PROGRAM VERIFICATION USING POLYNOMIALS OVER MODULAR ARITHMETIC

a thesis submitted to
The University of Kent
in the subject of computer science
for the degree
of doctor of philosophy.

By
Thomas Seed
January 2021

Abstract

As program verification has matured as a discipline, so distinct topics have emerged and then developed into thriving sub-disciplines, each with their own language and focus. In Satisfiability Modulo Theories (SMT) solving the focus is on deciding the satisfiability of formulae over predicates (constraints) drawn from a background theory. If a SMT formula encodes the existence of a problematic path through a program, then a model of the formula will expose a fault as demonstrated with a counter-example. In abstract interpretation, on the other hand, the objective is typically to infer invariants for a program so as to demonstrate the absence of a fault. These complementary sub-disciplines do not exist in silos completing against one another: one sub-discipline informs the other. This thesis illustrates how these sub-disciplines cross-fertilise in both directions: presenting a new abstract domain that draws on techniques from SMT solving, namely solving systems of symbolic equations (theory solving). One fundamental operation used in the domain construction applies a propagation technique that suggests how the satisfiability the SMT formulae can be reduced to that of deciding the satisfiability of a compact SAT instance. This leads to a new technique for SMT solving.

Although developed in tandem, for sake of presentation the thesis first addresses the satisfiability of systems of polynomial equations over bit-vectors. Instead of conventional bit-blasting, we exploit word-level inference to translate these systems into non-linear pseudo-boolean constraints. We derive the pseudo-booleans by simulating bit assignments through the addition of (linear) polynomials and applying a strong form of propagation by computing Gröbner bases, which provide an analog of a triangular form for systems of polynomials. By handling bit assignments symbolically, the number of Gröbner basis calculations, along with the number of assignments, is reduced. The final Gröbner basis yields an assignment

to the bit-vectors, expressed parametrically in terms of the symbolic bits, together with non-linear pseudo-boolean constraints on the symbolic variables, modulo a power of two. The pseudo-booleans can be solved by translation into classical linear pseudo-boolean constraints (without a modulo) or by encoding them as propositional formulae, for which a novel translation process is described. This aspect of the thesis has a practical bias.

The dual theme of the thesis on abstract domain construction has a theoretical bias. The thesis presents MPAD, the modulo polynomial abstract domain, whose invariants are systems of polynomial equations that hold modulo 2^ω where ω is bit-width. MPAD systems over d variables symbolically represent sets of points in the d-dimensional space $(\mathbb{Z}_{2^\omega})^d$ as their solutions, and provide a way of representing and inferring polynomial invariants in the presence of wrap-around arithmetic. The domain operations of MPAD are computed using Gröbner bases, but are founded on a closure operation, mirroring a construction familiar in numeric abstraction. Given an input system of polynomials, and their associated solutions, closure derives a finite polynomial representation of all polynomials that satisfy these solutions. Closure is necessary for faithfully computing join and projection, operations that preserve it. Meet does not maintain closure, hence the need for an algorithm for computing it. Unlike convention polynomial abstraction, MPAD satisfies the ascending chain condition, finessing the need for widening. It also remedies the disparity in handling of equality but not disequality in guards, normally found in numeric abstraction: the structure of MPAD allowing the addition of a single polynomial disequality to be reexpressed using closure and join. We demonstrate that MPAD can derive invariants necessary for verifying the correctness of algorithms which exploit integrality, that were previously out of reach.

As a whole, the thesis makes contributions to SMT solving and abstract interpretation, two complementary themes of program verification, both of which draw on common techniques from algebraic computation, namely Gröbner bases.

Acknowledgements

First and foremost I would like to thank my supervisor Andy King for his patience and dedication in helping me submit this thesis. I would also like to thank my girlfriend Georgia for her unending care for me throughout this process - you've given up a lot and I couldn't have done it without you. Thanks also to my family and Georgia's family who have been a great support to me throughout.

I would also like to thank Paul Subotic, for giving me a home at Amazon and encouraging me in my pursuit of program verification and abstract interpretation, Chris Coppins who helped me get up with speed with Scala when I needed to rapidly develop a Gröbner engine, and finally Neil from AWE who saw a spark in the Gröbner basis and encouraged us to develop it into an abstract domain.

Contents

List of Figures

Chapter 1

Introduction

This thesis makes contributions to both SMT solving and abstract interpretation, the first by developing a new architecture for solving systems of polynomials where the modulo is a power of two, and the second by devising a new abstract domain of systems of polynomials, again where the modulo is a power of two. To provide context for both developments, this chapter provides a gentle introduction to both SMT and abstract interpretation. Work that closely relates to a technical development given in a latter chapter is provided close to the development itself, at the end of the respective chapter. The notable exception is the chapter on Gröbner bases over modular integers, which is a common underlying theme, that warrants its own extended primer.

1.1 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) [51] is concerned with deciding the satisfiability of formulae over constraints drawn from a background theory. Examples of such theories include, but are not limited to difference logic [67], uninterpreted functions [7] and bit-vectors and arrays [31]. The dominant approach to SMT solving, referred to as the lazy approach [68], seeks to simplify the decision problem by separating the logical structure of a formula from its interpretation in a given theory. The logical form is described by a propositional formula, for which candidate truth assignments can be proposed by a SAT solver. On the theory side, such

assignments correspond to conjunctions of atomic constraints and their negations, which can then be solved by a domain-specific solver. The approach can exploit the extreme efficiency of modern SAT solvers. Simultaneously, new theories can be handled by providing a domain-specific solver just for conjunctions of theory literals, rather than arbitrary formulae, greatly simplifying the development effort. The remainder of this section discusses lazy SMT solving in more detail, first addressing the Boolean satisfiability problem, before showing how it can be extended with a background theory.

1.1.1 SAT solving

The Boolean satisfiability problem (SAT) is the problem of determining whether there is an assignment of boolean values to the variables in a propositional formula under which it evaluates to true. To illustrate, consider the propositional formula

$$f = (u \vee \neg w) \wedge (x \vee \neg y \vee z) \wedge (\neg u \vee v \vee \neg z)$$

defined over the set of propositional variables $X = \{u, v, w, x, y, z\}$. A truth assignment is a partial mapping $\theta : X \to \{\mathsf{true}, \mathsf{false}\}$, which is said to satisfy f if evaluating f with these assignments yields true. In the present case, $\theta = \{u \mapsto \mathsf{true}, x \mapsto \mathsf{true}, z \mapsto \mathsf{false}\}$ is a satisfying assignment. The goal of SAT is to decide if a propositional formula f is satisfiable, and if so to determine a satisfying assignment.

Most modern SAT solvers are based on the Davis, Putman, Logemann, Loveland (DPLL) algorithm [20], for which a recursive formulation presented in Fig. 1, adapted from [79]. The first input to the algorithm is a propositional formula f, assumed to be a conjunctive normal form (CNF). A CNF formula is a conjunction of clauses, where a clause is a disjunction of literals and a literal is either a variable or the negation of a variable. The second input to the algorithm is truth assignment θ. Intuitively, the call $\mathsf{DPLL}(f, \theta)$ determines the satisfiability of the formula f under the (partial) truth assignment θ. The algorithm either returns $\bot$, indicating that f is unsatisfiable under this assignment, or else an extension $\theta' \supseteq \theta$ of θ to a satisfying assignment. By calling $\mathsf{DPLL}(f, \emptyset)$, where $\emptyset$ denotes the empty truth function, the (unconditional) satisfiability of f can thus be decided.

```
function DPLL(f : CNF formula, θ : truth assignment)
begin
    θ1 := θ ∪ unit_propagation(f, θ)
    if (is_conflicting(f, θ1))
        return ⊥
    else if (is_satisfied(f, θ1))
        return θ1
    else
        x := choose_free_variable(f, θ1)
        θ2 := DPLL(f, θ1 ∪ {x ↦ true})
        if (θ2 ≠ ⊥)
            return θ2
        else
            return DPLL(f, θ1 ∪ {x ↦ false})
        end if
    end if
end
```

Figure 1: The DPLL algorithm

DPLL first attempts to derive assignments that must necessarily hold for f to be satisfiable To illustrate, consider f as above and $\theta = \{v \mapsto \mathsf{false}, w \mapsto \mathsf{true}\}$. Then, since $w \mapsto \mathsf{true}$, the assignment $u \mapsto \mathsf{true}$ must hold to ensure the clause $(u \vee \neg w)$ evaluates to true. Consequently, the assignment $z \mapsto \mathsf{false}$ must hold for the clause $(\neg u \vee v \vee \neg z)$ to evaluate to true. By contrast, after these two assignments, the satisfiability of the clause $(x \vee \neg y \vee z)$ still depends on two unknowns x and y hence this clause yields no further information. It follows that θ may be extended to $\theta_1 = \theta \cup \{u \mapsto \mathsf{true}, z \mapsto \mathsf{false}\}$ without affecting the satisfiability of f under θ. This process, referred to a unit propagation, is carried out in the call $\mathsf{unit\_propagation}(f, \theta)$ which thus returns θ_1. Note that unit propagation only applies when there is a single unassigned variable in a clause. This situation can be effectively detected by maintaining a reference to two unassigned variables in each clause using watched literals [61].

After unit propagation, DPLL tests for early termination. First, if f contains a clause for which every literal is unsatisfiable under θ_1 then f is itself unsatisfiable and $\bot$ is returned. This situation is detected by the call $\mathsf{is\_conflicting}(f, \theta_1)$. To

illustrate, consider $g = (\neg x) \wedge (x \vee y) \vee (\neg y)$ and $\theta = \{x \mapsto \mathsf{false}, y \mapsto \mathsf{false}\}$. Then, each literal in the second clause is unsatisfiable under θ_1, hence g is unsatisfiable. Conversely, if every clause of f contains at least one literal that is satisfied under θ_1 then f is also satisfied under θ_1, which is thus returned. This situation is detected by the call $\mathsf{is\_satisfied}(f, \theta_1)$. To illustrate, consider $h = (x \vee \neg y) \wedge (y \vee z)$ and $\theta_1 = \{y \mapsto \mathsf{false}, z \mapsto \mathsf{true}\}$. Then, the literal $\neg y$ is satisfied in $(x \vee \neg y)$ and the literal z is satisfied in $(y \vee z)$, hence h is satisfiable under θ_1.

If neither of these situations applies, a currently unassigned variable x is selected, as effected through $\mathsf{choose\_free\_variable}(f, \theta_1)$. A recursive call is then made with the augmented assignment $\theta_1 \cup \{x \mapsto \mathsf{true}\}$. This either yields a satisfying assignment, or else a recursive call is made under the assignment $\theta_1 \cup \{x \mapsto \mathsf{false}\}$. The return value form this call determines the satisfiability of the original formula. Termination of this procedure is ensured since the number of unassigned variables strictly reduces with each recursive call. To illustrate, with f and θ_1 as above, neither of the calls $\mathsf{is\_conflicting}(f, \theta_1)$ and $\mathsf{is\_satisfied}(f, \theta_1)$ succeeds. The unassigned variable x is thus selected and the recursive call $\mathsf{DPLL}(f, \theta_1 \cup \{x \mapsto \mathsf{true}\})$ is made. Since $\theta_2 = \theta_1 \cup \{x \mapsto \mathsf{true}\}$ is already a satisfying assignment for f, θ_2 is thus returned. On top of this framework, modern SAT solvers apply a range of techniques to improve performance, for instance, static and dynamic variable orderings [61], non-chronological backtracking [56] and clause-learning [55].

1.1.2 Lazy SMT solving

To illustrate the lazy approach to SMT solving, consider the following formula, drawn from the theory of linear real arithmetic:

$$\phi = (a < b) \wedge (a = 0 \vee a = 1) \wedge (b = 0 \vee b = 1) \wedge \neg(1 \leq a + b)$$

The set of atomic constraints in ϕ is $\Sigma = \{a < b, a = 0, a = 1, b = 0, b = 1, 1 \leq a + b\}$. The propositional skeleton $e(\phi)$ of ϕ is constructed by mapping each atomic constraint in Σ to a propositional variable. Formally, we consider a bijective mapping $e : \Sigma \to X$ where X is a set of propositional variables. For

instance, letting $X = \{u, v, w, x, y, z\}$, such a mapping can be defined

$$\begin{array}{lll} e(a < b) = u & e(a = 0) = v & e(a = 1) = w \\ e(b = 0) = x & e(b = 1) = y & e(1 \leq a + b) = z \end{array}$$

The mapping e then lifts to formulae ϕ whose atomic constraints are drawn from Σ, by replacing each atomic constraint c in ϕ with $e(c)$ while preserving propositional structure of ϕ. For instance, for ϕ as above this yields

$$e(\phi) = u \wedge (v \vee w) \wedge (x \vee y) \wedge \neg z$$

To decide the satisfiability of ϕ, first a satisfying assignment θ to $e(\phi)$ is sought. Note if no such assignment exists then ϕ is unsatisfiable by virtue of its propositional structure. Otherwise, θ corresponds to a conjunction of theory literals, $T(\theta, e)$, which contains the literal l whenever $\theta(e(l)) = \mathsf{true}$ and the literal $\neg l$ whenever $\theta(e(l)) = \mathsf{false}$. In the present example, $\theta = \{u \mapsto \mathsf{true}, v \mapsto \mathsf{true}, w \mapsto \mathsf{true}, x \mapsto \mathsf{true}, y \mapsto \mathsf{true}, z \mapsto \mathsf{false}\}$ is a satisfying assignment for $e(\phi)$ and $T(\theta, e)$ is defined

$$T(\theta, e) = (a < b) \wedge (a = 0) \wedge (a = 1) \wedge (b = 0) \wedge (b = 1) \wedge \neg(1 \leq a + b)$$

The satisfiability of $T(\theta, e)$ can be determined by a specialised solver for linear real arithmetic. The solver will return either $\top$, to indicate $T(\theta, e)$ is satisfiable, else a clause t which illustrates why it is unsatisfiable. In the first case, it follows that ϕ itself is satisfiable, whereas in the second, the chosen assignment θ does not to yield a satisfying assignment to ϕ. In the present case, the solver might return $t = \neg(a = 0) \vee \neg(a = 1)$, thus providing an explanation for why $T(\theta, e)$ is unsatisfiable.

If the assignment θ did not yield a solution, then a new assignment is sought. However, to avoid rediscovering same assignment, ϕ is strengthened with a new blocking clause, which serves to guide search away from previously discovered solutions. Concretely, a new formula f is defined $f = e(t) \wedge \phi$, where $e(t)$ codifies the inconsistent clause t as a propositional clause. In the present case, the blocking clause is $e(\neg(a = 0) \vee \neg(a = 1)) = \neg v \vee \neg w$, which prevents further solutions

```
function DPLL_T(f : CNF formula, e : Σ → X)
begin
    θ := DPLL(f, ∅)
    if (θ = ⊥)
        return ⊥
    else
        t := deduce(T(θ, e))
        if (t = ⊤)
            return ⊤
        else
            return DPLL_T(f ∧ e(t), e)
        end if
    end if
end
```

Figure 2: The SMT framework

being found in which both v and w are assigned true. The while procedure is then repeated on f and any further strengthenings of f that are inferred from inconsistent assignments along the way. Eventually, either the theory solver succeeds on some assignment, hence ϕ is satisfiable, or else the formula f becomes unsatisfiable, in which case ϕ is unsatisfiable.

Fig. 2 presents an algorithm that formalises this approach, based on a recursive reformulation of Algorithm 3.3.1 from [51]. The algorithm is parameterised by a theory T and accepts two inputs. The first is a propositional formula f, initially the propositional skeleton of ϕ, and the second a propositional encoding $e : \Sigma \to X$, as described above. The algorithm either returns $\top$, indicating satisfiability of ϕ, else $\bot$, indicating unsatisfiability. The procedure applies DPLL to discover satisfying assignments to f, and a theory-specific procedure deduce for detecting satisfiability of the conjunction of theory literals $T(\theta, e)$.

1.2 Abstract Interpretation

Abstract interpretation [14] provides a rigorous basis for static program analysis. The essential idea is to synthesise an abstraction of data and deploy domain

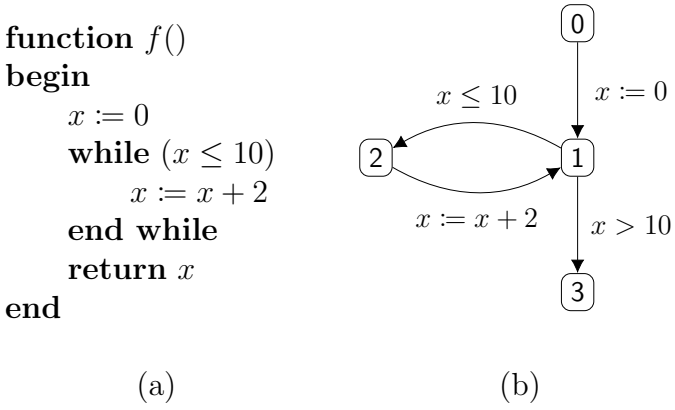


Figure 3: A toy program (a) and its flow graph (b)

operations that trace that abstraction over the paths of the program to automatically derive program invariants. By formally relating the abstract and concrete domain operations, abstract interpretation can be applied to show that an analysis is sound. Moreover, by expressing the analysis as a fixpoint of a system of equations, the outcome of an analysis can be computed iteratively. Thus abstract interpretation is not only a methodology for designing and justifying analyses: it provides a pathway for realising them too.

The beautiful simplicity of abstract interpretation is often lost in the formality of its presentation, hence this section will motivate the essential elements of abstract interpretation through the toy program presented in Fig. 3(a). The variable x stores an (unbounded) integer and an abstract interpretation of the program aims to discover the values x can assume at each program point. Fig. 3(b) presents the control-flow graph of this program. Each node in this graph corresponds to a location in the program, either prior to the initial first assignment (0), prior to the loop test (1), prior to the second assignment (2), or prior to return (3).

Concrete semantics An abstract interpretation begins by modelling the concrete states of the program. First, a concrete domain $\mathcal{C}$ is introduced, which serves as a data-type to describe the concrete property of interest. In the present case, the concrete domain can be taken as $\wp(\mathbb{Z})$ since, at each program point, the values

obtained by x constitute a set of integers. To relate this to the execution of the program, the concrete (collecting) semantics is introduced, which serves as a reference against which the soundness of a proposed analysis can be judged. The concrete semantics defines for each program point i the set $C_i \in \mathcal{C}$ of all concrete states obtained at that program point over all execution paths of the program. For realistic programs, the concrete semantics is uncomputable, or at least prohibitatively expensive to compute. For the toy, however, it can be seen that:

$$C_0 = \mathbb{Z} \quad C_1 = \{0, 2, \ldots, 12\} \quad C_2 = \{0, 2, \ldots, 10\} \quad C_3 = \{12\}$$

where C_i is the set of values obtained by x at program point i. In particular, the value of x is arbitrary at program point 0 (x is assumed initialised to some unknown value), an even number between 0 and 12 at program point 1, an even number between 0 and 10 at program point 2 and exactly 12 at program point 3.

Abstract semantics Even if the concrete semantics cannot be effectively computed, it may still be approximated. This is achieved by introducing an abstract domain $\mathcal{A}$, whose elements are abstractions of sets of concrete program states. For the present example, the interval domain [14] will be employed. Elements of this domain are either intervals $[l, u]$ where $l \in \mathbb{Z} \cup \{-\infty\}$, $u \in \mathbb{Z} \cup \{\infty\}$ and $l \leq u$, or else $\bot$ indicating an empty set. Here, an interval is considered simply as a pair of bounds, giving a compact representation of the (potentially infinite) set it represents. Note in particular that intervals cannot represent arbitrary sets of points, hence there is an inherent loss of information (precision) induced by this choice of domain. This is a general theme in abstract interpretation, and is the tradeoff needed for tractability. A rich variety of abstract domains have been proposed in the literature, for instance the congruence domain [34], the polyhedral domain [16] and the Octagon domain [59], and these for describing numerical properties alone. Each has its own unique characteristics that make it suitable for certain applications, and unsuitable for others. The craft of abstract interpretation is in choosing, or designing, an appropriate abstraction for the given application, carefully balancing performance against precision.

The relationship between the abstract and concrete domains is made plain

through a mapping $\gamma : \mathcal{A} \to \mathcal{C}$, called concretisation. Intuitively, if $a \in \mathcal{A}$ is an abstract domain element, then $\gamma(a)$ is the concrete domain element it represents. For instance, in the interval domain, $\gamma([0, \infty]) = \{0, 1, 2, \ldots\}$, crystallising the idea that $[0, \infty]$ describes the set of non-negative integers. Moreover, just as the concrete domain is associated with a concrete semantics, so too the abstract domain is associated with an abstract semantics. The abstract semantics specifies for each program point i an abstract domain element $A_i \in \mathcal{A}$, which describes the set C_i of concrete states at that point. This specification is sound at program point i if $C_i \subseteq \gamma(A_i)$. For instance, for the toy, the interval $A_1 = [0, 12]$ is a sound approximation of the set $C_1 = \{0, 2, \ldots, 12\}$ of concrete states at program point 1, since $\{0, 2, \ldots, 12\} \subseteq \gamma([0, 12]) = \{0, 1, \ldots, 12\}$. Note, however, this does not exclude the possibly of odd assignments, which is a consequence of abstraction.

In certain situations, a second map $\alpha : \mathcal{C} \to \mathcal{A}$ can be defined called abstraction which provides the best abstraction of C by an element of $\mathcal{A}$. This leads to the Galois connection approach to abstract interpretation [14] which provides a development methodology where γ can be synthesised from α and vice versa.

Fixpoint formulation An abstract semantics for the program is typically formulated as the solution of a system of fixpoint equations, which provide conditions on the sets $\gamma(A_i)$ in order that the abstract semantics be sound. To illustrate with the toy example, these equations could be defined

$$\begin{array}{ll}
\gamma(A_1) \supseteq \{0\} \text{ if } \gamma(A_0) \neq \emptyset & (x \coloneqq 0) \\
\gamma(A_2) \supseteq \{a \in \gamma(A_1) \mid a \leq 10\} & (x \leq 10) \\
\gamma(A_1) \supseteq \{a + 2 \mid a \in \gamma(A_2)\} & (x \coloneqq x + 2) \\
\gamma(A_3) \supseteq \{a \in \gamma(A_2) \mid a > 10\} & (x > 10)
\end{array}$$

Each equation is derived from a program statement, as indicated on the right. To illustrate, suppose $a \in \gamma(A_1)$. Now if A_1 is sound approximation at program point 1 then a might be a valid assignment to x at that point. Therefore, if $a \leq 10$, then the loop test $(x \leq 10)$ would succeed and a would also be a valid assignment to x at program point 2, hence $a \in \gamma(A_2)$ is required for soundness. Conversely, if $a > 10$ then the loop test would fail, hence a would be a valid assignment to x

at program point 3 and so $a \in \gamma(A_3)$ should hold. This justifies the second and fourth equations. Similarly, if $\gamma(A_0) \neq \emptyset$ then x might be assigned some value at program point 0, which must therefore be reachable. In this case, x would obtain the value 1 at program point 1 under the assignment $x := 1$, hence $1 \in \gamma(A_1)$. Similarly, if $a \in \gamma(A_2)$ then a might be a valid assignment to x at program point 2, in which case $a + 2$ would be a valid assignment to x at program point 1 under the assignment $x := x + 2$, hence $a + 2 \in \gamma(A_1)$. This justifies the first and third equations.

Fixpoint solution To compute the abstract semantics, an iterative procedure is applied. Intuitively, values for each A_i are proscribed and then the semantic equations are repeated checked and reevaluated. If they all hold, then a valid solution has been found which is returned. Otherwise, some A_i is updated to ensure the violated equation is satisfied, and the process is repeated. The abstract elements are initialised

$$A_0 = [-\infty, \infty] \quad A_1 = \bot \quad A_2 = \bot \quad A_3 = \bot$$

which reflects the fact that x might take any value at program point 0, but no assignments to x have yet been witnessed at any other program point.

Now, since $\gamma(A_0) \neq \emptyset$, the first equation above must apply, hence $\gamma(A_1) \supseteq \{0\}$. But $A_1 = \bot$ hence $\gamma(A_1) = \emptyset \not\supseteq \{0\}$ and the equation is thus violated. Hence, A_1 is reassigned to $[0, 0]$ to ensure $\gamma(A_1) \supseteq \{0\}$, yielding

$$A_0 = [-\infty, \infty] \quad A_1 = [0, 0] \quad A_2 = \bot \quad A_3 = \bot$$

Next, the second equation applies to show $\gamma(A_2) \supseteq \{a \in \gamma(A_1) \mid a \leq 10\} = \{0\}$. But, as above, $A_2 = \bot$ hence $\gamma(A_2) = \emptyset \not\supseteq \{0\}$ and the equation is violated. Hence, A_2 is reassigned to $[0, 0]$ to ensure $\gamma(A_2) \supseteq \{0\}$ yielding

$$A_0 = [-\infty, \infty] \quad A_1 = [0, 0] \quad A_2 = [0, 0] \quad A_3 = \bot$$

Now, the third equation applies to show $\gamma(A_1) \supseteq \{a + 2 \mid a \in \gamma(A_2)\} = \{2\}$. But since $\gamma(A_1) = \{0\} \not\supseteq \{2\}$ the equation is violated. In this case, it is already known

that $\gamma(A_1) \supseteq \{0\}$ is necessary, hence it now follows $\gamma(A_2) \supseteq \{0, 2\}$ must hold. To ensure this, A_2 is updated to $[0, 2]$, for which $\gamma(A_2) = \{0, 1, 2\} \supseteq \{0, 2\}$, hence

$$A_0 = [-\infty, \infty] \quad A_1 = [0, 2] \quad A_2 = [0, 0] \quad A_3 = \bot$$

The most recent assignment to A_1 again violates the second equation, which is then corrected by reassigning A_2 to $[0, 2]$, which then leads to further relaxation of A_1. Thus a sequence of alternating updates to A_1 and A_2 occurs until finally

$$A_0 = [-\infty, \infty] \quad A_1 = [0, 12] \quad A_2 = [0, 10] \quad A_3 = \bot$$

At this point, the second equation requires $\gamma(A_2) \supseteq \{a \in \gamma(A_1) \mid a \leq 10\} = \{0, \ldots, 10\}$ and since $\gamma(A_2) = \{0, \ldots, 10\}$ this already holds. Similarly, the third equation requires $\gamma(A_1) \supseteq \{a + 2 \mid a \in \gamma(A_2)\} = \{0, \ldots, 12\}$ and since $\gamma(A_2) = \{0, \ldots, 12\}$ this already holds as well. The first equation also applies trivially, hence only the fourth equation remains to validate. In this case, it is required $\gamma(A_3) \supseteq \{a \in \gamma(A_2) \mid a > 10\} = \{11, 12\}$. Since $A_3 = \bot$ this does not currently hold, so A_3 is updated to $[11, 12]$, yielding

$$A_0 = [-\infty, \infty] \quad A_1 = [0, 12] \quad A_2 = [0, 10] \quad A_3 = [11, 12]$$

At this point, all four equations are satisfied, hence the analysis terminates. Note particularly that for each i it holds that $C_i \subseteq \gamma(I_i)$, hence the inferred analysis is sound. However, the analysis is not fully precise. For instance, the analysis can only infer that x might take the value 11 or 12 at program point 3. To overcome this, a more refined abstract domain must be used. For instance, a product domain construction [10] between intervals and the congruence domain [34] could be employed to reason also about the value of x modulo 2.

Note, the description of the analysis presented here is somewhat simplified. First, the calculation of the updates to the abstract semantics occurred by reasoning about the concretisations $\gamma(A_i)$. In practice this is not possible, hence computational procedures must be designed to compute the updates solely in terms of the abstract domain elements. This is formalised through the notion of abstract transfer functions. Second, the analysis presented here terminated rapidly. In

certain situations, however, termination may not occur, or at least the number of iterations is so great that a fixpoint is impractical to compute. In this case, fixpoint acceleration techniques, for instance widening [15], may be applied. This approach computes a sound over-approximation to the fixed-point, ensuring termination at the cost of a potential loss of precision. To an extent, this precision can be regained through the complementary technique of narrowing [15], illustrating the interplay that often arises between the fidelity of the abstraction and the fixpoint technique.

1.3 Contributions

The present work turns its attention first to SMT, specifically the decision problem for systems of polynomial equalities over bit-vectors. Conventional approaches to this problem employ bit-blasting, where constraints are translated to propositional formulae by modelling them as circuits, then solved with a SAT solver. However, in the presence of bit-vector multiplication, the resulting formulae can be prohibitatively large, and the opportunity to exploit word-level reasoning is lost.

The first novel contribution of this thesis seeks to address these limitations, presenting a new architecture for solving systems of polynomial equalities over bit-vectors. Rather than converting to SAT and bit-blasting, the method simulates bit assignments through the addition of certain polynomials to the system. Computing a Gröbner basis for the resulting system can then infer new entailed constraints which, in turn, expose the values of other bits, a technique termed bit-sequence propagation. Moreover, by assigning symbolic truth values to bits, the procedure can avoid backtracking, yielding instead a residue system of non-linear pseudo-boolean constraints modulo a power of two. The residue system can be solved through a novel translation procedure that converts it to a compact SAT instance. Overall, the architecture provides a principled method for compiling high-level polynomials to low-level pseudo-boolean constraints, and then to SAT.

Next, the work turns its attention to abstract interpretation, specifically the inference of polynomial invariants over fixed-width integers. The development centres on MPAD: the Modulo Polynomial Abstract Domain, whose domain operations are founded on a closure operator, that extends a set of polynomials with all

other polynomials that share their solutions. Preserving closure is key to maintaining the precision of domain operations, yet is not itself preserved by meet, hence closure must be explicitly computed. To that end, the technique underpinning bit-sequence propagation finds fresh application. Indeed, by judiciously setting bits of individual variables, and applying propagation, a system can be reduced to a collection of simpler systems from which closure can be computed directly.

Using closure, we provide abstract transfer functions for MPAD for the class of polynomial programs, demonstrating particularly that the transfer function for polynomial assignment is optimal. Coupled with the finiteness of MPAD, it follows that MPAD will infer all polynomial invariant for programs consisting solely of polynomial assignments. MPAD is unique in that it tracks polynomials in the context of modular arithmetic, complementing the new approach to SMT solving for modular polynomials, both building on bit-sequence propagation.

1.4 Roadmap

This thesis is structured as follows: Chapter 2 introduces Gröbner bases over modulo integers, which are the fundamental tool employed in both the SMT and MPAD work. Chapter 3 applies Gröbner bases over modular arithmetic to derive a decision procedure for polynomial equalities over bit-vector arithmetic. Chapter 4 formally introduces MPAD before defining polynomial programs and providing an illustrative example of MPAD in deriving a quadratic loop invariant. Chapter 5 discusses the computational aspects of MPAD, explaining how the abstract domain operations are organised and realised. Chapter 6 summarises the contributions of this thesis, and reviews strands of inquiry that emerged during this thesis work which are yet be explored. Finally, to maintain a rapid pace of presentation, proofs for the major results, and even statements and proofs for minor results, are relegated to Appendix A. Their position should not be interpreted as a comment on their importance.

Chapter 2

Gröbner Bases

This chapter is a primer on the theory of Gröbner bases. In particular, the nuances of calculating Gröbner bases over modulo integers are discussed, with reference to a variant of Buchberger's algorithm originally proposed for circuit verification [5]. The chapter also distills prior work on null-polynomials [35], which have been reinvented repeatedly within the mathematics literature over the last century [47, 43, 75]. Of prime importance in this development is an enumeration of a minimal (irredundant) Gröbner basis for nulls, which is used, amongst other things, in the calculation of closure.

The chapter commences by formally introducing the set of modulo integers $\mathbb{Z}_m$ where $m = 2^\omega$. Divisibility in $\mathbb{Z}_m$ can be elegantly formulated in terms of the concept of rank, which is discussed next. After introducing the core concepts of leading terms and reduction, Gröbner bases are formally defined, following a conventional development. Thereafter, when algorithmic properties are of interest, the presentation diverges from the norm. In particular, Buchberger's classic criterion, which combines pairs of polynomials to eliminate leading terms, is extended by scaling single polynomials by powers of two, in order to likewise eliminate leading terms. Next, a way to finitely enumerate the set of nulls for any given number of variable and bit-width is presented. The section concludes by discussing polynomial substitution, that is the act of systematically replacing a variable in a polynomial with a polynomial expression, needed again in the formulation of closure. Proofs are omitted for results which are standard, and otherwise explicit citations are given

to literature where proofs can be found.

2.1 Modular Arithmetic

Throughout the following, let $\omega \geq 1$ and $m = 2^\omega$. Following [64, 65], $\mathbb{Z}_m = \{0, \ldots, m-1\}$ is taken as an abstraction of machine arithmetic over ω-bit integers. The relation $\equiv_m \subseteq \mathbb{Z} \times \mathbb{Z}$ is defined by $x \equiv_m y$ if there exists $k \in \mathbb{Z}$ such that $x - y = km$. Atop, the operation $\cdot \pmod{m} : \mathbb{Z} \to \mathbb{Z}_m$ is defined by: $x \pmod{m} = y$ where $y \in \mathbb{Z}_m$ uniquely satisfies $x \equiv_m y$. The unary operation $- : \mathbb{Z}_m \to \mathbb{Z}_m$ and the dyadic operations $+, \cdot : \mathbb{Z}_m \times \mathbb{Z}_m \to \mathbb{Z}_m$ are then defined:

$$-x = (\hat{-}x) \pmod{m} \qquad x + y = (x \mathbin{\hat{+}} y) \pmod{m} \qquad x \cdot y = (x \mathbin{\hat{\cdot}} y) \pmod{m}$$

where $\hat{-}, \hat{+}, \hat{\cdot}$ denote the classical operations over $\mathbb{Z}$. If $x \in \mathbb{Z}_m$ then $y \in \mathbb{Z}_m$ is a multiplicative inverse of x if $x \cdot y = 1$. Note that $x \in \mathbb{Z}_m$ has a multiplicative inverse iff it is odd, in which case the inverse is unique. In particular, if $\omega > 1$ then $\mathbb{Z}_m$ is not a field, since 2 has no inverse. The inverse x^{-1} can be found as a stationary point of the sequence $y_1 = 1$, $y_{n+1} = y_n(2 - xy_n)$ [64].

Example 1. In $\mathbb{Z}_8$, each odd element of $\mathbb{Z}_8$ is self-inverse:

$$1 \cdot 1 = 1 \quad 3 \cdot 3 = 1 \quad 5 \cdot 5 = 1 \quad 7 \cdot 7 = 1$$

Note this property does not hold generally, since in $\mathbb{Z}_{16}$:

$$1{\cdot}1 = 1 \quad 3{\cdot}11 = 1 \quad 5{\cdot}13 = 1 \quad 7{\cdot}7 = 1 \quad 9{\cdot}9 = 1 \quad 11{\cdot}3 = 1 \quad 13{\cdot}5 = 1 \quad 15{\cdot}15 = 1$$

2.2 Rank and divisibility in $\mathbb{Z}_m$

Let $\mid \subseteq \mathbb{Z}^2$ denote the divisibility relation over integers: $a \mid b$ iff b is divisible by a. The rank [64] of $a \in \mathbb{Z}_m$ is defined: $\mathsf{rank}_\omega(a) = \max\{j \in \mathbb{N} \mid 2^j \mid a\}$ if $a > 0$ otherwise ω, and can be computed by counting the number of trailing zeros in the binary representation of a [78].

Example 2. In $\mathbb{Z}_{256}$ where $\omega = 8$, $\mathsf{rank}_8(0) = 8$, $\mathsf{rank}_8(15) = 0$ and $\mathsf{rank}_8(56) = 3$.

If $a \in \mathbb{Z}_m$ then $a = 2^{\mathsf{rank}_\omega(a)}d$ for some odd d. If $a \neq 0$ then $d = a/2^{\mathsf{rank}_\omega(a)}$ is unique and the expression $2^{\mathsf{rank}_\omega(a)}d$ is referred to as the rank decomposition of a. When $a = 0$, d can be arbitrary; for definiteness, we declare $2^\omega \cdot 1$ be the rank decomposition of 0.

Example 3. In $\mathbb{Z}_{256}$, $0 = 2^8 \cdot 1$, $15 = 2^0 \cdot 15$ and $56 = 2^3 \cdot 7$ are rank decompositions.

For $a_1 \in \mathbb{Z}_m$ and $a_2 \in \mathbb{Z}_m \setminus \{0\}$, a_1 is divisible by a_2 if $a_1 = ba_2$ for some divisor $b \in \mathbb{Z}_m$. This occurs iff $\mathsf{rank}_\omega(a_1) \geq \mathsf{rank}_\omega(a_2)$, in which case, if $a_i = 2^{k_i}d_i$ is the rank decomposition of each a_i, then $b = 2^{k_1-k_2}d_1{d_2}^{-1}$.

Example 4. Recall $15 = 2^0 \cdot 15$ and $56 = 2^3 \cdot 7$ are rank decompositions in $\mathbb{Z}_{256}$. Since $\mathsf{rank}_8(56) = 3 \geq 0 = \mathsf{rank}_8(15)$, it follows 56 is divisible by 15 in $\mathbb{Z}_{256}$. Moreover, the divisor can be found by $2^{3-0} \cdot 7 \cdot 15^{-1} = 8 \cdot 7 \cdot 239 = 72$ and indeed $56 = 72 \cdot 15$.

2.3 Polynomials

Let $\vec{x} = \langle x_1, \ldots, x_d \rangle$ be a vector of variables. A monomial over $\vec{x}$ is an expression $\vec{x}^{\vec{\alpha}} = x_1^{\alpha_1} \cdots x_d^{\alpha_d}$ where $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_d \rangle \in \mathbb{N}^d$. A term over $\vec{x}$ is an expression $t = c\vec{x}^{\vec{\alpha}}$ where $c \in \mathbb{Z}_m$ is the coefficient and $\vec{x}^{\vec{\alpha}}$ the monomial of t. A polynomial over $\vec{x}$ is an expression $t_1 + \cdots + t_s$ where each t_i is a term over $\vec{x}$, the case $s = 0$ corresponding to the 0 polynomial. The set of polynomials over $\vec{x}$ is denoted $\mathbb{Z}_m[\vec{x}]$.

