} \\ \sigma_X & = & \{X \mapsto \lambda\widetilde{y}.X(\widetilde{y}) \land X_{\Downarrow}(\widetilde{x}, \widetilde{y})\} \cup \{X_i \mapsto \lambda\widetilde{y}.X_i(\top, \widetilde{x}, \widetilde{y}) \mid i = 1, \dots, m\} \\ \sigma_i & = & \{X \mapsto \lambda\widetilde{y}.X(\widetilde{y}) \land (b_i \Rightarrow X_{\Downarrow}(\widetilde{x}, \widetilde{y}))\} \cup \{X_j \mapsto \lambda\widetilde{y}.X_j(b_i, \widetilde{x}, \widetilde{y}) \mid j = 1, \dots, m\} \end{array}$$

As remarked before, $\mathbf{elim}_\mu(\phi, \mathcal{P}, \mathcal{K})$ is inspired by an analogous reduction done in *binary reachability analysis* [Cook et al. 2006; Grebenshchikov et al. 2012; Kuwahara et al. 2014] for termination verification. Next, we describe the main ideas of the reduction. The third argument of $\mathbf{elim}_\mu$ accumulates generated fresh well-founded predicate variables. The base case of $\mathbf{elim}_\mu(\phi, \mathcal{P}, \mathcal{K})$ just returns the converted ν-only μCLP $(\phi, \mathcal{P})$ with free predicate variables $\mathcal{K}$. In the recursive step, for the definition $X(\widetilde{x}) =_\mu \phi'$ of the right-most inductive (i.e., μ) predicate X in the input μCLP, we generate a fresh well-founded predicate variable $X_{\downdownarrows}$ and use it as the guard for each co-recursion in the converted co-inductive definition $X(\widetilde{x}) =_\nu \sigma_X(\phi')$: we use the substitution σ_X to replace each call $X(\widetilde{t})$ in the body ϕ' of X with $X(\widetilde{t}) \wedge X_{\downdownarrows}(\widetilde{x}, \widetilde{t})$ that checks that the formal arguments $\widetilde{x}$ of X and the actual arguments $\widetilde{t}$ of the co-recursion are related by the well-founded relation represented by $X_{\downdownarrows}$.

At the same time, we extend the formal arguments of each co-inductive (i.e., ν) predicate X_i in the right-hand side of the equation for X with arguments $\widetilde{x}$ of the same sort as the formal arguments of X and a Boolean argument b_i, where we assume that the formal arguments $\widetilde{x_i}$ of X_i are α-renamed to avoid a name conflict between $\widetilde{x_i}$ and $\widetilde{x}, b_i$. The extended formal arguments $\widetilde{x}$ of X_i are used to receive the actual arguments previously passed to a call to the inductive predicate X and are related by $X_{\downdownarrows}$, in the converted definition of X_i, with the actual arguments passed to each indirect recursive call to X in X_i. Dummy values are passed as $\widetilde{x}$ when no such previous call to X exists and the extended Boolean formal argument b_i of X_i indicates whether there indeed is such a call to X and its actual arguments are passed as $\widetilde{x}$ to X_i ($b_i = \top$) or the dummy values are passed as $\widetilde{x}$ to X_i ($b_i = \bot$). In fact, we use the substitution σ_0 to replace each call $X_i(\widetilde{t})$ in the query ϕ and the definition of the predicates $\mathcal{P}$ in the left-hand side of the equation for X with $X_i(\bot, \widetilde{v}, \widetilde{t})$ for some sequence $\widetilde{v}$ of dummy values of the same sorts as the formal arguments $\widetilde{x}$ of X. For the definition $X(\widetilde{x}) =_\mu \phi'$, we use the substitution σ_X to replace each call $X_i(\widetilde{t})$ in X with $X_i(\top, \widetilde{x}, \widetilde{t})$.

Finally, for the definition $X_j(\widetilde{x_j}) =_\nu \phi_j$ of each co-inductive predicate X_j in the right-hand side of the equation for X, we use σ_j to replace each call $X_i(\widetilde{t})$ in X_j with $X_i(b_j, \widetilde{x}, \widetilde{t})$ and each call $X(\widetilde{t})$ with $X(\widetilde{t}) \wedge (b_j \Rightarrow X_{\downdownarrows}(\widetilde{x}, \widetilde{t}))$ that checks that if $\widetilde{x}$ are not dummy (i.e., $b_j = \top$), the actual arguments of a previous call to X passed around to X_j as its extended formal arguments $\widetilde{x}$ are related by $X_{\downdownarrows}$ with the actual arguments $\widetilde{t}$ of the indirect recursive call to X. In the resulting pfwCSP, the generated well-founded predicate variables occur only positively.

Example 5.1. Let us consider the μCLP $(\phi, \mathcal{P}_X;\ \mathcal{P}_Y)$ where $\phi \triangleq \forall x. X(x) \wedge Y(x)$, $\mathcal{P}_X \triangleq (X(x) =_\mu Y(x-1))$, and $\mathcal{P}_Y \triangleq (Y(y) =_\mu y \leq 0 \vee X(y-1))$. We obtain $\mathbf{elim}_\exists(\phi, \mathcal{P}_X;\ \mathcal{P}_Y, \emptyset) = (\phi, \mathcal{P}_X;\ \mathcal{P}_Y, \emptyset)$ and

$$\begin{array}{l}
\mathbf{elim}_\mu(\phi, \mathcal{P}_X;\ \mathcal{P}_Y, \emptyset) = \mathbf{elim}_\mu(\phi, \mathcal{P}_X;\ (Y(y) =_\nu y \leq 0 \vee X(y-1)), \emptyset) = \\
\quad (\forall x. X(x) \wedge Y(\bot, 0, x), \mathcal{P}, \{X_{\downdownarrows} \mapsto \downdownarrows\}) \text{ where } \mathcal{P} = \\
X(x) =_\nu Y(\top, x, x-1));\ Y(b, x, y) =_\nu y \leq 0 \vee X(y-1) \wedge (b \Rightarrow X_{\downdownarrows}(x, y-1))
\end{array}$$

Here, in the first step of the transformation, the inductive definition of Y is simply replaced by the co-inductive definition because the body of Y has no recursive call to Y. The indirect recursive call to X in Y is properly handled in the second step by adding the formal arguments b and x to Y. Note also that in the call $Y(\bot, 0, x)$ in the query, 0 is used as a dummy value for the extended formal argument x of Y. We thus get the pfwCSP $\mathbf{red}(\phi, \mathcal{P}) = (C, \mathcal{K})$ where

$$\begin{array}{l}
C \triangleq \left\{ \begin{array}{l} \underline{X}(x), \underline{Y}(\bot, 0, x), \neg \underline{X}(x) \vee \underline{Y}(\top, x, x-1), \neg \underline{Y}(b, x, y) \vee y \leq 0 \vee \underline{X}(y-1), \\ \neg \underline{Y}(b, x, y) \vee y \leq 0 \vee \neg b \vee X_{\downdownarrows}(x, y-1)) \end{array} \right\} \\
\mathcal{K} \triangleq \{\underline{X} \mapsto \bullet, \underline{Y} \mapsto \bullet, X_{\downdownarrows} \mapsto \downdownarrows\}
\end{array}$$

The reduction presented here can be optimized removing unnecessary arguments (cf. Appendix C). The soundness and the completeness of the reduction is stated below (cf. Appendix D for proof).

THEOREM 5.2 (SOUNDNESS AND COMPLETENESS OF THE REDUCTION FROM μCLP TO pfwCSP). *$(\phi, \mathcal{P})$ is valid if and only if* $\mathbf{red}(\phi, \mathcal{P})$ *is satisfiable.*

REMARK 2. *Our reduction generates* co-CHCs*-style clauses that have only one negatively occurring ordinary predicate variable. We have adopted this style because it the more natural direction from a fixpoint logic point of view. Note that one can systematically convert back-and-forth from the* co-CHCs *style to the* CHCs *style (i.e., only one positively occurring ordinary predicate variable in each clause) by replacing each literal of the form $X(\widetilde{t})$ with $\neg X^{\neg}(\widetilde{t})$ and $\neg X(\widetilde{t})$ with $X^{\neg}(\widetilde{t})$ for $X \in fpv(C)$, where $X^{\neg}$ is a fresh predicate variable that represents the negation of the ordinary predicate variable X.*

5.2 Synthesizing Sound Bounds from Partial Solutions

We now describe how the parallel pfwCSP constraint solving processes synthesize sound lower-and-upper-bounds from partial solutions to reduce each other's solution spaces. As described in Sections 2.2 and 4.1, the pfwCSP solver PCSAT generates in each of its CEGIS iteration a candidate solution to the given pfwCSP instance. In an ordinary CEGIS-based approach, such a candidate solution, when it is found not to be a genuine solution, is only used to obtain additional counterexamples and then forgotten. However, in our novel modular primal-dual approach, we also check whether the (non-genuine) candidate solution is a *partial solution* that satisfies some subset of the clauses. As we discussed in Sections 1 and 2.2, and shall describe in more detail next, such partial solutions can be used to generate sound lower- and upper- bounds to reduce the solution spaces of the primal and the dual instances.

We next formalize when a candidate solution is a partial solution. For this, we define some preliminary notions. In what follows, we assume that for any pfwCSP instance $(C, \mathcal{K})$ and an ordinary predicate variable X of $(C, \mathcal{K})$, there is a unique sequence of variables $\widetilde{x}$ such that any clause $X(\widetilde{y}) \Rightarrow \psi \in C$ satisfies $\widetilde{y} = \widetilde{x}$. Note that any pfwCSP instance generated by our reduction method described in Section 5.1 can be converted to such a form because each clause has at most one negatively occurring ordinary predicate variable. Also, for an ordinary predicate variable X, we define the *definition* of X, denoted by $\mathit{Defs}_C(X)$, to be the set of formulas $\{\psi \mid X(\widetilde{x}) \Rightarrow \psi \in C\}$. We omit C and simply write $\mathit{Defs}(X)$ when it is clear from the context. We say that a substitution ρ is *well-kinded* for $(C, \mathcal{K})$ if $\rho \models \mathit{WF}(X)$ for all X such that $\mathcal{K}(X) = \Downarrow$, and $\rho \models \mathit{FN}(X)$ for all X such that $\mathcal{K}(X) = \lambda$. We write $\mathit{fpvord}(\phi)$ for the set of free ordinary predicate variables of ϕ (we assume that $\mathcal{K}$ is clear from the context). We extend the notation sets of clauses in the obvious way: $\mathit{fpvord}(C) = \bigcup_{\phi \in C} \mathit{fpvord}(\phi)$. We are now ready to formally define partial solutions.