A polynomial $p = t_1 + \cdots + t_s$ is normalised if either $s = 0$ or else for all $t_i = c_i\vec{x}^{\vec{\alpha}_i}$ and $t_j = c_j\vec{x}^{\vec{\alpha}_j}$ it holds that $c_i \neq 0$ and if $i \neq j$ then $\vec{\alpha}_i \neq \vec{\alpha}_j$. By repeatedly combining the coefficients of terms with equal monomials, and deleting terms with coefficient 0, a polynomial can be transformed into a normalised form. Two polynomials are considered equal if they have equal normal forms, up to the ordering of terms. In the following, polynomials will always be considered to be normalised. We write $t \in p$ to indicate that the term t is present in the normalised form of p.

Let $\vec{a} \sqsubseteq \vec{b}$ denote the sub-sequence relation, that is, if $\vec{b} = \langle b_1, \ldots, b_s \rangle$ then $\vec{a} = \langle b_{j_1}, \ldots, b_{j_\ell} \rangle$ where $1 \leq j_1 < j_2 < \ldots < j_\ell \leq s$. Then the inclusion $\mathbb{Z}_m[\vec{x}] \sqsubseteq \mathbb{Z}_m[\vec{y}]$ will be considered to hold if $\vec{x} \sqsubseteq \vec{y}$. If $c\vec{x}^{\vec{\alpha}}$ is a term then $\mathsf{vars}(c\vec{x}^{\vec{\alpha}}) = \{x_i \mid \alpha_i > 0\}$,

which is extended to polynomials by $\mathsf{vars}(p) = \bigcup_{t \in p} \mathsf{vars}(t)$. In general, a term t will be printed over just those variables $x_i \in \mathsf{vars}(t)$. In particular, if $t = c\vec{x}^{\vec{\alpha}}$ and $\mathsf{vars}(t) = \emptyset$ then t will simply be denoted c.

The product of two terms $t_1 = c_1\vec{x}^{\vec{\alpha}_1}$ and $t_2 = c_2\vec{x}^{\vec{\alpha}_2}$ it defined $t_1 \cdot t_2 = c_1c_2\vec{x}^{\vec{\alpha}_1+\vec{\alpha}_2}$. Negation of a term $t = c\vec{x}^{\vec{\alpha}}$ is defined $-t = -c\vec{x}^{\vec{\alpha}}$. Arithmetic over $\mathbb{Z}_m[\vec{x}]$ can then be defined $p_1 + p_2 = \sum_{t \in p_1} t + \sum_{t \in p_2} t$, $p_1 \cdot p_2 = \sum_{t_1 \in p_1, t_2 \in p_2} t_1 \cdot t_2$ and $-p = \sum_{t \in p} -t$.

2.4 Bases

A set $B \subseteq \mathbb{Z}_m[\vec{x}]$ is considered to represent the set of polynomials generated thus:

Definition 1. If $B \subseteq \mathbb{Z}_m[\vec{x}]$ then

$$\langle B \rangle_{\vec{x}} = \left\{ \sum_{i=1}^{s} u_i p_i \,\middle|\, s \in \mathbb{N}, p_i \in B, u_j \in \mathbb{Z}_m[\vec{x}] \right\}$$

The set of polynomials $\langle B \rangle_{\vec{x}}$ is an ideal [2] in that it is closed under addition with a polynomial from B and multiplication with an arbitrary polynomial (not necessarily drawn from B). The ideal $\langle B \rangle_{\vec{x}}$ is said to be generated by B, which is called the basis. As will be seen in the following, the principle role of Gröbner bases is to provide a test for membership in the set $\langle B \rangle_{\vec{x}}$.

2.5 Monomial orderings

Gröbner bases are founded on the concept of reduction, which is a rewrite procedure for simplifying a polynomial with respect to a set of polynomials. To define reduction it is necessary to order the terms in a polynomial, leading to the concept of monomial ordering:

Definition 2. A monomial ordering over $\vec{x}$ is a total order $\prec$ over monomials $\vec{x}^{\vec{\alpha}}$ satisfying:

- $1 \prec \vec{x}^{\vec{\alpha}}$ for all $\vec{\alpha} > \vec{0}$

- If $\vec{x}^{\vec{\alpha}_1} \prec \vec{x}^{\vec{\alpha}_2}$ then $\vec{x}^{\vec{\alpha}_1}\vec{x}^{\vec{\beta}} \prec \vec{x}^{\vec{\alpha}_2}\vec{x}^{\vec{\beta}}$ for all $\vec{x}^{\vec{\alpha}_1}$, $\vec{x}^{\vec{\alpha}_2}$ and $\vec{x}^{\vec{\beta}}$.

If $\prec$ is a monomial ordering then $\preceq$ will denote its non-strict version. Note that monomial orderings are well-orderings, hence there is no infinite decreasing chain $\vec{x}^{\vec{\alpha}_1} \succ \vec{x}^{\vec{\alpha}_2} \succ \cdots$ of monomials.

Example 5. Let $\vec{y}$ be a permutation of $\vec{x}$, so $\vec{y} = \langle x_{j_1}, \ldots, x_{j_d} \rangle$ where $j_{i_1} \neq j_{i_2}$ if $i_1 \neq i_2$, and $<$ denote (strict) lexicographical ordering over $\mathbb{N}^d$. Then,

- The ordering $\prec_{\vec{y}}$ defined $\vec{x}^{\vec{\alpha}} \prec_{\vec{y}} \vec{x}^{\vec{\beta}}$ if $\langle \alpha_{j_1}, \ldots, \alpha_{j_d} \rangle < \langle \beta_{j_1}, \ldots, \beta_{j_d} \rangle$ is a monomial ordering, referred to as the lexicographical ordering with respect to $\vec{y}$.
- The ordering $\prec_{\mathsf{deglex}[\vec{y}]}$ defined $\vec{x}^{\vec{\alpha}} \prec_{\mathsf{deglex}[\vec{y}]} \vec{x}^{\vec{\beta}}$ if $\sum_{i=1}^{d} \alpha_i < \sum_{i=1}^{d} \beta_i$, or else $\sum_{i=1}^{d} \alpha_i = \sum_{i=1}^{d} \beta_i$ and $\langle \alpha_{j_1}, \ldots, \alpha_{j_d} \rangle < \langle \beta_{j_1}, \ldots, \beta_{j_d} \rangle$ is a monomial ordering, referred to as the degree lexicographical ordering with respect to $\vec{y}$.

Monomial orderings provide additional structure to polynomials: specifically, if $p \neq 0$ then p can be uniquely expressed as $p = c\vec{x}^{\vec{\alpha}} + q$ where $c \neq 0$ and all monomials $\vec{x}^{\vec{\beta}}$ in q satisfy $\vec{x}^{\vec{\beta}} \prec \vec{x}^{\vec{\alpha}}$. Making use of this additional structure we define:

Definition 3. Let $\prec$ be a monomial ordering over $\vec{x}$ and $p = c\vec{x}^{\vec{\alpha}} + q$ where $c \neq 0$ and all monomials $\vec{x}^{\vec{\beta}}$ in q satisfy $\vec{x}^{\vec{\beta}} \prec \vec{x}^{\vec{\alpha}}$. Then,

- $\mathsf{lt}_{\prec}(p) = c\vec{x}^{\vec{\alpha}}$,
- $\mathsf{lc}_{\prec}(p) = c$,
- $\mathsf{lm}_{\prec}(p) = \vec{x}^{\vec{\alpha}}$,

are respectively the leading term, coefficient and monomial of p with respect to $\prec$.

Leading terms play a foundational role in reduction, introduced in the following section.

2.6 Reduction

As noted above, reduction provides a mechanism to simplify one polynomial by another:

Definition 4. Let $p, q, r \in \mathbb{Z}_m[\vec{x}]$, $p \neq 0$, $q \neq 0$ and $\prec$ a monomial ordering. Then, p is $\prec$-reducible by q to r, denoted $p \to_{\prec,q} r$, if $\mathsf{lt}_\prec(p) = t\,\mathsf{lt}_\prec(q)$ and $p = tq + r$ for some term t.

Reducibility lifts to sets $B \subseteq \mathbb{Z}_m[\vec{x}]$ by $\to_{\prec,B} = \bigcup_{p\in B} \to_{\prec,p}$. Furthermore, let $\to^+_{\prec,B}$ (resp. $\to^*_{\prec,B}$) denote the transitive (resp. transitive, reflexive) closure of $\to_{\prec,B}$. If $p \to^+_{\prec,B} r$ for some r then p is said to be $\prec$-reducible by B, otherwise $\prec$-irreducible by B, denoted $p \not\to_{\prec,B}$.

Example 6. Let $\vec{x} = \langle x, y, a\rangle$ and $B \subseteq \mathbb{Z}_{16}[\vec{x}]$ where

$$B = \left\{ \begin{matrix} x + a^2 + 7a + 7 \;\; (p_1), & y + a^2 + 7a + 7 \;\; (p_2), & \\ a^3 + a^2 + 7a + 7 \;\; (p_3), & 2a^2 + 14 \qquad\quad (p_4), & 8a + 8 \;\; (p_5) \end{matrix} \right\}$$

Now, let $p = xa + 15 \in \mathbb{Z}_{16}[\vec{x}]$ and $\prec = \prec_{\vec{x}}$. Then, $\mathsf{lt}_\prec(p) = xa = a\,\mathsf{lt}_\prec(p_1)$ and $p = ap_1 + r_1$ where $r_1 = 15a^3 + 9a^2 + 9a + 15$, hence $p \to_{\prec,p_1} r$. Similarly, $\mathsf{lt}_\prec(r_1) = 15a^3 = 15\,\mathsf{lt}_\prec(p_3)$ and $r_1 = 15p_3 + r_2$ where $r_2 = 10a^2 + 6$, hence $r_1 \to_{\prec,p_3} r_2$. Finally, $\mathsf{lt}_\prec(r_2) = 10a^2 = 5\,\mathsf{lt}_\prec(p_4)$ and $r_2 = 5p_4 + r_3$ where $r_3 = 0$, hence $r_2 \to_{\prec,p_4} r_3$. Thus, $p \to_{\prec,p_1} r_1 \to_{\prec,p_3} r_2 \to_{\prec,p_4} r_3$, hence $p \to^+_{\prec,B} 0$.

Note p is $\prec$-reducible by B iff $\mathsf{lt}_\prec(p)$ is divisible by $\mathsf{lt}_\prec(q)$ for some $q \in B$. Here, a term t_1 is divisible by a term t_2 if $t_1 = t_2 t_3$ for some term t_3; letting $t_1 = c_1\vec{x}^{\vec{\alpha}}$ and $t_2 = c_2\vec{x}^{\vec{\beta}}$, this occurs iff c_1 is divisible by c_2 and $\vec{\alpha} \geq \vec{\beta}$ pointwise, in which case $t_3 = c_3\vec{x}^{\vec{\delta}}$ where $c_1 = c_3c_2$ and $\vec{\delta} = \vec{\alpha} - \vec{\beta} \geq \vec{0}$. Moreover, reduction eliminates the leading term of a polynomial, leaving a residue polynomial comprised of strictly smaller terms with respect to $\prec$:

Lemma 1. If $p \to^+_{\prec,B} r \neq 0$ then $\mathsf{lm}_\prec(r) \prec \mathsf{lm}_\prec(p)$.

Since monomial orderings are well-orderings, the previous result implies that a sequence of reductions cannot continue ad infinitum and must eventually terminate with the 0 polynomial. In this case, it follows that $p \in \langle B\rangle_{\vec{x}}$, hence reduction provides a test for membership in an ideal:

Proposition 1. If $p \rightarrow^*_{\prec,B} 0$ then $p \in \langle B \rangle_{\vec{x}}$.

In fact, the sequence of reductions itself be used to concretely demonstrate membership of $\langle B \rangle_{\vec{x}}$, as illustrated in the following example:

Example 7. With the setup from Example 6, it holds that $p \rightarrow^*_{\prec,B} 0$, thus from the previous result $p \in \langle B \rangle_{\vec{x}}$. To demonstrate this concretely, recall $p \rightarrow_{\prec,p_1} r_1 \rightarrow_{\prec,p_3} r_2 \rightarrow_{\prec,p_4} r_3 = 0$, where $p = ap_1 + r_1$, $r_1 = 1p_3 + r_2$, $r_2 = 5p_4 + r_3$ and $r_3 = 0$. By chaining these equalities it follows $p = ap_1 + r_1 = ap_1 + 1p_3 + r_2 = ap_1 + 1p_3 + 5p_4 + r_3 = ap_1 + p_3 + 5p_4 \in \langle B \rangle_{\vec{x}}$.

The converse of this result does not generally hold, as demonstrated by the following example:

Example 8. Let $\vec{x} = \langle x, y \rangle$, $\prec = \prec_{\vec{x}}$ and $p \in \mathbb{Z}_{16}[\vec{x}]$ be defined $p = 4x$. Moreover, let $B = \{p_1, p_2\} \subseteq \mathbb{Z}_{16}[\vec{x}]$ where $p_1 = 2x^2y + 2x^2 + 6xy + x$ and $p_2 = 4y + 4$. Then, $p = 12p_1 + (10x^2 + 10x)p_2 \in \langle B \rangle_{\vec{x}}$, yet $p \not\rightarrow_{\prec,B}$.

Thus, reduction against an arbitrary basis B does not lead to a complete test for membership in $\langle B \rangle_{\vec{x}}$. Overcoming this limitation is a key motivation for the definition of Gröbner basis.

2.7 Gröbner bases

With reduction in place, the fundamental concept of Gröbner basis can be defined:

Definition 5. Let $B \subseteq \mathbb{Z}_m[\vec{x}]$ and $\prec$ a monomial ordering over $\vec{x}$. Then, $G \subseteq \langle B \rangle_{\vec{x}}$ is a Gröbner basis for $\langle B \rangle_{\vec{x}}$ with respect to $\prec$ if for all $p \in \langle B \rangle_{\vec{x}}$, if $p \neq 0$ then p is $\prec$-reducible by G.

Note that a Gröbner basis for $\langle B \rangle_{\vec{x}}$ is also a basis for $\langle B \rangle_{\vec{x}}$. As suggested above, Gröbner bases provide a compete test for membership in $\langle B \rangle_{\vec{x}}$, as asserted in the following result:

Lemma 2. If G is a Gröbner basis for $\langle B \rangle_{\vec{x}}$ with respect to $\prec$ then for all $p \in \langle B \rangle_{\vec{x}}$, $p \rightarrow^*_{\prec,G} 0$.

Example 9. Consider again the setup of Example 8. Then, $p \in \langle B \rangle_{\vec{x}}$ yet $p \not\to_{\prec,B}$, hence B is a not a Gröbner basis for $\langle B \rangle_{\vec{x}}$ with respect to $\prec$. It will be shown in Example 11 that if $p_3 = 6x$ and $p_4 = 3x$ then $G = \{p_1, p_2, p_3, p_4\}$ is a Gröbner basis for $\langle B \rangle_{\vec{x}}$ with respect to $\prec$. Note that p is $\prec$-reducible by $p_4 \in G$. Indeed, $p \to_{\prec,p_4} 0$, hence $p \to^*_{\prec,G} 0$, as predicted by the previous result.

Gröbner bases are not only useful for detecting membership of $\langle B \rangle_{\vec{x}}$; they also play a role in understanding zero sets of polynomials. Intuitively, if $B \subseteq \mathbb{Z}_m[\vec{x}]$ then $\vec{a} \in \mathbb{Z}_m^d$ is a zero of B if each polynomial $p \in B$ evaluates to 0 at $\vec{a}$, where evaluation is achieved by substituting each x_i for a_i in p and simplifying the result. This concept will be formalised later. For now, let $\gamma(B)$ denote the solution set of B. It is straightforward to show that $\gamma(B) = \gamma(\langle B \rangle_{\vec{x}})$, hence if $\langle B \rangle_{\vec{x}} = \langle B' \rangle_{\vec{x}}$ then $\gamma(B) = \gamma(B')$.

Now, let $B \subseteq \mathbb{Z}_m[\vec{x}]$ and suppose $\langle B \rangle_{\vec{x}}$ contains a non-zero constant polynomial c. Then, since c evaluates universally to $c \neq 0$, it follows $\gamma(\langle B \rangle_{\vec{x}}) = \emptyset$, hence $\gamma(B) = \emptyset$. But now, let G be a Gröbner basis for $\langle B \rangle_{\vec{x}}$. Then, since a constant polynomial is only reducible by a constant polynomial, it follows that G must contain a constant polynomial too. In particular, even though B may not itself contain a non-zero constant polynomial, G does, hence the fact that B has no solutions can be inferred directly from a Gröbner basis for $\langle B \rangle_{\vec{x}}$.

2.8 Buchberger's algorithm

Classically, Gröbner bases are computed with Buchberger's algorithm [8], which is defined in terms of S-polynomials:

Definition 6. Let $\prec$ be a monomial ordering over $\vec{x}$. The S-polynomial of $p_1, p_2 \in \mathbb{Z}_m[\vec{x}]$ with respect to $\prec$ is defined:

$$\mathsf{S}_\prec(p_1, p_2) = d_2 2^{k-k_1} \vec{x}^{\vec{\alpha}-\vec{\alpha}_1} p_1 - d_1 2^{k-k_2} \vec{x}^{\vec{\alpha}-\vec{\alpha}_2} p_2$$

where, if $p_i = 0$ then $k_i = \omega$, $d_i = 1$ and $\vec{\alpha}_i = \vec{0}$, else $2^{k_i} d_i$ is the rank decomposition of $\mathsf{lc}_\prec(p_i)$ and $\vec{x}^{\vec{\alpha}_i} = \mathsf{lm}_\prec(p_i)$, $k = \max(k_1, k_2)$ and $\vec{\alpha} = \max(\vec{\alpha_1}, \vec{\alpha_2})$.

function $\mathbf{gb}_{\prec}(B = \{p_1, \ldots, p_s\} \subseteq \mathbb{Z}_m[\vec{x}])$
begin
  $G :- B$
  $S :- \{(p_i, p_j) \mid 1 \leq i < j \leq s\} \cup \{(p_i, 0) \mid 1 \leq i \leq s\}$
  while $(S \neq \emptyset)$
    let $s = (f_1, f_2) \in S$
    $S :- S \setminus \{s\}$
    $p :- \mathsf{S}_{\prec}(f_1, f_2)$
    let $p \rightarrow^*_{\prec,G} r$ **where** $r \not\rightarrow_{\prec,G}$
    if $(r \neq 0)$
      $S :- S \cup \{(g, r) \mid g \in G\} \cup \{(r, 0)\}$
      $G :- G \cup \{r\}$
    end if
  end while
  return G
end

Figure 4: Gröbner basis algorithm over integers modulo 2^ω

Example 10. Let $\vec{x} = \langle x, y \rangle$, $\prec = \prec_{\vec{x}}$ and $p_1, p_2 \in \mathbb{Z}_{16}[\vec{x}]$ be defined $p_1 = 2x^2y + 2x^2 + 6xy + x$ and $p_2 = 4y + 4$. Then, $\mathsf{S}_{\prec}(p_1, p_2) = 2(2xy^2 + 6xy + 2y^2 + y) - y^2(4x + 4) = 12xy + 2y$ and $\mathsf{S}_{\prec}(p_1, 0) = 8(2xy^2 + 6xy + 2y^2 + y) - xy^2(0) = 8y$.

Note that $\mathsf{lt}_{\prec}(d_2 2^{k-k_1} \vec{x}^{\vec{\alpha}-\vec{\alpha}_1} p_1) = \mathsf{lt}_{\prec}(d_1 2^{k-k_2} \vec{x}^{\vec{\alpha}-\vec{\alpha}_2} p_2)$, hence $\mathsf{S}_{\prec}(p_1, p_2)$ leads to a cancellation of leading terms. In particular, the S-polynomial $\mathsf{S}_{\prec}(p_1, 0)$ eliminates the leading term of p_1, and possible other terms as well. This deviates from the classical case of fields, where only multiplying by 0 can eliminate a leading term. It is this addition of S-polynomials with 0 that gives Gröbner bases over $\mathbb{Z}_m$ their own unique flavour. S-polynomials then yield an effective criterion [5, Theorem 30] to determine if a given basis is a Gröbner basis.

Theorem 1 (Buchberger's criterion)**.** Let $\prec$ be a monomial ordering and $B = \{p_1, \ldots, p_s\} \subseteq \mathbb{Z}_m[\vec{x}]$. If $\mathsf{S}_{\prec}(p_i, p_j) \rightarrow^*_{\prec,B} 0$ for all $1 \leq i < j \leq s$ and $\mathsf{S}_{\prec}(p_i, 0) \rightarrow^*_{\prec,B} 0$ for all $1 \leq i \leq s$ then B is a Gröbner basis for $\langle B \rangle_{\vec{x}}$ with respect to $\prec$.

Buchberger's criterion justifies Buchberger's algorithm for constructing Gröbner bases. Fig. 4 presents a version of Buchberger's algorithm [5] for modulo integers, that takes $B \subseteq \mathbb{Z}_m[\vec{x}]$ and a monomial ordering $\prec$ over $\vec{x}$ as input and returns

a Gröbner basis for $\langle B\rangle_{\vec{x}}$ with respect to $\prec$. The algorithm maintains a basis G, initialised to B, and a set of S-polynomials S, initialised to the set of S-polynomials derived from elements in B. Intuitively, the algorithm attempts to verify that G is a Gröbner basis by reducing each S-polynomial pair in S against it. If some S-polynomial does not reduce, it yields a new element which is added to G, and generates further S-polynomials. Observe that the pair $(r, 0)$ is added to S to eliminate the leading term of r, which is likewise reflected in the way S is primed. The algorithm terminates when all S-polynomials for the current basis reduce to 0, at which point Buchberger's criterion applies to show the resulting system G is a Gröbner basis.

Example 11. Let $\vec{x} = \langle x, y\rangle$, $\prec = \prec_{\vec{x}}$ and $B = \{p_1, p_2\} \subseteq \mathbb{Z}_{16}[\vec{x}]$ where $p_1 = 2x^2y + 2x^2 + 6xy + x$ and $p_2 = 4y + 4$. The table in Fig. 5 summarises the execution of $\mathbf{gb}_{\prec}(B)$. The k-th row displays the values of G and S before the k-th iteration of the main loop, as well as the reduction $p \to^*_{\prec,G} r$ that occurs during that iteration. It follows from the last step that $\{p_1, p_2, p_3, p_4\}$ is a Gröbner basis for $\langle B\rangle_{\vec{x}}$ with respect to $\prec$.

Example 12. The previous example shows that $G = \{2x^2y + 2x^2 + 6xy + x, 4y + 4, 6x, 3x\}$ is a Gröbner basis for $\langle B\rangle_{\vec{x}}$ with respect to $\prec = \prec_{\vec{x}}$, where $B = \{2x^2y + 2x^2 + 6xy + x, 4y + 4\}$. In fact, the subset $\{4y + 4, 3x\} \subseteq G$ is also a Gröbner basis for $\langle B\rangle_{\vec{x}}$. To see this, note that $\mathsf{lt}_{\prec}(2x^2y + 2x^2 + 6xy + x) = 2x^2y$ is divisible by $\mathsf{lt}_{\prec}(3x) = 3x$, since $2x^2y = (6xy)3x$. It follows if $p \in \langle B\rangle_{\vec{x}}$ is $\prec$-reducible by $2x^2y + 2x^2 + 6xy + x$ then it is also $\prec$-reducible by $3x$. In particular, $2x^2y + 2x^2 + 6xy + x$ can be removed from G without compromising its status as a Gröbner basis. The same observation applies to $6x$, hence $\{4y + 4, 3x\}$ is a Gröbner basis for $\langle B\rangle_{\vec{x}}$.

These observations motivate the following notion:

Definition 7. A Gröbner basis G with respect to $\prec$ is minimal if $p \not\to_{\prec, G\setminus\{p\}}$ for all $p \in G$.

As indicated above, minimality can be achieved by successively removing polynomials $p \in G$ that are $\prec$-reducible by $G \setminus \{p\}$, a process that can be carried out after Buchberger's algorithm terminates. In the following, we shall assume that this process is always carried out, hence only present minimal Gröbner bases.

G	S	$p \to^*_{\prec,G} r$
$\{p_1, p_2\}$	$\{(p_1,p_2), (p_1,0), (p_2,0)\}$	$\mathsf{S}_\prec(p_1,p_2) = 12xy + 2x \to^*_{\prec,G} 6x = p_3$
$\{p_1, p_2, p_3\}$	$\{(p_1,0), (p_2,0), (p_1,p_3), (p_2,p_3), (p_3,0)\}$	$\mathsf{S}_\prec(p_1,0) = 8x \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3\}$	$\{(p_2,0), (p_1,p_3), (p_2,p_3), (p_3,0)\}$	$\mathsf{S}_\prec(p_2,0) = 0 \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3\}$	$\{(p_1,p_3), (p_2,p_3), (p_3,0)\}$	$\mathsf{S}_\prec(p_1,p_3) = 6x^2 + 2xy + 3x \to^*_{\prec,G} 3x = p_4$
$\{p_1, p_2, p_3, p_4\}$	$\{(p_2,p_3), (p_3,0), (p_1,p_4), (p_2,p_4), (p_3,p_4), (0,p_4)\}$	$\mathsf{S}_\prec(p_2,p_3) = 12x \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3, p_4\}$	$\{(p_3,0), (p_1,p_4), (p_2,p_4), (p_3,p_4), (0,p_4)\}$	$\mathsf{S}_\prec(p_3,0) = 0 \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3, p_4\}$	$\{(p_1,p_4), (p_2,p_4), (p_3,p_4), (p_4,0)\}$	$\mathsf{S}_\prec(p_1,p_4) = 6x^2 + 2xy + 3x \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3, p_4\}$	$\{(p_2,p_4), (p_3,p_4), (p_4,0)\}$	$\mathsf{S}_\prec(p_2,p_4) = 12x \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3, p_4\}$	$\{(p_3,p_4), (p_4,0)\}$	$\mathsf{S}_\prec(p_3,p_4) = 0 \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3, p_4\}$	$\{(p_4,0)\}$	$\mathsf{S}_\prec(p_4,0) = 0 \to^*_{\prec,G} 0$
$\{p_1, p_2, p_3, p_4\}$	$\emptyset$	$-$

Figure 5: Execution of $\mathbf{gb}_\prec(B)$

2.9 Null polynomials

Null polynomials are polynomials that evaluate identically to 0, hence are universally valid:

Definition 8. Let $n \in \mathbb{Z}_m[\vec{x}]$. Then, n is a null polynomial iff $\gamma_{\vec{x}}(n) = \mathbb{Z}_m^d$.

The set of null polynomials in $\mathbb{Z}_m[\vec{x}]$ will be denoted $\mathrm{Null}_m[\vec{x}]$. Null polynomials play a fundamental role in the development of MPAD: they constitute the domain element $\top$ of MPAD and arise in computing closure. In this section, it will be demonstrated how a finite basis for null polynomials can be constructed. To build towards this development, consider:

Example 13. Let $\vec{x} = \langle x, y\rangle$ and $n_1, n_2 \in \mathbb{Z}_{16}[\vec{x}]$ be defined $n_1 = 2x^4 + 4x^3 + 6x^2 + 4x$ and $n_2 = 4x^2y^2 + 12x^2y + 12xy^2 + 4xy$. Note that n_1 and n_2 factor as

$$n_1 = 2x(x-1)(x-2)(x-3) \quad \text{and} \quad n_2 = 4x(x-1)y(y-1)$$

Now, let $\vec{a} = \langle a, b\rangle \in \mathbb{Z}_{16}^2$. First note $a(a-1)(a-2)(a-3)$ is a product of 4 consecutive numbers, hence must be divisible by 8. It follows $[\![n_1]\!]_{\vec{x}}(\vec{a}) = 2a(a-1)(a-2)(a-3)$ must be divisible by $2 \cdot 8 = 16$, hence $[\![n_1]\!]_{\vec{x}}(\vec{a}) = 0$. Similarly, the products $a(a-1)$ and $b(b-1)$ are formed from two consecutive numbers, hence both are divisible by 2. It follows $[\![n_1]\!]_{\vec{x}}(\vec{a}) = 4a(a-1)b(b-1)$ must be divisible by $4 \cdot 2 \cdot 2 = 16$, hence $[\![n_2]\!]_{\vec{x}}(\vec{a}) = 0$. Therefore $\{n_1, n_2\} \subseteq \mathrm{Null}_{16}[\vec{x}]$.

The previous example suggests a way to systematically construct null polynomials, formalised in the following definition and result:

Definition 9. For a term $c\vec{x}^{\vec{\alpha}}$, the factor polynomial $\eta_{c\vec{x}^{\vec{\alpha}}}$ is defined:

$$\eta_{c\vec{x}^{\vec{\alpha}}} = c\prod_{i=1}^{d}\prod_{j=0}^{\alpha_i - 1}(x_i - j)$$

Example 14. In $\mathbb{Z}_{16}[\vec{x}]$ where $\vec{x} = \langle x, y\rangle$,

$$\eta_{3x^2y^3} = 3x(x-1)y(y-1)(y-2) = 3x^2y^3 + 7x^2y^2 + 6x^2y + 13xy^3 + 9xy^2 + 10xy$$

Note that $3x^2y^3$ is a term of $\eta_{3x^2y^3}$. Moreover, each monomial $\vec{x}^{\vec{\alpha}}$ in $\eta_{3x^2y^3}$ takes the form $x^{k_1}y^{k_2}$ where $k_1 \leq 2$ and $k_2 \leq 3$, hence divides x^2y^3. This division is strict unless $\vec{x}^{\vec{\alpha}} = x^2y^3$.

As the previous example suggests, if $\vec{x}^{\vec{\beta}}$ is a monomial occurring in $\eta_{c\vec{x}^{\vec{\alpha}}}$ and $\vec{x}^{\vec{\beta}} \neq \vec{x}^{\vec{\alpha}}$ then $\vec{x}^{\vec{\beta}}$ strictly divides $\vec{x}^{\vec{\alpha}}$. Thus, letting $\vec{\delta} = \vec{\alpha} - \vec{\beta} > \vec{0}$, it follows from the defining properties of a monomial ordering $\prec$ that $1 \prec \vec{x}^{\vec{\delta}}$, hence $\vec{x}^{\vec{\beta}} = 1\vec{x}^{\vec{\beta}} \prec \vec{x}^{\vec{\delta}}\vec{x}^{\vec{\beta}} = \vec{x}^{\vec{\alpha}}$. In particular, it follows $\mathsf{lt}_{\prec}(\eta_{c\vec{x}^{\vec{\alpha}}}) = c\vec{x}^{\vec{\alpha}}$ independently of $\prec$. The following result, which determines conditions on c and $\vec{\alpha}$ to ensure a factor polynomial is null, is a straightforward consequence of [35, Lemma 3.1]:

Proposition 2. If $2^{\omega - \mathsf{rank}_\omega(c)} \mid \alpha_1! \cdots \alpha_d!$ in $\mathbb{N}$ then $\eta_{c\vec{x}^{\vec{\alpha}}} \in \mathrm{Null}_m[\vec{x}]$.

Example 15. Continuing with Example 13, recall $n_1 = \eta_{2x^4}$ and $n_2 = \eta_{4x^2y^2}$. Since $2^{4-\mathsf{rank}_4(2)} = 2^{4-1} = 8 \mid 24 = 4!\,0!$ and $2^{4-\mathsf{rank}_4(2)} = 2^{4-2} = 4 \mid 4 = 2!\,2!$, Proposition 2 provides an alternative justification as to why n_1 and n_2 are both null. Observe too that $\mathsf{lt}_\prec(\eta_{2x^4}) = \mathsf{lt}_\prec(n_1) = 2x^4$ and $\mathsf{lt}_\prec(\eta_{4x^2y^2}) = \mathsf{lt}_\prec(n_2) = 4x^2y^2$.

The following result presents a partial converse to Proposition 2, demonstrating that a null polynomial is always reducible by a null factor polynomial. Restated, this demonstrates the polynomials $\eta_{c\vec{x}^{\vec{\alpha}}}$ form a Gröbner basis for $\mathrm{Null}_m[\vec{x}]$, as formalised in the corollary:

Proposition 3. Let $\prec$ be a monomial ordering over $\vec{x}$ and $p \in \mathrm{Null}_m[\vec{x}]$ with $\mathsf{lt}_\prec(p) = c\vec{x}^{\vec{\alpha}}$. Then, $2^{\omega-\mathsf{rank}_\omega(c)} \mid \alpha_1!\cdots\alpha_d!$. In particular, p is $\prec$-reducible by $\eta_{c\vec{x}^{\vec{\alpha}}} \in \mathrm{Null}_m[\vec{x}]$.

Corollary 1. The set $\{\, \eta_{c\vec{x}^{\vec{\alpha}}} \mid 2^{\omega-\mathsf{rank}_\omega(c)} \mid \alpha_1!\cdots\alpha_d! \,\}$ is a Gröbner basis for $\mathrm{Null}_m[\vec{x}]$ with respect to any monomial ordering $\prec$.

These results, which follow from [35, Lemma 3.2], demonstrate the existence of a basis for $\mathrm{Null}_m[\vec{x}]$, but not a finite one. Note, however, the set of factor polynomials $\eta_{c\vec{x}^{\vec{\alpha}}}$ is not a minimal Gröbner basis. To see this, suppose $\eta_{c\vec{x}^{\alpha}}$ is null, c divides c' and $\vec{\alpha} \le \vec{\alpha}'$. Then, $\mathsf{rank}_\omega(c) \le \mathsf{rank}_\omega(c')$ hence $2^{\omega-\mathsf{rank}_\omega(c')}$ divides $2^{\omega-\mathsf{rank}_\omega(c)}$, which divides $\alpha_1!\cdots\alpha_d!$, which divides $\alpha'_1!\cdots\alpha'_d!$. Thus, Proposition 2 implies $\eta_{c'\vec{x}^{\vec{\alpha}'}}$ is also null. But also, since $\mathsf{lt}_\prec(\eta_{c\vec{x}^{\vec{\alpha}}}) = c\vec{x}^{\vec{\alpha}}$ and $\mathsf{lt}_\prec(\eta_{c'\vec{x}^{\vec{\alpha}'}}) = c'\vec{x}^{\vec{\alpha}'}$ it follows $\eta_{c'\vec{x}^{\vec{\alpha}'}}$ is $\prec$-reducible by $\eta_{c\vec{x}^{\vec{\alpha}}}$. In particular, the set of null factor polynomials excluding $\eta_{c'\vec{x}^{\vec{\alpha}'}}$ is still a Gröbner basis for $\mathrm{Null}_m[\vec{x}]$. This motivates defining a criterion for a null factor polynomial to be irreducible by another null factor polynomial. To that end:

Definition 10. A null factor polynomial $\eta_{2^k\vec{x}^{\vec{\alpha}}}$ is principle if all the following hold:

- $0 \le k < \omega$,
- If $k > 0$ then $2^{\omega-(k-1)} \nmid \alpha_1!\cdots\alpha_d!$,
- If $\alpha_\ell > 0$ then $2^{\omega-k} \nmid \alpha_1!\cdots(\alpha_\ell - 1)!\cdots\alpha_d!$

Example 16. Returning to Example 15, $n_1 = \eta_{2x^4} = \eta_{2^{k_1}\vec{x}^{\vec{\alpha}_1}}$ where $k_1 = 1 < \omega$ and $\vec{\alpha}_1 = \langle 4, 0\rangle$. Now,

- $2^{\omega-(k_1-1)} = 2^{4-0} = 16 \nmid 24 = 4!\,0!$, hence the second property holds,
- $2^{\omega-k_1} = 2^{4-1} = 8 \nmid 6 = (4-1)!\,0!$, hence the third property holds for $\ell = 1$ (it holds vacuously for $\ell = 2$).

It follows n_1 is a principle null factor polynomial. Similarly, $n_2 = \eta_{4x^2y^2} = \eta_{2^{k_2}\vec{x}^{\vec{\alpha}_2}}$ where $k_2 = 2 < \omega$ and $\vec{\alpha}_2 = \langle 2, 2\rangle$. Now,

- $2^{\omega-(k_2-1)} = 2^{4-1} = 8 \nmid 4 = 2!\,2!$, hence the second property holds,
- $2^{\omega-k_2} = 2^{4-2} = 4 \nmid 2 = (2-1)!\,2! = 2!\,(2-1)!$, hence the third property holds.

It follows n_2 is also a principle null factor polynomial.

Let $B_{\mathsf{Null}_m[\vec{x}]}$ denote the set of all principle null factor polynomials. Note that this set is finite. To see this, suppose $\alpha_\ell > \omega + 1$ for some ℓ. Then, for any $0 \leq k < \omega$, it follows $2^{\omega-k}$ divides 2^ω, which divides $(\omega+1)!$, which divides $(\alpha_\ell - 1)!$, which divides $\alpha_1! \cdots (\alpha_\ell - 1)! \cdots \alpha_d!$. Thus, $2^{\omega-k} \mid \alpha_1! \cdots (\alpha_\ell - 1)! \cdots \alpha_d!$, hence the third property above cannot hold. In particular, if $\eta_{c\vec{x}^{\vec{\alpha}}}$ is principle then $\vec{\alpha} \leq \langle \omega+1, \ldots, \omega+1\rangle$ pointwise, which implies $B_{\mathsf{Null}_m[\vec{x}]}$ is finite. . In fact, from [35, Theorem 3.3] it follows the criteria described in Definition 10 guarantee that no principle null factor polynomial is $\prec$-reducible by another principle null factor polynomial, thus establishing:

Theorem 2. $B_{\mathsf{Null}_m[\vec{x}]}$ is a minimal Gröbner basis for $\mathrm{Null}_m[\vec{x}]$.

In particular, it follows $B_{\mathsf{Null}_m[\vec{x}]}$ is a finite basis for null polynomials. Indeed, this justifies that the basis B defined in Example 25 is a basis for $\mathrm{Null}_{16}[\vec{x}]$, since it is precisely $B_{\mathsf{Null}_{16}[\vec{x}]}$.

2.10 Substitution

Polynomial substitution will play a key role in the later development of closure:

Definition 11. Let $p, q \in \mathbb{Z}_m[\vec{x}]$. Then $p[x_i \mapsto q]$ denotes the polynomial constructed by substituting q for every instance of x_i in p.

Example 17. Let $\vec{x} = \langle x, y \rangle$ and $p = xy + 2x + 3 \in \mathbb{Z}_{16}[\vec{x}]$. Then,

- $p[x \mapsto y + 1] = (y + 1)y + 2(y + 1) + 3 = y^2 + 3y + 5$.
- $p[x \mapsto x + 1] = (x + 1)y + 2(x + 1) + 3 = xy + 2x + y + 5$.

Substitution and reduction are closely linked. The following result relies on reducing a polynomial p by $\{x_1 - W_1, \ldots, x_d - W_d\}$ to eliminate all occurrences of the variables x_i. The example that follows illustrates the idea behind the lemma.

Lemma 3. Let $p \in \mathbb{Z}_m[\vec{x} : \vec{w}]$ and $\vec{W} \in \mathbb{Z}_m[\vec{w}]^d$. Then, $p = q_1(x_1 - W_1) + \cdots + q_d(x_d - W_d) + r$ for some $q_\ell \in \mathbb{Z}_m[\vec{x} : \vec{w}]$ and $r \in \mathbb{Z}_m[\vec{w}]$.

Example 18. Let $p = x^2 + 3y \in \mathbb{Z}_{16}[x, y]$ and $\vec{W} \in \mathbb{Z}_{16}[w_1, w_2]$. Then,

$$\begin{aligned} p &= x^2 + 3y \\ &= x(x - W_1) + xW_1 + 3y \\ &= (x + W_1)(x - W_1) + 3y + W_1^2 \\ &= (x + W_1)(x - W_1) + 3(y - W_2) + W_1^2 + 3W_2 \\ &= q_1(x - W_1) + q_2(y - W_2) + r \end{aligned}$$

where $q_1 = x + W_1$, $q_2 = 3$ and $r = W_1^2 + 3W_2$. Note that $r = p[x \mapsto W_1, y \mapsto W_2]$.

2.11 Related work

Gröbner bases Originally introduced for rings of polynomials with coefficients are drawn from a field [52] (where non-zero elements have multiplicative inverses) the theory of Gröbner bases has subsequently been extended to more general rings [2, 45, 60]. Magma [4] provides a variant of Buchberger for modulo arithmetic, though details of the implementation are proprietary. Although the primary focus of [5] is on Boolean Gröbner bases, for which $\omega = 1$ and $\mathbb{Z}_m$ is actually a field, this study reports a Buchberger algorithm which adds the annihilator of a polynomial. This algorithm also use so-called field equations that express invariants $x_i = x_i^2$ for all $x_i \in \mathbb{Z}_2$, which are a particular instance of null polynomials. An alternative approach is to add the equation $2^\omega = 0$ to a system of polynomial equations to simulate modulo behaviour with arbitrary-precision integers [45]. In recent

years, signature-based approaches [23, 28], which have improved the efficiency of computing Gröbner bases for polynomials over fields, have been extended to more general settings [24, 25, 30]. Gröbner base engines have also recently shown promise for realising word-level propagation [72].

Null polynomials Null polynomials have a long history [47, 75], inspired by the desire to detect semantic equivalence of two polynomials mod m (where m is not necessarily 2^ω). Unary null polynomials were studied almost a century ago [47], then rediscovered fifty years later [75], where they were presented in a more accessible way. These results were extended to multi-variate polynomials, first for the case of arbitrary m [43], then, bizarrely, for $m = 2^\omega$ [73] (although [73] overtook [43] in publication order). More recently, these results were distilled [35] to construct a minimal Gröbner basis for the set of null polynomials.

2.12 Concluding Discussion

Buchberger's classic S-polynomial based criterion for characterising a Gröbner base can be extended to modular arithmetic by scaling each polynomial in the basis by a power of two to eliminate its leading term. This extension amounts to relaxing the form of S-polynomials from $\mathsf{S}_\prec(p_1, p_2)$ where $p_1 \neq 0$ and $p_2 \neq 0$ to allow $\mathsf{S}_\prec(p_1, 0)$ where $p_1 \neq 0$, since the latter form eliminates the leading term of p_1. This extension of Buchberger's criterion, leads to a natural extension of Buchberger's algorithm. Less endearing is the handling of null polynomials, which are polynomials that are universally valid. Although vacuous semantically, they still need to considered for reduction, hence must be enumerated. To this end, a way to finitely enumerate the set of nulls is presented for any given number of variable and bit-width. The enumeration is tight in that it yields a Gröbner basis which is minimal.

Chapter 3

SMT for Modular Polynomials

3.1 Introduction

Some of the most influential algorithms in algebraic computation, such as Buchberger's algorithm [8] and Collin's Cylindrical Algebraic Decomposition algorithm [12], were invented long before the advent of SMT. SMT itself has evolved from its origins in SAT into a largely independently branch of symbolic computation. Yet the potential of cross-fertilising one branch with the other has been repeatedly observed [1, 6, 19], and a new class of SMT solvers is beginning to emerge that apply both algebraic and satisfiability techniques in tandem [39, 44, 76]. The problem, however, is that algebraic algorithms do not readily fit into the standard SMT architecture [68] because they are not normally incremental or backtrackable, and rarely support learning [1].

For application to software verification, the SMT background theory of bit-vectors is of central interest. Solvers for bit-vectors conventionally translate bit-vector constraints into propositional formulae by replacing constraints with propositional circuits that realise them, a technique evocatively called bit-blasting. However, particularly for constraints involving multiplication, the resulting formulae can be prohibitively large. Moreover, bit-blasting foregoes the advantages afforded by reasoning at the level of bit-vectors [3, 36].

In this chapter we present a new architecture for solving systems of polynomial equalities over bit-vectors. Rather than converting to SAT and bit-blasting, the

method sets bits in order of least significance through the addition of certain polynomials to the system. Computing a Gröbner basis [5] for the resulting system effects a kind of high-level propagation, which we have called bit-sequence propagation, in which the values of other bits can be automatically inferred. Furthermore, we show how the procedure can be carried out with symbolic truth values without giving up bit-sequence propagation, thus unifying Gröbner basis calculations that would otherwise be separate.

Once all bits are assigned truth values (symbolic or otherwise), the resulting Gröbner basis prescribes an assignment to the bit-vectors which is a function of the symbolic truth values. The remaining polynomials in the basis relate the symbolic truth values and correspond to non-linear pseudo-boolean constraints modulo a power of two. These constraints can be solved either by translation into classical linear pseudo-boolean constraints (without a modulo) or else by encoding them as propositional formulae, for which a novel translation process is described. Either way, the algebraic Gröbner basis computation is encapsulated in the phase that emits the pseudo-boolean constraints, hence the Gröbner basis engine [5] does not need to be backtrackable, incremental or support learning. Overall, the architecture provides a principled method for compiling high-level polynomials to low-level pseudo-boolean constraints.

In summary, this chapter makes the following contributions:

- We introduce bit-sequence propagation, in which a bit is set by adding a suitable constraint to the system and computing a Gröbner basis, demonstrating how it can lead to other bits being set automatically;
- We show how bit assignments can be handled symbolically in order to unify distinct Gröbner basis computations, eventually yielding a residue system of non-linear pseudo-boolean constraints;
- We show how the resulting pseudo-boolean systems can be solved by employing a novel rewrite procedure for converting non-linear modulo pseudo-booleans to propositional formulae.

The chapter is structured as follows: Section 4.3 illustrates bit-sequence propagation through a concrete example. The pseudo-boolean encoding are detailed

in Section 3.3. Experimental results are given in Section 3.4, Section 3.5 surveys related work and Section 3.6 concludes.

3.2 Bit-sequence Propagation in a Nutshell

Classically, that is for polynomials over algebraically closed fields, unsatisfiability can be decided by Hilbert's Nullstellensatz [18]. This equates unsatisfiability with the existence of a non-zero constant polynomial in a Gröbner basis for the polynomials. The concept of Gröbner basis is inextricably linked with that of an ideal [18]. The ideal for a given system (set) of polynomials is the least set closed under the addition of polynomials drawn from the set and multiplication of an arbitrary polynomial with a polynomial from the set; an ideal shares the same zeros as the system from which it is derived, but is not finite. A Gröbner basis is merely a finite representation of an ideal, convenient because, among other things, it enables satisfiability to be detected, at least over a field.

Unary bit-vectors constitute a field, but Nullstellensatz does not hold for bit-vectors with multiple bits. To see this, consider the polynomial equation $x^2+2=0$ where the arithmetic is 3-bit (modulo 8). Any solution x to this equation must be even. But, $0^2+2=4^2+2=2$ and $2^2+2=6^2+2=6$. Hence $x^2+2=0$ has no solutions, yet the Gröbner basis $\{x^2+2\}$ does not contain a non-zero constant polynomial. Moreover, even for a Gröbner basis of a satisfiable system, such as $\{x^2+4\}$, the solutions to the system cannot be immediately read off from the basis. The force of these observations is that Gröbner bases need to be augmented with search to test satisfiability and discover models. To illustrate this we consider a more complicated system:

$$B = \left\{ \begin{array}{rr} y^2+120x^2+123x+48=0, & yx+65x^2+50x+32=0, \\ 2y+63x^2+59x+128=0, & x^3+135x^2+100x+64=0, \\ 64x^2+192x=0 & \end{array} \right\}$$

where $x, y \in \mathbb{Z}_{256}$. Henceforth we follow convention and omit $=0$ from systems.

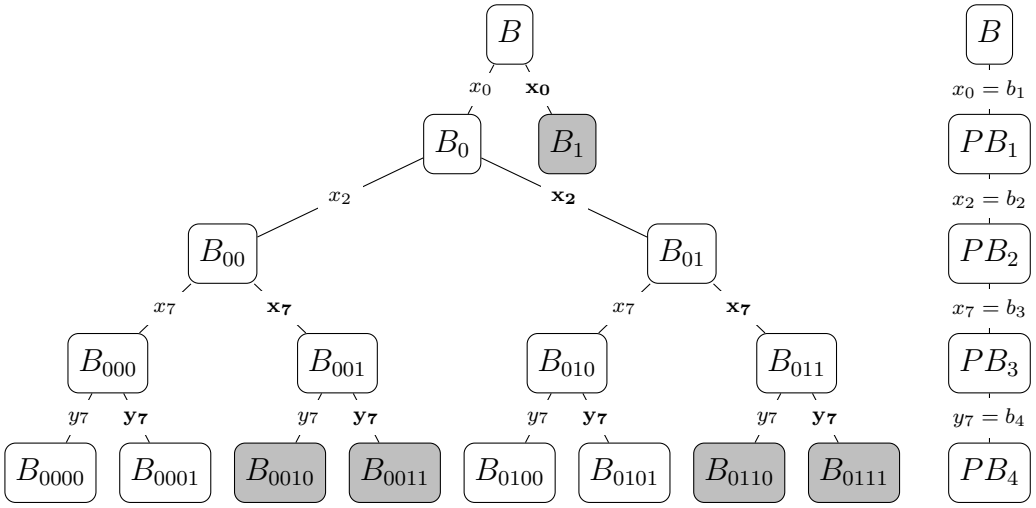


Figure 6: Bit-assignments and word-level propagation: 0/1 bits and symbolic bits

3.2.1 Solving using 0/1 truth values

Since $\mathbb{Z}_{256}$ is finite, this system can be solved by viewing the problem [58] as a finite domain constraint satisfaction problem. In this setting, each bit-vector is associated with a set of values that is progressively pruned using word-level constraint propagation rules. The search tree in the left-hand side of Fig 6 illustrates how pruning is achieved by setting and inferring bits in the order of least-significance, starting with the bits of x then those of y. On a left branch of the tree one bit, x_i or y_j, is set to 0; on a right branch the bit is set to 1 (indicated in bold). Each node is labelled with a Gröbner basis that encodes the impact of setting a bit on all other bits. Gröbner bases are indexed by their position in the tree. Grey nodes correspond to the solutions of B.

Computing B_0 Setting the least significant bit of x, bit 0, to 0 can be achieved by imposing $x = 2w$ for some otherwise unconstrained variable w. Hence, we add $2w - x$ to B and compute a Gröbner basis with respect to the lexicographical ordering on variables $w \succ y \succ x$, yielding:

$$\left\{ \begin{array}{llll} wx + 86x + 96, & 2w + 255x, & y^2 + 219x + 48, & \\ yx + 134x + 96, & 2y + 231x + 64, & x^2 + 172x + 192, & 64x \end{array} \right\}$$

To eliminate dependence on w, polynomials involving w are removed, giving:

$$B_0 = \left\{ \begin{array}{lll} y^2 + 219x + 48, & yx + 134x + 96, & 2y + 231x + 64, \\ x^2 + 172x + 192, & 64x & \end{array} \right\}$$

Note that B_0 contains $64x$ (representing $64x = 0$) which indicates that x is a multiple of 4. Thus bit 1 is also clear, although we did not actively impose it.

Now, observe the constraint $64x = 0$ implies $0 = 2^6(x - 0)$ hence $x - 0 = 2^2 w'$ for some w'. To clear the next bit, put $w' = 2w$ which gives $x - 0 = 8w$ yielding the polynomial $8w - x$. Otherwise, to set the next bit put $w' = 2w + 1$ giving the polynomial $8w - x + 4$.

Computing B_{00} Augmenting B_0 with $8w - x$, calculating a Gröbner basis, and then eliminating w gives:

$$B_{00} = \left\{ \begin{array}{lll} y^2 + 219x + 48, & yx + 128, & 2y + 231x + 64, \\ x^2, & 2x + 160 & \end{array} \right\}$$

Since B_{00} includes $2x + 160$ (representing $2x + 160 = 0$) it follows that only bit 7 is undetermined. To constrain it, observe $0 = 2(x - 48)$ thus $x - 48 = 2^7 w'$ for some w'. Putting $w' = 2w$ gives $x - 48 = 256w = 0$ hence the polynomial $x - 48$. Conversely, putting $w' = 2w + 1$ gives $x - 48 = 256w + 128 = 128$ thus $x - 176$.

Computing B_{000} and B_{001} Adding $x - 48$ and $x - 176$ to B_{00}, computing a Gröbner basis, and eliminating w (a vacuous step), respectively yields:

$$B_{000} = \left\{ y^2 + 64, \quad 2y + 144, \quad x + 208 \right\} \; B_{001} = \left\{ y^2 + 192, \quad 2y + 16, \quad x + 80 \right\}$$

Both systems contain a single constraint on x which uniquely determines its value, hence we move attention to y. Both B_{000} and B_{001} contain equations with leading terms $2y$ and thus only bit 7 of y must be constrained. Following the same

procedure as before, we obtain:

$$B_{0000} = \begin{Bmatrix} y + 200, \\ x + 208, \\ 128 \end{Bmatrix} B_{0001} = \begin{Bmatrix} y + 72, \\ x + 208, \\ 128 \end{Bmatrix} B_{0010} = \begin{Bmatrix} y + 136, \\ x + 80 \end{Bmatrix} B_{0011} = \begin{Bmatrix} y + 8, \\ x + 80 \end{Bmatrix}$$

These Gröbner bases all completely constrain x and y, hence are leaf nodes. Note that B_{0000} and B_{0001} contain the non-zero, constant polynomial 128, indicating unsatisfiability. Hence, only B_{0010} and B_{0011} actually yield solutions (highlighted in grey), namely $x \mapsto 176, y \mapsto 120$ and $x \mapsto 176, y \mapsto 248$ respectively.

Computing B_* The general principle is that if $2^k(x - \ell)$ is in the basis and ω is the bit width, then the linear polynomial $2^{\omega-k+1}w - x + \ell$ is added for some fresh w to set the next undermined bit to 0. Conversely, to set the next bit to 1, the polynomial $2^{\omega-k+1}w - x + 2^{\omega-k} + \ell$ is added. We name this tactic bit-sequence propagation. Using this tactic to flesh out the rest of the tree gives the following satisfiable bases (also marked in grey in the figure):

$$B_1 = \Big\{y + 183, x + 91\Big\} \quad B_{0110} = \Big\{y + 158, x + 92\Big\} \quad B_{0111} = \Big\{y + 30, x + 92\Big\}$$

yielding $x \mapsto 165, y \mapsto 73$, $x \mapsto 164, y \mapsto 98$ and $x \mapsto 164, y \mapsto 226$ respectively.

3.2.2 Solving using symbolic truth values

To reduce the total number of Gröbner basis calculations, we observe that it is sufficient to work with symbolic bits. The right-hand side of Figure 6 illustrates how this reduces the number of bases calculated to 4, albeit at the cost of carrying symbolic bits in the basis. Bit-sequence propagation generalises via the single rule: if $2^k(x - \ell)$ is in the basis and ω is the bit width, then the polynomials $2^{\omega-k+1}w - x + 2^{\omega-k}b + \ell$ and $b^2 - b$ are added to the basis. This sets the next undermined bit to the symbolic value b; the polynomial $b^2 - b$ merely asserts that

each symbolic b can only be 0 or 1. This construction gives:

$$PB_1 = \left\{ \begin{array}{rrr} y^2 + 219x + 216b_1 + 48, & yx + 6x + 181b_1 + 96, & yb_1 + 183b_1, \\ 2y + 103x + 203b_1 + 64, & x^2 + 44x + 139b_1 + 192, & xb_1 + 91b_1, \\ 64x + 192b_1, & b_1^2 + 255b_1 & \end{array} \right\}$$

$$\vdots$$

$$PB_4 = \left\{ \begin{array}{rrrr} y + 128b_4 + 192b_3 + 214b_2 + 153b_1 + 200, & & x + 12b_2 + 255b_1 + 80, & \\ b_4^2 + 255b_4, & 128b_4b_1 + 128b_1, & b_3^3 + 255b_3, & 64b_3b_1, \\ 128b_3 + 128b_2 + 128, & b_2^2 + 255b_2, & 2b_2b_1 + 254b_1, & b_1^2 + 255b_1 \end{array} \right\}$$

The final PB_4 expresses x and y as combinations of b_4, b_3, b_2 and b_1:

$$y \equiv_{256} -128b_4 - 192b_3 - 214b_2 - 153b_1 - 200 \qquad x \equiv_{256} -12b_2 - 255b_1 - 80$$

Observe that the remaining polynomials are non-linear pseudo-boolean constraints over b_4, b_3, b_2 and b_1 modulo 256. The polynomials $b_i^2 + 255b_i$, which assert that each b_i is binary, are subsequently ignored.

3.2.3 Solving using SAT

These pseudo-booleans can be simplified by observing that when all coefficients in the constraint are divisible by a power of 2 then the modulo can be lowered:

$$\begin{array}{rcl} 128b_4b_1 + 128b_1 \equiv_{256} 0 & \iff & b_4b_1 + b_1 \equiv_2 0 \\ 64b_3b_1 \equiv_{256} 0 & \iff & b_3b_1 \equiv_4 0 \\ 128b_3 + 128b_2 + 128 \equiv_{256} 0 & \iff & b_3 + b_2 + 1 \equiv_2 0 \\ 2b_2b_1 + 254b_1 \equiv_{256} 0 & \iff & b_2b_1 + 127b_1 \equiv_{128} 0 \end{array}$$

Since the reduced versions of the first and third constaints are modulo 2 they can be mapped immediately to the propositional formulae:

$$\begin{array}{rcl} b_4b_1 + b_1 \equiv_2 0 & \iff & (b_4 \wedge b_1) \oplus b_1 \\ b_3 + b_2 + 1 \equiv_2 0 & \iff & \neg(b_3 \oplus b_2) \end{array}$$

where the negation is introduced because of the constant 1. The second and fourth

constraints cannot be handled so directly because the modulus is not 2. However, for the second, we can use the fact that the left-hand side is a single term to infer either b_3 or b_1 must be 0, yielding the formula $\neg b_3 \vee \neg b_1$. Finally, for the fourth constraint, we do a case split on b_2. Setting $b_2 = 0$ simplifies the constraint to $127b_1 \equiv_{128} 0$, from which $b_1 = 0$ is inferred. Conversely, setting $b_2 = 1$ simplifies the constraint to $128b_1 \equiv_{128} 0$ which is vacuous. Overall, we derive the formula $(\neg b_2 \wedge \neg b_1) \vee b_2$ for the fourth constraint. There are 5 truth assignments for the formula assembled from the above 4 sub-formulae, yielding 5 assignments to x and y that concur with those given previously.

The reasoning exemplified here has been distilled into a series of rules, presented in Section 3.3, for encoding non-linear modulo pseudo-booleans into SAT. An alternative approach finds the values for b_4, b_3, b_2 and b_1 using a cutting-plane pseudo-boolean solver [53] alongside a modulo elimination transformation [29, Section 3]. Regardless of the particular method employed to solve this system, observe that search has been isolated in the SAT/pseudo-boolean solver; the Gröbner bases are calculated in an entirely deterministic fashion.

3.3 Encoding pseudo-boolean constraints

Figure 7 presents rules for translating a polynomial in the form $\vec{c} \cdot \vec{X} \equiv_{2^r} d$ to a propositional formula such that $\vec{c} \in \mathbb{Z}_m^\ell$, $d \in \mathbb{Z}_m$ and $\vec{X} \in \wp(\cup\vec{x})^\ell$, where $\vec{x}$, recall, is the vector of variables and ℓ is the arity of the vectors $\vec{c}$ and $\vec{X}$. This form of constraint, although restrictive, is sufficient to express the pseudo-booleans which arise in the final Gröbner basis, as illustrated below:

Example 19. Returning to PB_4 of Section 4.3 the polynomials $128b_4b_1 + 128b_1$ and $128b_3 + 128b_2 + 128$ can be written as $\langle 128, 128\rangle \cdot \langle \{b_1, b_4\}, \{b_1\}\rangle \equiv_{256} 0$ and $\langle 128, 128\rangle \cdot \langle \{b_3\}, \{b_2\}\rangle \equiv_{256} 128$ since $128 = -128 \pmod{256}$.

The rules of Figure 7 collectively reduce the problem of encoding a constraint to that of encoding one or more strictly simpler constraints. For brevity, we limit the commentary to the more subtle rules. The false rule handles constraints which are unsatisfiable because the coefficients $\vec{c}$ are all even and d is odd. The scale rule reduces the encoding problem to that for an equi-satisfiable constraint obtained

$$\text{true}\ \frac{}{\epsilon \cdot \epsilon \equiv_{2^r} 0 \ \rightarrow\ \mathsf{true}} \qquad \text{false}\ \frac{\forall c_i \in \vec{c}.\ \mathsf{rank}(c_i) > 0 \quad \mathsf{rank}(d) = 0}{\vec{c} \cdot \vec{X} \equiv_{2^r} d \ \rightarrow\ \mathsf{false}}$$

$$\text{xor}\ \frac{}{\vec{1} \cdot \vec{X} \equiv_2 1 \ \rightarrow\ \bigoplus_{X \in \vec{X}} (\bigwedge X)} \qquad \text{iff}\ \frac{\vec{c} \cdot \vec{X} \equiv_2 1 \ \rightarrow\ f}{\vec{c} \cdot \vec{X} \equiv_2 0 \ \rightarrow\ \neg f}$$

$$\text{scale}\ \frac{\vec{c} \cdot \vec{X} \equiv_{2^r} d \ \rightarrow\ f}{(2^s \vec{c}) \cdot \vec{X} \equiv_{2^{r+s}} (2^s d) \ \rightarrow\ f} \qquad \text{set}\ \frac{\mathsf{rank}(d) = 0 \quad \exists! c_i \in \vec{c}.\ \mathsf{rank}(c_i) = 0 \quad (\vec{c} \cdot \vec{X} \equiv_{2^r} d)[x \mapsto 1 \mid x \in X_i] \ \rightarrow\ f}{\vec{c} \cdot \vec{X} \equiv_{2^r} d \ \rightarrow\ (\bigwedge X_i) \wedge f}$$

$$\text{clear}\ \frac{\mathsf{rank}(d) > 0 \quad \exists! c_i \in \vec{c}.\ \mathsf{rank}(c_i) = 0 \quad \forall x \in X_i.\ (\vec{c} \cdot \vec{X} \equiv_{2^r} d)[x \mapsto 0] \ \rightarrow\ f_x}{\vec{c} \cdot \vec{X} \equiv_{2^r} d \ \rightarrow\ \bigvee_{x \in X_i} (\neg x \wedge f_x)}$$

$$\text{split}\ \frac{x \in \bigcup \vec{X} \quad (\vec{c} \cdot \vec{X} \equiv_{2^r} d)[x \mapsto 0] \ \rightarrow\ f_0 \quad (\vec{c} \cdot \vec{X} \equiv_{2^r} d)[x \to 1] \ \rightarrow\ f_1}{\vec{c} \cdot \vec{X} \equiv_{2^r} d \ \rightarrow\ (\neg x \wedge f_0) \vee (x \wedge f_1)}$$

Figure 7: Reduction rules for pseudo-boolean polynomials modulo 2^r

by dividing the modulo, coefficients and constant by a power of 2. The set rule handles constraints where d is odd and there is a unique term $c_i X_i$ for which c_i is odd. In this circumstance all the variables of X_i must be 1 for the constraint to be satisfiable. Conversely, clear deals with constraints for which d is even and there exists a unique $c_i X_i$ for which c_i is odd since then one variable of X_i must be 0 for satisfiability. When none of above are applicable, split is applied to reduce to encoding problem to that of two strictly smaller constraints.

3.4 Experimental results

Our aim is to apply high-level algebraic reasoning to systematically reduce polynomials to compact systems of low-level constraints. Our experimental work thus assesses how the complexity of the low-level constraints relate to that of the input polynomials. Although we provide timings for our Buchberger algorithm, which as far as we know is state-of-the-art, this is not our main concern. Indeed, fast algorithms for calculating Gröbner bases over fields have emerged in the last two

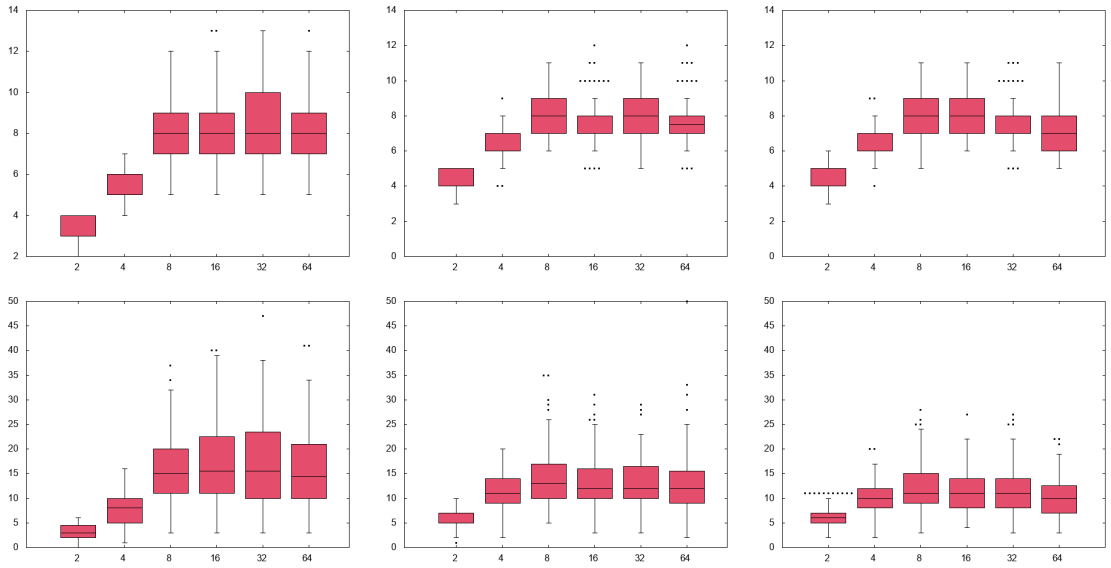


Figure 8: Top: number of symbolic variables (y) against ω (x) for $n = 2$, 3 and 4; Bottom: number of pseudo-booleans (y) against ω (x) for $n = 2$, 3 and 4

decades [27, 28], and similar performance gains seem achievable for modulo arithmetic. In light of this, our Buchberger algorithm is implemented in Scala 2.13.0 (compiled to JVM) using BigInt for complete generality. Experiments were performed on a 2.7GHz Intel i5 Macbook with 16Gbyte of SDRAM.

Datasets To exercise the symbolic method, polynomial systems were generated for different numbers of bit-vectors n and different bit-widths ω. For each $\omega \in \{2, 4, 8, 16, 32, 64\}$ and $n \in \{2, 3, 4\}$, 100 polynomial systems were constructed by randomly generating points in $\mathbb{Z}_{2^\omega}^n$ and deriving a system with these points as their zeros. First, each point was described as a system of n (linear) polynomials. Second, a single system was then formed with n points as its zeros through the introduction of $n-1$ fresh variables [2]. Third, the fresh variables were eliminated by calculating a Gröbner base to derive a basis constituting a single datapoint.

Symbolic variable and pseudo-boolean count Figure 8 presents box and whisker diagrams that summarise the numbers of symbolic variables and pseudo-booleans appearing in the derived pseudo-boolean systems. For each box and whisker, the lower and upper limits of the box indicate the first and third quartiles, the central line the median. The inter-quartile range (IQR) is the distance from the top to the bottom of the box. By convention the whiskers extend to 1.5 times the IQR above and below the median value; any point falling outside of this range is considered to be an outlier and is plotted as an individual point. Figure 8 was derived from datapoints generated from 6 random points. Similar trends are observed with fewer points and appear to be displayed for more points, but variable elimination impedes dataset generation and large-scale evaluation.

For both the number of symbolic variables and the number of pseudo-booleans, the medians level off after an initial increase and then appear to be relatively independent of ω. This surprising result suggests that algebraic methods have a role in reducing the complexity of polynomials for bit-vectors, which is sensitive to ω for bit-blasting. This implication is that the number of Gröbner base calculations also stablises with ω since this tallies with the number of symbolic variables. We also observe that the number of symbolic bits employed is typically only a fraction of the total number of bits occurring in the bit-vectors, hence setting a single symbolic bit is often sufficient to infer values for many other bits.

Pseudo-boolean versus multiplication count The upper row of figure 9 presents a fine-grained analysis of the number of pseudo-booleans, comparing this count to the number of bit-vector multiplications in the datasets. Multiplications are counted as follows: An occurrence of a monomial x^3yz, say, in polynomial system contributes $2 + 1 + 1$ to its multiplication count, irrespective of whether it occurs singly or multiply. The term $42x^3yz$ also contributes 4 to the count, so simple multiplications with constants are ignored. Addition is also not counted, the rationale being to compare the number of pseudo-booleans against the number of bit-vector multiplications which are not amenable to specialisation in a reasonably smart encoding. The x-axis of the histograms of Figure 9 divides the different ratios into bins, the first column giving the number of datasets for which the ratio falls within $[0, 0.25)$. As n increases the ratios bunch more tightly around the bin

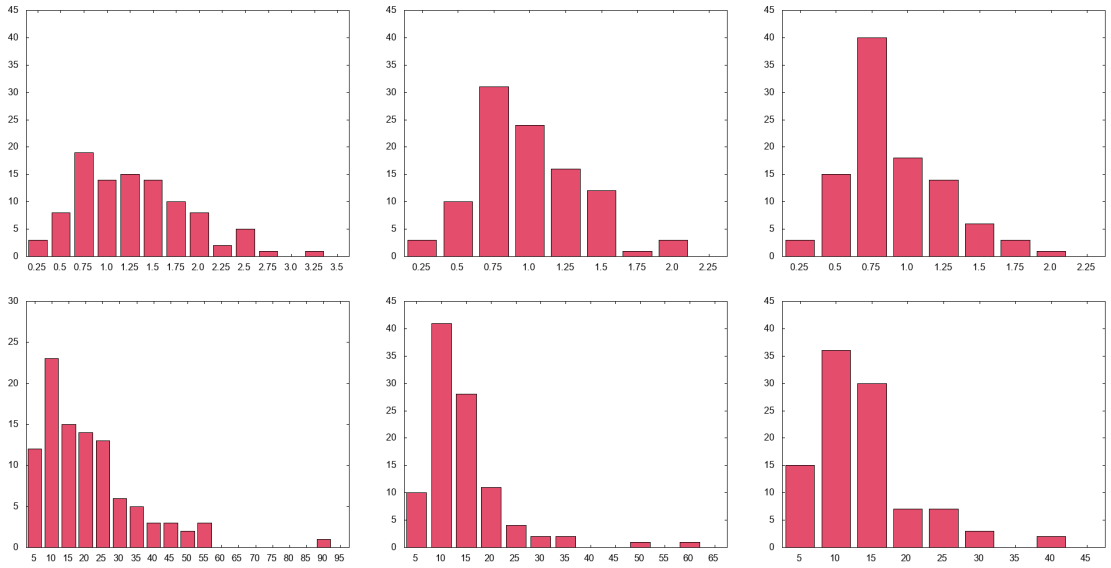


Figure 9: Histograms for the ratio of the number of pseudo-booleans (top)/logical connectives (bottom) to the multiplication count for $n = 2$, 3, 4 and $\omega = 32$

$[0.5, 0.75)$ and, more significantly, the number of pseudo-booleans rarely exceed twice the multiplication count, at least for $\omega = 32$.

Logical connectives versus multiplication count The lower row of Figure 9 examines the complexity of the resulting pseudo-boolean systems from another perspective: the number of logical connectives required to encode them. The pseudo-boolean systems were translated to propositional formulae using the reduction rules of Figure 7 and their complexity measured by counting the number of logical connectives used within them. The histograms present a frequency analysis of ratios of the number logical connectives to the multiplication count. Remarkably, histograms show that typically no more than 25 logical connectives are required per multiplication for $\omega = 32$.

Timing Although the number of symbolic variables is proxy for complexity, it ignores that Gröbner base calculation increases in cost with the number of symbolic variables. Fig. 10 is intended to add clarity, plotting the time in seconds

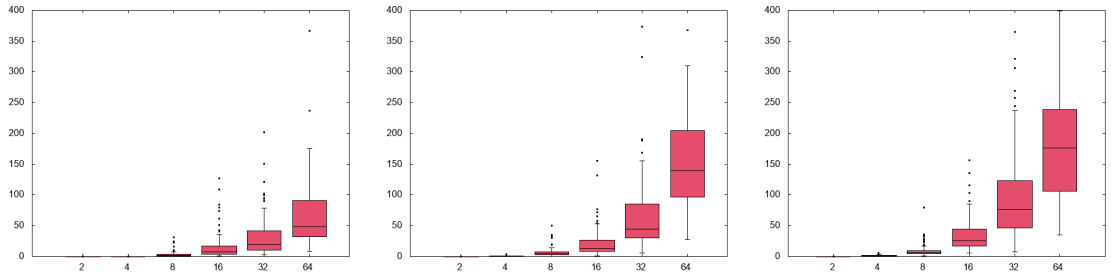


Figure 10: Timings for Buchberger in seconds (y) against ω (x) for $n = 2, 3, 4$

to calculate the pseudo-booleans against ω. As expected, the median runtimes increase with ω for any given n, though not alarmingly for an implementation based on Buchberger rather than a modern, fast engine such as F5 [28]. It should be emphasised that the Gröbner bases computation is the dominating overhead: the resulting SAT instances are almost trivial for our datapoints since the instances are defined over the symbolic variables, which are few in number because of bit-sequence propagation.

3.5 Related Work

Momentum may be growing [1, 6, 19] for combining algebraic and SMT techniques but work at this intersection has mainly focused on CAD [44, 76]. Gröbner bases have been used [3], however, for interpolating non-linear constraints over bit-vectors by use of symbolic conversion predicates. These predicates are used to lazily convert between bit-vectors and rationals, over which Gröbner bases are computed. A closer integration of Gröbner bases with bit-vectors is offered by Buchberger's algorithm [8] which has been adapted to operate on polynomials integers with arbitrary moduli [5], work which is developed in this current chapter. Further afield efficient, and also motivated by the desire to bypass bit-blasting, portfolio solvers have been developed for bit-vectors [77] which combine learning with word-level propagators [58] which iteratively restrict the values which can be assigned to a bit-vector. In contrast to our work, the propagators are designed to run in constant time and make use of low-level bit-twiddling operations [78].

3.6 Concluding Discussion

This chapter argues for translating polynomial equalities over bit-vectors into pseudo-boolean constraints, the central idea being to use Gröbner bases to expose the consequences of setting an individual bit on the bit-vectors over which a polynomial system is defined. The resulting technique, named bit-sequence propagation, typically infers the values of many bits from setting a single bit, even in the context of symbolic bit assignments. The symbolic bits enable the Gröbner bases to be calculated in an entirely deterministic fashion, with search, and its all associated complexity, encapsulated within the pseudo-boolean solver, whether one is employed directly or a reduction to SAT is used.

Chapter 4

The Modular Polynomial Abstract Domain

4.1 Introduction

Numeric abstract domains evolve at a surprisingly slow pace considering their ubiquity in optimisation, transformation and verification. One evolutionary step is when an abstract domain, originally conceived for idealised, arbitrary-precision arithmetic, is adapted to machine arithmetic to better suit its working environment. This adaption is more often a leap than a step since the domain operations typically need to be fundamentally reimagined to model modular arithmetic.