Definition 5.3 (Partial Solution). For a candidate solution ρ for a pfwCSP instance $(C, \mathcal{K})$ and an ordinary predicate variable $X \in \mathit{fpvord}(C)$, the property that ρ is a *partial solution* for X, written $\rho \rhd_{(C,\mathcal{K})} X$, is defined co-inductively as the largest relation satisfying the following condition: $\rho \rhd_{(C,\mathcal{K})} X$ iff (1) ρ is well-kinded and (2) there exists a set of ordinary predicate variables $S \subseteq \mathit{fpvord}(\varphi)$ where $\varphi = \bigwedge \mathit{Defs}_C(X)$ such that

(2a) $\models \rho(X)(\widetilde{x}) \Rightarrow \rho_S(\varphi)$ where $\rho_S = \{Y \mapsto \mathit{LB}(Y) \mid Y \in S\} \cup \{Y \mapsto \rho(Y) \mid Y \notin S\}$, and
(2b) for each $Y \in \mathit{fpvord}(\varphi) \setminus S$, $\rho \rhd_{(C,\mathcal{K})} Y$.

Here, $\mathit{LB}(Y)$ is a *lower-bound* of the ordinary predicate variable Y that is initialized to be $\bot$ and soundly updated during the constraint solving process as we shall describe in Section 5.3. As we shall show in Example 5.6, roughly, using a non-empty S soundly cuts dependencies among predicate variable definitions by under-approximating some predicate variables by concrete predicates that denote their lower-bounds.

A partial solution for a predicate variable is generally not an actual solution of the given pfwCSP because it needs not to satisfy clauses besides (under-approximations of) the ones that define that

Algorithm 2 PartialSolCheck

```
 1: function PartialSolCheck($(\rho, (\mathcal{C}, \mathcal{K}))$)
 2:     for each $X \in \mathit{fpvord}(\mathcal{C})$ do
 3:         let $\varphi = \bigwedge \mathit{Defs}_{\mathcal{C}}(X)$ in
 4:         $\mathit{SAT2a}(X) \longleftarrow \{S \mid S \subseteq \mathit{fpvord}(\varphi) \land S$ satisfies condition (2a)$\}$
 5:     end for
 6:     $\mathit{NotPsol} \longleftarrow \emptyset$
 7:     repeat
 8:         $T \longleftarrow \mathit{NotPsol}$
 9:         for each $X \in \mathit{fpvord}(\mathcal{C}) \setminus \mathit{NotPsol}$ do
10:             let $\varphi = \bigwedge \mathit{Defs}_{\mathcal{C}}(X)$ in
11:             if $\forall S \in \mathit{SAT2a}(X).(\mathit{fpvord}(\varphi) \setminus S) \cap \mathit{NotPsol} \neq \emptyset$ then
12:                 $\mathit{NotPsol} \longleftarrow \mathit{NotPsol} \cup \{X\}$
13:             end if
14:         end for
15:     until $T = \mathit{NotPsol}$
16:     return $\mathit{fpvord}(\mathcal{C}) \setminus \mathit{NotPsol}$
17: end function
```

predicate variable. Namely, it need not to satisfy any *goal clause* which is (typically a unique) clause with no negative occurrence of ordinary predicate variables (e.g., $x_2 > 2 \lor \underline{I}(x_1, x_2)$ in $\mathcal{C}_{\text{term}}$ from Section 2.2). However, a partial solution for a predicate variable is guaranteed to *under-approximate* the least (resp. greatest) fixpoint of the corresponding inductive (resp. co-inductive) predicate, provided that *LB* also under-approximates the corresponding predicates in the same sense. We state this property formally as the theorem below, which follows from the proof of Theorem 5.2.

Theorem 5.4. *Suppose that $(\mathcal{C}, \mathcal{K})$ is reduced from a* μCLP *validity problem instance $(\phi, \mathcal{P})$, for each $\underline{Y} \in \mathit{fpvord}(\mathcal{C})$, $\models \mathit{LB}(\underline{Y}) \Rightarrow [\![\mathcal{P}]\!](\emptyset)(Y)(\widetilde{x})$ where Y is the predicate variable in $\mathcal{P}$ corresponding to $\underline{Y}$, and ρ is a partial solution for $\underline{X} \in \mathit{fpvord}(\mathcal{C})$. Then, $\models \rho(\underline{X})(\widetilde{x}) \Rightarrow [\![\mathcal{P}]\!](\emptyset)(X)(\widetilde{x})$ where X is the predicate variable in $(\phi, \mathcal{P})$ corresponding to $\underline{X}$.*

Next, we describe an algorithm, PartialSolCheck, that computes, given a candidate solution ρ, the set of ordinary predicate variables for which ρ is a partial solution. We assume that condition (1), that is, well-kindedness of ρ, is satisfied[10]. The pseudo-code of the algorithm is given in Algorithm 2.

Roughly, PartialSolCheck works by inductively marking ordinary predicate variables for which ρ is *not* a partial solution. More formally, the algorithm takes a partial solution ρ along with a pfwCSP instance $(\mathcal{C}, \mathcal{K})$ as input. It then builds the table *SAT2a* which stores for each ordinary predicate variable X, the set of sets of ordinary predicate variables that appear in the definition of X such that if $\mathit{fpvord}(\bigwedge \mathit{Defs}_{\mathcal{C}}(X)) \setminus S$ contains a predicate variable for which ρ is not a partial solution for every set of predicate variables S in the set, then ρ also cannot be a partial solution for X. More concretely, as shown in line 4, $\mathit{SAT2a}(X)$ is created by checking for each set of ordinary predicate variables $S \subseteq \mathit{fpvord}(\varphi)$ whether S satisfies condition (2a) mentioned above and letting such S be a member of $\mathit{SAT2a}(X)$[11]. Next, the algorithm initializes *NotPsol* to the empty set (line 6). *NotPsol* is used to store the set of ordinary predicate variables for which ρ is currently known *not* to be a partial solution. Then, the algorithm repeats the loop of lines 7–15 until no new predicate variables for which ρ is not a partial solution are discovered. Each iteration of the loop checks for each

[10] This is ensured by construction for candidate solutions synthesized in PCSat (cf. Unno et al. [2021]).

[11] As remarked in Remark 3, *SAT2a* can be optimized to only include *maximal* sets of ordinary predicate variables.

ordinary predicate variable $X \notin \mathit{NotPsol}$, whether it should be put into *NotPsol* based on the table *SAT2a* and the current *NotPsol*. Finally, the algorithm returns the set of ordinary predicate variables that do not belong to *NotPsol* (line 16). It is easy to see that the algorithm works as intended, that is, $\textsc{PartialSolCheck}(\rho, (C, \mathcal{K}))$ returns the set of predicate variables $\{X \in \mathit{fpvord}(C) \mid \rho \rhd_{(C,\mathcal{K})} X\}$.

From Theorem 5.4, it follows that if ρ is a partial solution for a predicate variable $\underline{X}$ in the pfwCSP obtained from the primal (resp. dual) μCLP and X is the (co-)inductive predicate in the μCLP corresponding to $\underline{X}$, then the complement of $\rho(\underline{X})$ is an *upper-bound* of the (co-)inductive predicate $X^{\neg}$ in the dual (resp. primal) μCLP. Therefore, adding the clause $\underline{X^{\neg}}(\widetilde{x}) \Rightarrow \neg\rho(\underline{X})(\widetilde{x})$ to the pfwCSP reduced from the dual (resp. primal) μCLP where $\underline{X^{\neg}}$ is the predicate variable corresponding to $X^{\neg}$ does not rule out any actual solution of the dual (resp. primal) pfwCSP because $\underline{X^{\neg}}$ represents an under-approximation of $X^{\neg}$. However, the addition of such a clause can hasten the convergence of the constraint solving process as it (conservatively) reduces the space of possible solutions. An analogy can be made to how clause-learning DPLL SAT solvers conservatively reduce the solution space by asserting learned clauses.

Example 5.5. Recall C_{term} from Section 2.2. $\mathit{Defs}_{C_{\text{term}}}(\underline{\mathcal{J}}) = \{\neg(x_2 \neq 3) \lor \underline{\mathcal{J}}(x_2+1) \land \mathcal{J}_{\Downarrow}(x_2, x_2+1)\}$. Then, for $S = \emptyset \subseteq \mathit{fpvord}(\bigwedge \mathit{Defs}_{C_{\text{term}}}(\underline{\mathcal{J}})) = \{\underline{\mathcal{J}}\}$, we have that $\models \rho(\underline{\mathcal{J}})(x_2) \Rightarrow \rho_S(\neg(x_2 \neq 3) \lor \underline{\mathcal{J}}(x_2+1) \land \mathcal{J}_{\Downarrow}(x_2, x_2+1))$ for any substitution ρ that satisfies clause (3) (note that $\rho_S = \rho$ when $S = \emptyset$). Therefore, any such substitution that is well-kinded is a partial solution for $\underline{\mathcal{J}}$. For example, $\rho_t = \{\underline{\mathcal{J}} \mapsto \lambda x_2.x_2 = 3, \mathcal{J}_{\Downarrow} \mapsto \lambda x_2, x_2'.\varphi\}$ where $\lambda x_2, x_2'.\varphi$ describes any well-founded relation over integers is a partial solution for $\underline{\mathcal{J}}$. Note that, in this case, it suffices to let $S = \emptyset$. The next example shows a case where a non-empty S is useful.