Modular domains It has taken more then two decades for each of the classical abstract domains of ranges [14, 37], difference constraints [22] and linear equalities [46], to be adjusted to a modular setting, as realised in, respectively, sign agnostic range analysis [33], modular difference constraints [32] and linear equalities modulo a power of two [65]. The tenor of these works is that operating over modular integers is not, in fact, a restriction, but rather the natural domain for deriving invariants over fixed-width integers, which are the norm in mainstream programming languages.

Working over modular integers not only allows a more faithful representation of concrete program operations, but can even expose invariants that exist but

would otherwise be missed. For instance, consider a program operating on two 32-bit unsigned integers x and y, and a merge point for three branches. Suppose $x = 1, y = 6$ at the end of the first branch; $x = 2, y = 13$ at the end of the second; and $x = 0, y = 2^{32} - 1$ at the end of the third. The linear modular constraint $y \equiv 7x - 1 (\mathrm{mod}\ 2^{32})$ holds for all three points and therefore summarises program state at the merge point. But $y = 7x - 1$ does not hold, nor any linear equality since the three points are not co-linear in Euclidean geometry.

Modular polynomials For inferring polynomial invariants, one might be forgiven for considering the additional complexity of modular arithmetic to be an irritation, justified only by the desire to faithfully model machine integers and avoid missing invariants. In this chapter we challenge this view by demonstrating how a novel abstract domain employing modular arithmetic can actually simplify the discovery of polynomial equalities. Contrary to non-modular approaches [9, 21, 41, 63, 49, 50, 70, 71], the Modular Polynomial Abstract Domain (MPAD) is a finite lattice, finessing the need for widening and ad hoc constructions, such as artificial degree bounds.

Closure of modular systems Fundamental to MPAD is the concept of a closed polynomial system. A system of polynomials is closed if it cannot be further augmented with polynomials without restricting its solution set. Mirroring a construction used for the Octagon domain [59], we demonstrate that join and projection can be calculated, without omitting polynomials that actually hold, when they are applied to closed systems. Moreover, the systems that result are also closed. This, in general, does not hold for meet, thus it is necessary to apply a closure operation, which takes a polynomial system as input and outputs a closed system with the same solution set.

Contributions To summarise, this chapter makes the following contributions:

- We propose MPAD, whose invariants are systems of polynomial equations modulo a power of two, providing formal definitions of the key abstract domain operations of meet, join and projection;

- We introduce a notion of closure, showing that it is preserved by join and projection but must be re-established after meet in order to retain all polynomial invariants;
- We demonstrate that MPAD can derive invariants that cannot be expressed with non-modular polynomial systems. Therefore, the techniques presented in this chapter represent a new point in the pantheon of abstract domains.

4.2 Modular Polynomial Abstract Domain

This section abstractly specifies MPAD, and its domain operations, with minimal mathematical machinery. The problem of how to finitely represent the elements of MPAD and finitely compute the domain operations is deferred until after an extended example which illustrates how the domain operations are deployed.

4.2.1 Concretisation

For $\vec{a} \in \mathbb{Z}_m^d$ and $p \in \mathbb{Z}_m[\vec{x}]$ let $[\![p]\!]_{\vec{x}}(\vec{a})$ denote evaluating p at $\vec{a}$ by substituting each a_i for x_i in p, and calculating the resulting arithmetical expression. Through this definition, a set of polynomials in $\mathbb{Z}_m[\vec{x}]$ is a symbolic description of a set of points, interpreted by $\gamma_{\vec{x}}$ as follows:

Definition 12. The concretisation map $\gamma_{\vec{x}} : \wp(\mathbb{Z}_m[\vec{x}]) \to \wp(\mathbb{Z}_m^d)$ where $d = |\vec{x}|$ is defined:

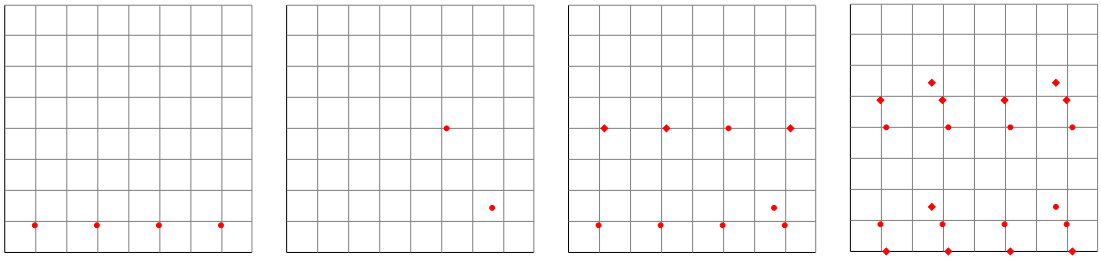
$$\gamma_{\vec{x}}(P) = \{\vec{a} \in \mathbb{Z}_m^d \mid [\![p]\!]_{\vec{x}}(\vec{a}) = 0 \text{ for all } p \in P\}$$

The set of points $\gamma_{\vec{x}}(P)$ is the solution (or zero) set of the set P of polynomials over $\vec{x}$. For a single $p \in \mathbb{Z}_m[\vec{x}]$, let $\gamma_{\vec{x}}(p) = \gamma_{\vec{x}}(\{p\})$.

Example 20. Let $P_1, P_2 \subseteq \mathbb{Z}_8[x]$ where $P_1 = \{x^2 + 7x\}$ and $P_2 = \{x + 6\}$. Then $\gamma_x(P_1) = \{0, 1\}$ and $\gamma_x(P_2) = \{2\}$.

Example 21. Let $\vec{x} = \langle x, y \rangle$ and $Q_1, Q_2 \subseteq \mathbb{Z}_{256}[\vec{x}]$ where

$$Q_1 = \left\{ \; 4x + 132, y + 228 \; \right\} \quad Q_2 = \left\{ \begin{array}{ll} x^2 + x + 123y + 130, & xy + 108y + 128, \\ 2x + 23y + 54, & y^2 + 82y, \\ 128y & \end{array} \right\}$$



(a) $\gamma_{\vec{x}}(Q_1)$ (b) $\gamma_{\vec{x}}(Q_2)$ (c) $\gamma_{\vec{x}}(Q_3)$ (d) $\gamma_{\vec{x}}(Q_4)$

Figure 11: Dyadic join with and without closure, where $Q_3 = \langle Q_1' \rangle_{\vec{x}} \cap \langle Q_2 \rangle_{\vec{x}}$ and $Q_4 = \langle Q_1 \rangle_{\vec{x}} \cap \langle Q_2 \rangle_{\vec{x}}$

The solutions sets $\gamma_{\vec{x}}(Q_1)$ and $\gamma_{\vec{x}}(Q_2)$ are plotted as points in $[0, 255]^2$ in Fig. 11(a) and Fig. 11(b) respectively. Here, the grid lines represent increments of 32. Although Q_1 is linear it has 4 solutions, namely (31, 28), (95, 28), (159, 28) and (223, 28), because $31 \cdot 4 \equiv_{256} 95 \cdot 4 \equiv_{256} 159 \cdot 4 \equiv_{256} 223 \cdot 4 \equiv_{256} 124 \equiv_{256} -132$.

Given $P = \langle B \rangle_{\vec{x}}$, the following result shows that we can reason about the solution sets of P and B interchangeably:

Lemma 4. If $P = \langle B \rangle_{\vec{x}}$ then $\gamma_{\vec{x}}(P) = \gamma_{\vec{x}}(B)$

4.2.2 Closure

Suppose $P \subseteq \mathbb{Z}_m[\vec{x}]$, $p \in \mathbb{Z}_m[\vec{x}]$ and $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p)$. Then $\gamma_{\vec{x}}(P \cup \{p\}) = \gamma_{\vec{x}}(P)$, thus P can be augmented with p without restricting its solution set. This is the intuition behind the following definition:

Definition 13. The operator $\uparrow_{\vec{x}}: \wp(\mathbb{Z}_m[\vec{x}]) \to \wp(\mathbb{Z}_m[\vec{x}])$ is defined by:

$$\uparrow_{\vec{x}} P = \{p \in \mathbb{Z}_m[\vec{x}] \mid \gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p)\}$$

The following result collects fundamental properties of $\uparrow_{\vec{x}}$. The first three together imply that $\uparrow_{\vec{x}}$ is a closure operator on $\langle \wp(\mathbb{Z}_m[\vec{x}]), \subseteq \rangle$. The fourth implies that $\uparrow_{\vec{x}}$ constructs a canonical representation of a system of polynomials. The fifth shows that the canonical representation preserves the solution set, hence it is sufficient to work with this representation alone.

Proposition 4. The operator $\uparrow_{\vec{x}}$ satisfies the following:

- $P \subseteq \uparrow_{\vec{x}} P$ (extensivity),
- if $P_1 \subseteq P_2$ then $\uparrow_{\vec{x}} P_1 \subseteq \uparrow_{\vec{x}} P_2$ (monotonicity),
- $\uparrow_{\vec{x}} \uparrow_{\vec{x}} P = \uparrow_{\vec{x}} P$ (idempotence),
- $\gamma_{\vec{x}}(P_1) = \gamma_{\vec{x}}(P_2)$ iff $\uparrow_{\vec{x}} P_1 = \uparrow_{\vec{x}} P_2$,
- $\gamma_{\vec{x}}(\uparrow_{\vec{x}} P) = \gamma_{\vec{x}}(P)$.

The closure operator $\uparrow_{\vec{x}}$ yields a canonical representation of a given set of polynomials, yet the representation is not finite. A tractable representation for $\uparrow_{\vec{x}} P$ could be obtained by demonstrating $\uparrow_{\vec{x}} P = \langle B \rangle_{\vec{x}}$ for some finite B. The following result presents the first step in establishing this:

Lemma 5. If $P = \uparrow_{\vec{x}} P$ then $P = \langle P \rangle_{\vec{x}}$.

In particular, since $\uparrow_{\vec{x}} P = \uparrow_{\vec{x}} \uparrow_{\vec{x}} P$ it holds that $\uparrow_{\vec{x}} P = \langle \uparrow_{\vec{x}} P \rangle_{\vec{x}}$, hence $\uparrow_{\vec{x}} P$ is an ideal. But it has long been known that ideals of polynomials admit a finite basis [38]. In particular, $\uparrow_{\vec{x}} P = \langle B \rangle_{\vec{x}}$ for some finite B. The domain operations developed later will build on this finite representation.

Example 22. Returning to Example 21, $\uparrow_{\vec{x}} Q_1$ and $\uparrow_{\vec{x}} Q_2$ admit the finite representations $\uparrow_{\vec{x}} Q_1 = \langle Q_1' \rangle_{\vec{x}}$ and $\uparrow_{\vec{x}} Q_2 = \langle Q_2 \rangle_{\vec{x}}$ where $Q_1' = \{x^2+2x+1, 4x+132, y+228\}$. Observe $31^2 + 2 \cdot 31 + 1 = 1024 \equiv_{256} 0$. Similarly it follows $\gamma_{\vec{x}}(x^2 + 2x + 1) \supseteq \{(31, y), (95, y), (159, y), (233, y) \mid y \in \mathbb{Z}_{256}\}$. Thus $x^2 + 2x + 1 \in \uparrow_{\vec{x}} Q_1$. However, $x^2 + 2x + 1 \notin \langle Q_1 \rangle_{\vec{x}}$. To see this, consider the expansion of the polynomial $p(4x+132)+q(y+228) = 4(xp+33p+57q)+yq$. Observe that any term t occuring in this polynomial that is independent of y must be a term of $4(xp + 33p + 57q)$. But then, the coefficient of t must be a multiple of 4. In particular, there cannot exist p, q for which $x^2 + 2x + 1 = p(4x + 132) + q(y + 228)$, since x^2 (and in fact $2x$ and 1 as well) is independent of y but has coefficient 1. Hence Q_1 must be enlarged to obtain a basis for $\uparrow_{\vec{x}} Q_1$.

4.2.3 MPAD

The closure operator characterises the elements of our abstract domain:

Definition 14. $\mathsf{MPAD}_m[\vec{x}] = \{P \subseteq \mathbb{Z}_m[\vec{x}] \mid \uparrow_{\vec{x}} P = P\}$

Elements of $\mathsf{MPAD}_m[\vec{x}]$ are said to be closed. If $P_1 \subseteq P_2$ then $\gamma_{\vec{x}}(P_1) \supseteq \gamma_{\vec{x}}(P_2)$ thus to align with $\langle \wp(\mathbb{Z}_m^d), \subseteq \rangle$ the domain $\mathsf{MPAD}_m[\vec{x}]$ adopts the superset ordering:

Proposition 5. $\langle \mathsf{MPAD}_m[\vec{x}], \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle$ is a finite lattice, where

$$\sqsubseteq = \supseteq \quad \bot = \mathbb{Z}_m[\vec{x}] \quad \top = \uparrow_{\vec{x}} \emptyset \quad P_1 \sqcap P_2 = \uparrow_{\vec{x}} (P_1 \cup P_2) \quad P_1 \sqcup P_2 = P_1 \cap P_2$$

Join and meet are specified set theoretically rather than algorithmically. Observe too that MPAD is finite even though there are no bounds, a priori, put on the degree of any polynomial. This follows from the finiteness of $\mathbb{Z}_m$ and the closure construction which underlies MPAD. To observe this, consider the function space $F = \{[\![p]\!]_{\vec{x}} \mid p \in \mathbb{Z}_m[\vec{x}]\} \subseteq \mathbb{Z}_m^d \to \mathbb{Z}_m$. Since the space $\mathbb{Z}_m^d \to \mathbb{Z}_m$ is finite there exists $p_1, \ldots, p_\ell \in \mathbb{Z}_m[\vec{x}]$ such that $F = \{[\![p_i]\!]_{\vec{x}} \mid i \in [1, \ell]\}$. To see how F determines the structure of $\mathsf{MPAD}_m[\vec{x}]$, define $p \equiv q$ iff $[\![q]\!]_{\vec{x}}(\vec{a}) = [\![p]\!]_{\vec{x}}(\vec{a})$ for all $\vec{a} \in \mathbb{Z}_m^d$. Let $P \in \mathsf{MPAD}_m[\vec{x}]$ and $p \in P$. Observe $p \equiv p_i$ for some $i \in [1, \ell]$ and $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p) = \gamma_{\vec{x}}(p_i)$ hence $p_i \in P$. Conversely, if $p_j \in P$ and $p_j \equiv q$ then $q \in P$. Therefore there exists $I \subseteq [1, \ell]$ such that $P = \{q \in \mathbb{Z}_m[\vec{x}] \mid q \equiv p_i, i \in I\}$. Thus $\mathsf{MPAD}_m[\vec{x}]$ only has a finite number of elements.

Example 23. Developing Example 20 further, $\langle P_1 \rangle_{\vec{x}} \sqcup \langle P_2 \rangle_{\vec{x}} = \langle P_1 \rangle_{\vec{x}} \cap \langle P_2 \rangle_{\vec{x}} = \langle P \rangle_{\vec{x}}$ where $P = \{x^3 + 5x^2 + 2x\}$ though at this stage we omit details of how to calculate the intersection of two ideals. Nevertheless observe $\gamma_{\vec{x}}(P) = \{0, 1, 2, 4, 6\}$ hence $\gamma_{\vec{x}}(\langle P_1 \rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(\langle P \rangle_{\vec{x}})$ and $\gamma_{\vec{x}}(\langle P_2 \rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(\langle P \rangle_{\vec{x}})$ as required.

Example 24. Continuing from Example 22, let

$$Q' = \left\{ \begin{array}{l} x^3 + x + 13y^2 + 11y + 126, \\ x^2y + xy + 14y^2 + 24y, \\ 2x^2 + xy + 19y^2 + 97y + 78, \\ xy^2 + 22y^2 + 116y, \\ 2xy + 19y^2 + 110y, \\ 4x + 2y^2 + 82y + 108, \\ y^3 + 22y^2 + 72y, \\ 32y^2 + 64y, \\ 128y, \end{array} \right\} \qquad Q = \left\{ \begin{array}{l} x^2y + xy + 14y^2 + 24y, \\ xy^2 + 22y^2 + 116y, \\ 2xy + 19y^2 + 110y, \\ 4x + 2y^2 + 82y + 108, \\ y^3 + 22y^2 + 72y, \\ 32y^2 + 64y, \\ 128y \end{array} \right\}$$

Then, $\langle Q'_1\rangle_{\vec{x}} \cap \langle Q_2\rangle_{\vec{x}} = \langle Q'\rangle_{\vec{x}}$ and $\langle Q_1\rangle_{\vec{x}} \cap \langle Q_2\rangle_{\vec{x}} = \langle Q\rangle_{\vec{x}}$. Again, we defer the discussion of how Q and Q' are calculated. Observe from Figs. 11(a), 11(b) and 11(c) that $\gamma_{\vec{x}}(\langle Q'_1\rangle_{\vec{x}}) \cup \gamma_{\vec{x}}(\langle Q_2\rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(\langle Q'\rangle_{\vec{x}})$ as required, the diamond points indicating those introduced by join itself. The diamonds in Figure 11(d) are extraneous points introduced by calculating $\langle Q_1\rangle_{\vec{x}} \cap \langle Q_2\rangle_{\vec{x}}$ rather than $\langle Q'_1\rangle_{\vec{x}} \cap \langle Q_2\rangle_{\vec{x}}$. This illustrates that operating on arbitrary bases is not generally sufficient to maintain precision, thus motivating the need for closure.

Finally, the following result asserts that MPAD enjoys mathematical properties that simplify the application of abstract interpretation:

Proposition 6. $\langle \wp(\mathbb{Z}_m^d), \subseteq\rangle \underset{\gamma_{\vec{x}}}{\overset{\alpha_{\vec{x}}}{\rightleftharpoons}} \langle \mathsf{MPAD}_m[\vec{x}], \sqsubseteq\rangle$ is a Galois insertion, where

$$\alpha_{\vec{x}}(A) = \{p \in \mathbb{Z}_m[\vec{x}] \mid A \subseteq \gamma_{\vec{x}}(p)\}$$

4.2.4 Null polynomials

Recall $\top = \uparrow_{\vec{x}} \emptyset = \{p \in \mathbb{Z}_m[\vec{x}] \mid \gamma_{\vec{x}}(\emptyset) \subseteq \gamma_{\vec{x}}(p)\}$. It follows $\top = \{p \in \mathbb{Z}_m[\vec{x}] \mid \forall \vec{a} \in \mathbb{Z}_m^d . [\![p]\!]_{\vec{x}}(\vec{a}) = 0\}$ because $\gamma_{\vec{x}}(\emptyset) = \mathbb{Z}_m^d$. Such polynomials are referred to as null polynomials and represent universally valid constraints.

Example 25. Let $\vec{x} = \langle x, y \rangle$. Then in $\mathbb{Z}_{16}[\vec{x}]$, $\top = \langle B \rangle_{\vec{x}}$ where

$$B = \left\{ \begin{array}{ccc} x^4y^2 + 15x^4y + 10x^3y^2 + 6x^3y + 11x^2y^2 + 5x^2y + 10xy^2 + 6xy, & & \\ x^2y^4 + 10x^2y^3 + 11x^2y^2 + 10x^2y + 15xy^4 + 6xy^3 + 5xy^2 + 6xy, & & \\ 4x^2y^2 + 12x^2y + 12xy^2 + 4xy, & & \\ x^6 + x^5 + 5x^4 + 15x^3 + 2x^2 + 8x, & 2x^4 + 4x^3 + 6x^2 + 4x, & 8x^2 + 8x, \\ y^6 + y^5 + 5y^4 + 15y^3 + 2y^2 + 8y, & 2y^4 + 4y^3 + 6y^2 + 4y, & 8y^2 + 8y \end{array} \right\}$$

It is tempting to remove null polynomials from bases, since they are vacuous as constraints. Unfortunately, this is not generally possible without sacrificing the canonical representation property of closure. Despite this, for brevity, we will follow the convention that null polynomials are omitted in the printed representation of bases.

4.3 Motivating Example

To demonstrate how a run of the analysis can infer a non-linear loop invariant, the class of polynomial programs is introduced. The syntax of this class is given in Section 4.3.1, followed by their reference semantics in Section 4.3.2, formulated over sets of points. Their semantics is then abstracted in Section 4.3.3, to reason about how paths through a program compose. A work-list algorithm is given in Section 4.3.4, which serves as a framework for inferring loop invariants. The whole section, however, majors on a run of the analysis itself, given over Sections 4.3.5, 4.3.6 and 4.3.7 which illustrate how the domain operations are applied within a work-list framework.

4.3.1 Syntax of polynomial programs

Let $\vec{x} = \langle x_1, \ldots, x_d \rangle$ denote a vector of program variables. A polynomial program over $\vec{x}$ is a graph $G = \langle N, E, n^* \rangle$ where N is a finite set of program points, $E \subseteq N \times \mathsf{Stmt} \times N$ is a finite set of annotated edges and $n^* \in N$ is the entry point into G. The set Stmt of program statements is defined:

$$x_j \coloneqq * \quad | \quad x_j \coloneqq p \quad | \quad \textbf{assume } (p = 0) \quad | \quad \textbf{assume } (p \neq 0)$$

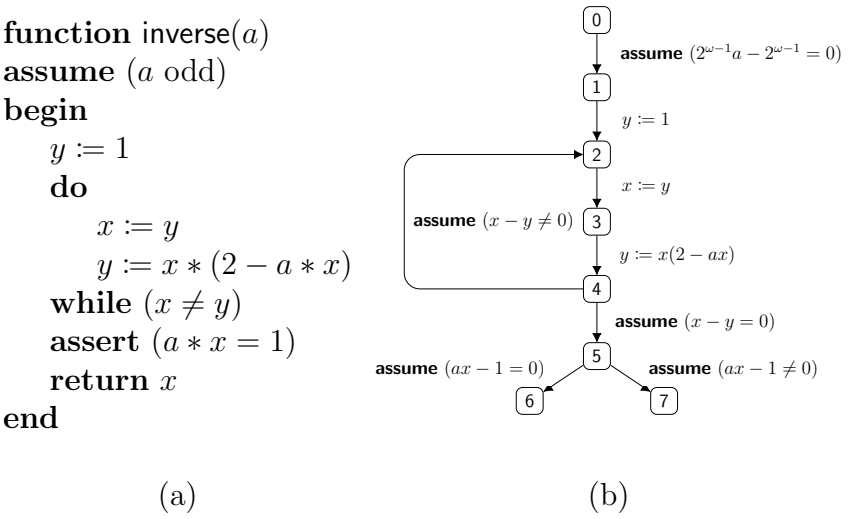


Figure 12: An algorithm (a) and flow graph (b) for computing the multiplicative inverse

where $x_j := *$ and $x_j := p$ denote, respectively, non-deterministic assignment to the variable x_j and polynomial assignment to x_j for some $p \in \mathbb{Z}_m[\vec{x}]$. The **assume** statements for $p = 0$ and $p \neq 0$ provide a linguistic abstraction for positive and negative guards, respectively expressing that p is satisfied, and conversely p is not satisfied, by an assignment to $\vec{x}$.

To illustrate a polynomial program, consider the algorithm [78] for computing the multiplicative inverse of an (odd) integer $a \in \mathbb{Z}_m$ listed in Fig. 12(a). The variables x, y and a all store a ω-bit (unsigned) machine integer. This algorithm is abstracted by $G = \langle N, E, n^* \rangle$ over $\vec{x} = \langle x, y, a \rangle$, where $N = \{0, \ldots, 7\}$, $n^* = 0$ and E is given in Fig. 12(b). The statement **assume** (a odd) is rendered as the edge $\langle 0, \textbf{assume}\ (2^{\omega-1}a - 2^{\omega-1} = 0), 1 \rangle$, where the (linear) polynomial $2^{\omega-1}a - 2^{\omega-1} = 0$ expresses that a is odd. The control-flow for the **do** ... **while** is represented as two edges $\langle 4, \textbf{assume}\ (x - y \neq 0), 2 \rangle$ and $\langle 4, \textbf{assume}\ (x - y = 0), 5 \rangle$, which, respectively, encode the loop condition $x \neq y$ and its negation. The control flow for the assert statement is expressed through two edges: the edge $\langle 5, \textbf{assume}\ (ax - 1 = 0), 6 \rangle$ and $\langle 5, \textbf{assume}\ (ax - 1 \neq 0), 7 \rangle$, where 7 is an error state which is reached if the assertion fails.

4.3.2 Collecting semantics of polynomial programs

The collecting semantics defines the actions of single statements and sequences of statements. The semantics for assignment is formulated as an update to $\vec{a} \in \mathbb{Z}_m^d$ at position j with a constant $c \in \mathbb{Z}_m$, an action denoted by: $\vec{a}[j \mapsto c] = \langle a_1, \ldots, a_{j-1}, c, a_{j+1}, \ldots, a_d \rangle$. A collecting semantics for single statements $[\![\cdot]\!] : \mathsf{Stmt} \to (\wp(\mathbb{Z}_m^d) \to \wp(\mathbb{Z}_m^d))$ can then be defined case-wise by:

$$\begin{aligned}
[\![x_j \coloneqq *]\!](A) &= \{(\vec{a})[j \mapsto c] \mid \vec{a} \in A, c \in \mathbb{Z}_m\} \\
[\![x_j \coloneqq p]\!](A) &= \{(\vec{a})[j \mapsto c] \mid \vec{a} \in A, c = [\![p]\!]_{\vec{x}}(\vec{a})\} \\
[\![\textbf{assume } (p = 0)]\!](A) &= \{\vec{a} \in A \mid [\![p]\!]_{\vec{x}}(\vec{a}) = 0\} \\
[\![\textbf{assume } (p \neq 0)]\!](A) &= \{\vec{a} \in A \mid [\![p]\!]_{\vec{x}}(\vec{a}) \neq 0\}
\end{aligned}$$

To lift the collecting semantics to a program $G = \langle N, E, n^* \rangle$, the set of paths Π_G through G is introduced. The set Π_G is defined simultaneously with the map $\mathsf{end} : \Pi_G \to N$ as follows:

$$\begin{array}{ll}
\epsilon \in \Pi_G & \mathsf{end}(\epsilon) = n^* \\
\pi \cdot e \in \Pi_G \text{ if } \pi \in \Pi_G, \mathsf{end}(\pi) = n, e = \langle n, s, n' \rangle \in E & \mathsf{end}(\pi \cdot \langle n, s, n' \rangle) = n'
\end{array}$$

A prefix relation $\preceq \subseteq \Pi_G \times \Pi_G$ is defined as the smallest partial order $\preceq$ such that for all $\pi \in \Pi_G$, $\epsilon \preceq \pi$ and if $\pi = \pi' \cdot e$ where $e \in E$ then $\pi' \preceq \pi$. The collecting semantics on single statements lifts to paths $[\![\cdot]\!] : \Pi_G \to (\wp(\mathbb{Z}_m^d) \to \wp(\mathbb{Z}_m^d))$ by:

$$\begin{aligned}
[\![\epsilon]\!](A) &= A \\
[\![\pi \cdot \langle n, s, n' \rangle]\!](A) &= [\![s]\!]([\![\pi]\!](A))
\end{aligned}$$

Thus $[\![\pi]\!]$ thus maps a set of points A at n^* to a set of points $[\![\pi]\!](A)$ at $\mathsf{end}(\pi)$. This semantics serves as a reference semantics for judging the correctness of an abstract semantics, given next.

4.3.3 Abstract semantics of polynomial programs

The collecting semantics defines the behaviour of sequences of statements, but does not provide an algorithm for inferring invariants. To this end, state maps are introduced where a state map for $G = \langle N, E, n^* \rangle$ over $\vec{x}$ is a map $\sigma : N \to$

function fixpoint($G = \langle N, E, n^* \rangle$)
begin
  $k \coloneqq 0$
  $w_0 \coloneqq \{\langle n, s, n' \rangle \in E \mid n = n^*\}$
  $\sigma_0 \coloneqq \lambda n.(\text{if } n = n^* \text{ then } \top \text{ else } \bot)$
  while $(w_k \neq \emptyset)$ **do**
    let $e_k = \langle n, s, n' \rangle \in w_k$
    let $P_k \in \mathsf{MPAD}_m[\vec{x}]$ **where** $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(P_k)$
    if $(P_k \sqsubseteq \sigma_k(n'))$ **then**
      $w_{k+1} \coloneqq w_k \setminus \{e_k\}$
      $\sigma_{k+1} \coloneqq \sigma_k$
    else
      $w_{k+1} \coloneqq (w_k \setminus \{e_k\}) \cup \{e' \in E \mid e' = \langle n', s', n'' \rangle\}$
      $\sigma_{k+1} \coloneqq \sigma_k[n' \mapsto \sigma_k(n') \sqcup P_k]$
    end if
    $k \coloneqq k + 1$
  end while
  return σ_k
end

Figure 13: Worklist-based fixpoint algorithm

$\mathsf{MPAD}_m[\vec{x}]$. The set of state maps, the function space $N \to \mathsf{MPAD}_m[\vec{x}]$, is ordered point-wise by $f_1 \sqsubseteq f_2$ iff $f_1(n) \sqsubseteq f_2(n)$ for all $n \in N$. Meet and join lift by $f_1 \sqcap f_2 = \lambda n.\ f_1(n) \sqcap f_2(n)$ and $f_1 \sqcup f_2 = \lambda n.\ f_1(n) \sqcup f_2(n)$ and bottom and top are defined $\bot = \lambda n.\ \bot$ and $\top = \lambda n.\ \top$. Since N is finite and $\mathsf{MPAD}_m[\vec{x}]$ is a finite lattice, it follows that $\langle N \to \mathsf{MPAD}_m[\vec{x}], \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle$ is also a finite lattice.

The connection with the collecting semantics is made by interpreting a state map σ as a description of a (possibly infinite) set $\gamma(\sigma)$ of sequences emanating from n^*, defined:

$$\gamma(\sigma) = \{\pi \in \Pi_G \mid \forall \rho \preceq \pi. [\![\rho]\!](\mathbb{Z}_m^d) \subseteq \gamma_{\vec{x}}(\sigma(\mathsf{end}(\rho)))\}$$

Note that the set of points at n^* is taken as $\mathbb{Z}_m^d$. The calculational problem is then to find a state map $\sigma \in N \to \mathsf{MPAD}_m[\vec{x}]$ that describes all paths Π_G, that is, $\Pi_G \subseteq \gamma(\sigma)$. This can be achieved with the work-list algorithm presented in Fig 13.

4.3.4 Calculating the abstract semantics (framework)

The work-list algorithm computes a non-decreasing sequence $\sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \ldots \sqsubseteq \sigma_k$ of state maps where $\sigma_0 = \lambda n.(\text{if } n = n^* \text{ then } \top \text{ else } \bot)$. The algorithm is driven by a work-list $w_k \subseteq E$ which is primed with the edges that flow from n^*. At each iteration of the algorithm, if $w_k \neq \emptyset$ then an edge $e = \langle n, s, n' \rangle$ is (non-deterministically) selected from w_k and, σ_{k+1} is computed as a relaxation of σ_k that ensures $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(\sigma_{k+1}(n'))$. The algorithm reduces this process to the selection of $P_k \in \mathsf{MPAD}_m[\vec{x}]$ satisfying $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n)) \subseteq \gamma_{\vec{x}}(P_k)$, but leaves open the question of how to choose P_k. Otherwise, if $w_k = \emptyset$, the algorithm terminates and returns σ_k. The following result, and its corollary, asserts that the work-list algorithm terminates to derive a state map that describes all paths Π_G:

Proposition 7. For any polynomial graph $G = \langle N, E, n^* \rangle$, the fixpoint algorithm terminates and returns a state map σ^* satisfying $[\![s]\!](\gamma_{\vec{x}}(\sigma^*(n))) \subseteq \gamma_{\vec{x}}(\sigma^*(n'))$ for all $\langle n, s, n' \rangle \in E$.

Corollary 2. If $\mathsf{fixpoint}(G) = \sigma^*$ then $\Pi_G \subseteq \gamma(\sigma^*)$

4.3.5 Calculating the abstract semantics: pre-loop

To illustrate the calculation of the abstract semantics, a concrete execution is considered for the polynomial program G represented in Fig. 12(b) where $\omega = 4$ and $\vec{x} = \langle x, y, a \rangle$. This execution is summarised in Fig. 14, where each row corresponds to iteration k of the fixpoint algorithm.

The second column displays the worklist w_k. For brevity, each edge $\langle n, s, n' \rangle$ in the worklist is abbreviated to $\langle n, n' \rangle$. Since for each pair $\langle n, n' \rangle$ there is at most one statement s for which $\langle n, s, n' \rangle \in E$, this causes no ambiguity. The selected edge is always the first listed in the worklist for a given step. For instance, at step 4, the edge $\langle 4, 2 \rangle$ is selected, rather than $\langle 4, 5 \rangle$.

The third column displays the state map σ_{k+1} as a function of the state map σ_k. This is either σ_k, if no update occured, or else $\sigma_k[n \mapsto Q]$, where $Q = \sigma_k(n) \sqcup P_k$. For brevity, polynomials that appear more than once are introduced with a label (p_a, p_b, etc.) which is used to denote them in the bases of subsequent iterations.

k	w_k	σ_{k+1}
0	$\{\langle 0,1\rangle\}$	$\sigma_0[1 \mapsto \langle x^4a + x^4 + 2x^3a + 2x^3 + 3x^2a + 3x^2 + 2xa + 2x\ (p_a),$ $2x^2y^2a + 2x^2y^2 + 2x^2ya + 2x^2y + 2xy^2a + 2xy^2 +$ $2xya + 2xy, x^2a^2 + 7x^2 + xa^2 + 7x\ (p_b),$ $4x^2a + 4x^2 + 4xa + 4x\ (p_c),$ $y^4a + y^4 + 2y^3a + 2y^3 + 3y^2a + 3y^2 + 2ya + 2y,$ $y^2a^2 + 7y^2 + ya^2 + 7y, 4y^2a + 4y^2 + 4ya + 4y,$ $a^3 + a^2 + 7a + 7\ (p_d), 2a^2 + 14\ (p_e), 8a + 8\ (p_f)\rangle_{\vec{x}}]$
1	$\{\langle 1,2\rangle\}$	$\sigma_1[2 \mapsto \langle p_a, p_b, p_c, y + 15\ (p_g), p_d, p_e, p_f\rangle_{\vec{x}}]$
2	$\{\langle 2,3\rangle\}$	$\sigma_2[3 \mapsto \langle x + 15\ (p_h), p_g, p_d, p_e, p_f\rangle_{\vec{x}}]$
3	$\{\langle 3,4\rangle\}$	$\sigma_3[4 \mapsto \langle p_h, y + a + 14, p_d, p_e, p_f\rangle_{\vec{x}}]$
4	$\{\langle 4,2\rangle, \langle 4,5\rangle\}$	$\sigma_4[2 \mapsto \langle p_a, p_b, p_c, xy + 15x + 7y + 9,$ $y^2 + ya + 5y + 7a + 2\ (p_i), ya^2 + 7y + a^2 + 7\ (p_j),$ $2ya + 2y + 6a + 6\ (p_k), 8y + 8\ (p_l), p_d, p_e, p_f\rangle_{\vec{x}}]$
5	$\{\langle 2,3\rangle, \langle 4,5\rangle\}$	$\sigma_5[3 \mapsto \langle x + 7y + 8\ (p_m), p_i, p_j, p_k, p_l, p_d, p_e, p_f\rangle_{\vec{x}}]$
6	$\{\langle 3,4\rangle, \langle 4,5\rangle\}$	$\sigma_6[4 \mapsto \langle x^2 + 2x + 3y + 3a + 7\ (p_n), xy + 3x + a + 11,$ $xa + 3x + y + 11, 4x + 2y + 2a + 8\ (p_o),$ $y^2 + 2y + a^2 + 6a + 6\ (p_q), ya + y + a^2 + 7a + 6\ (p_r),$ $4y + 4a + 8\ (p_s), p_d, p_e, p_f\rangle_{\vec{x}}]$
7	$\{\langle 4,2\rangle, \langle 4,5\rangle\}$	$\sigma_7[2 \mapsto \langle p_a, x^2y + 7x^2 + 8x + 7y + 9, p_b, p_c,$ $xy^2 + 15x + y^2 + 15,$ $xya + xy + 7xa + 7x + ya + y + 7a + 7,$ $2xy + 14x + y^2 + ya + 3y + 7a + 4,$ $y^3 + y^2 + 7y + 7\ (p_t), y^2a + y^2 + 7a + 7\ (p_u),$ $2y^2 + 14\ (p_v), p_j, p_k, p_l, p_d, p_e, p_f\rangle_{\vec{x}}]$
8	$\{\langle 2,3\rangle, \langle 4,5\rangle\}$	$\sigma_8[3 \mapsto \langle p_m, p_t, p_u, p_v, p_j, p_k, p_l, p_d, p_e, p_f\rangle_{\vec{x}}]$
9	$\{\langle 3,4\rangle, \langle 4,5\rangle\}$	$\sigma_9[4 \mapsto \langle p_n, xy + xa + 2x + 3y + 3a + 6, xa^2 + 3x + 2y + a^2 +$ $2a + 7, 2xa + 2x + 6a + 6, p_o, p_q, p_r, p_s, p_d, p_e, p_f\rangle_{\vec{x}}]$
10	$\{\langle 4,2\rangle, \langle 4,5\rangle\}$	σ_{10}
11	$\{\langle 4,5\rangle\}$	$\sigma_{11}[5 \mapsto \langle x + a^2 + 7a + 7\ (p_w), y + a^2 + 7a + 7\ (p_x), p_d, p_e, p_f\rangle_{\vec{x}}]$
12	$\{\langle 5,6\rangle, \langle 5,7\rangle\}$	$\sigma_{12}[6 \mapsto \langle p_w, p_x, p_d, p_e, p_f\rangle_{\vec{x}}]$
13	$\{\langle 5,7\rangle\}$	σ_{13}

Figure 14: Updates to the state map

Assume statement (positive case) When $k = 0$, the edge $\langle 0,1\rangle$ is selected from w_0, corresponding to the statement **assume** $(8a + 8 = 0)$. To process this edge, $P_0 = \langle B_0\rangle_{\vec{x}} \in \mathsf{MPAD}_{16}[\vec{x}]$ is computed such that $[\![$**assume** $(8a + 8 =$

$0)]\!](\gamma_{\vec{x}}(\sigma_0(0))) \subseteq \gamma_{\vec{x}}(P_0)$. Note that $\sigma_0(0) = \top = \langle N \rangle_{\vec{x}}$ where, as discussed in Section 4.2.4, N is a basis for null-polynomials in $\mathbb{Z}_m[\vec{x}]$. To construct B_0, the closure of the basis $N \cup \{8a + 8\}$ is calculated, which yields

$$B_0 = \left\{ \begin{array}{c} x^4a + x^4 + 2x^3a + 2x^3 + 3x^2a + 3x^2 + 2xa + 2x, \\ 2x^2y^2a + 2x^2y^2 + 2x^2ya + 2x^2y + 2xy^2a + 2xy^2 + 2xya + 2xy, \\ x^2a^2 + 7x^2 + xa^2 + 7x, \quad 4x^2a + 4x^2 + 4xa + 4x, \\ y^4a + y^4 + 2y^3a + 2y^3 + 3y^2a + 3y^2 + 2ya + 2y, \\ y^2a^2 + 7y^2 + ya^2 + 7y, \quad 4y^2a + 4y^2 + 4ya + 4y, \\ a^3 + a^2 + 7a + 7, \quad 2a^2 + 14, \quad 8a + 8 \end{array} \right\}$$

Intuitively, adjoining the polynomial $8a+8$ to N imposes the constraint $8a+8 = 0$. The closure algorithm is applied to guarantee that $P_0 = \langle B_0 \rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$, a property that it not guaranteed of $\langle N \cup \{8a + 8\} \rangle_{\vec{x}}$. An algorithm for calculating closure will be detailed in Section 5.2.