Example 5.6. Let C consist of the clauses below over ordinary predicate variables X_1, X_2, Y_1, Y_2 and a functional predicate variable F:

$$\begin{array}{llll}
\text{(a)} & X_1(x,y) \Rightarrow X_2(x,y+1) & \text{(d)} & Y_1(x,y) \Rightarrow x > 100 \lor Y_1(x+1,y) \\
\text{(b)} & X_2(x,y) \Rightarrow x = 0 \lor Y_1(x,y) & \text{(e)} & Y_2(x,y) \Rightarrow \neg F(x,y,z) \lor z > 0 \\
\text{(c)} & X_2(x,y) \Rightarrow x \neq 0 \lor Y_2(x,y) & \text{(f)} & Y_2(x,y) \Rightarrow \neg F(x,y,z) \lor X_1(x-z,y)
\end{array}$$

Let ρ be a substitution such that $\rho(Z) = \lambda x.x > 0$ for each $Z \in \{X_1, X_2, Y_1, Y_2\}$. We show that ρ is a partial solution for X_2 and Y_1 with $LB(Y_2) = \bot$ (assuming that $\rho(F)$ describes some total function from pairs of integers to integers). First, ρ is a partial solution for Y_1 because it directly satisfies clause (d) that defines it and only Y_1 itself appears in the definition. Secondly, with $S = \{Y_2\} \subseteq \{Y_1, Y_2\} = \mathit{fpvord}(\bigwedge \mathit{Defs}_C(X_2))$, we have that $\rho_S(\bigwedge \mathit{Defs}_C(X_2)) = (x = 0 \lor \rho(Y_1)(x,y)) \land (x \neq 0 \lor LB(Y_2)(x,y))) = x > 0$. Therefore condition (2a) for X_2 is satisfied with that S. Then, because ρ is a partial solution for Y_1 as described above, condition (2b) is also satisfied with the S for X_2. Thus, ρ is also a partial solution for X_2. Note that ρ cannot be a partial solution for X_2 if S in Definition 5.3 is restricted to be the empty set because ρ does not satisfy clauses (e) and (f) that define Y_2 (regardless of what $\rho(F)$ is), which would be needed if Y_2 in clause (c) was not under-approximated by its concrete lower-bound $LB(Y_2)$ in $\rho_S(\bigwedge \mathit{Defs}_C(X_2))$. Here, using the non-empty S cut the dependency on Y_2 (and X_1) from the definition of X_2.

Remark 3. *To compute the table SAT2a, for each ordinary predicate variable $X \in \mathit{fpvord}(C)$,* $\textsc{PartialSolCheck}$ *checks condition (2a) for each $S \subseteq \mathit{fpvord}(\varphi)$ where $\varphi = \bigwedge \mathit{Defs}_C(X)$. As checking (2a) incurs a call to an SMT solver to decide $\models \rho(X)(\widetilde{x}) \Rightarrow \rho_S(\varphi)$, the computation of $SAT2a(X)$ makes a number of SMT calls exponential in $|\mathit{fpvord}(\varphi)|$ and can be a performance bottleneck, The cost can be alleviated by exploiting the observation that $SAT2a(X)$ only needs to contain "maximal" sets of predicate variables. That is, it suffices to restrict $SAT2a(X)$ to contain $S \subseteq \mathit{fpvord}(\varphi)$ such that S satisfies condition (2a) and there is no $S \subset S' \subseteq \mathit{fpvord}(\varphi)$ with S' satisfying condition (2a). Such*

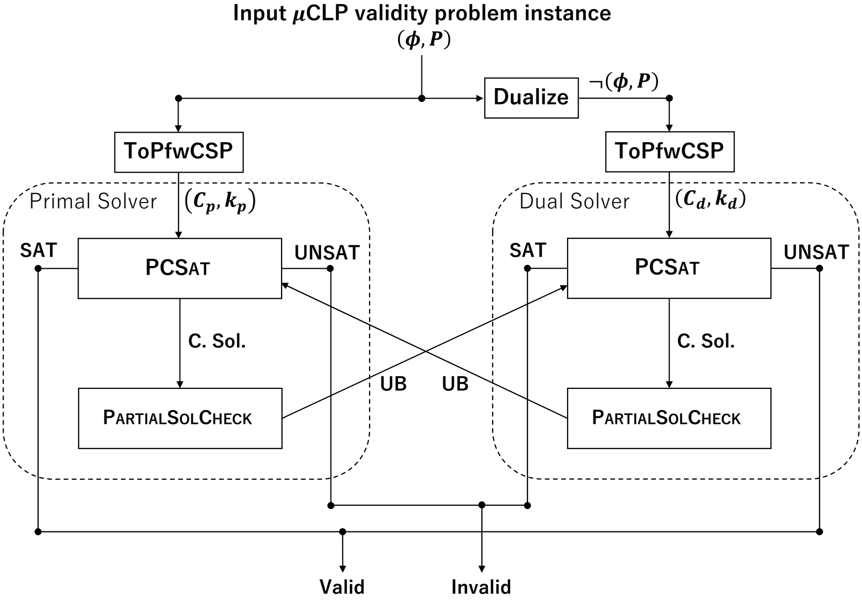
Input μCLP validity problem instance
$(\boldsymbol{\phi}, \boldsymbol{P})$
Dualize
$\neg(\boldsymbol{\phi}, \boldsymbol{P})$
ToPfwCSP
ToPfwCSP
Primal Solver
$(\boldsymbol{C_p}, \boldsymbol{k_p})$
Dual Solver
$(\boldsymbol{C_d}, \boldsymbol{k_d})$
SAT
PCSAT
UNSAT
SAT
PCSAT
UNSAT
C. Sol.
C. Sol.
UB
UB
PARTIALSOLCHECK
PARTIALSOLCHECK
Valid
Invalid

Fig. 2. Overview of MuVal, the modular primal-dual μCLP validity solver

maximal sets can be efficiently computed by a greedy algorithm that starts from $\mathcal{S} = \{fpvord(\varphi)\}$, checks if all sets in $\mathcal{S}$ satisfies condition (2a) and if so lets $SAT2a(X) \longleftarrow \mathcal{S}$ and otherwise replaces the sets in $\mathcal{S}$ that do not satisfy the condition with sets obtained by removing one predicate variable from each such set, and repeat the process until we reach $\mathcal{S}$ whose elements all satisfy condition (2a).

5.3 A Detailed Description of the Modular Primal-Dual Semi-Algorithm

We now formalize MuVal, the modular primal-dual semi-algorithm for semi-deciding μCLP validity. Figure 2 shows the overview of the semi-algorithm. Given an input μCLP validity problem instance $(\phi, \mathcal{P})$, MuVal makes its dual instance $\neg(\phi, \mathcal{P})$ (depicted by the box Dualize in Figure 2), and reduces both the primal (i.e. input) and the dual instances to pfwCSP constraint sets $(C_p, \mathcal{K}_p)$ and $(C_d, \mathcal{K}_d)$ via the method presented in Section 5.1, respectively (depicted by the boxes ToPfwCSP). Then, the primal and the dual constraint sets are sent to the primal solver and the dual solver, respectively. The two solvers are symmetric and run in parallel while exchanging sound upper-bounds synthesized from partial solutions. Each solver uses a modified version of PCSat as a sub-process, obtained by adding the following lines before line 12 in Algorithm 1:

```
i :   $S \longleftarrow \textsc{PartialSolCheck}(\rho, (C, \mathcal{K}))$
ii :  $send(\{X^{\neg}(\widetilde{x}) \Rightarrow \neg\rho(X)(\widetilde{x}) \mid X \in S\})$
iii : **for** each $X \in S$ **do** $LB(X) \longleftarrow LB(X) \vee \rho(X)(\widetilde{x})$ **endfor**
```

Additionally, the modified PCSat adds the line iv : $C \longleftarrow recv() \cup C$ somewhere in the CEGIS loop (lines 3–17) of Algorithm 1. Here, $send(C')$ sends the set of clauses C' to the other solver process, and $recv()$ returns the set of clauses it received. The communication is non-blocking and buffered so that $recv()$ returns the union of the sets of clauses that have been received since the last time it was called. With the modification, PCSat, whenever it finds the given candidate solution ρ to be non-genuine, computes the set of predicate variables S for which ρ is a partial solution by calling the PartialSolCheck sub-routine (line i, and depicted by the arrows labeled C. Sol. in Figure 2). Then, it communicates the learned information to the other process by sending the clauses $\{X^{\neg}(\widetilde{x}) \Rightarrow \neg\rho(X)(\widetilde{x}) \mid X \in S\}$ (line ii, and depicted by the arrows labeled UB). As explained in Section 5.2, the clauses stipulate sound upper-bounds for the predicate variables of the other process. Finally, it updates the lower-bound information LB by disjuncting $LB(X)$ with $\rho(X)(\widetilde{x})$ for each $X \in S$ (line iii). As explained in Section 5.2, LB is used by PartialSolCheck.

MuVal terminates by returning Valid if either the PCSat sub-process of the primal solver returns SAT or that of the dual solver returns UNSAT, and terminates by returning Invalid if either the PCSat sub-process of the primal solver returns UNSAT or that of the dual solver returns SAT. The soundness and the completeness of MuVal immediately follows from the soundness and completeness of the reduction from μCLP validity to pfwCSP satisfiability (Theorem 5.2), the soundness and completeness of PCSat (Theorem 4.2), and Theorem 5.4.

Theorem 5.7 (Soundness and Completness of MuVal). *MuVal returns* Valid *only if the given* μCLP *validity problem instance is valid and returns* Invalid *only if it is invalid.*

Example 5.8. Recall C_{term} and its partial solution ρ_t from Example 5.5 and Section 2.2. As explained there, ρ_t is a partial solution for $\underline{\mathcal{J}}$. Additionally, $\mathcal{J}$ is the predicate in the input μCLP $\mathcal{P}_{\text{term}}$ corresponding to $\underline{\mathcal{J}}$, $N\mathcal{J}$ is the dual predicate of $\mathcal{J}$ in the dual μCLP $\mathcal{P}_{\text{nterm}}$, and $\underline{N\mathcal{J}}$ is the predicate variable in the pfwCSP C_{nterm} reduced from $\mathcal{P}_{\text{nterm}}$ corresponding to $N\mathcal{J}$. Thus, $\neg\rho_t(\underline{\mathcal{J}})(x_2) = \neg(x_2 \neq 3)$ is asserted as a sound upper-bound of $\underline{N\mathcal{J}}(x_2)$ by adding the clause $\underline{NJ}(x_2) \Rightarrow \neg(x_2 = 3)$ to the pfwCSP constraint set of the dual process. Likewise, $\rho_t(\underline{\mathcal{J}}) = x_2 \neq 3$ is a sound lower-bound of $\underline{\mathcal{J}}$, and is used to update LB by $LB(\underline{\mathcal{J}}) \leftarrow x_2 \neq 3 \vee LB(\underline{\mathcal{J}})$.