To complete the abstract execution of this statement, the new state map σ_1 and worklist w_1 are computed. Since $\sigma_0(1) = \bot$, the test $P_0 \sqsubseteq \sigma_0(1)$ fails, hence the **else** clause is executed. But also since $\sigma_0(1) = \bot$, it follows $\sigma_0(1) \sqcup P_0 = P_0$. Thus, $\sigma_1 = \sigma_0[1 \mapsto P_0]$, as recorded in the first row of the table, and $w_1 = (w_0 \setminus \{\langle 0, 1\rangle\} \cup \{\langle n, n'\rangle \in E \mid n = 1\} = \{\langle 1, 2\rangle\}$, as recorded in the second row of the table. Execution then continues with $k = 1$.

Polynomial assignment When $k = 1$, the edge $\langle 1, 2\rangle$ is selected, corresponding to the statement $y \coloneqq 1$. To process this edge, $P_1 \in \mathsf{MPAD}_{16}[\vec{x}]$ is computed such that $[\![y \coloneqq 1]\!](\gamma_{\vec{x}}(\sigma_1(1))) \subseteq \gamma_{\vec{x}}(P_1)$. Recall from above that $\sigma_1(1) = P_0 = \langle B_0 \rangle_{\vec{x}}$. To effect the assignment $y \coloneqq 1$, first the basis B_0 is adjoined with the polynomial $w-1$. Here, w is a new variable that represents the value of y after the assignment and the polynomial $w-1$ expresses that this value must equal 1. Then, y is eliminated from $B_0 \cup \{w-1\}$, to reflect that y is overwritten during the assignment. This elimination step is achieved in two phases and exploits the concept of Gröbner basis, introduced in Section **??**. First, a Gröbner basis is computed for $\langle B_0 \cup \{w - 1\} \rangle_{\vec{x}}$ with respect

to a lexicographical ordering $\langle y, x, w, a\rangle$ over the variables, yielding

$$\left\{\begin{array}{c} y^4a + y^4 + 2y^3a + 2y^3 + 3y^2a + 3y^2 + 2ya + 2y, \\ 2y^2x^2a + 2y^2x^2 + 2y^2xa + 2y^2x + 2yx^2a + 2yx^2 + 2yxa + 2yx, \\ y^2a^2 + 7y^2 + ya^2 + 7y, \quad 4y^2a + 4y^2 + 4ya + 4y, \\ x^4a + x^4 + 2x^3a + 2x^3 + 3x^2a + 3x^2 + 2xa + 2x, \\ x^2a^2 + 7x^2 + xa^2 + 7x, \quad 4x^2a + 4x^2 + 4xa + 4x, \\ w + 15, \quad a^3 + a^2 + 7a + 7, \quad 2a^2 + 14, \quad 8a + 8 \end{array}\right\}$$

Then, all polynomials involving y are deleted:

$$\left\{\begin{array}{c} x^4a + x^4 + 2x^3a + 2x^3 + 3x^2a + 3x^2 + 2xa + 2x, \\ x^2a^2 + 7x^2 + xa^2 + 7x, \quad 4x^2a + 4x^2 + 4xa + 4x, \\ w + 15, \quad a^3 + a^2 + 7a + 7, \quad 2a^2 + 14, 8a + 8 \end{array}\right\}$$

Note that it is essential for the original basis to be a Gröbner basis to ensure this deletion does not lose information. To finalise the assignment, w is renamed to y, yielding:

$$B_1 = \left\{\begin{array}{c} x^4a + x^4 + 2x^3a + 2x^3 + 3x^2a + 3x^2 + 2xa + 2x, \\ x^2a^2 + 7x^2 + xa^2 + 7x, \quad 4x^2a + 4x^2 + 4xa + 4x, \\ y + 15, \quad a^3 + a^2 + 7a + 7, \quad 2a^2 + 14, 8a + 8 \end{array}\right\}$$

In the sequel it will be shown that $P_1 = \langle B_1\rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$ hence closure need not be reapplied after polynomial assignment.

Since $\sigma_1(2) = \bot$ it follows as in the previous case that $P_1 \not\sqsubseteq \sigma_1(2)$ and $\sigma_1(2) \sqcup P_1 = P_1$. Thus, $\sigma_2 = \sigma_1[2 \mapsto P_1]$ and $w_2 = (w_1 \setminus \{\langle 1, 2\rangle\} \cup \{\langle n, n'\rangle \in E \mid n = 2\} = \{\langle 2, 3\rangle\}$, as recorded in the second and third rows of the table respectively. Execution then continues with $k = 2$.

4.3.6 Calculating the abstract semantics: loop

The assignments $x \coloneqq y$ and $y \coloneqq x(2 - ax)$ are handled in the same way as above and yield σ_3 and σ_4, as recorded in the fourth and fifth rows of the table. At this point, execution reaches node 4 and $w_4 = \{\langle 4, 2\rangle, \langle 4, 5\rangle\}$, corresponding to the two

edges rooted at node 4.

Assume statement (negative case) When $k = 4$, the edge $\langle 4, 2\rangle$ is selected, corresponding to the statement **assume** $(x - y \neq 0)$. To process this edge, $P_4 \in \mathsf{MPAD}_{16}[\vec{x}]$ is computed such that $[\![\textbf{assume } (x - y \neq 0)]\!](\gamma_{\vec{x}}(\sigma_4(4))) \subseteq \gamma_{\vec{x}}(P_4)$. Note that, from the table it follows $\sigma_3(4) = \langle B_3\rangle_{\vec{x}}$ where

$$B_3 = \Big\{x + 15, \quad y + a + 14, \quad a^3 + a^2 + 7a + 7, \quad 2a^2 + 14, \quad 8a + 8\Big\}$$

To effect the operation, closure is separately applied to four bases:

$$\begin{array}{lll}
B_3 \cup \{8(x-y)+8\} & \rightarrow_{\mathsf{cl}[\vec{x}]} & B_{4,1} = \{1\} \\
B_3 \cup \{4(x-y)+8\} & \rightarrow_{\mathsf{cl}[\vec{x}]} & B_{4,2} = \{x + 15, \quad y + a + 14, \quad a^2 + 2a + 1, \quad 4a + 4\} \\
B_3 \cup \{2(x-y)+8\} & \rightarrow_{\mathsf{cl}[\vec{x}]} & B_{4,3} = \{x + 15, \quad y + a + 14, \quad a^2 + 7, \quad 2a + 6\} \\
B_3 \cup \{(x-y)+8\} & \rightarrow_{\mathsf{cl}[\vec{x}]} & B_{4,4} = \{x + 15, \quad y + 7, \quad a + 7\}
\end{array}$$

The intuition is that each $\gamma_{\vec{x}}(B_{4,k})$ is the subset of $\vec{a} \in \gamma_{\vec{x}}(B_3)$ for which the k least-significant bits of $[\![x - y]\!]_{\vec{x}}(\vec{a})$ store the value 2^{k-1}. Thus $\gamma_{\vec{x}}(B_{4,1})$ is the subset of $\vec{a} \in \gamma_{\vec{x}}(B_3)$ for which the least bit of $[\![x-y]\!]_{\vec{x}}(\vec{a})$ is $2^0 = 1$; $\gamma_{\vec{x}}(B_{4,2})$ is the subset for which the 2 least bits of $[\![x - y]\!]_{\vec{x}}(\vec{a})$ store $2^1 = 2$, etc. Since $[\![x - y]\!]_{\vec{x}}(\vec{a}) \neq 0$ holds precisely when at least one bit is set, it follows $P_4 = \bigsqcup_{k=1}^{4}\langle B_{4,k}\rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$ satisfies the property above (the procedure for calculating join is discussed shortly). In fact, in this case $P_4 = P_3$, hence the abstract execution of **assume** $(x - y \neq 0)$ does not strengthen the polynomial constraints even though $B_{4,1} = \{1\}$ reveals that the difference between x and y is never odd.

Join As for the previous updates, the inclusion $P_4 \sqsubseteq \sigma_4(2)$ does not hold, hence the join $\sigma_4(2) \sqcup P_4 = \langle B_1\rangle_{\vec{x}} \sqcup \langle B_4\rangle_{\vec{x}}$ must be computed. Contrary to the previous updates, however, $\sigma_4(2) \neq \bot$ hence the join cannot be inferred immediately. To compute it, the basis $\{wp \mid p \in B_2\} \cup \{(1 - w)p \mid p \in B_3\}$ is constructed, where w

is a fresh variable, before w is eliminated, yielding:

$$B' = \left\{ \begin{array}{c} x^4a + x^4 + 2x^3a + 2x^3 + 3x^2a + 3x^2 + 2xa + 2x, \\ x^2a^2 + 7x^2 + xa^2 + 7x, \quad 4x^2a + 4x^2 + 4xa + 4x \\ xy + 15x + 7y + 9, \quad y^2 + ya + 5y + 7a + 2, \\ ya^2 + 7y + a^2 + 7, \quad 2ya + 2y + 6a + 6, \quad 8y + 8 \\ a^3 + a^2 + 7a + 7, \quad 2a^2 + 14, \quad 8a + 8 \end{array} \right\}$$

Then, $\sigma_5 = \sigma_2[2 \mapsto \langle B' \rangle_{\vec{x}}]$, as recorded in the fifth row of the table.

Loop stability This update propagates to nodes 3 and 4 by re-evaluating the assignments $x := y$ and $y := x(2 - ax)$, as recorded in the table for $k = 5$ and $k = 6$. The update to node 4 is propagated back to node 2 via the statement **assume** $(x - y \neq 0)$. Processing this statement leads to a further relaxation at node 2 which again propagates to nodes 3 and 4 under the assignments $x := y$ and $y := x(2 - ax)$. These updates are recorded in the rows of the table corresponding to $k = 7$, $k = 8$ and $k = 9$. When $k = 10$, the edge $\langle 4, 2 \rangle$, corresponding to the statement **assume** $(x - y \neq 0)$, is processed for the third time and $P_{10} \in \mathsf{MPAD}_m[\vec{x}]$ is computed such that $[\![\textbf{assume } (x - y \neq 0)]\!](\gamma_{\vec{x}}(\sigma_{10}(4))) \subseteq \gamma_{\vec{x}}(P_{10})$, analogously to before. In this case, however, it holds that $P_{10} \sqsubseteq \sigma_{10}(2)$ and thus $\sigma_{11} = \sigma_{10}$ and a fixpoint is reached for the loop body in 3 iterations.

4.3.7 Calculating the abstract semantics: post-loop

When, $k = 11$, 12 and 13, the statements **assume** $(x - y = 0)$ and **assume** $(ax - 1 = 0)$, and **assume** $(ax - 1 \neq 0)$ are processed analogously to before, and the associated updates are recorded in the table. After executing these statements, the worklist becomes empty hence the algorithm terminates. In the final state map σ_{14} it holds that $\sigma_{14}(7) = \bot$, hence node 7 is unreachable. In particular, the assertion $\mathsf{assert}\ (a * x = 1)$ must succeed.

4.4 Related Work

Modular domains Momentum for migrating abstract domains from idealised arithmetic to machine arithmetic is growing [26, 33, 32, 48, 64, 65], driven by the desire to soundly model program behaviour, particularly with regards to low-level code. Invariants over fixed-width integers can be represented with machine integers, which can speed up domain operations and allow coefficients, constants and bounds to be stored in constant space reducing overall memory consumption.

Polynomial invariants Early work [63, 71] on deriving polynomial invariants use polynomials with symbolic coefficients whose degree is fixed a priori. These works provide iterative [63] and direct constraint-solving [71] methods for inferring polynomial invariants, the former propagating polynomial preconditions against the control-flow, using Buchberger's algorithm to test for loop stability. The latter method [71] instead uses polynomial templates for invariants whose coefficients are linear expressions over template variables. Parametric linear equalities are solved and, where necessary, cylindrical algebraic decomposition methods [11] are applied to compute the coefficients. Neither method is complete and [63] conclude, "It is a challenging open problem whether or not the set of all polynomial relations can be computed not just ones of some given form".

This challenge [63] has motivated subsequent work [41, 42, 70, 50], which restrict the form of programs that can be analysed, either to those containing only simple loops [70], P-solvable loops [50] or affine programs [41]. Simple loops [70] are loops for which the body is a set of alternative assignments where each assignment simultaneously updates a subset of variables with an affine map on that subset summed with a polynomial over the other variables. Matrices which encode the affine maps are required to have positive rational eigenvalues, but the conjecture [70, Section 9] is that this is not necessary for termination.

P-solvable loops [50] are a class of loop for which the values of the variables can be expressed as polynomials of the initial values, loop counters and exponents of the counters. Polynomial invariants can be derived by solving recurrence equations in the loop counters, and then eliminating the counters and exponential terms [49]. The approach has been generalised to extended P-solvable loops, which allow

multiplication between program variables and the loop counter [42], though this generates more complex recurrences again.

State-of-the-art in computing all polynomial relations focuses on affine programs [41] where a variable is assigned to an affine expression. The problem is reduced by an ingenious construction to that of computing the Zariski closure of the semigroup generated by a finite set of rational square matrices. However, it is not clear how this approach extends to general polynomial assignments, particularly those in a modulo setting. It is also not evident how the construction can be combined with a conventional fixpoint engine which traces invariants in the direction of control-flow, typically using an approximate reduced cardinal product to combine numeric and symbolic domains [17].

In an attempt to side-step Gröbner bases [57], linear algebra [9, 21] has been proposed for inferring polynomial invariants, again at the cost of bounding the degree of the invariants. However, as [13] notes, even work over polynomial ideals of unrestricted degree is not sufficient for completeness since ideals should ideally be closed under radicals. Thus, if an ideal includes the polynomial p^d for some power d then it should include p, which is not dissimilar to our closure operator.

4.5 Concluding Discussion

Although Müller-Olm and Seidl [63] effectively threw down the gauntlet on the problem of how to compute the set of all polynomial relations, one solution – that set out in this chapter – has its roots in their own work on the (linear) analysis of modular arithmetic [64, 65], as is reflected in the title of our work on MPAD.

Working over modular integers is not merely more realistic, but reshapes the domain operations which can and need be applied. Widening is unnecessary because modular integers induce an abstract domain of polynomial invariants which satisfies the ascending chain condition. Conversely, negative polynomial guards can be supported by partitioning the solution set of a polynomial disequality into sets of integers whose least bits represent a power of two. To illustrate the novelty of this domain, and how it extends the scope of invariant discovery, we show how MPAD can be used to automatically derive a quadratic loop invariant for a classic algorithm for calculating the multiplicative inverse of a modular integer.

Chapter 5

Domain Operation Algorithms for MPAD

The design of an abstract domain divides into two phases: the specification of the high-level domain operations such as meet and join, and then the detailed algorithmics of how the operations are actually realised. But domain operations do not exist in isolation and there is often latitude to shift the complexity, whether conceptual or computational, from one operation and into another. MPAD adopts a centralised architecture, akin to that used in Octagons [59], in which there is a single auxiliary domain operation, closure, that streamlines and supports the other domain operations. Closure then localises and encapsulates much of the complexity of the domain.

Closure is formulated in terms of join, which itself is reduced to variable elimination that is, in turn, calculated using Gröbner bases. The development of the domain operations is thus layered: commencing with variable elimination, then moving onto join before majoring on closure itself.

Once closure is in place, the transfer functions of MPAD can be formulated in a natural and systematic fashion by combining variable elimination with closure. When polynomial assignment is constructed in this way, it even comes with an optimality guarantee, which generalises a result for affine approximation [62]. Rather surprisingly, MPAD provides transfer functions for both positive and negative polynomial guards, that is, statements of the form **assume** $(p = 0)$ and

assume $(p \neq 0)$ where p is an arbitrary polynomial. Thus the abstract transfer functions which result from this development are unusually rich. To summarise, this chapter makes the following contributions:

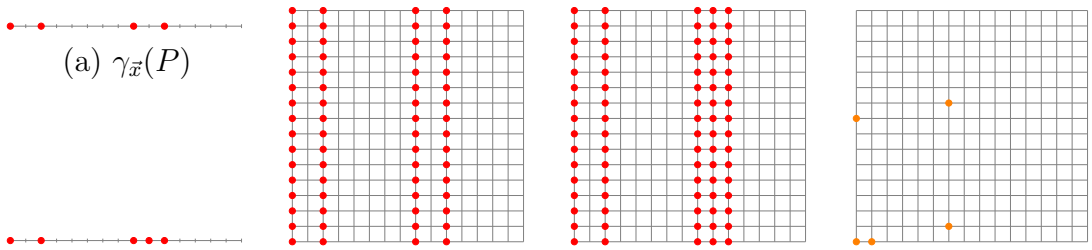
- We introduce variable elimination, demonstrating how its calculation can be reduced to the calculation of a Gröbner basis;
- We introduce the notion of covers, presenting a divide-and-conquer algorithm for computing them and introducing reductions and shortcuts that simplify its calculation;
- We demonstrate how closure can be computed for the individual sub-systems of a cover, in a new application for null-polynomials. Thus, closure can be computed from a cover of a system;
- We demonstrate how the abstract transfer functions for MPAD can be reduced to variable elimination and closure. In particular, we show how polynomial disequalities can be handled by a novel partitioning scheme;
- We show that polynomial assignment is optimal. Coupled with the finiteness of MPAD, it follows that MPAD will infer all polynomial invariants for programs consisting solely of polynomial assignments.

5.1 Calculating variable elimination and join

Variable elimination is fundamental to computing the domain operations of MPAD, many operations deploying it or reducing to it, join being one such example. This section explains how variable elimination can be computed using Gröbner bases, and how variable elimination can be combined with a relaxation to compute the join of two ideals finitely represented as bases.

5.1.1 Concretisation and closure: reprise

A generic projection function $\pi_i(\langle a_1, \ldots, a_\ell \rangle) = \langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_\ell \rangle$ is used to formulate elimination. It maps a vector of dimension ℓ to another of dimension



(a) $\gamma_{\vec{x}}(P)$ (b) $\gamma_{\vec{x}}(Q)$ (c) $\gamma_{\vec{y}}(P)$ (d) $\gamma_{\vec{y}}(Q)$ (e) $\gamma_{\vec{w}}(B)$

Figure 15: Concretisation of $P = \{x^2 + 14x\}$, $Q = \{x^3 + 5x^2 + 2x\}$ and $B = \{wx + 10w, 15wx^2 + wx + x^2 + 15x\}$ for $\vec{x} = \langle x \rangle$ and $\vec{y} = \langle x, y \rangle$ and $\vec{w} = \langle x, w \rangle$

$\ell - 1$. Elimination and join likewise relate objects of different dimensionality. The elimination of x_j from a system $P \subseteq \mathbb{Z}_m[\vec{x}]$ is the derivation of a system $S \subseteq \mathbb{Z}_m[\vec{y}]$ where $\vec{y} = \pi_j(\vec{x})$. Hence $|\vec{y}| = |\vec{x}| - 1$. The join of two systems $P, Q \subseteq \mathbb{Z}_m[\vec{x}]$ is calculated using a relaxation $R \subseteq \mathbb{Z}_m[\vec{y}]$ where $\vec{y} = \langle w, x_1, \ldots, x_d \rangle$. Hence $|\vec{y}| = |\vec{x}|+1$. To reason about variable elimination and join it is therefore necessary to relate systems over $\vec{x}$ and $\vec{y}$ where $\vec{x} \sqsubseteq \vec{y}$ and $|\vec{x}| \neq |\vec{y}|$. Here, as before, $\sqsubseteq$ denotes the subsequence relation. To this end, the first two results assert how the subset ordering between $\gamma_{\vec{x}}(P)$ and $\gamma_{\vec{x}}(Q)$ for $P, Q \subseteq \mathbb{Z}_m[\vec{x}]$ is preserved by $\gamma_{\vec{y}}(P)$ and $\gamma_{\vec{y}}(Q)$ when $\vec{x} \sqsubseteq \vec{y}$. The second is an immediate consequence of the first.

Lemma 6. If $\vec{x} = \pi_i(\vec{y})$ and $P, Q \subseteq \mathbb{Z}_m[\vec{x}]$ such that $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(Q)$ then $\gamma_{\vec{y}}(P) \subseteq \gamma_{\vec{y}}(Q)$.

Corollary 3. If $\vec{x} \sqsubseteq \vec{y}$ and $P, Q \subseteq \mathbb{Z}_m[\vec{x}]$ such that $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(Q)$ then $\gamma_{\vec{y}}(P) \subseteq \gamma_{\vec{y}}(Q)$.

Example 26. Let $\vec{x} = \langle x \rangle$ and $\vec{y} = \langle x, y \rangle$ so that $\vec{x} \sqsubseteq \vec{y}$ and consider $P = \{x^2 + 14x\}$ and $Q = \{x^3 + 5x^2 + 2x\}$. Fig. 15(a)–(d) illustrate $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(Q)$ and how it is mirrored by $\gamma_{\vec{y}}(P) \subseteq \gamma_{\vec{y}}(Q)$. Fig. 15(e) will be discussed below.

Closure $\uparrow_{\vec{x}}$ is defined in terms of $\gamma_{\vec{x}}$ and thus is also parameterised by $\vec{x}$. The next result, which is a direct consequence of the above corollary, explains how subset ordering between $\uparrow_{\vec{x}}(P)$ and $\uparrow_{\vec{x}}(Q)$ over $\vec{x}$ is likewise preserved by $\uparrow_{\vec{y}} P$ and $\uparrow_{\vec{y}} Q$ when $\vec{x} \sqsubseteq \vec{y}$. Together, these results show how concretisation $\gamma_{\vec{x}}$, which provides an interpretation for polynomials of $\mathbb{Z}_m[\vec{x}]$, and closure $\uparrow_{\vec{x}}$, which provides a representation for them, both extend from $\vec{x}$ to $\vec{y}$ where $\vec{x} \sqsubseteq \vec{y}$.

Corollary 4. $P, Q \subseteq \mathbb{Z}_m[\vec{x}]$, $\uparrow_{\vec{x}} P \subseteq \uparrow_{\vec{x}} Q$ and $\vec{x} \sqsubseteq \vec{y}$ then $\uparrow_{\vec{y}} P \subseteq \uparrow_{\vec{y}} Q$.

5.1.2 Variable elimination

The presentation of elimination itself commences with a syntactic form of variable elimination, which simply removes polynomials that contains a given variable.

Definition 15. (Syntactic) variable elimination $\mathsf{elim}[x_j] : \wp(\mathbb{Z}_m[\vec{x}]) \to \wp(\mathbb{Z}_m[\pi_j(\vec{x})])$ is defined

$$\mathsf{elim}[x_j](P) = P \cap \mathbb{Z}_m[\pi_j(\vec{x})]$$

The following result demonstrates that abstraction and elimination commute. The result is formulated in terms of the natural lifting of π_j from the function space $\mathbb{Z}_m^d \to \mathbb{Z}_m^{d-1}$ to $\wp(\mathbb{Z}_m^d) \to \wp(\mathbb{Z}_m^{d-1})$.

Proposition 8. If $A \subseteq \mathbb{Z}_m^d$ then $\mathsf{elim}[x_j](\alpha_{\vec{x}}(A)) = \alpha_{\pi_j(\vec{x})}(\pi_j(A))$.

It follows from this result that elimination preserves closure:

Corollary 5. If $P \in \mathsf{MPAD}_m[\vec{x}]$ then $\mathsf{elim}[x_j](P) \in \mathsf{MPAD}_m[\pi_j(\vec{x})]$.

Example 27. Consider $B = \{wx + 10w, 15wx^2 + wx + x^2 + 15x\} \subseteq \mathbb{Z}_{16}[w, x]$ and observe $\mathsf{elim}[w](B) = \emptyset$. However $(x^2 + 7x + 8)(wx + 10w) + (x + 2)(15wx^2 + wx + x^2 + 15x) = x^3 + x^2 + 14x$ hence $x^3 + x^2 + 14x \in \langle B \rangle_{\langle w,x \rangle}$. Since $w \notin \mathsf{vars}(x^3 + x^2 + 14x)$ it follows $x^3 + x^2 + 14x \in \mathsf{elim}[w](\langle B \rangle_{\langle w,x \rangle})$. In particular, $\mathsf{elim}[w](\langle B \rangle_{\langle w,x \rangle}) \neq \{0\} = \langle \emptyset \rangle_{\langle x \rangle} = \langle \mathsf{elim}[w](B) \rangle_{\langle x \rangle}$.

The previous example shows that syntactic variable elimination is not well-behaved with respect to ideal generation, thus motivating the following definition:

Definition 16. (Semantic) variable elimination is a relation $\to_{\mathsf{elim}[x_j]} \subseteq \wp(\mathbb{Z}_m[\vec{x}]) \times \wp(\mathbb{Z}_m[\pi_j(\vec{x})])$ defined $B \to_{\mathsf{elim}[x_j]} B'$ iff $\mathsf{elim}[x_j](\langle B \rangle_{\vec{x}}) = \langle B' \rangle_{\pi_j(\vec{x})}$.

Proposition 9. Let $B \subseteq \mathbb{Z}_m[\vec{x}]$ and B' be a Gröbner basis for $\langle B \rangle_{\vec{x}}$ with respect to $\prec_{\vec{y}}$ where $\vec{y}$ is a permutation of $\vec{x}$ and $y_1 = x_j$. Then $B \to_{\mathsf{elim}[x_j]} \mathsf{elim}[x_j](B')$.

The previous result can be stated more generally in terms of elimination orderings [2]; the restriction to lexicographical ordering is adopted merely to simplify the presentation. Consistent with this choice, $\mathbf{gb}_{\prec_{\vec{y}}}$ is henceforth abbreviated to $\mathbf{gb}_{\vec{y}}$, again purely to streamline the exposition.

Example 28. Let $B = \{wx + 10w, 15wx^2 + wx + x^2 + 15x\} \subseteq \mathbb{Z}_{16}[w, x, y]$. Then,

$$\mathbf{gb}_{\langle w,x,y\rangle}(B) = \{wx + 3x^2 + 13x, 2w + x^2 + 15x, x^3 + x^2 + 14x\}$$

It follows $B \to_{\mathsf{elim}[w]} \{x^3 + x^2 + 14x\}$.

Example 29. Let $B = \{w(x+3), w(y+9), (1-w)(x+6), (1-w)(y+2)\}$. Then,

$$\begin{aligned}\mathbf{gb}_{\langle w,x,y\rangle}(B) &= \{w + 7y + 14, \quad x + 5y, \quad y^2 + 11y + 2\}\\ \mathbf{gb}_{\langle w,y,x\rangle}(B) &= \{w + 5x + 14, \quad y + 13x, \quad x^2 + 9x + 2\}\end{aligned}$$

Thus $B \to_{\mathsf{elim}[w]} B'$ and $B \to_{\mathsf{elim}[w]} B''$ where $B' = \{x + 5y, y^2 + 11y + 2\}$ and $B'' = \{y + 13x, x^2 + 9x + 2\}$ illustrating why $\to_{\mathsf{elim}[w]}$ is defined as a relation. To see $\langle B'\rangle_{\langle x,y\rangle} = \langle B''\rangle_{\langle x,y\rangle}$ observe $x + 5y \to_{y+13x} 0$ and

$$y^2{+}11y{+}2 \to_{y+13x} 3xy{+}11y{+}2 \to_{y+13x} 9x^2{+}11y{+}2 \to_{x^2+9x+2} 15x{+}11y \to_{y+13x} 0$$

Similarly, $p \to_{B'} 0$ for all $p \in B''$.

5.1.3 Join

Once variable elimination is in place, join can be calculated by adapting a standard relaxation [2] to the current setting. The result, which provides a way of intersecting ideals, hence calculating join, is stated in terms of a lifted product $qP = \{qp \mid p \in P\}$ where $P \subseteq \mathbb{Z}_m[\vec{x}]$ and $q \in \mathbb{Z}_m[\vec{x}]$:

Proposition 10. If $w \notin \mathsf{vars}(B_1 \cup B_2)$ then $\langle B_1\rangle_{\vec{x}} \cap \langle B_2\rangle_{\vec{x}} = \langle B\rangle_{\vec{x}}$ whenever

$$wB_1 \cup (1 - w)B_2 \to_{\mathsf{elim}[w]} B$$

Example 30. Let $\vec{x} = \langle x, y\rangle$ and $B_1, B_2 \subseteq \mathbb{Z}_{16}[\vec{x}]$ where $B_1 = \{x + 10\}$, $B_2 = \{x^2 + 15x\}$ and $I_i = \langle B_i\rangle_{\vec{x}}$. Both I_i are closed, that is, $I_i = \uparrow I_i$. Let

$$B = wB_1 \cup (1 - w)B_2 = \{wx + 10w, 15wx^2 + wx + x^2 + 15x\}$$

From example 27, $B \to_{\mathsf{elim}[w]} \{x^3{+}x^2{+}14x\}$, hence $\langle B_1\rangle_{\vec{x}} \sqcup \langle B_2\rangle_{\vec{x}} = \langle x^3{+}x^2{+}14x\rangle_{\vec{x}}$.

Figs. 16(a), 16(b) and 16(i) depict $\gamma_{\vec{x}}(I_1)$, $\gamma_{\vec{x}}(I_2)$ and $\gamma_{\vec{x}}(I_1 \sqcup I_2)$ respectively. Observe $(8, y) \in \gamma_{\vec{x}}(I_1 \sqcup I_2)$ but $(8, y) \notin \gamma_{\vec{x}}(I_1) \cup \gamma_{\vec{x}}(I_2)$ for any $y \in \mathbb{Z}_{16}$. These additional points, which are introduced by join itself, stem not from the relaxation $wB_1 \cup (1 - w)B_2$ which introduces w, but the elimination of w from $\mathbf{gb}_{\langle w,x,y\rangle}(B)$ which derives a unary polynomial representation over x alone. To see this, Fig. 15(e) depicts $\gamma_{\vec{w}}(B)$, where $\vec{w} = \langle x, w\rangle$ and the w-axis is vertical and the x-axis is horizontal. Observe that $\gamma_{\vec{w}}(B)$ contains points with x-coordinates of 0, 1 and 6, and no others. These x-coordinates concur with $\gamma_{\vec{x}}(I_1)$ and $\gamma_{\vec{x}}(I_2)$. But $\alpha_{\vec{x}}(A) = \uparrow_{\vec{x}} \{x^3 + x^2 + 14x\}$ for $A = \{(x, y) \in \mathbb{Z}_{16}^2 \mid x = 0 \vee x = 1 \vee x = 6\}$. Thus there is no better unary polynomial representation of A than $\{x^3 + x^2 + 14x\}$. In particular, $\gamma_{\vec{x}}(I_1 \sqcup I_2)$ cannot exclude points $(8, y)$ for $y \in \mathbb{Z}_{16}$.

Example 31. Fig. 16 presents a series of examples of join on $\mathbb{Z}_{16}[\vec{x}]$ for $\vec{x} = \langle x, y\rangle$. Figs. 16(a) - (h) depict $\gamma_{\vec{x}}(I_i)$ for $I_i = \langle B_i\rangle_{\vec{x}}$ where $I_i = \uparrow I_i$ and B_i are as follows:

$$B_3 = \left\{x + 3, \quad y + 9\right\} \qquad B_6 = \left\{x^2, \quad xy^4 + xy^2 + 2xy, \quad 2xy^2 + 2xy, \quad 4x\right\}$$
$$B_4 = \left\{x + 6, \quad y + 2\right\} \qquad B_7 = \left\{x^4y + x^2y + 2xy, \quad 2x^2y + 2xy, \quad y^2, \quad 4y\right\}$$
$$B_5 = \left\{x^2, \quad 4x, \quad y\right\} \qquad B_8 = \left\{x + y\right\}$$

For comparison, the yellow points give the best abstraction of $\gamma_{\vec{x}}(I_i)$ using systems of linear congruences modulo 16 (linear polynomials).

Figs. 16(i) - (p) depict $\gamma_{\vec{x}}(I_i \sqcup I_j)$ for various combinations of $i, j \in \{1, 8\}$, illustrating where a polynomial representation introduces additional points through join. Again, the yellow points give the join of the best linear abstractions, which can be computed by combining a relaxation with variable elimination [48]. To illustrate the working, consider B_3 and B_4 rewritten as follows:

$$B_3 = \begin{cases} x \equiv_{16} -3 \\ y \equiv_{16} -9 \end{cases} \qquad B_4 = \begin{cases} x \equiv_{16} -6 \\ y \equiv_{16} -2 \end{cases}$$

The relaxation introduces fresh variables x', y', x'', y'' and μ:

$$x \equiv_{16} x' + x'' \qquad x' \equiv_{16} -3\mu \qquad x'' \equiv_{16} -6(1 - \mu)$$
$$y \equiv_{16} y' + y'' \qquad y' \equiv_{16} -9\mu \qquad y'' \equiv_{16} -2(1 - \mu)$$

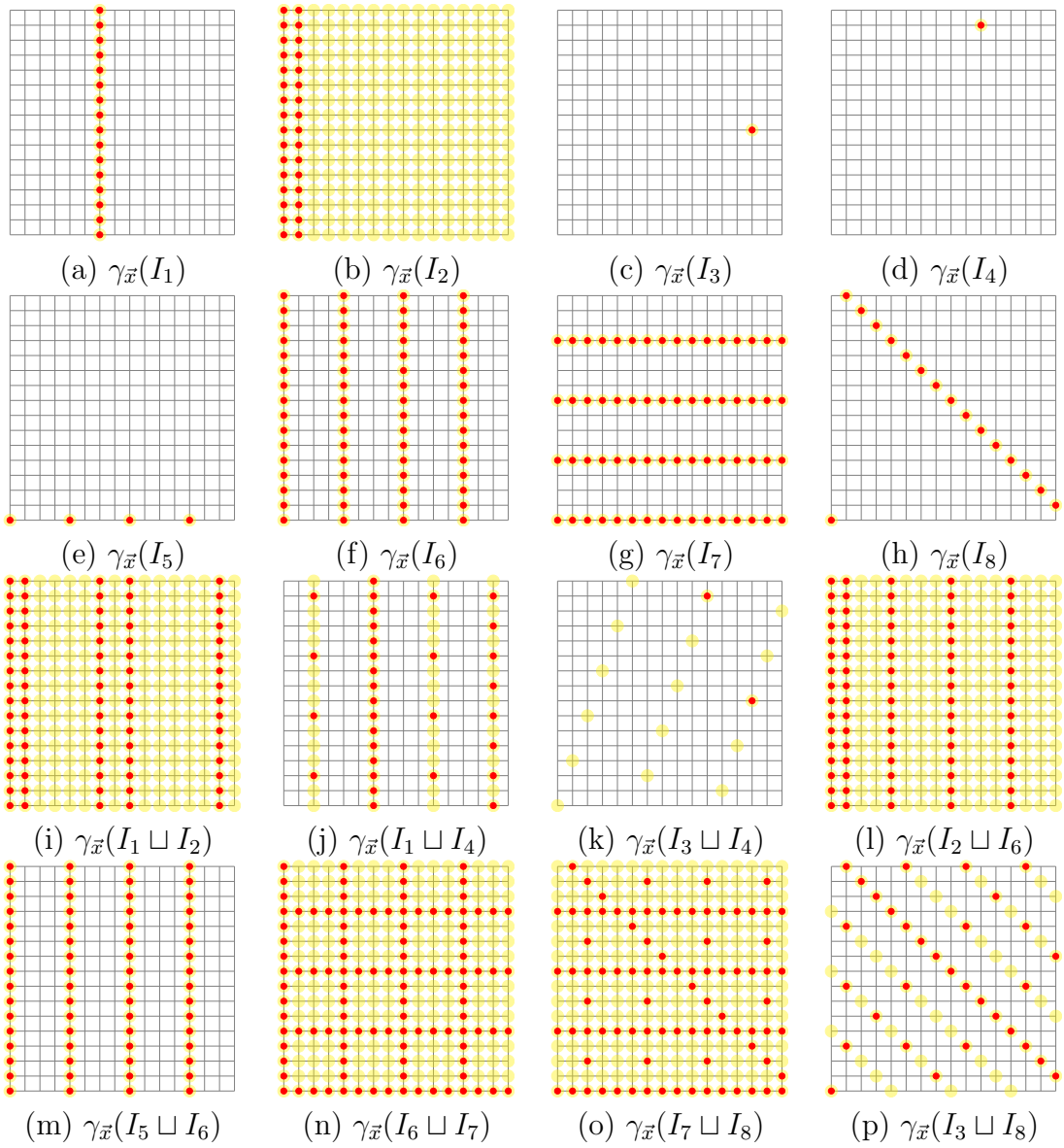
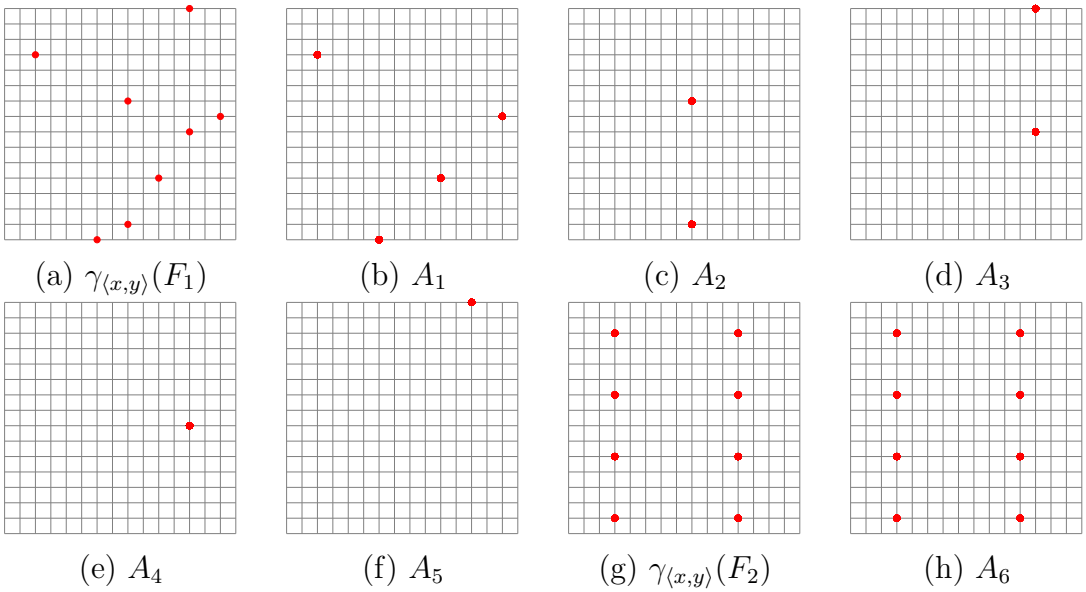


Figure 16: Examples of join on $\mathbb{Z}_{16}[\vec{x}]$ for $\vec{x} = \langle x, y \rangle$

Eliminating x', y', x'' and y'' gives a system of two congruences: $x \equiv_{16} 3\mu - 6$ and $y \equiv_{16} -7\mu - 2$. Rearranging for μ gives $\mu \equiv_{16} 2 - 5x$ hence $y \equiv_{16} 3x$ as illustrated in Figure 16(k). The other linear joins are computed likewise.