As remarked in Section 2.2, used with a modular encoding of a verification problem, our primal-dual solving method is able to synthesize information about individual program or specification sub-component of the given program or specification. Namely, in this example, $\mathcal{J}$ and $N\mathcal{J}$ represents the terminating and non-terminating behavior of the inner loop of the program, respectively. Correspondingly, the bounds synthesized for them give information about the states from which the loop diverges and terminates, and therefore can be culled from the solution spaces of the dual process and the primal process, respectively.

Remark 4. *While we use partial solutions synthesized by one side to reduce the solution space of the other side, one may wonder if they can also be used to reduce the solution space of the same side. That is, since a partial solution $\rho(\underline{X})$ represents a lower-bound for the (co-)inductive predicate X corresponding to $\underline{X}$, one may consider restricting the solution space of the same side by asserting the clause $\rho(\underline{X})(\widetilde{x}) \Rightarrow \underline{X}(\widetilde{x})$. The clause asserts that any solution for $\underline{X}$ is lower-bounded by $\rho(\underline{X})$. While this does not rule out the actual semantic fixpoint of X from the solution space, it may rule out sound syntactic solutions and can adversely affect the performance of the semi-algorithm. Namely, a solution for $\underline{X}$ need not be the actual fixpoint of X but only needs to be its under-approximation that is sufficient to satisfy the given set of clauses.*

Remark 5. *As remarked before, PCSat synthesizes a candidate solution by finding a solution that is consistent with the current set of counterexamples, which are ground clauses obtained by instantiating the term variables of the clauses in the given* pfwCSP. *While having more counterexamples make the candidate solution more likely to be an actual solution, it can preclude synthesis of useful partial solutions, especially in the unsatisfiable side because there is no actual solution there. To this end, MuVal implements a heuristic where partial solutions consistent with only some subsets of the counterexamples are synthesized. Specifically, in each iteration, MuVal synthesizes two kinds of substitutions as candidates for partial solutions: those consistent with (1) all counterexamples, and (2) for each ordinary predicate variable X, only the counterexamples obtained by instantiating the*

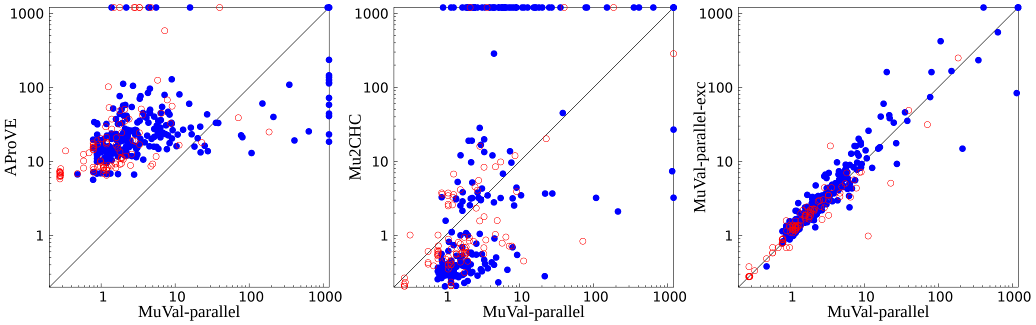

Fig. 3. Comparison with AProVE and Mu2CHC on termination/non-termination benchmarks.

clauses that (transitively) define X and some additional clauses of the form $X(\tilde{t})$ where $X(\tilde{t})$ appears positively in a counterexample.

6 IMPLEMENTATION AND EVALUATION

We present the evaluation with MuVal. MuVal is implemented in Multicore OCaml and supports the combined theory of Booleans, integer and rational arithmetic, and algebraic data types. We report on two experimental evaluation. The supplementary material contains the benchmark set.

(Non-)Termination Verification. We experimented with MuVal on the 335 (non-)termination verification problems from Termination Competition 2021 (C Integer track). For these benchmarks, we encode the problems by first translating the given C program to a labeled transition system (LTS) in T2 format [Brockschmidt et al. 2016] using Clang and llvm2KITTeL [Falke et al. 2011] and systematically encoding the termination verification problem for LTSs as μCLP validity checking problems following the encoding approach described in Section 3.3 (by restricting the temporal property to termination/non-termination). We compared MuVal with the state-of-the-art (non-)termination verification tools AProVE [Giesl et al. 2017], iRankFinder [Ben-Amram and Genaim 2014], UltimateAutomizer [Heizmann et al. 2014], and Mu2CHC [Kobayashi et al. 2019].

Mu2CHC is a tool for semi-deciding the validity problem for Mu-Arithmetic, which is an instance of μCLP with integer arithmetic as the background theory. Mu2CHC is based on an incomplete reduction to CHCs and embarrassingly parallel solving of primal and dual problems (cf. Section 7 for more detail). For an ablation study, we ran MuVal with four configurations: **primal** (resp. **dual**) only solves the primal (resp. dual) μCLP, **parallel** solves **primal** and **dual** problems in an embarrassingly parallel manner without exchanging sound upper-bounds, and **parallel/exc** further enables the exchange of sound upper-bounds. The experiments were conducted on StarExec (CentOS Linux release 7.7.1908, Intel(R) Xeon(R) CPU E5-2609 @ 2.40GHz with 27 GiB RAM) with 1,200 seconds time limit.

The cactus plot shown in Figure 4 plots the number of solved instances (x-axis) against the time taken to solve the instances (y-axis), non-cumulatively, for the above tools. MuVal's **parallel/exc** solved 210 terminating + 108 non-terminating instances while **parallel** solved 211 + 108, **primal** solved 211 + 61, and **dual** solved 8 + 107. AProVE solved 216 terminating + 100 non-terminating instances, iRankFinder solved 208 + 93, UltimateAutomizer solved 200 + 99, and Mu2CHC solved 145 + 105. One problem instance was incorrectly reported non-terminating by iRankFinder. Note that **parallel** (and **parallel/exc**) solved a similar number of instances as AProVE, the winner of the C Integer track of the competition in 2018, 2020, and 2021. The left-most scatter plot in

Figure 3 compares them in detail. The scatter plot is in log-scale. Here the color of dots indicates terminating (blue) and non-terminating (red) instances. We can see an advantage of **parallel** over AProVE with respect to the elapsed time.

Note also that **parallel** significantly outperforms Mu2CHC with respect to the number of solved instances. As for the elapsed time, Mu2CHC often solved easy instances faster than **parallel** as shown in the middle plot of Figure 3. This is partly because Mu2CHC uses SPACER [Komuravelli et al. 2016] as the backend CHCs solver. SPACER, being a highly-tuned mature tool, is often substantially faster than PCSat that MuVal uses as the backend pfwCSP solver (on CHCs subset of pfwCSP). The right-most plot of Figure 3 compares the configurations **parallel** and **parallel/exc**. There, we cannot observe the advantage of **parallel/exc** over **parallel**. This is partly because the encoded μCLP problems here tend to contain a small number of predicates with low modularity which suppressed the exchange of learned upper-bounds.

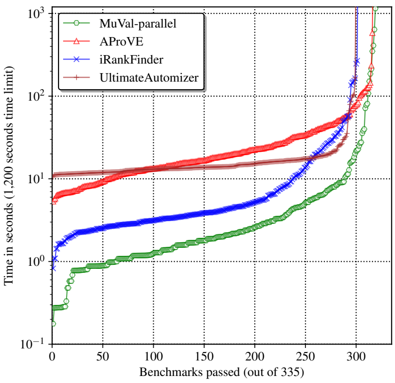

Fig. 4. Cactus Plot for Termination Verification.

Temporal Verification. We also experimented with MuVal on 202 μCLP validity checking problems that encode a diverse collection of temporal verification problems: (1) 41 LTL verification problems from Dietsch et al. [2015], (2) 61 small and 56 industrial CTL verification problems from Cook and Koskinen [2013], (3) 28 Mu-Arithmetic problems from Kobayashi et al. [2019] encoding some properties of integer arithmetic (Problems 1–6), linear-time temporal properties of first-order functional programs (Problems 7–22), branching-time temporal properties (some only expressible in CTL* or modal-μ) of imperative programs (Problems 23–28), and (4) 16 termination verification problems of FuncTion [Urban 2013; Urban and Miné 2014]. For (1) and (2), we automatically generated μCLP from the target C code using the reduction method in Kobayashi et al. [2019] where we used LTL3BA [Babiak et al. 2012] to translate the given LTL formula into a Büchi automaton. For (4), we applied encoding approach described in Section 3.2 (manually, at the time of writing) to translate the problems to μCLP validity problems.

On the entire benchmark set (i.e., (1)-(4)), we compared with Mu2CHC which, to our knowledge, is the only tool besides our MuVal that can handle all those classes of benchmarks. On the class (1) of LTL benchmarks, we compared MuVal with the state-of-the-art LTL verifier UltimateLTLAutomizer [Dietsch et al. 2015]. The experiments were conducted on Amazon Linux 2, Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz CPU and 32 GiB RAM with 600 seconds time limit.

The cactus plot shown in Figure 6 compares the 4 configurations of MuVal with Mu2CHC. MuVal's **parallel/exc** solved 102 VALID + 92 INVALID instances while **parallel** solved 98 + 90, **primal** solved 99 + 62, **dual** solved 41 + 86, and Mu2CHC solved 86 + 87. Note that **parallel** (and **parallel/exc**) significantly outperforms Mu2CHC: Mu2CHC failed to solve 23 instances that were solved by MuVal and require synthesis of predicates with a complex shape and large coefficients (e.g., piecewise-defined and/or lexicographic affine ranking functions). This shows a limitation of Mu2CHC that relies on an incomplete reduction, separately synthesizes ranking functions and inductive invariants, and cannot quickly feedback a failure of invariant synthesis to ranking function synthesis and vice versa.

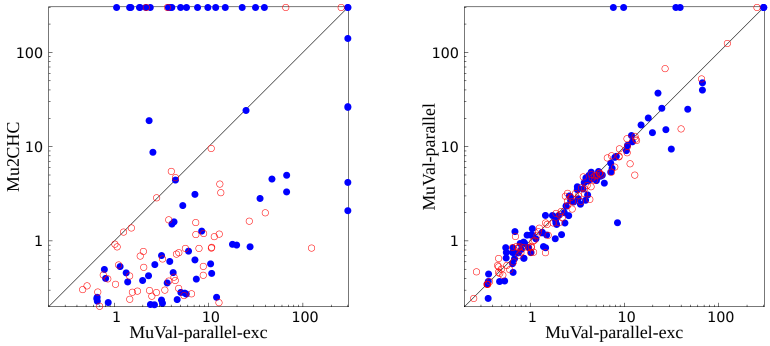

Fig. 5. Comparison with Mu2CHC on temporal verification benchmarks.

Note also that **parallel/exc** further improves **parallel**: **parallel/exc** solved 4 hard instances that were not solved with **parallel** by exchanging sound upper-bounds[12]. The scatter plots comparing Mu2CHC, **parallel** and **parallel/exc** are shown in Figure 5.

Finally, we compare MuVal with UltimateLTLAutomizer on the 41 LTL verification problems. MuVal **parallel/exc** solved 18 VALID + 18 INVALID instances while MuVal **parallel** solved 16 + 16, MuVal **primal** solved 16 + 11, MuVal **dual** solved 4 + 13, and UltimateLTLAutomizer solved 20 + 15. MuVal **parallel/exc** solved 4 instances that were not solved by UltimateLTLAutomizer. Among the instances, MuVal even discovered that the LTL benchmark files contained incorrect classifications, claiming "safe" when they actually are not. On the other hand, MuVal failed to solve 3 instances that were solved by UltimateLTLAutomizer. A reason is that the instances are relatively large and the constraint preprocessor in the backend solver PCSat is not efficient to handle them.

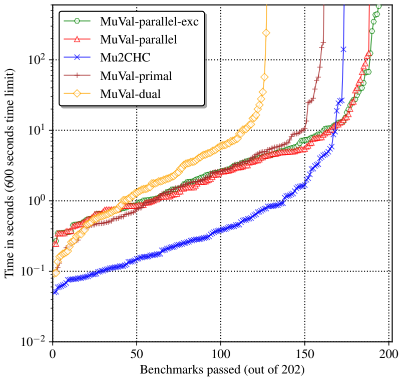

Fig. 6. Cactus Plot for Temporal Verification.

7 RELATED WORK

Our μCLP is a first-order fixpoint logic with background theories. As remarked before, μCLP can be seen as constraint logic programming (CLP) but extended with arbitrarily nested (co-)inductive predicate definitions and quantifiers. CLP has been extensively studied (see, e.g., Jaffar and Maher [1994]). It is well known that the validity problem of CLP and the satisfiability problem of CHCs are inter-reducible. When the background theory is that of integer arithmetic, by the results of Tsukada [2020], CHCs is Π^0_1-complete (in the analytical hierarchy) but μCLP $\notin \Pi^0_1$, which imply that μCLP is strictly more expressive than CLP or CHCs and so there is no sound and complete reduction from μCLP to CHCs. Unno et al. [2021] introduced pfwCSP as an extension of CHCs. The sound and complete reduction from μCLP to pfwCSP in this paper shows that pfwCSP is at least as expressive as μCLP and therefore strictly more expressive than CHCs. In

[12] Valid instances `sas2010_mod_3` (the running example program from Section 2) and `05-toylinarith2.c` are respectively solved by exchanging $7 \rightleftharpoons 16$ and $60 \rightleftharpoons 68$ bounds (*primal* $\rightleftharpoons$ *dual*). Invalid instances `coolant_basis_4_1_unsafe_sftyliveness` and `coolant_basis_4_2_unsafe_sftyliveness` are solved by $0 \rightleftharpoons 12$ and $0 \rightleftharpoons 13$ bounds.

fact, it can be shown that μCLP is in Δ^1_2 while pfwCSP is Σ^1_2-complete. It is an open problem whether there is a class of predicate constraint satisfaction problems that is computationally equivalent to μCLP. μCLP is also related to the *Horn μ-calculus* of Charatonik et al. [1998] which is an extension of ordinary (i.e., not constraint) logic programming by arbitrarily nested (co-)inductive predicate definitions. That is, μCLP can be seen as an extension of the Horn μ-calculus with constraints and quantifiers.

Unno et al. [2021] introduced pfwCSP for the purpose of verifying relational properties of infinite state programs. They also presented the PCSat semi-algorithm for soundly-and-relatively-completely solving pfwCSP that we also use within our MuVal (but with modifications for our novel primal-dual solving method). Their approach directly uses pfwCSP to encode the verification problems and is different from our approach that uses μCLP to encode verification problems and translate them to pfwCSP via our novel reduction. An advantage of our approach is that going through μCLP allows modular encoding of verification problems in which program components are expressed by separate (co-)inductive predicates whose modularity is preserved by our reduction. Another important advantage of our approach is that, thanks to the fact that a μCLP instance can be complemented by the standard De Morgan complement, it enables the novel primal-dual constraint solving method that solves the pfwCSPs reduced from the given μCLP and its dual in parallel while exchanging bounds to reduce each others' solution spaces. By contrast, the approach of Unno et al. [2021] is neither modular nor primal-dual. However, by using pfwCSP directly, Unno et al. [2021] is able to verify difficult relational properties including k-safety properties requiring *semantic schedulers* and hyperliveness properties such as co-termination and generalized non-interference. We leave for future work to investigate whether the ideas of this paper can be adopted to verification of such relational properties.

There has been much work on temporal property verification of infinite state programs. These include approaches for termination [Ben-Amram and Genaim 2014; Cook et al. 2006; Fedyukovich et al. 2018; Giesl et al. 2017; Heizmann et al. 2014; Urban 2013; Urban et al. 2016; Urban and Miné 2014], non-termination [Chen et al. 2014; Cook et al. 2014; Gupta et al. 2008], LTL [Cook and Koskinen 2011; Dietsch et al. 2015], ∀CTL [Cook et al. 2011], CTL [Beyene et al. 2013; Cook and Koskinen 2013; Urban et al. 2018], fair CTL [Cook et al. 2015a; Tellez and Brotherston 2020], and CTL* [Cook et al. 2015b, 2017]. However, few support the full class of modal μ calculus properties. A notable exception is Kobayashi et al. [2019] that presents a method called Mu2CHC. Mu2CHC modularly encodes modal μ model checking of infinite state programs to the problem of deciding the validity of Mu-Arithmetic [Bradfield 1999; Lubarsky 1993] which is a first-order fixpoint logic with the integer arithmetic as the background theory. We also use the same encoding to verify modal μ calculus properties. However, Mu2CHC works by soundly-but-*in*completely reducing the validity checking problem to (ordinary) CHCs which are then solved by an off-the-shelf CHCs solver. By contrast, our MuVal soundly and completely reduces the problem to the extended class of pfwCSP. As a consequence of the incomplete reduction, Mu2CHC cannot conclude anything when the given set of constraints is found unsatisfiable, whereas our complete reduction allows MuVal to conclude invalidity (resp. validity) of the given μCLP instance when the primal (resp. dual) pfwCSP is found unsatisfiable. Additionally, while Mu2CHC also solves the primal and the dual problems in parallel, it simply runs the two processes in an embarrassingly parallel manner without exchanging information. By contrast, our novel modular primal-dual method allows the parallel constraint solving processes to mutually improve each other by synthesizing and exchanging sound bounds to reduce each others' solution spaces.

The duality of verification and refutation has been observed and exploited in many prior works. For example, IC3/PDR [Bradley 2011] is a method for safety property verification in which the search for an inductive invariant (called *frames* in that approach) is guided by the search for states that reach an error state. Recent works further investigate related forms of duality in safety verification such as that casted as a backtracking interpreter for CLP [McMillan 2014] and that between state space exploration and *bounded-width induction* [Padon et al. 2022]. Perhaps more commonly, the duality is also implicit in the classic CEGAR (counterexample-guided abstraction refinement) [Ball and Rajamani 2002; Clarke et al. 2000; Henzinger et al. 2004] and CEGIS based approaches in which the search for a counterexample and that for a proof mutually guide each other. In that sense, the CEGIS-based solver PCSat [Unno et al. 2021] that MuVal uses as a sub-process can also been considered as a primal-dual approach, and therefore, MuVal can be considered to possess two levels of duality: the inner one inside of PCSat and the outer one in which the two primal and dual PCSat processes execute in parallel while exchanging learned upper-bounds. A key difference of our MuVal's (outer) duality from those in prior approaches is that it is formulated as the standard De Morgan duality of a first-order fixpoint logic. Thanks to the generality of the logic and De Morgan duality, our approach extends the benefit of duality to many classes of verification problems, including those beyond safety.

Another difference from many of the prior primal-dual approaches is that ours can exploit *modularity* by modularly encoding verification problems in the first-order fixpoint logic. That is, as we have shown, the logic allows can encode verification problems in which individual program and property components are expressed as different (but possibly nested) (co-)inductive predicates, all of which will be subject to the under- and over-approximation of our primal-dual method. In this respect, the primal-dual approaches of Godefroid et al. [2010] and Le et al. [2015] are also modular, however, are specialized to (non-)safety verification and (non-)termination verification, respectively[13]. By contrast, our MuVal supports a wide range of temporal properties which is due in no small part to the generality of the first-order fixpoint logic μCLP that underlies our approach.