Note in particular the loss of precision in using linear, rather than polynomial, abstractions. For instance, the set $\gamma_{\vec{x}}(I_2)$ can only be approximated by a trivial (unconstrained) linear system, which leads to a complete loss of information.



(a) $\gamma_{\langle x,y\rangle}(F_1)$ (b) A_1 (c) A_2 (d) A_3

(e) A_4 (f) A_5 (g) $\gamma_{\langle x,y\rangle}(F_2)$ (h) A_6

Figure 17: Covers of F_1 over $\langle w_1\rangle$ and F_2 over $\langle w_1, w_2\rangle$

Moreover, as demonstrated in Figs. 16(j) - (k) and Figs. 16(n) - (p), even if the arguments to a (polynomial) join are precisely representable via linear systems, the result may not be. This has particular consequences for abstract interpretation, where joins typically arise from a merge of control flow, for instance at the entry point of a loop. When employing a linear abstraction [65], the presence of such merge points can lead to a significant loss of precision compared to the corresponding polynomial abstraction.

5.2 Calculating closure and meet

This section addresses how to finitely compute closure. The problem is reduced to that of computing a cover of a system of polynomials. A cover provides a way to decompose closure to sub-problems for which closure can be computed directly. A divide-and-conquer algorithm is introduced for computing a cover, which exploits a simplification procedure based on Gröbner bases, to avoid superfluous work. The section concludes by showing how meet can be computed using closure.

5.2.1 Covering

An algorithm for computing closure is formulated in terms of the concept of a cover, which is itself defined through a pointwise lifting of polynomial evaluation $[\![p]\!]_{\vec{x}}(\vec{a})$ to a vector of polynomials $\vec{p} = \langle p_1, \ldots, p_n \rangle$ by $[\![\vec{p}]\!]_{\vec{x}}(\vec{a}) = \langle [\![p_1]\!]_{\vec{x}}(\vec{a}), \ldots, [\![p_n]\!]_{\vec{x}}(\vec{a}) \rangle$.

Definition 17. Let $\mathcal{W} \subseteq \mathbb{Z}_m[\vec{w}]^d$, $A \subseteq \mathbb{Z}_m^d$ and $F \subseteq \mathbb{Z}_m[\vec{x}]$. Then

- $\mathcal{W}$ is a cover of A over $\vec{w}$ iff $A = \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{W} \in \mathcal{W} \wedge \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\}$
- $\mathcal{W}$ is a cover of F over $\vec{w}$ iff $\mathcal{W}$ is a cover of $\gamma_{\vec{x}}(F)$ over $\vec{w}$

Example 32. Figs. 17(a) and (e) depict $\gamma_{\vec{x}}(F_1)$ and $\gamma_{\vec{x}}(F_2)$ for $\vec{x} = \langle x, y \rangle$ where

$$F_1 = \begin{Bmatrix} x + 3y^3 + 4y^2 + 7y + 10, \\ y^4 + 7y^2 + 8y \end{Bmatrix} \qquad F_2 = \begin{Bmatrix} 2x + 10, \\ 4y + 12 \end{Bmatrix}$$

Figs. 17(b), (c) and (d) illustrate $A_i = \{[\![\vec{W}_i]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^1\}$ for $\vec{w} = \langle w_1 \rangle$ where

$$\vec{W}_1 = \langle 4w_1 + 6, 4w_1 \rangle \qquad \vec{W}_2 = \langle 8, 8w_1 + 1 \rangle \qquad \vec{W}_3 = \langle 12, 8w_1 + 7 \rangle$$

Observe $\{\vec{W}_i\}$ is a cover of A_i and since $\gamma_{\vec{x}}(F_1) = A_1 \cup A_2 \cup A_3$, $\{\vec{W}_1, \vec{W}_2, \vec{W}_3\}$ is a cover of F_1 over $\vec{w}$. The set of 4 vectors $\{\vec{W}_1, \vec{W}_2, \vec{W}_4, \vec{W}_5\}$ where $\vec{W}_4 = \langle 12, 7 \rangle$ and $\vec{W}_5 = \langle 12, 15 \rangle$ is also a cover of F_1, illustrating that covers are not unique. The polynomial vectors $\vec{W}_4$ and $\vec{W}_5$ define single points and suggest how a cover can be constructed for an arbitrary $F \subseteq \mathbb{Z}_m[\vec{w}]$ by putting $\mathcal{W} = \{\vec{a} \mid \vec{a} \in \gamma_{\vec{x}}(F)\}$. The vector $\vec{w}$ is not necessarily unary as the cover $\{\vec{W}_6\}$ of F_2 over $\vec{w} = \langle w_1, w_2 \rangle$ illustrates where $\vec{W}_6 = \langle 8w_1 + 3, 4w_2 + 1 \rangle$ and $\gamma_{\vec{x}}(F_2) = A_6 = \{[\![\vec{W}_6]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^2\}$, and $\gamma_{\vec{x}}(F_2)$ and A_6 are illustrated in Figs. 17(g) and (h) respectively.

The challenge is compute a cover over some $\vec{w}$ for arbitrary $F \subseteq \mathbb{Z}_m[\vec{x}]$ without naively enumerating all points of $\gamma_{\vec{x}}(F)$. To this end, Fig. 18 presents a divide-and-conquer algorithm that recursively decomposes $\gamma_{\vec{x}}(F)$ into subsets following the structure of F. Ultimately the function computes a cover $\mathcal{W} \subseteq \mathbb{Z}_m[\vec{w}]^d$ for F over $\vec{w}$ where $|\vec{w}| = d = |\vec{x}|$. The function **cover** depends on three auxiliary functions, **simplify constrain** and **safe** all of which are listed in Fig. 19. The function **cover** and its auxiliaries operate on pairs $S = \langle \vec{W}, F \rangle$ where $\vec{W} \in \mathbb{Z}_m[\vec{w}]^d$

function cover$(F \subseteq \mathbb{Z}_m[\vec{x}])$
begin
  let $\vec{w} = \langle w_1, \ldots, w_d \rangle$
  return cover$(\vec{w}, F[x_1 \mapsto w_1, \ldots, x_d \mapsto w_d])$
end
function cover$(S \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}]))$
begin
  $S' =$ **simplify**(S)
  if $(S' =$ **nil**$)$ **return** $\emptyset$
  else
    let $S' = \langle \vec{W}, F \rangle$
    if $(F = \emptyset)$ **return** $\{\vec{W}\}$
    else
      let $w_i \in \mathsf{vars}(F)$
      $S'_0 =$ **constrain**$(S', 1, w_i, 0)$ (* $F \cup \{w_i - 2^1 w\}$ *)
      $S'_1 =$ **constrain**$(S', 1, w_i, 1)$ (* $F \cup \{w_i - 2^1 w + 1\}$ *)
      return cover$(S'_0) \cup$ **cover**(S'_1)
    end if
  end if
end

Figure 18: The **cover** algorithm

is a vector of polynomials and $F \subseteq \mathbb{Z}_m[\vec{w}]$ is a system. The vector $\vec{W}$ provides a lens to interpret the solutions of F, as formalised in the following:

Definition 18. The concretisation map $\gamma_{\vec{w}} : \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}]) \to \mathbb{Z}_m^d$ is defined:

$$\gamma_{\vec{w}}(\langle \vec{W}, F \rangle) = \{ [\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \gamma_{\vec{w}}(F) \}$$

Example 33. Consider $S_b = \langle \vec{W}_b, F_b \rangle$ and $S_c = \langle \vec{W}_c, F_c \rangle$, where $\vec{W}_b = \langle w_1, 2w_2 \rangle$, $\vec{W}_c = \langle w_1, 4w_2 \rangle$ and

$$F_b = \begin{Bmatrix} w_1^2 + w_1 + 6w_2 + 12, \\ 2w_1w_2 + 4w_1, \quad 4w_2^2, \quad 8w_2 \end{Bmatrix} \qquad F_c = \begin{Bmatrix} w_1^2 + w_1 + 12w_2 + 12, \\ 4w_1w_2 + 4w_1 \end{Bmatrix}$$

Fig. 20(b) illustrates $\gamma_{\vec{w}}(F_b)$ as large, translucent points and $\gamma_{\vec{w}}(S_b)$ as small, opaque points. Observe $\langle 8, 2 \rangle, \langle 8, 10 \rangle \in \gamma_{\vec{w}}(F_b)$ and $[\![\vec{W}]\!]_{\vec{w}}(\langle 8, 2 \rangle) = \langle 8, 4 \rangle =$

function simplify$(\langle \vec{W}, F\rangle \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}]))$
begin
  $F' = \mathbf{gb}_{\vec{w}}(F)$
  $S' = \langle \vec{W}, F'\rangle$
  if $(c \in F'$ **where** $c \in \mathbb{Z}_m \setminus \{0\})$
    return nil
  else if $(2^{\omega-j}(w_i + r) \in F'$ **where** $j > 0 \wedge r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d] \wedge$ **safe**$(\vec{W}, w_i, r))$
    $S'' =$ **constrain**(S', j, w_i, r) (* $F \cup \{w_i - 2^j w + r\}$ *)
    return simplify(S'')
  else
    return S'
  end if
end
function constrain$(\langle \vec{W}, F\rangle \in \mathbb{Z}_m[\vec{w}]^d \times \mathbb{Z}_m[\vec{w}], j \in \mathbb{N}, w_i \in \vec{w}, r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d])$
begin
  $F \cup \{w_i - 2^j w + r\} \rightarrow_{\mathsf{elim}[w_i]} F'$
  $\vec{W}' = \vec{W}[w_i \mapsto 2^j w - r]$
  if $(W_i' = 2^\omega w + q \wedge q \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d])$ $F'' = F'[w \mapsto 0]$
  else $F'' = F'[w \mapsto w_i]$
  return $\langle \vec{W}'[w \mapsto w_i], F''\rangle$
end
function safe$(\vec{W} \in \mathbb{Z}_m[\vec{w}]^d, w_i \in \vec{w}, r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d])$
begin
  let $\vec{W} = \langle 2^{k_1} w_1 + q_1, \ldots, 2^{k_d} w_d + q_d\rangle$
  if $(c\vec{y}^{\vec{\alpha}} \in r, w_\ell \in \mathsf{vars}(\vec{y})$ **where** $k_i + \mathsf{rank}(c) < k_\ell)$ **return false**
  else return true
end

Figure 19: The **simplify**, **constrain** and **safe** functions

$[\![\vec{W}]\!]_{\vec{w}}(\langle 8, 10\rangle)$ hence, in general, there is many-to-one relationship between $\gamma_{\vec{w}}(F_b)$ and $\gamma_{\vec{w}}(S_a)$. Fig. 20(c) depicts $\gamma_{\vec{w}}(F_c)$ and $\gamma_{\vec{w}}(S_c)$ using the same convention. Observe too that $\gamma_{\vec{w}}(S_b) = \gamma_{\vec{w}}(S_c)$ but the cardinality of $\gamma_{\vec{w}}(F_c)$ is 4-fold that of $\gamma_{\vec{w}}(S_c)$ since $\vec{W}_c = \langle w_1, 4w_2\rangle$.

Observe that if $\mathcal{W} \subseteq \mathbb{Z}_m[\vec{w}]^d$ is a cover for $F \subseteq \mathbb{Z}_m[\vec{x}]$ over $\vec{w}$ then $\gamma_{\vec{x}}(F) = \cup\{\gamma_{\vec{w}}(\langle \vec{W}, \emptyset\rangle) \mid \vec{W} \in \mathcal{W}\}$. Thus a cover is formed from pairs $\langle \vec{W}, F\rangle$ that are degenerate in the sense that $F = \emptyset$. The rationale behind **cover** is therefore to

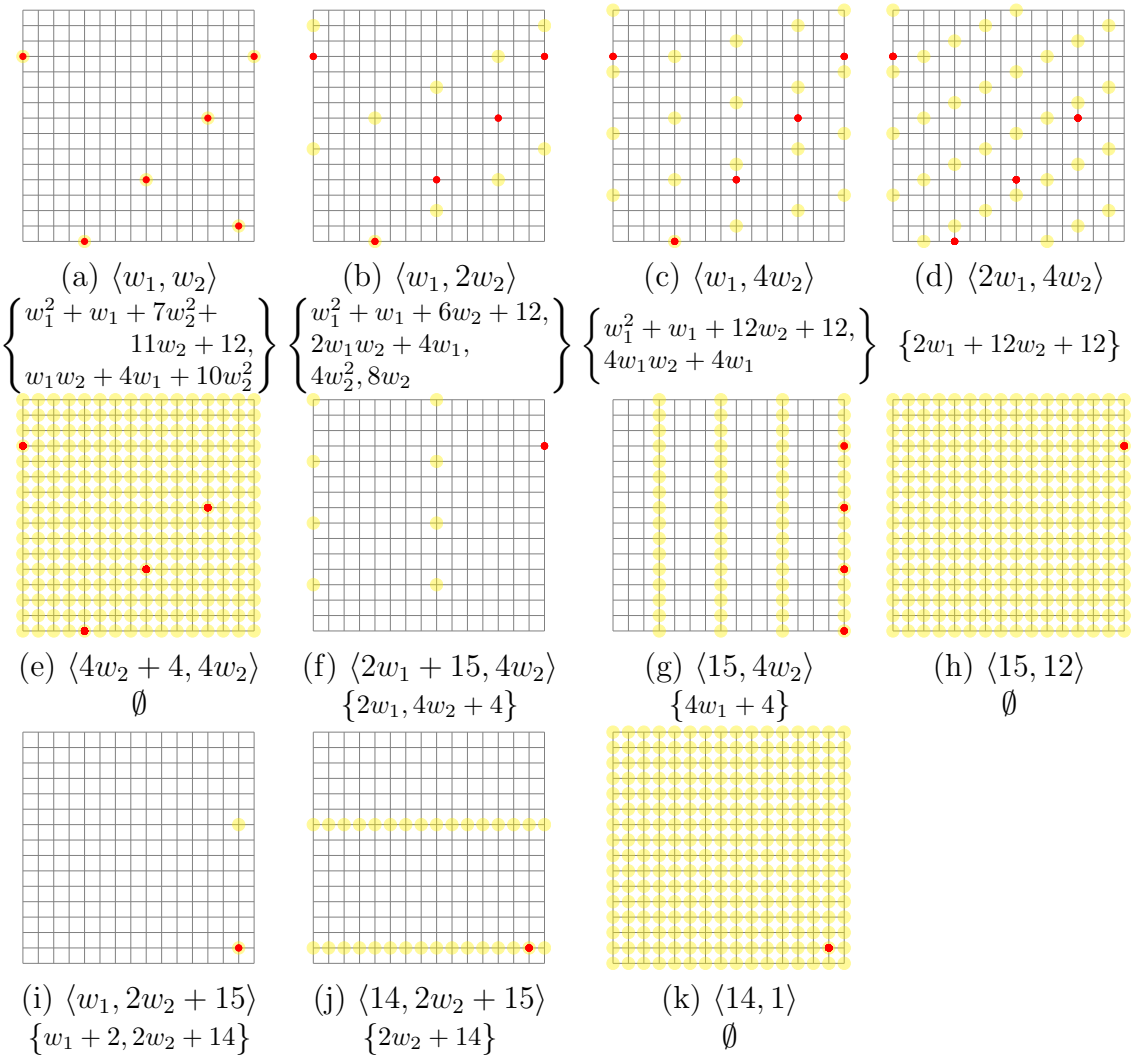


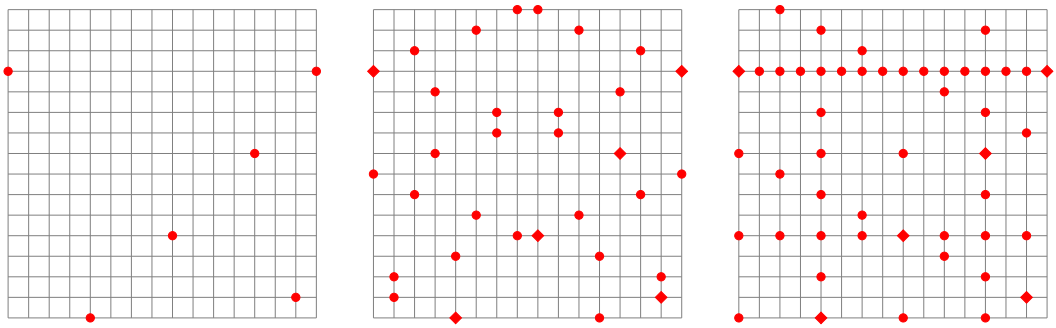
Figure 20: Covering F: $\gamma_{\vec{y}}(F_n)$ (large, translucent points) and $\gamma_{\vec{y}}(S_n)$ (small, opaque points) for $S_n = \langle \vec{W}_n, F_n\rangle$

decompose a single pair $\langle \vec{W}, F\rangle$ where $\vec{W} = \vec{w}$ into a collection of degenerate pairs:

Example 34. Consider computing a cover for the system

$$F = \left\{\begin{matrix} x^2 + x + 7y^2 + 11y + 12, \\ xy + 4x + 10y^2 \end{matrix}\right\}$$

over $\vec{w} = \langle w_1, w_2\rangle$. The set $\gamma_{\vec{x}}(F)$ is plotted in Figure 21(a). The top-level **cover**



(a) $\gamma_{\vec{x}}(F)$ (b) $\gamma_{\vec{x}}(F_3)$ (c) $\gamma_{\vec{x}}(F_4)$

Figure 21: Solution sets for F, F_3 and F_4

function expresses F as the pair $S_a = \langle \vec{W}_a, F_a \rangle$ where

$$\vec{W}_a = \vec{w} \qquad F_a = \left\{ \begin{array}{c} w_1^2 + w_1 + 7w_2^2 + 11w_2 + 12, \\ w_1 w_2 + 4w_1 + 10w_2^2 \end{array} \right\}$$

Since $\vec{W}_a$ is the identity, that is $[\![\vec{W}]\!]_{\vec{w}}(\vec{b}) = \vec{b}$ for all $\vec{b} \in \mathbb{Z}_m^2$, it follows $\gamma_{\vec{w}}(S_a) = \gamma_{\vec{x}}(F)$.

The **cover** function invokes both **simplify** and **constrain**. The function **simplify** performs simplification, either returning **nil**, indicating $\gamma_{\vec{w}}(\langle \vec{W}, F \rangle) = \emptyset$, or $S' = \langle \vec{W}', F' \rangle$ where $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S')$ (possibly with $S = S'$). The first substantive action of **simplify** is to calculate a Gröbner base F' for the ideal $\langle F \rangle_{\vec{w}}$ using the variable ordering $\vec{w}$. If there exists a constant polynomial $c \in F'$ such that $c \neq 0$ then this reveals $\gamma_{\vec{w}}(F) = \gamma_{\vec{w}}(F') = \emptyset$ hence $\gamma_{\vec{w}}(S) = \emptyset$. Otherwise, **constrain** is invoked if F' contains a polynomial of the form $2^{\omega - j}(w_i + r)$ where $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, $0 < j \leq \omega$ and the safety check **safe**$(\vec{W}, w_i, r)$ is satisfied. The added polynomial $w_i - 2^j w + r$ asserts that $w_i + r$ is a multiple of 2^j, which is a direct consequence of $2^{\omega - j}(w_i + r)$. The safety check ensures that the addition of $2^{\omega - j}(w_i + r)$ does not induce a coupling between the variables of $\vec{w}$, specifically those arising in r, that would compromise the termination argument behind **simplify** and **cover**. The safety check is vacuously satisfied if $\mathsf{vars}(r) = \emptyset$.

Simplification is used in tandem with splitting, the latter employed by **cover** only when the former cannot infer new information. When **constrain** is invoked from **cover**, two pairs S'_0 and S'_1 are derived from $S' = \langle \vec{W}', F' \rangle$ for which $\gamma_{\vec{w}}(S') =$

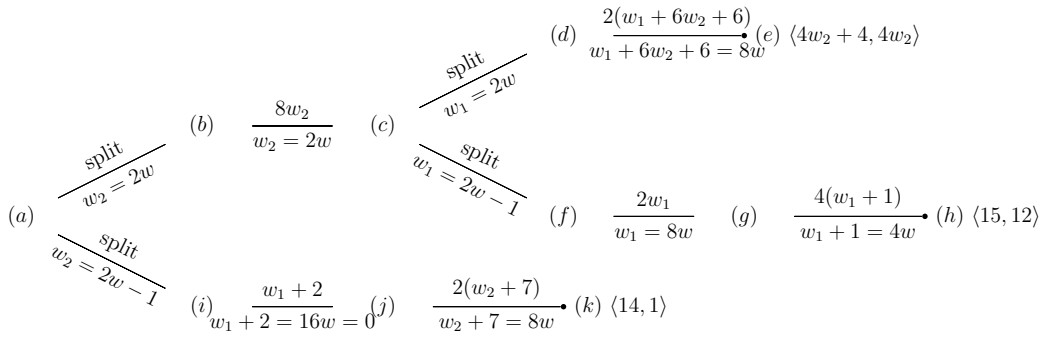


Figure 22: Covering F: the simplification and splitting actions

$\gamma_{\vec{w}}(S'_0) \cup \gamma_{\vec{w}}(S'_1)$. The pairs S'_0 and S'_1 are formed by adding $w_i - 2w + 0$ and $w_i - 2w + 1$ to F', which stipulate, respectively, whether w_i takes an even or an odd value. Note, in this case, **constrain**$(S', 1, w_i, r)$ is called with $\mathsf{vars}(r) = \emptyset$, hence **safe**$(\vec{W}, w_i, r)$ holds independently of $\vec{W}$ and w_i and need not be deployed within the body of **cover** itself. The **cover** function is then recursively applied to S'_0 and S'_1 to compute two covers, which are combined by set union. The function returns a singleton set $\{\vec{W}\}$ when $F = \emptyset$ (though the check $F = \emptyset$ can be relaxed to $F \subseteq \top$ to allow early termination for when F only contains null polynomials).

Example 35. Fig. 22 presents the simplification and splitting actions that arise during a run of the algorithm on the pair $S_a = \langle \vec{W}_a, F_a \rangle$ introduced in Example 34. The actions are presented as a tree rooted at node a where the leaves, nodes e, g and h, are each decorated with a single polynomial vector. Together these 3 vectors constitute the cover. Fig. 20 augments Fig. 22 with details of $S_n = \langle \vec{W}_n, F_n \rangle$ for each node n of the tree: $\vec{W}_n$ written above F_n. In each diagram $\gamma_{\vec{w}}(F_n)$ is represented as large, translucent points and $\gamma_{\vec{w}}(S_n)$ as small, opaque points. Observe that F_a does not contain any polynomial of the general form $2^{\omega-j}(w_i + r)$ hence **cover** immediately splits the problem into calculating a cover for $\langle \vec{W}_b, F_b \rangle$ and a cover for $\langle \vec{W}_i, F_i \rangle$. Note how splitting doubles a leading constant: $\vec{W}_a = \vec{w}$ whereas $\vec{W}_b = \langle w_1, 2w_2 \rangle$ and $\vec{W}_i = \langle w_1, 2w_2 + 1 \rangle$. This form of scaling by a power of 2 is a general pattern. By comparing the number of small, opaque points in Fig. 20(a) against those in (b) and (i), observe that the solutions of $\gamma_{\vec{w}}(S_a)$ are preserved by the split, that is, $\gamma_{\vec{w}}(S_a) = \gamma_{\vec{w}}(S_b) \cup \gamma_{\vec{w}}(S_i)$.

The system F_b contains $8w_2 = 2^{4-1}(w_2 + r)$ where $r = 0$ hence **cover** deploys simplification to derive $S_c = \langle \vec{W}_c, F_c \rangle$ from S_b. Since $\mathsf{vars}(r) = \emptyset$ the check **safe**$(\vec{W}, w_i, r)$ is vacuously satisfied. Recall from example 33 that $\gamma_{\vec{w}}(S_b) = \gamma_{\vec{w}}(S_c)$. Observe too how a leading constant is again doubled, with a commensurate doubling in the cardinality of $\gamma_{\vec{w}}(F_c)$ over $\gamma_{\vec{w}}(F_b)$. Since F_c does not contain any polynomial $2^{\omega-j}(w_i + r)$ splitting is again applied to give a total of three branches that emanate from a. Observe $F_e = F_g = F_h = \emptyset$ hence the pairs $\langle \vec{W}_e, F_e \rangle$, $\langle \vec{W}_g, F_g \rangle$ and $\langle \vec{W}_h, F_h \rangle$ are degenerate and thereby define the final cover $\{\vec{W}_e, \vec{W}_g, \vec{W}_h\}$ over $\vec{w}$.

Example 36. Fig. 22 serves to illustrate the application of the check **safe**$(\vec{W}, w_i, r)$ within **simplify**. Observe that $\mathsf{vars}(r) = \emptyset$ in all but one of the simplification steps. For the step that applies $2(w_1 + 6w_2 + 6)$, $r = 6w_2 + 6$ and $\vec{W} = \langle 2^1 w_1, 2^2 w_2 \rangle$. The polynomial r contains a single term $6w_2$, which contains the single variable w_2. The test **safe**$(\vec{W}, w_1, r)$ thus reduces to a single inequality $k_1 + \mathsf{rank}(6) < k_2$ which is false since $k_1 = 1$, $\mathsf{rank}(6) = 1$ and $k_2 = 2$. Thus **safe** returns **true**.

The **cover** function, and its auxiliaries, are justified by two independent sets of results, the first establishing termination of **simplify** and **cover** and the second proving that **cover** indeed computes a cover. Both sets are founded on two results, Proposition 11 and Proposition 12, which establish fundamental properties of **constrain**. These properties are then reflected in the functions, **simplify** and **cover**, which call it.

Proposition 11 asserts syntactic properties of the polynomials constituting $\vec{W}$ and how they are preserved by **constrain**. The result gives weight to the observation that each polynomial of $\vec{W}$ assumes the form $W_\ell = 2^{k_\ell} w_\ell + q_\ell$ where $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$. On exit from **constrain**, the result shows how the powers of 2 in leading constants of $\vec{W}$ are preserved, with the exception of $W_i' = 2^{k_i'} w_i + q_i'$, for which $k_i < \min(k_i + j, \omega) = k_i'$. These powers of 2 are therefore related by $\langle k_1, \ldots, k_d \rangle < \langle k_1', \ldots, k_d' \rangle$ where ¡ denotes the pointwise ordering, which provides the basis for a termination argument.

Proposition 11. Let $S = \langle \vec{W}, F \rangle \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ and suppose for each $1 \leq \ell \leq d$

- $W_\ell = 2^{k_\ell} w_\ell + q_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F)$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F)$
- If $k_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F)$

where $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ and $0 \leq k_\ell \leq \omega$. Suppose $\mathbf{constrain}(S, j, w_i, r) = \langle \vec{W}'', F'' \rangle$ where $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, $\{w_i\} \cup \mathsf{vars}(r) \subseteq \mathsf{vars}(F)$ and $\mathbf{safe}(\vec{W}, w_i, r)$ holds. Then, for each $1 \leq \ell \leq d$,

- $W''_\ell = 2^{k'_\ell} w_\ell + q'_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F'')$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k'_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F'')$
- If $k'_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F'')$

where $q'_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ and $k'_\ell = \begin{cases} \min(k_i + j, \omega) & \text{if } i = \ell \\ k_\ell & \text{otherwise} \end{cases}$

The proposition also explains how families of solutions are preserved and extended by an application of **constrain**. The result asserts that if each solution $\vec{v} \in \gamma_{\vec{w}}(F)$ arises in a family $V \subseteq \gamma_{\vec{w}}(F)$ of solutions generated thus:

$$V = \{\vec{v} + \vec{\delta} \mid \vec{\delta} = \langle c_1 2^{\omega - k_1}, \ldots, c_d 2^{\omega - k_d} \rangle, 0 \leq c_i < 2^{k_i}\}$$

then each solution $\vec{v}' \in \gamma_{\vec{w}}(F')$ generates an analogous family of solutions with respect to the k'_ℓ. Quite apart for accounting for the regular nature of $\gamma_{\vec{w}}(F')$, it follows that the cardinality of $\gamma_{\vec{y}}(F')$ is $2^{\min(j, \omega - k_i)}$-fold that of $\gamma_{\vec{y}}(F)$. Moreover, if $k'_1 = \ldots = k'_d = \omega$ then $\gamma_{\vec{y}}(F')$ has either 0 or $(2^\omega)^d$ solutions. The proposition also clarifies that if $w_i \in \mathsf{vars}(F)$ then $k_i < \omega$. This provides a progress condition in that if w_i is selected for splitting in **cover** then $k_i < \min(k_i + j, \omega) = k'_i$. Since $k'_\ell = k_\ell$ for all $\ell \neq i$, it then follows $\langle k_1, \ldots, k_d \rangle < \langle k'_1, \ldots, k'_d \rangle$ ensuring a variable is selected for splitting only a finite number of times.

The **cover** function is primed with $\vec{W} = \vec{w}$ so initially $W_\ell = 2^{k_\ell} w_\ell + q$ where $k_\ell = 0$ and $q = 0$. This ensures that the first property of Lemma 11 holds when **constrain** is initially called. But since $k_\ell = 0$ for all $1 \leq \ell \leq d$, the second and third properties hold too, albeit vacuously. The following corollary is a consequence

of this initialisation, and Lemma 11 which shows how these three properties are perpetuated by **constrain**:

Corollary 6. If **cover** calls **constrain**(S, j, w_i, r), **simplify**(S) or **cover**(S) where $S = \langle \vec{W}, F \rangle$ then for each $1 \leq \ell \leq d$,

- $W_\ell = 2^{k_\ell} w_i + q_\ell$,
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F)$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F)$
- If $k_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F)$

where $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ and $0 \leq k_\ell \leq \omega$.

The force of the corollary is that it provides the basis of a termination argument for **simplify** and **cover** both of which are recursive.

Theorem 3. simplify and **cover** terminate

The correctness argument is likewise organised in a bottom-up fashion. First, semantic properties are derived for **constrain**. These properties are then used to justify **simplify**, whose properties are then, in turn, deployed in the correctness argument of the top-level function **cover**. The following proposition asserts that **constrain**(S, j, w_i, r) is only used to augment F of $S = \langle \vec{W}, F \rangle$ with a polynomial of the form $2^{\omega - j}(w_i + r)$: the context of the calls ensuring that **safe**$(\vec{W}, w_i, r)$ holds, either because it is validated on-the-fly or because $r = 0$ or $r = 1$.

Proposition 12. Let $S = \langle \vec{W}, F \rangle \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ and consider a call $\langle \vec{W}'', F'' \rangle =$ **constrain**(S, j, w_i, r) made from **cover**. Then,

$$\gamma_{\vec{w}}(\langle \vec{W}, F \cup \{2^{\omega - j}(w_i + r)\} \rangle) = \gamma_{\vec{w}}(\langle \vec{W}'', \vec{F}'' \rangle)$$

The following result explains that if **simplify**$(S) = S'$ then either $S' =$ **nil** and $\gamma_{\vec{w}}(S) = \emptyset$ or $\gamma_{\vec{w}}(S') = \gamma_{\vec{w}}(S)$. The theorem asserts that **cover**(F) does indeed compute a cover for F over $\vec{w}$.

Corollary 7. Let $S = \langle \vec{W}, F \rangle \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ and **simplify**$(S) = S'$.

- If $S' \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ then $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S')$

- If $S' = \mathbf{nil}$ then $\gamma_{\vec{w}}(S) = \emptyset$

Theorem 4. Let $F \in \mathbb{Z}_m[\vec{x}]^d$ and $\mathbf{cover}(F) = \mathcal{W} \subseteq \mathbb{Z}_m[\vec{w}]$. Then $\mathcal{W}$ is a cover of F over $\vec{w}$.

Example 37. Returning again to F_1 and F_2 of example 32, **cover** computes

$$\mathcal{W}_1 = \{\langle 4w_2 + 6, 4w_2\rangle, \langle 8, 8w_2 + 1\rangle, \langle 12, 8w_2 + 7\rangle\} \qquad \mathcal{W}_2 = \{\langle 8w_1 + 3, 4w_2 + 1\rangle\}$$

over $\vec{w} = \langle w_1, w_2\rangle$, where the 3 vectors of $\mathcal{W}_1$ corresponding to A_1, A_2 and A_3 respectively and the single vector constituting $\mathcal{W}_2$ corresponding to A_6 of figure 17, but with a different choice of parametric variable w_2 from w_1 used previously in example 32,

5.2.2 Closure

This section explains how a cover provides a vehicle for computing closure. A closed set of polynomials can be represented by different bases, and therefore a relation is introduced to express when one basis represents the closure of another:

Definition 19. The relation $\rightarrow_{\mathsf{cl}[\vec{x}]} \subseteq \wp(\mathbb{Z}_m[\vec{x}])^2$ is defined $B \rightarrow_{\mathsf{cl}[\vec{x}]} B'$ iff $\uparrow_{\vec{x}} \langle B\rangle_{\vec{x}} = \langle B'\rangle_{\vec{x}}$.

The following lemma provides a method for computing $\uparrow_{\vec{x}} \langle F\rangle_{\vec{x}}$ when $\{\vec{W}\}$ is a singleton cover for F. The lemma is stated by lifting the elimination relation to vectors of variables defined thus $B \rightarrow_{\mathsf{elim}[\epsilon]} B$ and $B \rightarrow_{\mathsf{elim}[y:\vec{y}]} B''$ iff $B \rightarrow_{\mathsf{elim}[y]} B'$ and $B' \rightarrow_{\mathsf{elim}[\vec{y}]} B''$. The computational tactic given in the lemma amounts to augmenting null polynomials with d polynomials which equate each variable x_ℓ with W_ℓ and then applying variable elimination:

Lemma 7. Let $\vec{W} \in \mathbb{Z}_m[\vec{w}]^d$ and suppose $\{x_1 - W_1, \ldots, x_d - W_d\} \cup B_{\mathsf{Null}_m[\vec{w}]} \rightarrow_{\mathsf{elim}[\vec{w}]} B \subseteq \mathbb{Z}_m[\vec{x}]$. Then, $\langle B\rangle_{\vec{x}} = \alpha_{\vec{x}}(\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\})$.