8 CONCLUSION

We have presented a novel modular primal-dual approach to automatically checking validity of a formula in a first-order logic with background theories and arbitrarily nested inductive and co-inductive predicate definitions. Our approach utilizes a novel reduction to the satisfiability of a recently proposed class pfwCSP of predicate constraint satisfaction problems, and further extends the existing constraint solving method by solving both the primal and the dual pfwCSP satisfaction problems that are obtained from the given and its dual validity problems. Our method solves the two problems in parallel while exchanging sound bounds synthesized from partial solutions to reduce each others' solution spaces. We have implemented the approach in the tool MuVal and obtained promising experimental results on diverse benchmark sets of temporal verification problems.

ACKNOWLEDGMENTS

We thank anonymous reviewers for useful comments. This work was supported by JSPS KAKENHI Grant Numbers JP20H04162, JP20K20625, JP22H03564, JP20H05703, and JP22H03570. Koskinen was partially supported by Office of Naval Research under Grant Nos. N00014-17-1-2787 and N00014-22-1-2643 and by NSF under Grant No. CCF-1618059.

REFERENCES

Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. 2012. LTL to Büchi Automata Translation: Fast and More Deterministic. In *TACAS '12*. Springer, 95–109.

Thomas Ball and Sriram K. Rajamani. 2002. The SLAM project: debugging system software via static analysis. In *POPL '02* (Portland, Oregon). ACM, 1–3.

[13] Also, unlike our work, Godefroid et al. [2010] makes no use of modern liveness verification techniques like (non-finitary) well-founded relations.

Amir M. Ben-Amram and Samir Genaim. 2014. Ranking Functions for Linear-Constraint Loops. *J. ACM* 61, 4, Article 26 (July 2014), 55 pages.

Tewodros A. Beyene, Corneliu Popeea, and Andrey Rybalchenko. 2013. Solving Existentially Quantified Horn Clauses. In *CAV '13 (LNCS, Vol. 8044)*. Springer, 869–882.

Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. 2015. Horn Clause Solvers for Program Verification. In *Fields of Logic and Computation II: Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday (LNCS, Vol. 9300)*. Springer, 24–51.

Julian C. Bradfield. 1999. Fixpoint Alternation and the Game Quantifier. In *CSL '99 (LNCS, Vol. 1683)*. Springer, 350–361.

Aaron R. Bradley. 2011. SAT-based Model Checking Without Unrolling. In *VMCAI '11* (Austin, TX, USA) *(LNCS, Vol. 6538)*. Springer, 70–87.

Marc Brockschmidt, Byron Cook, Samin Ishtiaq, Heidy Khlaaf, and Nir Piterman. 2016. T2: Temporal Property Verification. In *TACAS '16*. Springer, 387–393.

Adrien Champion, Tomoya Chiba, Naoki Kobayashi, and Ryosuke Sato. 2018. ICE-Based Refinement Type Discovery for Higher-Order Functional Programs. In *TACAS '18 (LNCS, Vol. 10805)*. Springer, 365–384.

Witold Charatonik, David A. McAllester, Damian Niwinski, Andreas Podelski, and Igor Walukiewicz. 1998. The Horn Mu-calculus. In *LICS '98*. IEEE Computer Society, 58–69.

Hong Yi Chen, Byron Cook, Carsten Fuhs, Kaustubh Nimkar, and Peter W. O'Hearn. 2014. Proving Nontermination via Safety. In *TACAS '14 (LNCS, Vol. 8413)*. Springer, 156–171.

Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-Guided Abstraction Refinement. In *CAV '00 (LNCS, Vol. 1855)*. Springer, 154–169.

Byron Cook, Carsten Fuhs, Kaustubh Nimkar, and Peter W. O'Hearn. 2014. Disproving termination with overapproximation. In *FMCAD '14*. IEEE, 67–74.

Byron Cook, Heidy Khlaaf, and Nir Piterman. 2015a. Fairness for Infinite-State Systems. In *TACAS '15*. Springer, 384–398.

Byron Cook, Heidy Khlaaf, and Nir Piterman. 2015b. On Automation of CTL* Verification for Infinite-State Systems. In *CAV '15*. Springer, 13–29.

Byron Cook, Heidy Khlaaf, and Nir Piterman. 2017. Verifying Increasingly Expressive Temporal Logics for Infinite-State Systems. *J. ACM* 64, 2, Article 15 (April 2017), 39 pages.

Byron Cook and Eric Koskinen. 2011. Making Prophecies with Decision Predicates. In *POPL '11* (Austin, Texas, USA). ACM, 399–410.

Byron Cook and Eric Koskinen. 2013. Reasoning About Nondeterminism in Programs. In *PLDI '13* (Seattle, Washington, USA). ACM, 219–230.

Byron Cook, Eric Koskinen, and Moshe Vardi. 2011. Temporal Property Verification As a Program Analysis Task. In *CAV '11* (Snowbird, UT). Springer, 333–348.

Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2006. Termination proofs for systems code. In *PLDI '06*. ACM, 415–426.

Giorgio Delzanno and Andreas Podelski. 2001. Constraint-based Deductive Model Checking. *International Journal on Software Tools for Technology Transfer* 3, 3 (2001), 250–270.

Daniel Dietsch, Matthias Heizmann, Vincent Langenfeld, and Andreas Podelski. 2015. Fairness Modulo Theory: A New Approach to LTL Software Model Checking. In *CAV '15*. Springer, 49–66.

Stephan Falke, Deepak Kapur, and Carsten Sinz. 2011. Termination Analysis of C Programs Using Compiler Intermediate Languages. In *RTA '11*, Vol. 10. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 41–50.

Grigory Fedyukovich, Yueling Zhang, and Aarti Gupta. 2018. Syntax-Guided Termination Analysis. In *CAV '18 (LNCS, Vol. 10981)*. Springer, 124–143.

Fabio Fioravanti, Alberto Pettorossi, Maurizio Proietti, and Valerio Senni. 2013. Generalization strategies for the verification of infinite state systems. *Theory and Practice of Logic Programming* 13, 2 (2013), 175–199.

Laurent Fribourg. 1999. Constraint Logic Programming Applied to Model Checking. In *LOPSTR '99*. Springer, 30–41.

Juergen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Pluecker, Peter Schneider-Kamp, Thomas Stroeder, Stephanie Swiderski, and Rene Thiemann. 2017. Analyzing Program Termination and Complexity Automatically with AProVE. *Journal of Automated Reasoning* 58 (2017), 3–31.

Patrice Godefroid, Aditya V. Nori, Sriram K. Rajamani, and SaiDeep Tetali. 2010. Compositional may-must program analysis: unleashing the power of alternation. In *POPL '10*, Manuel V. Hermenegildo and Jens Palsberg (Eds.). ACM, 43–56.

Erich Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, Moshe Y. Vardi, Y. Venema, and Scott Weinstein. 2005. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*. Springer.

Sergey Grebenshchikov, Nuno P. Lopes, Corneliu Popeea, and Andrey Rybalchenko. 2012. Synthesizing Software Verifiers from Proof Rules. In *PLDI '12* (Beijing, China). ACM, 405–416.

Ashutosh Gupta, Thomas A. Henzinger, Rupak Majumdar, Andrey Rybalchenko, and Ru-Gang Xu. 2008. Proving non-termination. In *POPL '08* (San Francisco, California, USA). ACM, 147–158.

Arie Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli, and Jorge A. Navas. 2015. The SeaHorn Verification Framework. In *CAV '15*. Springer, 343–361.

Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2014. Termination Analysis by Learning Terminating Programs. In *CAV '14*. Springer, 797–813.

Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. 2004. Abstractions from proofs. In *POPL '04* (Venice, Italy). ACM, 232–244.

Hossein Hojjat and Philipp Rümmer. 2018. The Eldarica Horn Solver. In *FMCAD '18*. IEEE, 1–7.

Joxan Jaffar and Michael J. Maher. 1994. Constraint logic programming: a survey. *The Journal of Logic Programming* 19 (1994), 503 – 581.

Ranjit Jhala and Kenneth L. McMillan. 2006. A Practical and Complete Approach to Predicate Refinement. In *TACAS '06 (LNCS, Vol. 3920)*. Springer, 459–473.

Temesghen Kahsai, Philipp Rümmer, Huascar Sanchez, and Martin Schäf. 2016. JayHorn: A Framework for Verifying Java programs. In *CAV '16*, Vol. 9779. Springer, 352–358.

Naoki Kobayashi, Takeshi Nishikawa, Atsushi Igarashi, and Hiroshi Unno. 2019. Temporal Verification of Programs via First-Order Fixpoint Logic. In *SAS '19*. Springer, 413–436.

Naoki Kobayashi, Takeshi Tsukada, and Keiichi Watanabe. 2018. Higher-Order Program Verification via HFL Model Checking. In *ESOP '18*. Springer, 711–738.

Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. 2014. SMT-Based Model Checking for Recursive Programs. In *CAV '14 (LNCS, Vol. 8559)*. Springer, 17–34.

Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. 2016. SMT-Based Model Checking for Recursive Programs. *Formal Methods in System Design* 48, 3 (June 2016), 175–205.

Stephan Kreutzer. 2004. Expressive equivalence of least and inflationary fixed-point logic. *Annals of Pure and Applied Logic* 130, 1 (2004), 61–78.

Takuya Kuwahara, Tachio Terauchi, Hiroshi Unno, and Naoki Kobayashi. 2014. Automatic Termination Verification for Higher-Order Functional Programs. In *ESOP '14 (LNCS, Vol. 8410)*. Springer, 392–411.

Ton Chanh Le, Shengchao Qin, and Wei-Ngan Chin. 2015. Termination and Non-termination Specification Inference. In *PLDI '15* (Portland, OR, USA). ACM, 489–498.

Leonid Libkin. 2004. *Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An Eatcs Series)*. Springer.

Robert S. Lubarsky. 1993. μ-Definable Sets of Integers. *Journal of Symbolic Logic* 58, 1 (1993), 291–313.

Kenneth L. McMillan. 2014. Lazy Annotation Revisited. In *CAV '14 (Lecture Notes in Computer Science, Vol. 8559)*, Armin Biere and Roderick Bloem (Eds.). Springer, 243–259.

Yoji Nanjo, Hiroshi Unno, Eric Koskinen, and Tachio Terauchi. 2018. A Fixpoint Logic and Dependent Effects for Temporal Property Verification. In *LICS '18*. ACM, 759–768.

Ulf Nilsson and Johan Lübcke. 2000. Constraint Logic Programming for Local and Symbolic Model-Checking. In *CL '00*. Springer, 384–398.

Oded Padon, James R. Wilcox, Jason R. Koenig, Kenneth L. McMillan, and Alex Aiken. 2022. Induction duality: primal-dual search for invariants. 6, POPL (2022), 1–29.

Yuki Satake, Hiroshi Unno, and Hinata Yanagi. 2020. Probabilistic Inference for Predicate Constraint Satisfaction. *AAAI '20* 34, 02 (Apr. 2020), 1644–1651.

Gadi Tellez and James Brotherston. 2020. Automatically Verifying Temporal Properties of Pointer Programs with Cyclic Proof. *Journal of Automated Reasoning* 64, 3 (2020), 555–578.

Tachio Terauchi and Hiroshi Unno. 2015. Relaxed Stratification: A New Approach to Practical Complete Predicate Refinement. In *ESOP '15 (LNCS, Vol. 9032)*. Springer, 610–633.

Takeshi Tsukada. 2020. On Computability of Logical Approaches to Branching-Time Property Verification of Programs. In *LICS '20* (Saarbrücken, Germany). ACM, 886–899.

Hiroshi Unno and Naoki Kobayashi. 2009. Dependent Type Inference with Interpolants. In *PPDP '09*. ACM, 277–288.

Hiroshi Unno, Yuki Satake, and Tachio Terauchi. 2017a. Relatively Complete Refinement Type System for Verification of Higher-order Non-deterministic Programs. *Proceedings of the ACM on Programming Languages* 2, POPL, Article 12 (Dec. 2017), 29 pages.

Hiroshi Unno, Yuki Satake, Tachio Terauchi, and Eric Koskinen. 2020. Program Verification via Predicate Constraint Satisfiability Modulo Theories. *CoRR* abs/2007.03656 (2020). arXiv:2007.03656 https://arxiv.org/abs/2007.03656

Hiroshi Unno, Tachio Terauchi, and Eric Koskinen. 2021. Constraint-Based Relational Verification. In *CAV '21*. Springer, 742–766.

Hiroshi Unno, Sho Torii, and Hiroki Sakamoto. 2017b. Automating Induction for Solving Horn Clauses. In *CAV '17*. Springer, 571–591.

Caterina Urban. 2013. The Abstract Domain of Segmented Ranking Functions. In *SAS '13 (LNCS, Vol. 7935)*. Springer, 43–62.

Caterina Urban, Arie Gurfinkel, and Temesghen Kahsai. 2016. Synthesizing Ranking Functions from Bits and Pieces. In *TACAS '16*. Springer, 54–70.

Caterina Urban and Antoine Miné. 2014. An Abstract Domain to Infer Ordinal-Valued Ranking Functions. In *ESOP '14*. Springer, 412–431.

Caterina Urban, Samuel Ueltschi, and Peter Müller. 2018. Abstract Interpretation of CTL Properties. In *SAS '18 (LNCS, Vol. 11002)*. Springer, 402–422.

A SOLUTIONS TO THE WELL-FOUNDED PREDICATE VARIABLES IN THE EXAMPLE OF SECTION 2.2

The solution for $I_{\Downarrow}(x_1, x_2, x_1', x_2')$ is as follows:

```
WF_I_(x1: int, x2: int, x1': int, x2': int) :=
(f(x1,x2) >= 0 /\ f(x1,x2) > f(x1',x2') \/
 g(x1,x2) >= 0 /\ g(x1,x2) > g(x1',x2') /\
 (f(x1,x2) < 0 /\ f(x1',x2') < 0 \/ f(x1,x2) >= f(x1',x2'))) /\
(D(x1',x2') => (f(x1,x2) >= 0 /\ f(x1,x2) > f(x1',x2') \/
                g(x1,x2) >= 0 /\ g(x1,x2) > h(x1',x2') /\
                (f(x1,x2) < 0 /\ f(x1',x2') < 0 \/ f(x1,x2) >= f(x1',x2')))) \/
D(x1,x2) /\
(f(x1,x2) >= 0 /\ f(x1,x2) > f(x1',x2') \/
 h(x1,x2) >= 0 /\ h(x1,x2) > g(x1',x2') /\
 (f(x1,x2) < 0 /\ f(x1',x2') < 0 \/ f(x1,x2) >= f(x1',x2'))) /\
(D(x1',x2') => (f(x1,x2) >= 0 /\ f(x1,x2) > f(x1',x2') \/
                h(x1,x2) >= 0 /\ h(x1,x2) > h(x1',x2') /\ (f(x1,x2) < 0 /\
                f(x1',x2') < 0 \/ f(x1,x2) >= f(x1',x2'))))

where
f(x1,x2) = 2 * x1 + x2
g(x1,x2) = x1
h(x1,x2) = 2 * x1
D(x1,x2) = x1 >= 0 /\ 1 >= x1 /\ x2 < 0
```

The solution for $J_{\Downarrow}(x_2, x_2')$ is as follows:

```
WF_J_(x2: int, x2': int) :=
f(x2) >= 0 /\ f(x2) > f(x2') /\
(f(x2) >= 0 /\ f(x2) > g(x2') \/ g(x2) > h(x2') /\ f(x2) >= g(x2')) \/
(g(x2) > f(x2') \/ h(x2) >= 0 /\ h(x2) > g(x2') /\ g(x2) >= f(x2')) /\
 h(x2) >= 0 /\ h(x2) > h(x2')

where
f(x) = 5 - x
g(x) = 0
h(x) = -x
```

B FRAGMENTS OF μCLP REDUCIBLE TO CHCS

This section discusses two fragments of μCLP whose validity is reducible to the satisfiability of CHCs.

We first note that the validity of μCLP

$$(\phi_0, ((X_1(\widetilde{x_1}) =_{\alpha_1} \phi_1);\ \cdots;\ (X_m(\widetilde{x_m}) =_{\alpha_m} \phi_m)))$$

has an equivalent CLP validity if all the following conditions are met: (1) $\alpha_i = \nu$ for all $i = 1, \dots, m$, (2) $X_1, \dots, X_m$ occur only positively in ϕ_0, and (3) universal (resp. existential) quantifiers occur only positively (resp. negatively) in ϕ_i's. We henceforth call this fragment of μCLP, *validity-reducible*. Similarly, the *invalidity* of μCLP has an equivalent CLP validity if: (1) $\alpha_i = \mu$ for all $i = 1, \dots, m$, (2) $X_1, \dots, X_m$ occur only positively in ϕ_0, (3a) ϕ_i's only free variables are $\widetilde{x_i}$, and (3b) universal (resp.

existential) quantifiers occur only negatively (resp. positively) in ϕ_i's. We call this μCLP fragment, *invalidity-reducible.*

We can then show that our reduction $\mathbf{red}(\phi, \mathcal{P})$ satisfies the following property for the fragments.

Proposition 1. *Let $(\phi, \mathcal{P})$ be a μCLP. If $(\phi, \mathcal{P})$ is validity-reducible, $\mathbf{red}(\phi, \mathcal{P})$ always generates* co-CHCs. *Similarly, if $(\phi, \mathcal{P})$ is invalidity-reducible, $\mathbf{red}(\phi, \neg(\mathcal{P}))$ always generates* co-CHCs.

Note also that the satisfiability of the co-CHCs C can be further reduced to that of the CHCs $C^{\neg}$, obtained from C by replacing each literal of the form $X(\widetilde{t})$ with $\neg X^{\neg}(\widetilde{t})$ and $\neg X(\widetilde{t})$ with $X^{\neg}(\widetilde{t})$ for $X \in \mathit{fpv}(C)$, where $X^{\neg}$ is a fresh predicate variable that represents the negation of X. Thus we can use off-the-shelf CHCs-solvers to discharge the validity-reducible and invalidity-reducible fragments of μCLP.

C OPTIMIZED $\mathbf{red}(\phi, \mathcal{P})$

Our μCLP validity checker MuVal optimizes $\mathbf{red}(\phi, \mathcal{P})$ so that unnecessary arguments are suppressed. For example, from the μCLP $(\phi, \mathcal{P})$ where $\phi \triangleq \forall x.X(x)$ and $\mathcal{P} \triangleq (X(x) =_{\mu} Y(x-1)); \; (Y(y) =_{\mu} Z(y-1)); \; (Z(z) =_{\mu} z \leq 0 \lor X(z-1))$, $\mathbf{red}(\phi, \mathcal{P})$ generates the pfwCSP $(C, \mathcal{K})$ where

$$C \triangleq \left\{ \begin{array}{l} \underline{X}(x), \underline{X}(x) \Rightarrow \underline{Y}(\top, x, x-1), \\ \underline{Y}(b_1, x, y) \Rightarrow \underline{Z}(b_1, x, \top, y, y-1), \\ \underline{Z}(b_1, x, b_2, y, z) \Rightarrow z \leq 0 \lor \underline{X}(z-1), \\ \underline{Z}(b_1, x, b_2, y, z) \land b_1 \Rightarrow z \leq 0 \lor X_{\Downarrow}(x, z-1)) \end{array} \right\}$$

$$\mathcal{K} \triangleq \left\{ \underline{X} \mapsto \bullet, \underline{Y} \mapsto \bullet, \underline{Z} \mapsto \bullet, X_{\Downarrow} \mapsto \Downarrow \right\}$$

By contrast, MuVal gets a simpler but equi-satisfiable pfwCSP $(C', \mathcal{K})$ where

$$C' \triangleq \left\{ \begin{array}{l} \underline{X}(x), \underline{X}(x) \Rightarrow \underline{Y}(x, x-1), \underline{Y}(x, y) \Rightarrow \underline{Z}(x, y-1), \\ \underline{Z}(x, z) \Rightarrow z \leq 0 \lor \underline{X}(z-1), \underline{Z}(x, z) \Rightarrow z \leq 0 \lor X_{\Downarrow}(x, z-1)) \end{array} \right\}$$

Note that the query ϕ calls X, X calls Y, Y calls Z, and Z recursively calls X. Thus, if we start from the query, Z is always passed the actual argument of the previous call to X and, therefore, b_1 is always $\top$ and so unneeded. Moreover, b_2 and y for $\underline{Z}$ are unneeded because Z does not call Y without going through X that is defined prior to Y.