Example 38. To illustrate this tactic, recall from Example 37 that $\{\vec{W}\}$ is a cover

of F_2 over $\vec{w} = \langle w_1, w_2 \rangle$ where $\vec{W} = \langle 8w_1 + 3, 4w_2 + 1 \rangle$. Now,

$$B_{\mathsf{Null}_{16}[\vec{w}]} = \left\{ \begin{array}{c} w_1^6 + w_1^5 + w_1^4 + 7w_1^3 + 6w_1^2 \ (p_1), \\ 2w_1^4 + 4w_1^3 + 6w_1^2 + 4w_1 \ (p_2), \\ w_1^4w_2^2 + w_1^4w_2 + 2w_1^3w_2^2 + 2w_1^3w_2 + \\ 3w_1^2w_2^2 + 3w_1^2w_2 + 2w_1w_2^2 + 2w_1w_2 \ (p_3), \\ w_1^2w_2^4 + 2w_1^2w_2^3 + 3w_1^2w_2^2 + 2w_1^2w_2 + \\ w_1w_2^4 + 2w_1w_2^3 + 3w_1w_2^2 + 2w_1w_2 \ (p_4), \\ 4w_1^2w_2^2 + 4w_1^2w_2 + 4w_1w_2^2 + 4w_1w_2, \qquad 8w_1^2 + 8w_1, \\ w_2^6 + w_2^5 + w_2^4 + 7w_2^3 + 6w_2^2 (p_5) \\ 2w_2^4 + 4w_2^3 + 6w_2^2 + 4w_2 \ (p_6), \qquad 8w_2^2 + 8w_2 \end{array} \right\}$$

and $\mathbf{gb}_{\vec{w}:\vec{x}}(\{x - W_1, y - W_2\} \cup B_{\mathsf{Null}_{16}[\vec{w}]}) = B$ where $\vec{x} = \langle x, y \rangle$ and

$$B = \left\{ \begin{array}{c} p_1, \quad p_2, \quad p_3, \quad p_4, \quad 2w_1y + 6w_1 + w_2x + w_2 + x + 3y + 10, \\ w_1^4y + w_1^4 + 4w_1^3 + w_1^2y + 5w_1^2 + 4w_1 + w_2x + w_2 + 3y + 13, \\ w_1^2w_2y + 3w_1^2w_2 + w_1w_2y + 3w_1w_2, \quad w_1x + 5w_1, \quad 8w_1 + x + 13, \\ \hline w_2^3y + 3w_2^3 + w_2^2y + 3w_2^2, w_2^2x + w_2^2 + w_2x + w_2y + y + 15, \\ p_5, \quad p_6, \quad 2w_2y + 2w_2 + y + 15, \quad 4w_2 + 3y + 13, \\ \hline x^2 + 7, \quad xy + x + y + 9, \quad 2x + 10, \quad y^2 + 2y + 13, \quad 4y + 12 \end{array} \right\}$$

The three regions delineate polynomials depending on both w_1 and w_2 (top), w_2 but not w_1 (middle) and neither w_1 nor w_2 (bottom). It follows $\{x - W_1, y - W_2\} \cup B_{\mathsf{Null}_{16}[w_1,w_2]} \rightarrow_{\mathsf{elim}[w_1]} B'$ where

$$B' = \left\{ \begin{array}{c} w_2^3y + 3w_2^3 + w_2^2y + 3w_2^2, w_2^2x + w_2^2 + w_2x + w_2y + y + 15, \\ p_5, \quad p_6, \quad 2w_2y + 2w_2 + y + 15, \quad 4w_2 + 3y + 13, \\ \hline x^2 + 7, \quad xy + x + y + 9, \quad 2x + 10, \quad y^2 + 2y + 13, \quad 4y + 12 \end{array} \right\}$$

Now, B' is also a Gröbner basis (with respect to $\prec_{\langle w,x,y \rangle}$), hence $B' \rightarrow_{\mathsf{elim}[w_2]} B''$ where

$$B'' = \{x^2 + 7, \quad xy + x + y + 9, \quad 2x + 10, \quad y^2 + 2y + 13, \quad 4y + 12\}$$

Composing the two elimination relations yields $\{x - W_1, y - W_2\} \cup B_{\mathsf{Null}_{16}[w_1,w_2]} \rightarrow_{\mathsf{elim}[\vec{w}]}$

B''. Note that it is only necessary to compute a single Gröbner basis to derive B''. Observe that each polynomial of B'' satisfies the points of $\gamma_{\vec{x}}(F_2)$ illustrated in Fig. 17(g).

The following theorem generalises this tactic to arbitrary covers:

Theorem 5. Let $B \subseteq \mathbb{Z}_m[\vec{x}]$ and $\mathcal{W} \subseteq \mathbb{Z}_m[\vec{w}]^d$ be a cover for B over $\vec{w}$. Suppose for each $\vec{W} \in \mathcal{W}$, $\{x_1 - W_1, \ldots, x_d - W_d\} \cup B_{\mathsf{Null}_m[\vec{w}]} \rightarrow_{\mathsf{elim}[\vec{w}]} B_{\vec{W}}$ and $\langle B' \rangle_{\vec{x}} = \bigsqcup_{\vec{W} \in \mathcal{W}} \langle B_{\vec{W}} \rangle_{\vec{x}}$. Then, $B \rightarrow_{\mathsf{cl}[\vec{x}]} B'$.

Example 39. Now recall from Example 35 that $\{\vec{W}_e, \vec{W}_h, \vec{W}_k\}$ is a cover of

$$F = \left\{ \begin{matrix} x^2 + x + 7y^2 + 11y + 12, \\ xy + 4x + 10y^2 \end{matrix} \right\}$$

over $\vec{w} = \langle w_1, w_2 \rangle$ where $\vec{W}_e = \langle 4w_2 + 4, 4w_2 \rangle$, $\vec{W}_h = \langle 15, 12 \rangle$ and $\vec{W}_k = \langle 14, 1 \rangle$.

To apply the theorem, $B_{\vec{W}_e}$ is derived by $\{x-(4w_2+4), y-4w_2\} \cup B_{\mathsf{Null}_{16}[\vec{w}]} \rightarrow_{\mathsf{elim}[\vec{w}]} B_{\vec{W}_e}$. Since $\vec{W}_e$ depends only on w_2, $B_{\vec{W}_e}$ can be computed by $\{x - (4w_2 + 4), y - 4w_2\} \cup B_{\mathsf{Null}_{16}[w_2]} \rightarrow_{\mathsf{elim}[w_2]} B_{\vec{W}_e}$. To that end, note $\mathbf{gb}_{\langle w_2, x, y \rangle}(\{x - (4w_2 - 4), y - 4w_2\} \cup B_{\mathsf{Null}_{16}[w_2]}) = B'_{\vec{W}_e}$ where

$$B'_{\vec{W}_e} = \left\{ \begin{matrix} w_2^6 + w_2^5 + w_2^4 + 3w_2^3 + w_2^2 y + 2w_2^2 + w_2 y, \\ 2w_2^4 + w_2^2 y + 2w_2^2 + w_2 y + y, \\ w_2^3 y + w_2 y + 2y, \quad 2w_2 y + 2y, \quad 4w_2 + 3y, \\ x + 3y + 12, \quad y^2, \quad 4y \end{matrix} \right\}$$

thus $B_{\vec{W}_e} = \{x + 3y + 12, y^2, 4y\}$ is computed whilst avoiding null polynomials containing w_1.

The bases $B_{\vec{W}_h}$ and $B_{\vec{W}_k}$ can be derived without recourse to elimination or null polynomials since $\vec{W}_h$ and $\vec{W}_k$ are independent of w_1 and w_2 hence it is sufficient to put

$$B_{\vec{W}_h} = \{x-15, y-12\} = \{x+1, y+4\} \qquad B_{\vec{W}_k} = \{x-14, y-1\} = \{x+2, y+15\}$$

By the theorem it follows $\uparrow_{\vec{x}} \langle F \rangle_{\vec{x}} = \langle B \rangle_{\vec{x}}$ where $\langle B \rangle_{\vec{x}} = \langle B_{\vec{W}_e} \rangle_{\vec{x}} \sqcup \langle B_{\vec{W}_h} \rangle_{\vec{x}} \sqcup$

$\langle B_{\vec{W}_k}\rangle_{\vec{x}}$ which gives

$$B = \left\{ \begin{matrix} x^2 + x + 7y^2 + 11y + 12, & xy + 4x + 10y^2, \\ y^3 + 7y^2 + 8y, & 4y^2 + 12y \end{matrix} \right\}$$

All the polynomials of B satisfy the points $\gamma_{\vec{x}}(F)$ plotted in Fig. 21(a). Note too that

$$F' = \left\{ x^2 + x + 7y^2 + 11y + 12, \quad xy + 4x + 10y^2, \quad y^3 + 7y^2 + 8y \right\}$$

is a Gröbner basis for $\langle F\rangle_{\vec{x}}$ with respect to $\prec_{\vec{x}}$. Since $4y^2 + 12y$ is irreducible by F' it follows $4y^2 + 12y \notin \langle F\rangle_{\vec{x}}$ which is why closure augments F with $4y^2 + 12y$.

5.2.3 Meet

Despite the central importance of meet, this section is relatively short, since the following proposition demonstrates how meet can be reduced to closure:

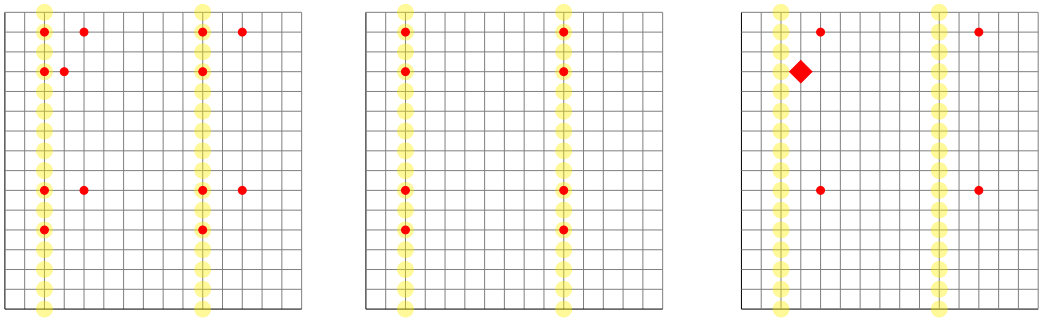
Proposition 13. If $B_1 \cup B_2 \to_{\mathsf{cl}[\vec{x}]} B$ then $\langle B_1\rangle_{\vec{x}} \sqcap \langle B_2\rangle_{\vec{x}} = \langle B\rangle_{\vec{x}}$.

Example 40. Consider $F_3, F_4 \subseteq \mathbb{Z}_{16}[x, y]$ where $F_3 = \{x^2+x+7y^2+11y+12\}$ and $F_4 = \{xy + 4x + 10y^2\}$ and let $F = F_3 \cup F_4$. The solution sets $\gamma_{\vec{x}}(F)$, $\gamma_{\vec{x}}(F_3)$ and $\gamma_{\vec{x}}(F_4)$ are plotted in Figs. 21(a), (b) and (c) respectively. The diamond points in Figs. 21(b) and (c) are those contained in both $\gamma_{\vec{x}}(F_3)$ and $\gamma_{\vec{x}}(F_4)$ and thus demonstrate $\gamma_{\vec{x}}(F) = \gamma_{\vec{x}}(F_1) \cap \gamma_{\vec{x}}(F_2)$.

Now, Example 39 shows $F \to_{\mathsf{cl}[\vec{x}]} B$ where

$$B = \left\{ \begin{matrix} x^2 + x + 7y^2 + 11y + 12, & xy + 4x + 10y^2, \\ y^3 + 7y^2 + 8y, & 4y^2 + 12y \end{matrix} \right\}$$

thus it follows $\langle F_3\rangle_{\vec{x}} \sqcap \langle F_4\rangle_{\vec{x}} = \langle B\rangle_{\vec{x}}$. As noted in Example 39, $4y^2 + 4y \notin \langle F\rangle_{\vec{x}}$ hence $\langle F\rangle_{\vec{x}} \neq \langle B\rangle_{\vec{x}}$.



(a) $A = \gamma(\langle B \rangle)$ (b) $[\![\textbf{assume } (p = 0)]\!](A)$ (c) $[\![\textbf{assume } (p \neq 0)]\!](A)$

Figure 23: Abstract assumes for $p = 2x - 4$

5.3 Calculating abstract transfer functions

The essense of abstract interpretation is to simulate the semantics $[\![s]\!](A) = A'$ of a statement s operating on concrete data $A \subseteq \mathbb{Z}_m^d$ with an an abstract version (an abstract transfer function for s) which given $B \in \mathsf{MPAD}_m[\vec{x}]$ such that $A \subseteq \gamma_{\vec{x}}(B)$ computes some $B' \in \mathsf{MPAD}_m[\vec{x}]$ such that $A' \subseteq \gamma_{\vec{x}}(B')$. This section provides abstract transfer functions for **assume** statements, non-deterministic assignments and polynomial assignments, all transfer functions satisfying the stronger property that $\alpha_{\vec{x}}([\![s]\!](\gamma_{\vec{x}}(B))) = \langle B' \rangle_{\vec{x}}$. The section concludes with a procedure for checking $\langle B \rangle_{\vec{x}} \sqsubseteq \langle B' \rangle_{\vec{x}}$ for finite bases B and B', necessary for detecting that a fixpoint is reached.

5.3.1 Assume for polynomial equality

The method for imposing a polynomial equality $p = 0$ is analogous to that for computing meet: the system of polynomials is augmented with p and then closure is applied:

Proposition 14. Suppose $\langle B \rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$ and $A = \gamma_{\vec{x}}(B)$. If $B \cup \{p\} \rightarrow_{\mathsf{cl}[\vec{x}]} B'$ then $\alpha_{\vec{x}}([\![\textbf{assume } (p = 0)]\!](A)) = \langle B' \rangle_{\vec{x}}$.

Example 41. To illustrate, let $\vec{x} = \langle x, y \rangle$, $p = 2x + 12 \in \mathbb{Z}_{16}[\vec{x}]$ and suppose

$B \subseteq \mathbb{Z}_{16}[\vec{x}]$ is defined:

$$B = \left\{ \begin{matrix} x^3 + x^2 + 3xy + 4x + 2y + 12, \\ x^2y + 4x + 4y + 8, \quad 4xy \\ 2x^2 + 3xy + 2x + 2y + 4, \\ y^2 + 6y + 8, \quad 8y \end{matrix} \right\}$$

The solution sets $\gamma_{\vec{x}}(B)$ and $\gamma_{\vec{x}}(p)$ are plotted in Fig. 23(a) as the small, opaque points and large, translucent points respectively. Now,

$$B \cup \{p\} \rightarrow_{\mathsf{cl}[\vec{x}]} B' = \{x^2 + 12, \quad xy + 6y, \quad 2x + 12, \quad y^2 + 6y + 8, \quad 8y\}$$

The solution sets $\gamma_{\vec{x}}(B')$ and $\gamma_{\vec{x}}(p)$ are plotted in Fig. 23(b) as the small, opaque points and large, translucent points respectively. As can be seen, $\gamma_{\vec{x}}(B')$ is a strict subset of $\gamma_{\vec{x}}(p)$, consistent with the semantics of the statement $[\![\textbf{assume } (p = 0)]\!]$.

5.3.2 Assume for polynomial disequality

The method for imposing a polynomial disequality $p \neq 0$ rests on a division of $\gamma_{\vec{x}}(B)$ into subsets $A_1, \ldots, A_\omega$ where the least k bits of $[\![p]\!]_{\vec{x}}(\vec{a})$ represent 2^{k-1} for all $\vec{a} \in A_k$. Bit k of $[\![p]\!]_{\vec{x}}(\vec{a})$ is set for all $a \in A_k$, hence $[\![p]\!]_{\vec{x}}(\vec{a}) \neq 0$, which provides a way of simulating disequality:

Proposition 15. Suppose $\langle B \rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$ and $A = \gamma_{\vec{x}}(B)$. If $B \cup \{2^{\omega-k}p + 2^{\omega-1}\} \rightarrow_{\mathsf{cl}[\vec{x}]} B_k$ for each $1 \leq k \leq \omega$ then $\alpha_{\vec{x}}([\![\textbf{assume } (p \neq 0)]\!](A)) = \bigsqcup_{k=1}^{\omega} \langle B_k \rangle_{\vec{x}}$.

For intuition, consider $A_i = \gamma_{\vec{x}}(\{2^{\omega-k}p + 2^{\omega-1}\})$ and observe $\vec{a} \in A_1$ iff the least bit of $[\![p]\!]_{\vec{x}}(\vec{a})$ is 1. Moreover, $\vec{a} \in A_2$ iff the least 2 bits of $[\![p]\!]_{\vec{x}}(\vec{a})$ are 10, and $\vec{a} \in A_\omega$ iff the ω bits of $[\![p]\!]_{\vec{x}}(\vec{a})$ are $10 \cdots 0$. Thus B is augmented with $2^{\omega-1}p + 2^{\omega-1}$, $2^{\omega-2}p + 2^{\omega-1}$, $\ldots$, $p + 2^{\omega-1}$. These ω separate systems are then closed and recombined by join, as is illustrate below:

Example 42. Consider again $p = 2x + 12 \in \mathbb{Z}_{16}[\vec{x}]$ and $B \subseteq \mathbb{Z}_{16}[\vec{x}]$ as specified in Example 41. Then, for each $1 \leq k \leq 4$, $B \cup \{2^{4-k}p + 8\} \rightarrow_{\mathsf{cl}[\vec{x}]} B_k$ where

$$B_1 = \{1\} \qquad B_2 = \{x + 13, \quad y + 4\}$$
$$B_3 = \{x^2, \quad xy + 8, \quad 2x + 8, \quad y^2 + 12, \quad 2y + 4\} \qquad B_4 = \{1\}$$

Thus $\gamma_{\vec{x}}(B_1) = \gamma_{\vec{x}}(B_4) = \emptyset$ but $\gamma_{\vec{x}}(B_2) = \{\langle 3, 12\rangle\}$. Observe $[\![p]\!]_{\vec{x}}(\langle 3, 12\rangle) = 2 \equiv 0010_2$. Moreover $\gamma_{\vec{x}}(B_3) = \{\langle 4, 6\rangle, \langle 4, 14\rangle, \langle 12, 6\rangle, \langle 12, 14\rangle\}$ and note $[\![p]\!]_{\vec{x}}(\langle 4, y\rangle) = [\![p]\!]_{\vec{x}}(\langle 12, y\rangle) = 4 \equiv 0100_2$ for all $y \in \mathbb{Z}_{16}$. The (non-empty) sets $\gamma_{\vec{x}}(B_2)$ and $\gamma_{\vec{x}}(B_3)$ are plotted in Fig. 23(c) as the lone diamond and the four small, opaque circles respectively. Observe that these two sets are disjoint. The set $\gamma_{\vec{x}}(p)$ is also plotted as the large, translucent points. It is disjoint from both $\gamma_{\vec{x}}(B_2)$ and $\gamma_{\vec{x}}(B_3)$.

Now, note $\bigsqcup_{k=1}^{4} \langle B_k\rangle_{\vec{x}} = \langle B''\rangle_{\vec{x}}$ where

$$B'' = \left\{ \begin{matrix} x^3 + x^2 + 2y + 4, & 2x^2 + 2x + 8, & xy + 6y + 4 \\ 4x + 6y + 12, & y^2 + 6y + 8, & 8y \end{matrix} \right\}$$

It thus follows that $\alpha_{\vec{x}}([\![\textbf{assume } (p \neq 0)]\!](\gamma_{\vec{x}}(B))) = \langle B''\rangle_{\vec{x}}$. Moreover, $\gamma_{\vec{x}}(B'') = \gamma_{\vec{x}}(B_2) \cup \gamma_{\vec{x}}(B_3)$ (though this does not hold in general). Thus the union of the diamond and the small, circular points in Fig. 23(c) is precisely the set $\gamma_{\vec{x}}(B'')$. Finally, observing Fig. 23(a), (b) and (c) together reveals that $\gamma_{\vec{x}}(B)$ is the disjoint union of $\gamma_{\vec{x}}(B'')$ of the $\gamma_{\vec{x}}(B')$ of the previous example.

5.3.3 Non-deterministic assignment

The following lemma demonstrates that the integrity of an abstraction B is preserved when a new variable y_j is introduced whose value is determined by a polynomial $p \in \mathbb{Z}_{16}[\vec{x}]$ where $y_j \notin \mathsf{vars}(\vec{x})$. The result supports the development of both non-deterministic and polynomial assignment.

Lemma 8. Suppose $\vec{x} \sqsubseteq \vec{y}$ where $\mathsf{vars}(\vec{y}) \setminus \{y_j\} = \mathsf{vars}(\vec{x})$. If $A \subseteq \mathbb{Z}_m^d$, $\alpha_{\vec{x}}(A) = \langle B\rangle_{\vec{x}}$, $p \in \mathbb{Z}_m[\vec{x}]$ and $A' = \{\langle a_1, \ldots, a_{j-1}, [\![p]\!]_{\vec{x}}(\vec{a}), a_j, \ldots, a_d\rangle \mid \vec{a} \in A\}$ then $\alpha_{\vec{y}}(A') = \langle B \cup \{y_j - p\}\rangle_{\vec{y}}$.

One might expect non-deterministic assignment to be modelled by eliminating polynomials which include the assigned variable. However, as the example below demonstrates, this is not generally sufficient since, paradoxically, it is possible for the variable to appear in a polynomial even when unconstrained. This situation necessitates the application of closure in the following result:

Proposition 16. Suppose $B \in \mathsf{MPAD}_m[\vec{x}]$ and $A = \gamma_{\vec{x}}(B)$. If $B \to_{\mathsf{cl}[w:\vec{x}]} B'' \to_{\mathsf{elim}[x_j]} B'''$ and $B''' \cup \{x_j - w\} \to_{\mathsf{elim}[w]} B'$ then $\alpha_{\vec{x}}([\![x_j \coloneqq *]\!](A)) = \langle B'\rangle_{\vec{x}}$.

Example 43. Let $\vec{x} = \langle x, y\rangle$ and consider $B \subseteq \mathbb{Z}_{16}[\vec{x}]$ defined $B = \{x^2, xy, 2x + 2y, y^2, 4y\}$. The set $\gamma_{\vec{x}}(B)$ is plotted in Fig. 24(a). Then, $B \to_{\mathsf{cl}[w:\vec{x}]} B''$ where

$$B'' = \begin{Bmatrix} y^2, & yx, & yw^2 + yw + xw^2 + xw, & 2y + 2x, \\ x^2, & xw^4 + xw^2 + 2xw, & 2xw^2 + 2xw, & 4x \end{Bmatrix}$$

Note this is also a Gröbner basis with respect to $\prec_{\langle y,w,x\rangle}$, hence $B'' \to_{\mathsf{elim}[y]} B'''$ where

$$B''' = \begin{Bmatrix} x^2, & xw^4 + xw^2 + 2xw, & 2xw^2 + 2xw, & 4x \end{Bmatrix}$$

Finally, noting $\mathbf{gb}_{\langle w,x,y\rangle}(B''' \cup \{y - w\}) = \{w - y, x^2, xy^4 + xy^2 + 2xy, 2xy^2 + 2xy, 4x\}$ it follows $B''' \cup \{y - w\} \to_{\mathsf{elim}[w]} B'$ where

$$B' = \begin{Bmatrix} x^2, & xy^4 + xy^2 + 2xy, & 2xy^2 + 2xy, & 4x \end{Bmatrix}$$

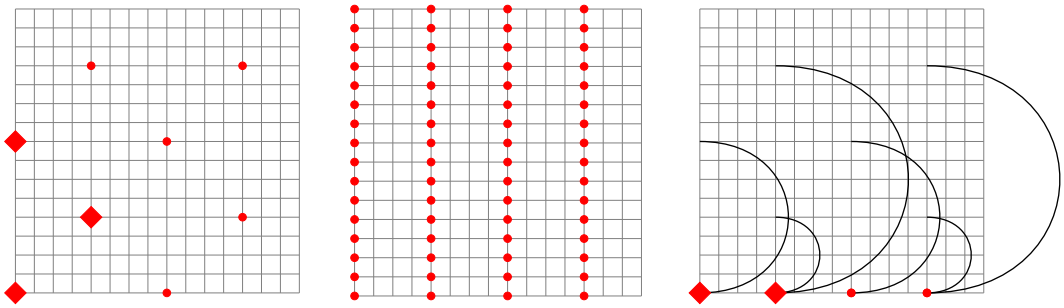
Thus, the previous result implies $\alpha_{\vec{x}}([\![y := *]\!](\gamma_{\vec{x}}(B)) = \langle B'\rangle_{\vec{x}}$, whose solutions are plotted in Fig. 24(b). Note that for each solution $\langle a_1, a_2\rangle$ in Fig. 24(a), the solution $\langle a_1, b\rangle$ is present in Fig. 24(c) for all $b \in \mathbb{Z}_{16}$, consistent with a non-deterministic assignment to y.

Interestingly, y occurs in B' even though it unconstrained. To see why, consider $2xy^2 + 2xy \in B'$. Any value a taken by x must be a multiple of 4 since $4x \in B$. But then, partially evaluating $2xy^2 + 2xy$ with $x = a = 4b$ yields $8by^2 + 8by = b(8y^2 + 8y)$. Since $8y^2 + 8y$ is a null-polynomial, so too is $b(8y^2 + 8y)$, independent of b. In particular, the polynomial $2xy^2 + 2xy$ does not constrain y, even though it is not itself null. If y had simply been eliminated from the basis then $2xy^2 + 2xy$ would not have been discovered, illustrating the need for closure.

5.3.4 Polynomial assignment

The correctness result for polynomial assignment takes a different form to the other program statements, stated in terms of the assumption $\alpha_{\vec{x}}(A) = \langle B\rangle_{\vec{x}}$. Note that if $\langle B\rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$ and $A = \gamma_{\vec{x}}(B)$ then $\alpha_{\vec{x}}(A) = \langle B\rangle_{\vec{x}}$ and the result applies. Thus this formulation is strictly stronger.

Proposition 17 (optimality)**.** Suppose $A \subseteq \mathbb{Z}_m^d$, $\alpha_{\vec{x}}(A) = \langle B\rangle_{\vec{x}}$, $B \cup \{w -$



(a) $A = \gamma_{\vec{x}}(B)$ (b) $[\![y := *]\!](A)$ (c) $[\![y := y^2]\!](A)$

Figure 24: Non-deterministic and polynomial assignment

$p\} \to_{\mathsf{elim}[x_j]} B''$ and $B'' \cup \{x_j - w\} \to_{\mathsf{elim}[w]} B'$, where $w \notin \mathsf{vars}(B)$. Then, $\alpha_{\vec{x}}([\![x_j := p]\!](A)) = \langle B' \rangle_{\vec{x}}$.

This lemma generalises a folklore result for affine approximation (linear equalities) [62, Lemma 2] which shows that affine approximation commutes with the transfer function for linear assignment, as modelled by an affine transformation. The above lemma strengthens this result to show an analogous optimality result for polynomial approximation and polynomial assignment, as realised in MPAD. The force of the folklore result is that if a program consists solely of linear assignments, then the affine approximation of the collecting semantics of the program coincides exactly with the linear systems derived by the transfer functions [62, Lemma 3]. Hence all linear equalities which hold for the collecting semantics are inferred when working at the more abstract level of linear equalities. The argumentation, which is standard in abstract interpretation, relies on linear equalities satisfying the ascending chain condition, a property that also carries over to MPAD. It should be noted, however, that these results only hold for programs which are devoid of **assume** statements, hence the full arguments are not rehearsed here.

Example 44. Let $\vec{x} = \langle x, y \rangle$ and $A' = \{\langle 0, 0 \rangle, \langle 0, 8 \rangle, \langle 4, 4 \rangle\} \subseteq \mathbb{Z}_{16}^2$ which is plotted as the three diamond points in Fig. 24(a). Then, with B as in Example 43, it follows $\alpha_{\vec{x}}(A') = \langle B \rangle_{\vec{x}}$. Then,

$$\mathbf{gb}_{\langle y,x,w \rangle}(B \cup \{w - y^2\}) = \{y^2, yx, 2y + 2x, x^2, 4x, w\}$$

hence $B \cup \{w - y^2\} \rightarrow_{\mathsf{elim}[y]} \{x^2, 4x, w\} = B''$. Moreover,

$$\mathbf{gb}_{\langle w,x,y\rangle}(B'' \cup \{y - w\}) = \{w, x^2, 4x, y\}$$

hence $B'' \cup \{y - w\} \rightarrow_{\mathsf{elim}[w]} B'$ where $B' = \{x^2, 4x, y\}$.

Now consider $A'' = [\![y \coloneqq y^2]\!](A') = \{\langle 0, 0\rangle, \langle 4, 0\rangle\}$, whose points are plotted as diamonds in Fig. 24(c). The arcs indicate how each point in A' is mapped to a corresponding point in A'', which is possibly the same. Abstraction of A'' introduces two additional points since $\alpha_{\vec{x}}(A'') = \langle x^2, 4x, y\rangle_{\vec{x}}$ and $\{\langle 8, 0\rangle, \langle 12, 0\rangle\} \subseteq \gamma_{\vec{x}}(\{x^2, 4x, y\})$. But note how these points are themselves mapped from points in $A \setminus A'$, as indicated by the arcs. Observe too that $\alpha_{\vec{x}}(A'') = B'$ as predicted by the proposition.

5.3.5 Fixpoint check

In order to detect a fixpoint, it is necessary to decide whether the relation $\sqsubseteq$ holds over $\mathsf{MPAD}_m[\vec{x}]$. For this, the theory of Gröbner bases provides a natural solution. First, note if $\langle B_1\rangle_{\vec{x}}, \langle B_2\rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$ then $\langle B_1\rangle_{\vec{x}} \sqsubseteq \langle B_2\rangle_{\vec{x}}$ iff $\langle B_2\rangle_{\vec{x}} \subseteq \langle B_1\rangle_{\vec{x}}$ iff $B_2 \subseteq \langle B_1\rangle_{\vec{x}}$. But this final inclusion can be decided if B_1 is a Gröbner basis for $\langle B_2\rangle_{\vec{x}}$ with respect to a monomial ordering $\prec$ since, in this case, $B_2 \subseteq \langle B_1\rangle_{\vec{x}}$ iff $p \rightarrow^*_{\prec,B_1} 0$ for all $p \in B_2$.

Example 45. To illustrate, let $\vec{x} = \langle x, y\rangle$ and consider checking $\langle B_1\rangle_{\vec{x}} \sqsubseteq \langle B_2\rangle_{\vec{x}}$ and $\langle B_2\rangle_{\vec{x}} \sqsubseteq \langle B_1\rangle_{\vec{x}}$ where

$$B_1 = \left\{\begin{matrix} x + 15y + 1, \\ y^2 + 11y + 4 \end{matrix}\right\} \quad B_2 = \left\{\begin{matrix} x + y^2 + 10y + 5, \\ y^3 + 11y + 4, \\ 2y^2 + 6y + 8 \end{matrix}\right\}$$

Then, B_1 and B_2 are Gröbner bases for $\langle B_1\rangle_{\vec{x}}$ and $\langle B_2\rangle_{\vec{x}}$ respectively, with respect to $\prec_{\vec{x}}$. But now, letting $p = x + 15y + 1$ and $q = y^2 + 11y + 4$ and abbreviating

$\rightarrow_{\prec_{\vec{x}},B}$ to $\rightarrow_B$, it follows

$$\begin{aligned} x + y^2 + 10y + 5 &\rightarrow_p y^2 + 11y + 4 \rightarrow_q 0 \\ y^3 + 11y + 4 &\rightarrow_q 5y^2 + 7y + 4 \rightarrow_q 0 \\ 2y^2 + 6y + 8 &\rightarrow_q 0 \end{aligned}$$

therefore $\langle B_1 \rangle_{\vec{x}} \sqsubseteq \langle B_2 \rangle_{\vec{x}}$. However, since $y^2 + 11y + 4 \in B_1$ is $\prec_{\vec{x}}$-irreducible by B_2 it follows $y^2 + 11y + 4 \notin \langle B_2 \rangle_{\vec{x}}$. In particular, $\langle B_2 \rangle_{\vec{x}} \not\sqsubseteq \langle B_1 \rangle_{\vec{x}}$.

5.4 Related work

The architecture of the domain operations of MPAD mirrors that of the so-called weakly relational abstract domains of Octagons [59] and the Two Variable per Inequality (TVPI) abstract domain [40, 74]. This architecture rests on a closure algorithm which strengthens a system of constraints with entailed constraints, whose presence allows other domain operations to be more syntactic and therefore simpler. Closure algorithms for sparse representations of both Octagons and TVPI have also now emerged [74], which are nearer again to the closure operation of MPAD that operates on polynomials which are also represented symbolically.

It is interesting too to see that aspects of the triangularisation algorithm for linear modular equations surface [65] in the argumentation of the correctness of closure. Recall that termination of covering follows by reasoning about powers of two in the leading terms of the polynomials constituting the vector $\vec{W}$, which resonates with the rank-based termination arguments used for linear modular equations.

The simplification and splitting techniqes at the heart of the covering algorithm, however, stem from the modular and finite nature of the arithmetic, and therefore it is perhaps not surprising that there is an absence of closely related work.

5.5 Concluding Discussion

The key operation in MPAD is closure since it underpins other domain operations, but closure is actually straightforward once a covering has been derived. The number of the null polynomials that need to be enumerated depends critically

on the number of variables of $\vec{w}$ which remain in the $\vec{W}$ vectors that constitute the cover. Thus further splitting could, paradoxically, confer a computational advantage by further reducing the number of variables hence the number of nulls. Nevertheless, the overarching principle behind the covering algorithm is to apply simplification aggressively, in order to defer, and ideally deter, splitting. Once closure is in place the transfer functions then slot into place too.

Chapter 6

Future Work and Conclusions

This thesis started with a discussion of how one sub-discipline of program verification cross-fertilises another, and concludes with a broader reflection on how algorithms used within verification cross-fertilise with techniques developed within another field of symbolic computation, namely computer algebra.

6.1 Future work

It is recognised [1] that classical algorithms for calculating Gröbner bases [8] or eliminating variables [11, 12] do not fit with the SMT computation model [68] because they support neither incrementality nor learning [54]. Incrementality allows single constraints to be added or retracted efficiently, and learning accelerates satisfiability checking by avoiding repeated search. Nevertheless, the potential afforded by combining symbolic computation with SMT has been noted in several places [1, 6, 19], and a new class of SMT solvers has appeared that apply algebraic and satisfiability techniques together [39, 44, 76]. Despite the promise of a tighter integration of computer algebra with SMT, progress has largely focussed on theories that are important in computer algebra, such as that of algebraically closed fields of polynomials, not theories which are routinely applied verification, such as that of bit-vectors. This thesis constitutes a first step towards bridging this gap, concentrating on multiplication of bit-vectors, and more generally polynomials over bit-vectors, which are problematic for bit-blasting.

Going further, an interesting research question is how systems of polynomials can express bit-wise operations. Ironically, at the very beginning of this thesis work, a study was undertaken calculating best transformers [69] for various bit-wise operations. For example, the following system is the best $\mathsf{MPAD}_8[\vec{x}]$ abstraction of the symbolic relation $z = x \;\&\; y$ over $\mathbb{Z}_8$ where $\vec{x} = \langle x, y, z\rangle$:

$$\left\{\begin{array}{ll}
x^4 + 6x^3 + 3x^2 + 6x, & xy^3 + xy^2 + 5yz^2 + 2xy + 5z^2 + 2z, \\
xz^3 + xz^2 + 5z^3 + 6xz + 5z^2 + 6z & y^4 + 6y^3 + 7y^2 + 2y, \\
yz^3 + 5yz^2 + z^3 + 2yz + 5z^2 + 2z, & z^4 + 6z^3 + 3z^2 + 6z, \\
x^2y + 5xy^2 + 6xz^2 + 6yz^2 + 6z^3 + 5xy + 3z^2, & \\
x^2z + 7xz^2 + 5xz + 3z^2, & xyz + 7xz^2 + 7yz^2 + z^3, \\
y^2z + 7yz^2 + yz + 7z^2, & 4x^2 + 4x, \\
4xy + 4z, \quad 4xz + 4z, \quad 4y^2 + 4y, & 4yz + 4z, \quad 4z^2 + 4z
\end{array}\right\}$$

A best transformer can be calculated for a bit-vector operation since $\mathsf{MPAD}_8[\vec{x}]$ satisfies the ascending chain condition. The intuition is that points $\langle x, y, z\rangle \in \mathbb{Z}_8^2$ are harvested which satisfy the bit-vector and join is applied to calculate a system of polynomials which summaries these points. The approach can be made demand-driven by finding a point $\langle x, y, z\rangle \in \mathbb{Z}_8^3$ which does not satisfy the summary computed thus far yet satisfies $z = x \;\&\; y$. The summary is then relaxed by calculating the join with this point, thus generating a sequence of summaries which progressively describe more and more points. When no further points can be found, the summary constitutes the best abstraction of the relation $z = x \;\&\; y$.

The value of these abstraction is that they provide a uniform scheme for handling bit-vector operations within the framework of modular polynomials. Bit-vector logic would potentially be translated into systems of polynomials to take better advantage of word-level propagation in SMT, mirroring the use of algebraic methods in SAT solving [39]. Best abstractions for bit-wise operations could also be useful in the verification of micro-controller code which frequently deploys bit-twiddling. However, it is not clear the extent to which these summaries can actually be derived.

Considering the large body of work on Gröbner bases over algebraically closed fields, there are relatively few studies [4, 5] on adapting Gröbner basis algorithms

to modular arithmetic, most notably using the F-series of algorithms [27, 28]. In particular, we know of no study which can side-step the enumeration of null polynomials, or even compute them on-demand in reduction. Null polynomials are an irritant for MPAD since they consume considerable space and impede scalability to full-size bit-widths. In our opinion, it would not seem prudent to invest more in implementation effort without a thorough reevaluation of the role of nulls.

The interaction between MPAD, and other numeric domains, is another promising field of study. It is worth noting that MPAD has potential for discovering polynomial invariants over arbitrary (idealised) integers. Observe that if a polynomial invariant holds over arbitrary integers then it also holds as a modular polynomial invariant. Hence, MPAD can be used to propose candidate invariants for arbitrary integers which are then checked as a post-processing (filtering) step. The covering algorithm which underpins closure performs propagation over systems of polynomials, but it would be interesting to examine whether this could be extended to other theories, Nelson-Oppen style [66], so that a covering could be computed for arbitrary SMT formulae.