D PROOF OF THEOREM 5.2

Theorem 5.2 follows from the soundness and the completeness lemmas for each steps of $\mathbf{red}(\phi, \mathcal{P})$: The following lemma for $\mathbf{elim}_{\exists}(\phi, \mathcal{P}, \emptyset)$, the first-step follows immediately from the well-known soundness and completeness of Skolemization for first-order logic.

Lemma D.1. *Suppose that $\mathbf{elim}_{\exists}(\phi, \mathcal{P}, \emptyset) = (\phi_{\mu}, \mathcal{P}_{\mu}, \mathcal{K}_{\mu})$. Then, $\mathcal{K}_{\mu}(X) = \lambda$ holds for all $X \in$* dom$(\mathcal{K}_{\mu})$. *We have $\models (\phi, \mathcal{P})$ if and only if there is an interpretation ρ for* dom$(\mathcal{K}_{\mu})$ *such that $\rho \models FN(X)$ for all $X \in$* dom(ρ) *and $\rho \models \mathbf{elim}_{\exists}(\phi_{\mu}, \mathcal{P}_{\mu})$.*

The following lemma for $\mathbf{elim}_{\nu}(\phi, \mathcal{P})$, the third-step follows from the maximality of the greatest fixpoints (i.e., co-induction principle) (see e.g., Corollary 1 in [Unno et al. 2017b] for a formal related discussion of least fixpoints that occur in negative positions).

Lemma D.2. *Suppose that $\mathbf{elim}_{\nu}(\phi_{\nu}, \mathcal{P}_{\nu}, \mathcal{K}_{\nu}) = (C, \mathcal{K})$. For any interpretation ρ for* dom$(\mathcal{K}_{\nu})$, *the* pfwCSP $(C, \mathcal{K})$ *has a semantic solution that extends ρ if and only if $\rho \models (\phi, \mathcal{P})$.*

The following lemma follows from the proof of Lemma D.2 and implies Theorem 5.4.

Lemma D.3. *Suppose that* $\mathbf{elim}_\nu(\phi_\nu, \mathcal{P}_\nu, \mathcal{K}_\nu) = (C, \mathcal{K})$. *For any interpretation* ρ *for* $\mathrm{dom}(\mathcal{K}_\nu)$, *if the* pfwCSP $(C, \mathcal{K})$ *has a semantic solution* ρ' *that extends* ρ, *then for any* $X \in \mathrm{dom}(\mathcal{P})$, $\rho' \uplus [[\mathcal{P}]](\rho) \models \underline{X} \Rightarrow X$.

We finally show the soundness and the completeness of $\mathbf{elim}_\mu(\phi, \mathcal{P})$, the second-step.

Lemma D.4. *Suppose that* $\mathbf{elim}_\mu(\phi_\mu, \mathcal{P}_\mu, \mathcal{K}_\mu) = (\phi_\nu, \mathcal{P}_\nu, \mathcal{K}_\nu)$. *Then,* $\mathcal{K}_\nu(X) = \Downarrow$ *holds for all* $X \in \mathrm{dom}(\mathcal{K}_\nu) \setminus \mathrm{dom}(\mathcal{K}_\mu)$. *For any interpretation* ρ *for* $\mathrm{dom}(\mathcal{K}_\mu)$, $\rho \models (\phi_\mu, \mathcal{P}_\mu)$ *if and only if there is an interpretation* ρ' *for* $\mathrm{dom}(\mathcal{K}_\nu) \setminus \mathrm{dom}(\mathcal{K}_\mu)$ *such that* $\rho' \models WF(X)$ *for all* $X \in \mathrm{dom}(\rho')$ *and* $\rho \uplus \rho' \models (\phi_\nu, \mathcal{P}_\nu)$.

This can be shown as a corollary of the following lemma.

Lemma D.5. $\rho \models (\phi, \mathcal{P};\ (X(\widetilde{x}) =_\mu \phi');\ (X_i(\widetilde{x_i}) =_\nu \phi_i)_{i=1}^m)$ *if and only if there is an interpretation* ρ' *of* $X_\Downarrow$ *such that* $\rho' \models WF(X_\Downarrow)$ *and* $\rho \uplus \rho' \models (\sigma_0(\phi), \sigma_0(\mathcal{P});\ (X(\widetilde{x}) =_\nu \sigma_X(\phi'));\ (X_i(b_i, \widetilde{x}, \widetilde{x_i}) =_\nu \sigma_i(\phi_i))_{i=1}^m)$ *where*

$$\begin{aligned}\sigma_0 &\triangleq \{ X_i \mapsto \lambda\widetilde{y}.X_i(\bot, \widetilde{v}, \widetilde{y}) \mid i = 1, \dots, m \}\\ \sigma_X &\triangleq \ \left\{X \mapsto \lambda\widetilde{y}.X(\widetilde{y}) \wedge X_\Downarrow(\widetilde{x}, \widetilde{y})\right\} \cup \{ X_i \mapsto \lambda\widetilde{y}.X_i(\top, \widetilde{x}, \widetilde{y}) \mid i = 1, \dots, m \}\\ \sigma_i &\triangleq \ \left\{X \mapsto \lambda\widetilde{y}.X(\widetilde{y}) \wedge (b_i \Rightarrow X_\Downarrow(\widetilde{x}, \widetilde{y}))\right\} \cup \{ X_j \mapsto \lambda\widetilde{y}.X_j(b_i, \widetilde{x}, \widetilde{y}) \mid j = 1, \dots, m \}\ .\end{aligned}$$

Recall that the semantics of μCLP defined in Section 3.1 is compositional in the sense that the semantics of $\mathcal{P}_2$ in $\mathcal{P}_1$; $\mathcal{P}_2$ is defined independently with $\mathcal{P}_1$ and is parameterized by an interpretation ρ for the free predicate variables in $\mathcal{P}_2$. Note also that the guard $X_\Downarrow(\widetilde{x}, \widetilde{y})$ in Lemma D.5 for a call to X is enabled only when the Boolean argument b_i is $\top$ (i.e., only for co-recursive calls caused (possibly indirectly) by X). Thus, by parameterizing ϕ and $\mathcal{P}$ away and focusing on the case where all b_i's are $\top$ (the other cases are straightforward), the proof of Lemma D.5 boils down to showing the following lemma.

Lemma D.6. *Let* $\mathcal{P} \triangleq (X(\widetilde{x}) =_\mu \phi);\ (X_i(\widetilde{x_i}) =_\nu \phi_i)_{i=1}^m$ *and* $\mathcal{P}' \triangleq (X(\widetilde{x}) =_\nu \sigma(\phi));\ (X_i(\widetilde{x}, \widetilde{x_i}) =_\nu \sigma(\phi_i))_{i=1}^m$ *where*

$$\sigma \triangleq \ \left\{X \mapsto \lambda\widetilde{y}.X(\widetilde{y}) \wedge X_\Downarrow(\widetilde{x}, \widetilde{y})\right\} \cup \{ X_i \mapsto \lambda\widetilde{y}.X_i(\widetilde{x}, \widetilde{y}) \mid i = 1, \dots, m \}\ .$$

We then have

(1) $[\![\mathcal{P}']\!](\rho \uplus \left\{X_\Downarrow \mapsto R\right\}) \subseteq [\![\mathcal{P}]\!](\rho)$ *for any well-founded relation* R *and*
(2) $[\![\mathcal{P}]\!](\rho) \subseteq [\![\mathcal{P}']\!](\rho \uplus \left\{X_\Downarrow \mapsto R\right\})$ *for some well-founded relation* R.

Proof:

(1) The soundness is shown by well-founded induction with R. Let F and G be the defining function of X in $\mathcal{P}$ and $\mathcal{P}'$, respectively. In other words, X in $\mathcal{P}$ and $\mathcal{P}'$ are respectively expressed as $\mu X.F(X)$ and $\nu X.G(X)$. Let A be a fixpoint of G. We then show $\forall \widetilde{x} \in A.\ \widetilde{x} \in \mu X.F(X)$ by well-founded induction with R. Since $A = G(A)$, we have $A(\widetilde{x}) = F(\{ \widetilde{y} \in A \mid R(\widetilde{x}, \widetilde{y}) \})$. Therefore, $\widetilde{x} \in A \Rightarrow \widetilde{x} \in F(\{ \widetilde{y} \in A \mid R(\widetilde{x}, \widetilde{y}) \})$. By the induction hypothesis, we have $\{ \widetilde{y} \in A \mid R(\widetilde{x}, \widetilde{y}) \} \subseteq \mu X.F(X)$. By the monotonicity of F, $F(\{ \widetilde{y} \in A \mid R(\widetilde{x}, \widetilde{y}) \}) \subseteq F(\mu X.F(X)) = \mu X.F(X)$. We thus get $\widetilde{x} \in A \Rightarrow \widetilde{x} \in \mu X.F(X)$.
(2) For proving the completeness, we construct such R as the *stage comparison relation* studied in finite model theory (see e.g., [Kreutzer 2004], [Grädel et al. 2005], or [Libkin 2004] for details). The *stage* of $\widetilde{v}$ for X in $\mathcal{P}$ is defined as the smallest ordinal number α such that $\widetilde{v} \in F^\alpha(\bot)$ holds where F is the defining function of X in $\mathcal{P}$. We then define R by $\{ (\widetilde{x}, \widetilde{y}) \mid \mathrm{stage}(\widetilde{x}) > \mathrm{stage}(\widetilde{y}) \}$. Let G be the defining function of X in $\mathcal{P}'$. We then get $F^\alpha(\bot) = G^\alpha(\bot) \subseteq G^\alpha(\top)$. Here, $F^\alpha(\bot) = G^\alpha(\bot)$ under the interpretation $\rho \uplus \left\{X_\Downarrow \mapsto R\right\}$ is shown by transfinite induction on α.

■

Received 2022-07-07; accepted 2022-11-07