Returning to the SMT problem of detecting the satisfiability of modular polynomials, it interesting to see that the decomposition proposed for handling negative guards is equally applicable to SMT formulae over both polynomial equalities and disequalities. A disequality could be reduced to a disjunct of ω separate equalities, each checking whether a particular bit is set, forgoing the $\omega - 1$ join calculations.

In summary, the thesis offers at least as many research questions as answers, possibly reflecting the richness of this vein of work at the intersection of SMT, abstract interpretation and computer algebra.

6.2 Conclusions

In terms of concrete answers, we review the main contributions of the thesis by way of a final concluding discussion.

The thesis first presents a new architecture for solving systems of polynomial equalities over bit-vectors which addresses the backtrackability, incrementality and learning issues which are normally associated with Gröbner basis engines [1]. Rather than converting to SAT and bit-blasting, the method sets bits in order of

least significance through the addition of polynomials. Computing a Gröbner basis for the resulting system realises bit-sequence propagation, in which the values of other bits can be automatically inferred. Furthermore, and perhaps rather surprisingly, we show how the procedure can be carried out with symbolic truth values without giving up bit-sequence propagation, thus unifying Gröbner basis calculations that would otherwise be separate. Once all bits are assigned truth values (symbolic or otherwise), the resulting Gröbner basis prescribes an assignment to the bit-vectors which is a function of the symbolic truth values. The remaining polynomials in the basis relate the symbolic truth values and correspond to non-linear pseudo-boolean constraints modulo a power of two. These constraints can be solved either by translation into classical linear pseudo-boolean constraints (without a modulo) or else by encoding them as propositional formulae, for which a novel translation process is described. Either way, the algebraic Gröbner basis computation is encapsulated in the phase that emits the pseudo-boolean constraints, hence the Gröbner basis engine does not need to be backtrackable, incremental or support learning. Overall, the architecture provides a principled method for compiling high-level polynomials to low-level pseudo-boolean constraints.

The second theme of the thesis focusses not on using Gröbner bases as a device for compilation, but as an engine for inferring polynomial invariants by abstract interpretation. This is idea is crystalised in MPAD: the modular polynomial abstract domain, which is a strict generalisation of linear equalities modulo a power of two [63]. We provide abstract transfer functions for MPAD, showing the transfer function for polynomial assignment is optimal. Coupled with the finiteness of MPAD, it follows that MPAD will infer all polynomial invariant for programs consisting solely of polynomial assignments. We introduce a notion of closure, showing that it is preserved by join and polynomial assignment. For meet, it is necessary to re-establish closure, hence a closure algorithm is provided, which is itself formulated in terms of covering. This algorithm for covering, like that for SMT solving, exploits structure in a system of polynomials by judiciously setting a single bit of a single variable to either 0 or 1 to derive two simpler systems of polynomials. Setting a single bit exposes the values of other bits of other variables, as with bit-sequence propagation. We show of how domain operations reduce to closure and demonstrate that MPAD can derive invariants that cannot be expressed

with non-modular polynomial systems. MPAD thus represents a new point in the pantheon of abstract domains, complementing the new approach to SMT solving for modular polynomials, both building on bit-sequence propagation.

Appendix A

Proofs

A.1 Proofs for domain operations

Lemma 9. If $P_1 \subseteq P_2$ then $\gamma_{\vec{x}}(P_2) \subseteq \gamma_{\vec{x}}(P_1)$.

Proof of Lemma 9. Let $\vec{a} \in \gamma_{\vec{x}}(P_2)$. Then, $[\![p]\!]_{\vec{x}}(\vec{a}) = 0$ for all $p \in P_2$. But since $P_1 \subseteq P_2$ it follows $[\![p]\!]_{\vec{x}}(\vec{a}) = 0$ for all $p \in P_1$. It follows $\vec{a} \in \gamma_{\vec{x}}(P_1)$, hence $\gamma_{\vec{x}}(P_2) \subseteq \gamma_{\vec{x}}(P_1)$. □

Proof of Proposition 4. The properties are proved in the alternative order 1, 2, 5, 4, 3, since some depend on others for their proof.

- $(P \subseteq \uparrow_{\vec{x}} P)$ Let $p \in P$. If $\vec{a} \in \gamma_{\vec{x}}(P)$ then $[\![p]\!]_{\vec{x}}(\vec{a}) = 0$, hence $\vec{a} \in \gamma_{\vec{x}}(p)$. It follows $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p)$. Thus, $p \in \uparrow_{\vec{x}} P$ so $P \subseteq \uparrow_{\vec{x}} P$.

- $(P_1 \subseteq P_2 \implies \uparrow_{\vec{x}} P_1 \subseteq \uparrow_{\vec{x}} P_2)$ Let $p \in \uparrow_{\vec{x}} P_1$. Then, $\gamma_{\vec{x}}(P_1) \subseteq \gamma_{\vec{x}}(p)$. But, since $P_1 \subseteq P_2$, it follows from Lemma 9 that $\gamma_{\vec{x}}(P_2) \subseteq \gamma_{\vec{x}}(P_2)$. Thus, $\gamma_{\vec{x}}(P_2) \subseteq \gamma_{\vec{x}}(p)$. It follows that $p \in \uparrow_{\vec{x}} P_2$, hence $\uparrow_{\vec{x}} P_1 \subseteq \uparrow_{\vec{x}} P_2$.

- $(\gamma_{\vec{x}}(\uparrow_{\vec{x}} P) = \gamma_{\vec{x}}(P))$ By monotonicity of $\uparrow_{\vec{x}}$ and Lemma 9, it follows $\gamma_{\vec{x}}(\uparrow_{\vec{x}} P) \subseteq \gamma_{\vec{x}}(P)$, hence it must ony be shown $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(\uparrow_{\vec{x}} P)$. For this, note if $p \in \uparrow_{\vec{x}} P$ then $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p)$, thus $[\![p]\!]_{\vec{x}}(\vec{a}) = 0$ for all $\vec{a} \in \gamma_{\vec{x}}(P)$. Thus $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(\uparrow_{\vec{x}} P)$ and the result follows.

- $(\gamma_{\vec{x}}(P_1) = \gamma_{\vec{x}}(P_2) \iff \uparrow_{\vec{x}} P_1 = \uparrow_{\vec{x}} P_2)$ If $\gamma_{\vec{x}}(P_1) = \gamma_{\vec{x}}(P_2)$, then for all $p \in \mathbb{Z}_m[\vec{x}]$, $\gamma_{\vec{x}}(P_1) \subseteq \gamma_{\vec{x}}(p)$ iff $\gamma_{\vec{x}}(P_2) \subseteq \gamma_{\vec{x}}(p)$. It follows that $\uparrow_{\vec{x}} P_1 = \uparrow_{\vec{x}} P_2$.

Conversely, if $\uparrow_{\vec{x}} P_1 = \uparrow_{\vec{x}} P_2$ then $\gamma_{\vec{x}}(P_1) = \gamma_{\vec{x}}(\uparrow_{\vec{x}} P_1) = \gamma_{\vec{x}}(\uparrow_{\vec{x}} P_2) = \gamma_{\vec{x}}(P_2)$ follows from the third property.

- ($\uparrow_{\vec{x}} \uparrow_{\vec{x}} P = \uparrow_{\vec{x}} P$) From the third result $\gamma_{\vec{x}}(\uparrow_{\vec{x}} P) = \gamma_{\vec{x}}(P)$. The fourth result then implies $\uparrow_{\vec{x}} \uparrow_{\vec{x}} P = \uparrow_{\vec{x}} P$.

□

Proof of Lemma 4. Since $B \subseteq \langle B \rangle_{\vec{x}}$ it follows $\gamma_{\vec{x}}(B) \subseteq \gamma_{\vec{x}}(\langle B \rangle_{\vec{x}})$ by Proposition 4. For the converse, let $p \in \langle B \rangle_{\vec{x}}$. Then $p = \sum_{i=1}^{s} u_i p_i$ for some $u_i \in \mathbb{Z}_m[\vec{x}]$ and $p_i \in B$. Observe $\gamma_{\vec{x}}(B) \subseteq \gamma_{\vec{x}}(p_i) \subseteq \gamma_{\vec{x}}(u_i p_i)$ hence $\gamma_{\vec{x}}(B) \subseteq \gamma_{\vec{x}}(p)$ therefore $\gamma_{\vec{x}}(B) \subseteq \gamma_{\vec{x}}(\langle B \rangle_{\vec{x}})$. □

Proof of Lemma 5. Let $p \in \langle P \rangle_{\vec{x}}$. Then $p = \sum_{i=1}^{s} u_i p_i$ for $u_i \in \mathbb{Z}_m[\vec{x}]$ and $p_i \in P$. Observe $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p_i) \subseteq \gamma_{\vec{x}}(u_i p_i)$ hence $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p)$ thus $p \in \uparrow_{\vec{x}} P = P$. It follows $\langle P \rangle_{\vec{x}} \subseteq P$. The converse is immediate and the result follows. □

Proof of Proposition 5. First it will be shown that $\bot, \top, P_1 \sqcap P_2, P_1 \sqcup P_2 \in \mathsf{MPAD}_m[\vec{x}]$ so that the domain operations are well-defined:

- ($\bot$) Since $\bot = \mathbb{Z}_m[\vec{x}]$ it follows from extensivity of $\uparrow_{\vec{x}}$ that $\uparrow_{\vec{x}} \bot \supseteq \bot = \mathbb{Z}_m[\vec{x}]$. Thus $\uparrow_{\vec{x}} \bot = \mathbb{Z}_m[\vec{x}] = \bot$, so $\bot \in \mathsf{MPAD}_m[\vec{x}]$.

- ($\top$ and $P_1 \sqcap P_2$) Note, if $P \subseteq \mathbb{Z}_m[\vec{x}]$ then $\uparrow_{\vec{x}} P \in \mathsf{MPAD}_m[\vec{x}]$, since $\uparrow_{\vec{x}} \uparrow_{\vec{x}} P = \uparrow_{\vec{x}} P$ by idempotency. It follows immediately that $\top, P_1 \sqcap P_2 \in \mathsf{MPAD}_m[\vec{x}]$.

- ($P_1 \sqcup P_2$) It suffices to show that $\uparrow_{\vec{x}} (P_1 \sqcup P_2) \subseteq P_1 \sqcup P_2$ since the converse holds by extensivity of $\uparrow_{\vec{x}}$. To this end, let $p \in \uparrow_{\vec{x}} (P_1 \sqcup P_2) = \uparrow_{\vec{x}} (P_1 \cap P_2)$. Then, $\gamma_{\vec{x}}(P_1 \cap P_2) \subseteq \gamma_{\vec{x}}(p)$. Now, for $i \in \{1, 2\}$, it follows from $P_1 \cap P_2 \subseteq P_i$ and Lemma 9 that $\gamma_{\vec{x}}(P_i) \subseteq \gamma_{\vec{x}}(P_1 \cap P_2)$, hence $\gamma_{\vec{x}}(P_i) \subseteq \gamma_{\vec{x}}(p)$. But this implies $p \in \uparrow_{\vec{x}} P_i$ and thus $p \in P_i$, since $P_i \in \mathsf{MPAD}_m[\vec{x}]$. Therefore, $p \in P_1 \cap P_2$, hence $\uparrow_{\vec{x}} (P_1 \sqcup P_2) \subseteq P_1 \cap P_2 = P_1 \sqcup P_2$, as required.

Now, note that if $P_1, P_2 \in \mathsf{MPAD}_m[\vec{x}]$ and $\gamma_{\vec{x}}(P_1) = \gamma_{\vec{x}}(P_2)$ then $\gamma_{\vec{x}}(\uparrow_{\vec{x}} P_1) = \gamma_{\vec{x}}(\uparrow_{\vec{x}} P_2)$, hence $P_1 = P_2$ by Proposition 4. In particular, since $\wp(\mathbb{Z}_m^n)$ is finite, so too must be $\mathsf{MPAD}_m[\vec{x}]$. Moreover, since $\subseteq$ is a partial order over $\mathsf{MPAD}_m[\vec{x}]$, so too is $\sqsubseteq$. Thus it only remains to show that the domain operation satisfy the defining properties of a lattice:

- ($\bot$) If $P \in \mathsf{MPAD}_m[\vec{x}]$ then $P \subseteq \bot$, thus $\bot \sqsubseteq P$.
- ($\top$) If $P \in \mathsf{MPAD}_m[\vec{x}]$ then $\emptyset \subseteq P$. Thus, by monotonicity of $\uparrow_{\vec{x}}$, $\top = \uparrow_{\vec{x}} \emptyset \subseteq \uparrow_{\vec{x}} P = P$, hence $\uparrow_{\vec{x}} P \sqsubseteq \top$.
- ($\sqcap$) If $P_1, P_2 \in \mathsf{MPAD}_m[\vec{x}]$ then for each $i \in \{1,2\}$, $P_i \subseteq P_1 \cup P_2 \subseteq \uparrow_{\vec{x}} (P_1 \cup P_2) = P_1 \sqcap P_2$. Thus, $P_1 \sqcap P_2 \sqsubseteq P_i$ for each $i \in \{1,2\}$, so $P_1 \sqcap P_2$ is a lower bound for P_1, P_2. To prove it is the greatest lower bound, let $Q \in \mathsf{MPAD}_m[\vec{x}]$ and suppose $Q \sqsubseteq P_i$ for each $i \in \{1,2\}$. Then, $P_i \subseteq Q$ for each $i \in \{1,2\}$ so $P_1 \cup P_2 \subseteq Q$. It follows $P_1 \sqcap P_2 = \uparrow_{\vec{x}} (P_1 \cup P_2) \subseteq \uparrow_{\vec{x}} Q = Q$. Thus $Q \sqsubseteq P_1 \sqcap P_2$ so $P_1 \sqcap P_2$ is the greatest lower bound of P_1, P_2.
- ($\sqcup$) If $P_1, P_2 \in \mathsf{MPAD}_m[\vec{x}]$ then $P_1 \sqcup P_2 = P_1 \cap P_2 \subseteq P_i$ for each $i \in \{1,2\}$. Thus, $P_1 \sqcup P_2$ is an upper bound for P_1, P_2. To prove it is the least upper bound, let $Q \in \mathsf{MPAD}_m[\vec{x}]$ and suppose $P_i \sqsubseteq Q$ for each $i \in \{1,2\}$. Then, $Q \subseteq P_i$ for each $i \in \{1,2\}$, so $Q \subseteq P_1 \cap P_2 = P_1 \sqcup P_2$. It follows $P_1 \sqcup P_2 \sqsubseteq Q$, so $P_1 \sqcup P_2$ is the least upper bound of P_1, P_2.

□

Proof of Proposition 6. It must be shown that $\alpha_{\vec{x}}(A) \sqsubseteq P$ iff $A \subseteq \gamma_{\vec{x}}(P)$, and moreover that $\alpha_{\vec{x}}\gamma_{\vec{x}}$ is the identity on $\mathsf{MPAD}_m[\vec{x}]$:

- Suppose $\alpha_{\vec{x}}(A) \sqsubseteq P$ and let $p \in P$. Then, $P \subseteq \alpha_{\vec{x}}(A)$ so $p \in \alpha_{\vec{x}}(A)$. It follows $A \subseteq \gamma_{\vec{x}}(p)$ and thus $A \subseteq \gamma_{\vec{x}}(P)$. For the converse, suppose $A \subseteq \gamma_{\vec{x}}(P)$ and let $p \in P$. Then $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p)$ so $A \subseteq \gamma_{\vec{x}}(p)$. It follows $p \in \alpha_{\vec{x}}(A)$, thus $P \subseteq \alpha_{\vec{x}}(A)$ and hence $\alpha_{\vec{x}}(A) \sqsubseteq P$.
- Let $P \in \mathsf{MPAD}_m[\vec{x}]$. Then, since $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(P)$, it follows from the previous property (applied to $A = \gamma_{\vec{x}}(P)$) that $\alpha_{\vec{x}}(\gamma_{\vec{x}}(P)) \sqsubseteq P$. Thus, $P \subseteq \alpha_{\vec{x}}(\gamma_{\vec{x}}(P))$. For the opposite inclusion, let $p \in \alpha_{\vec{x}}(\gamma_{\vec{x}}(P))$. Then, $\gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(p)$, from which it follows $p \in \uparrow_{\vec{x}} P = P$. Therefore $\alpha_{\vec{x}}(\gamma_{\vec{x}}(P)) \subseteq P$ and the result follows.

□

A.2 Proofs for worklist algorithm

for proposition 7. For termination, observe that $\langle N \to \mathsf{MPAD}_m[\vec{x}], \sqsubseteq\rangle$ is a finite lattice, hence the sequence (σ_k) must become stationary. Thus, there exists $\ell \in \mathbb{N}$ for which $\sigma_\ell = \sigma_{\ell+1} = \ldots$, hence $w_\ell \supset w_{\ell+1} \supset \ldots$ is a strictly decreasing sequence. It follows $w_{\ell'} = \emptyset$ for some $\ell' \geq \ell$, hence the algorithm terminates. Put $\sigma^* = \sigma_{\ell'}$.

For the second property, it is sufficient to prove that for each k, $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(\sigma_k(n'))$ for all $\langle n, s, n'\rangle \in E \setminus w_k$. The result then follows by setting $k = \ell'$, since $\sigma_{\ell'} = \sigma^*$ and $E \setminus w_{\ell'} = E \setminus \emptyset = E$. Proof is by induction.

- Suppose $k = 0$ and let $e = \langle n, s, n'\rangle \in E \setminus w_0$. Then $n \neq n^*$ hence $\gamma_{\vec{x}}(\sigma_0(n)) = \gamma_{\vec{x}}(\bot) = \emptyset$. It follows $[\![s]\!](\gamma_{\vec{x}}(\sigma_0(n))) = \emptyset \subseteq \gamma_{\vec{x}}(\sigma_0(n'))$ as required.

- Suppose the result holds for $k \geq 0$ and let $e = \langle n, s, n'\rangle \in E \setminus w_{k+1}$. It follows $\sigma_{k+1}(n) = \sigma_k(n)$ and therefore $[\![s]\!](\gamma_{\vec{x}}(\sigma_{k+1}(n))) = [\![s]\!](\gamma_{\vec{x}}(\sigma_k(n)))$. Thus, the result follows if $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(\sigma_{k+1}(n'))$. To show this:

  - Suppose $e = e_k$.
    - Suppose the **then** case is taken at step k. Then, by the defining property of P_k it holds $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(P_k)$. Moreover, $P_k \sqsubseteq \sigma_k(n')$ and $\sigma_k = \sigma_{k+1}$ hence $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(\sigma_{k+1}(n'))$.
    - Suppose the **else** case is taken at step k. Then $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(P_k)$ and $\sigma_{k+1}(n') = \sigma_k(n') \sqcup P_k$ hence $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(\sigma_{k+1}(n'))$.
  - Suppose $e \neq e_k$. Then $e \in E \setminus w_k$ hence by induction $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(\sigma_k(n'))$ but $\sigma_k \sqsubseteq \sigma_{k+1}$ hence $[\![s]\!](\gamma_{\vec{x}}(\sigma_k(n))) \subseteq \gamma_{\vec{x}}(\sigma_{k+1}(n'))$.

□

for corollary 2. Proceed by structural induction on Π_G. First, $[\![\epsilon]\!](\mathbb{Z}_m^d) = \mathbb{Z}_m^d = \gamma(\top) = \gamma(\sigma_0(n^*)) = \gamma(\sigma^*(n^*)) \subseteq \gamma(\sigma^*(\mathsf{end}(\epsilon)))$, thus $\epsilon \in \gamma(\sigma^*)$. Now, suppose $\pi \in \gamma(\sigma^*)$ and let $e = \langle n, s, n'\rangle \in E$ where $n = \mathsf{end}(\pi)$. Then, $[\![\pi \cdot e]\!](\mathbb{Z}_m^d) = [\![s]\!]([\![\pi]\!](\mathbb{Z}_m^d)) \subseteq [\![s]\!](\gamma(\sigma^*(\mathsf{end}(\pi)))) = [\![s]\!](\gamma(\sigma^*(n))) \subseteq \gamma(\sigma^*(n')) = \gamma(\sigma^*(\mathsf{end}(\pi \cdot e)))$, where the second step follows by inductive hypothesis and the fourth by

Proposition 7. Now let $\omega \preceq \pi$. By induction it follows $[\![\omega]\!](\mathbb{Z}_m^d) \subseteq \gamma(\sigma^*(\mathsf{end}(\omega)))$. Therefore $[\![\omega]\!](\mathbb{Z}_m^d) \subseteq \gamma(\sigma^*(\mathsf{end}(\omega)))$ for all $\omega \preceq \pi \cdot e$. Thus $\pi \cdot e \in \gamma(\sigma^*)$. □

A.3 Proofs for Gröbner bases

Proof of Lemma 3. Let $\prec = \prec_{\vec{x}:\vec{w}}$ and $d = |\vec{x}|$. First, note if $p \in \mathbb{Z}_m[\vec{w}]$ then the result holds immediately with each $q_\ell = 0$ and $r = p$. In particular, the result holds for $p = 0$, so suppose $p \neq 0$ and let $\mathsf{lt}_\prec(p) = c\vec{x}^{\vec{\alpha}}\vec{w}^{\vec{\beta}}$. Proceed by induction on $\vec{\alpha}$:

- Suppose $\vec{\alpha} = \vec{0}$. Since $\mathsf{lt}_\prec(p) = c\vec{x}^{\alpha}\vec{w}^{\vec{\beta}}$ it follows for all terms $d\vec{x}^{\vec{\alpha}'}\vec{w}^{\vec{\beta}'}$ in p satisfy $\vec{x}^{\vec{\alpha}'}\vec{w}^{\vec{\beta}'} \preceq \vec{x}^{\vec{\alpha}}\vec{w}^{\vec{\beta}}$. But then, by definition of $\prec$ this implies $\vec{\alpha}' \leq \vec{\alpha} = \vec{0}$ lexicographically, hence $\vec{\alpha}' = \vec{0}$. It follows $\mathsf{vars}(p) \subseteq \mathsf{vars}(\vec{w})$ hence $p \in \mathbb{Z}_m[\vec{w}]$ and result then follows from argument above.

- Suppose $\vec{\alpha} > \vec{0}$. Then, by assumption $\alpha_\ell > 0$ for some $1 \leq \ell \leq d$. But then, p is $\prec$-reducible by $x_\ell - W_\ell$. In particular, $p = t(x_\ell - W_\ell) + q$ for some $q \in \mathbb{Z}_m[\vec{x} : \vec{w}]$ where either $q = 0$ or else $\mathsf{lm}_\prec(q) \prec \vec{x}^{\vec{\alpha}}\vec{w}^{\vec{\beta}}$, by Lemma 1. If $q = 0$ then the result follows immediately with $q_\ell = t$, $q_i = 0$ for $i \neq \ell$ and $r = 0$. Otherwise, by the inductive hypothesis it holds $q = q_1(x_1 - W_\ell) + \cdots + q_d(x_d - W_d) + r$ where $r \in \mathbb{Z}_m[\vec{w}]$. It thus follows $p = t(x_\ell - W_\ell) + q = q_1(x_1 - W_1) + \cdots + (q_\ell + t)(x_\ell - W_\ell) + \cdots + q_d(x_d - W_d) + r$ and the result follows.

□

A.4 Proofs for variable elimination and join

Proof for lemma 6. Let $\vec{a} \in \gamma_{\vec{x}}(P)$. Then $[\![p]\!]_{\vec{x}}(\vec{a}) = 0$ for all $p \in P$. Because $P \subseteq \mathbb{Z}_m[\vec{x}]$ it follows $[\![p]\!]_{\vec{x}}(\pi_i(\vec{a})) = 0$ for all $p \in P$ thus $\pi_i(\vec{a}) \in \gamma_{\vec{x}}(P) \subseteq \gamma_{\vec{x}}(Q)$. Hence $[\![q]\!]_{\vec{x}}(\pi_i(\vec{a})) = 0$ for all $q \in Q$. Because $Q \subseteq \mathbb{Z}_m[\vec{x}]$ it follows $[\![q]\!]_{\vec{y}}(\vec{a}) = 0$ for all $q \in Q$ therefore $\vec{a} \in \gamma_{\vec{y}}(Q)$. □

Proof of Corollary 5. If $A = \gamma_{\vec{x}}(P)$ then $\mathsf{elim}[x_j](\alpha_{\vec{x}}(A)) = \alpha_{\pi_j(\vec{x})}(\pi_j(A))$ by Proposition 8. Since $P = \uparrow_{\vec{x}} P = \alpha_{\vec{x}}(\gamma_{\vec{x}}(P))$ then $\mathsf{elim}[x_j](P) = \mathsf{elim}[x_j](\alpha_{\vec{x}}(A))$. But the range of $\alpha_{\pi_j(\vec{x})}$ is $\mathsf{MPAD}_m[\pi_j(\vec{x})]$ so the result follows. □

Proof for corollary 4. Let $\vec{a} \in \gamma_{\vec{x}}(Q)$ and $p \in P$. Then $p \in P \subseteq \uparrow_{\vec{x}} P \subseteq \uparrow_{\vec{x}} Q$. Therefore $\vec{a} \in \gamma_{\vec{x}}(Q) \subseteq \gamma_{\vec{x}}(p)$. Thus $\vec{a} \in \gamma_{\vec{x}}(P)$ hence $\gamma_{\vec{x}}(Q) \subseteq \gamma_{\vec{x}}(P)$. By corollary 3 it follows $\gamma_{\vec{y}}(Q) \subseteq \gamma_{\vec{y}}(P)$. Now let $q \in \uparrow_{\vec{y}} P$. Then $\gamma_{\vec{y}}(Q) \subseteq \gamma_{\vec{y}}(P) \subseteq \gamma_{\vec{y}}(q)$ hence $q \in \uparrow_{\vec{y}} Q$. □

Proof of Proposition 8. Let $P = \alpha_{\vec{x}}(A)$ and $Q = \alpha_{\pi_j(\vec{x})}(\pi_j(A))$. To show $\mathsf{elim}[x_j](P) \subseteq Q$, let $p \in \mathsf{elim}[x_j](P)$ and $\vec{a} \in \pi_j(A)$. Then, there exists $\vec{b} \in A$ such that $\pi_j(\vec{b}) = \vec{a}$. It thus follows $[\![p]\!]_{\pi_j(\vec{x})}(\vec{a}) = [\![p]\!]_{\vec{x}}(\vec{b}) = 0$ and so $\pi_j(A) \subseteq \gamma_{\pi_j(\vec{x})}(\mathsf{elim}[x_j](P))$. Thus, from Proposition 6 it follows $\alpha_{\pi_j(\vec{x})}(\pi_j(A)) \sqsubseteq \mathsf{elim}[x_j](P)$, or equivalently $\mathsf{elim}[x_j](P) \subseteq Q$.

To show $Q \subseteq \mathsf{elim}[x_j](P)$, let $q \in Q$ and $\vec{b} \in A$. Then, $[\![q]\!]_{\vec{x}}(\vec{b}) = [\![q]\!]_{\pi_j(\vec{x})}(\pi_j(\vec{b})) = 0$, since $\pi_j(\vec{b}) \in \pi_j(A) \subseteq \gamma_{\pi_j(\vec{x})}(q)$. Therefore, $A \subseteq \gamma_{\vec{x}}(q)$, from which it follows $q \in \uparrow_{\vec{x}} P = P$. Since $q \in \uparrow_{\vec{x}} P$ and $q \in \mathbb{Z}_m[\pi_j(\vec{x})]$ it thus follows $q \in \mathsf{elim}[x_j](P)$, hence $Q \subseteq \mathsf{elim}[x_j](P)$. □

Proof for proposition 9. It must be shown that $\mathsf{elim}[x_j](\langle B\rangle_{\vec{x}}) = \langle \mathsf{elim}[x_j](B')\rangle_{\pi_j(\vec{x})}$.

- Let $p \in \langle \mathsf{elim}[x_j](B')\rangle_{\pi_j(\vec{x})}$. Then $p = \sum_{i=1}^{s} u_i p_i$ for $p_i \in \mathsf{elim}[x_j](B')$ and $u_i \in \mathbb{Z}_m[\pi_j(\vec{x})]$. Since p_j and u_j are independent of x_j so is p too. But because $\mathsf{elim}[x_j](B') \subseteq B' \subseteq \langle B\rangle_{\vec{x}}$ it follows $p \in \langle B\rangle_{\vec{x}}$. Since p is independent of x_j it follows $p \in \mathsf{elim}[x_j](\langle B\rangle_{\vec{x}})$, as required.

- Let $p \in \mathsf{elim}[x_j](\langle B\rangle_{\vec{x}})$. Since $\langle B'\rangle_{\vec{x}} = \langle B\rangle_{\vec{x}}$ it follows $p \in \mathsf{elim}[x_j](\langle B'\rangle_{\vec{x}})$. Since B' is a Gröbner basis, by repeatedly reducing p by elements of B' it follows $p = \sum_{i=1}^{s} u_i p_i$ where $p_i \in B'$ and $u_i \in \mathbb{Z}_m[\vec{x}]$. Let $p_i = p_i' + x_j p_i''$

where p'_i is independent of x_j. Then $p = \sum_{i=1}^{s} u_i p'_i + x_j \sum_{i=1}^{s} u_i p''_i$. Since p is independent of x_j, $\sum_{i=1}^{s} u_i p''_i = 0$ hence $p = \sum_{i=1}^{s} u_i p'_i$. Repeating the argument for $u_i = u'_i + x_j u''_i$ where u'_i is independent of x_j it follows $p = \sum_{i=1}^{s} u'_i p'_i$ where $p'_i \in \mathsf{elim}[x_j](B')$ and $u'_i \in \mathbb{Z}_m[\pi_j(\vec{x})]$. Thus $p \in \langle \mathsf{elim}[x_j](B') \rangle_{\pi_j(\vec{x})}$, as required.

□

Proof for proposition 10. The assumption means $\mathsf{elim}[w](\langle wB_1 \cup (1-w)B_2 \rangle_{\vec{y}}) = \langle B \rangle_{\vec{x}}$ where $\vec{x} = \langle x_1, \ldots, x_d \rangle$ and $\vec{y} = \langle w, x_1, \ldots, x_d \rangle$

- Let $p \in \langle B_1 \rangle_{\vec{x}} \cap \langle B_2 \rangle_{\vec{x}}$ hence $p \in \langle B_1 \rangle_{\vec{x}}$ and $p \in \langle B_2 \rangle_{\vec{x}}$. It follows $p = \sum_{i=1}^{s} u_i p_i$ for $p_i \in B_1$ and $u_i \in \mathbb{Z}_m[\vec{x}]$ and $p = \sum_{j=1}^{t} v_j q_j$ for $q_j \in B_2$ and $v_j \in \mathbb{Z}_m[\vec{x}]$. Thus $p = wp + (1-w)p = \sum_{i=1}^{s} u_i(wp_i) + \sum_{j=1}^{t} v_j(1-w)q_j \in \langle wB_1 \cup (1-w)B_2 \rangle_{\vec{y}}$. Since $w \notin \mathsf{vars}(p)$ it thus follows $p \in \mathsf{elim}[w](\langle wB_1 \cup (1-w)B_2 \rangle_{\vec{y}})$.

- Let $p \in \langle B \rangle_{\vec{x}}$. Then $p = \sum_{i=1}^{s} u_i(wp_i) + \sum_{j=1}^{t} v_j(1-w)q_j$ for $p_i \in B_1$, $q_j \in B_2$ and $u_i, v_j \in \mathbb{Z}_m[\vec{y}]$. Since $w \notin \mathsf{vars}(p)$ it follows that $p = p[w \mapsto 0] = \sum_{j=1}^{t} v'_j q_j$ where $v'_j = v_j[w \mapsto 0]$ and $o[w \mapsto x]$ denotes substituting x for w throughout the object o. Likewise $p = p[w \mapsto 1] = \sum_{i=1}^{s} u'_i p_i$ where $u'_i = u_i[w \mapsto 1]$. Since $u'_i, v'_j \in \mathbb{Z}_m[\vec{x}]$ it follows $p \in \langle B_1 \rangle_{\vec{x}}$ and $p \in \langle B_2 \rangle_{\vec{x}}$, hence $p \in \langle B_1 \rangle_{\vec{x}} \cap \langle B_2 \rangle_{\vec{x}}$, as required.

□

A.5 Proofs for cover and closure

Lemma 10. Let $F \subseteq \mathbb{Z}_m[\vec{w}]$ and suppose $F \cup \{w_i - 2^j w + r\} \to_{\mathsf{elim}[w_i]} F'$, where $w \notin \mathsf{vars}(\vec{w})$ and $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$. Then,

$$\gamma_{w:\pi_i(\vec{w})}(F') = \{b : \pi_i(\vec{a}) \mid b : \vec{a} \in \gamma_{w:\vec{w}}(F \cup \{w_i - 2^j w + r\})\}$$

Proof of Lemma 10. For brevity, put $G = F \cup \{w_i - 2^j w + r\}$ and recall $\gamma_{w:\vec{w}}(G) = \gamma_{w:\vec{w}}(\langle G\rangle_{w:\vec{w}})$. The assumption $G \to_{\mathsf{elim}[w_i]} F'$ implies $\mathsf{elim}[w_i](\langle G\rangle_{w:\vec{w}}) = \langle F'\rangle_{w:\pi_i(\vec{w})}$. Thus, $\gamma_{w:\pi_i(\vec{w})}(F') = \gamma_{w:\pi_i(\vec{w})}(\langle F'\rangle_{w:\pi_i(\vec{w})}) = \gamma_{w:\pi_i(\vec{w})}(\mathsf{elim}[w_i](\langle G\rangle_{w:\vec{w}}))$. It thus suffices to show

$$\gamma_{w:\pi_i(\vec{w})}(\mathsf{elim}[w_i](\langle G\rangle_{w:\vec{w}})) = \{b : \pi_i(\vec{a}) \mid b : \vec{a} \in \gamma_{w:\vec{w}}(\langle G\rangle_{w:\vec{w}})\}.$$

To that end:

- Let $b : \vec{a} \in \gamma_{w:\vec{w}}(\langle G\rangle_{w:\vec{w}})$ and $f \in \mathsf{elim}[w_i](\langle G\rangle_{w:\vec{w}})$. Then, $f \in \langle G\rangle_{w:\vec{w}}$, so $[\![f]\!]_{w:\vec{w}}(b : \vec{a}) = 0$. But since also $f \in \mathbb{Z}_m[w : \pi_i(\vec{w})]$, it follows $[\![f]\!]_{w:\pi_i(\vec{w})}(b : \pi_i(\vec{a})) = [\![f]\!]_{w:\vec{w}}(b : \vec{a}) = 0$ hence $b : \pi_i(\vec{a}) \in \gamma_{w:\pi_i(\vec{w})}(\mathsf{elim}[w_i](\langle G\rangle_{w:\vec{w}})$.
- Let $b : \vec{a}' \in \gamma_{w:\pi_i(\vec{w})}(\mathsf{elim}[w_i](\langle G\rangle_{w:\vec{w}})$ and $f \in \langle G\rangle_{w:\vec{w}}$. Then, $f = u(w_i - 2^j w + r) + q$ for some $u \in \mathbb{Z}_m[w : \vec{w}]$ and $q \in \mathbb{Z}_m[w : \pi_i(\vec{w})]$. Since $q = f - u(w_i - 2^j w + r)$ it follows $q \in \langle G\rangle_{w:\vec{w}}$, thus $q \in \mathsf{elim}[w_i](\langle G\rangle_{w:\vec{w}})$. Now, let $\vec{a} = \langle a'_1, \ldots, a'_{i-1}, a', a'_{i+1}, \ldots, a'_d\rangle$ where $a' = 2^j b - [\![r]\!]_{w:\vec{w}}(b : \vec{a}')$. Then, $[\![q]\!]_{w:\vec{w}}(b : \vec{a}) = [\![q]\!]_{w:\pi_i(\vec{w})}(b : \pi_i(\vec{a})) = [\![q]\!]_{w:\pi_i(\vec{w})}(b : \vec{a}') = 0$ and also $[\![w_i - 2^j w + r]\!]_{w:\vec{w}}(b : \vec{a}) = 0$. Thus $[\![f]\!]_{w:\vec{w}}(b : \vec{a}) = 0$ hence $b : \vec{a} \in \gamma_{w:\vec{w}}(\langle G\rangle_{w:\vec{w}})$ where $\vec{a}' = \pi_i(\vec{a})$.

□

Lemma 11. Let $S = \langle \vec{W}, F\rangle \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ and suppose for each $1 \leq \ell \leq d$

- $W_\ell = 2^{k_\ell} w_\ell + q_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d\rangle \in \gamma_{\vec{w}}(F)$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k_\ell}, \ldots, v_d\rangle \in \gamma_{\vec{w}}(F)$

where $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ and $0 \leq k_\ell \leq \omega$. Let $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$ satisfy $\mathbf{safe}(\vec{W}, w_i, r)$, $\vec{W}' = \vec{W}[w_i \mapsto 2^j w - r]$ and $F \cup \{w_i - 2^j w + r\} \rightarrow_{\mathsf{elim}[w_i]} F'$, where $w \notin \mathsf{vars}(\vec{w})$. Then, for $1 \leq \ell \leq d$,

- $W'_\ell = 2^{k'_\ell} w'_\ell + q'_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d \rangle \in \gamma_{\vec{w}'}(F')$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k'_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}'}(F')$

where $\vec{w}' = \vec{w}[i \mapsto w]$, $q'_\ell \in \mathbb{Z}_m[w'_{\ell+1}, \ldots, w'_d]$ and $k'_\ell = \begin{cases} \min(k_i + j, \omega) & \text{if } i = \ell \\ k_\ell & \text{otherwise} \end{cases}$

Proof of Lemma 11. Let $1 \leq \ell \leq d$. To prove the first property:

- Suppose $\ell \neq i$. Then $W'_\ell = W_\ell[w_i \mapsto 2^j w - r] = (2^{k_\ell} w_\ell + q_\ell)[w_i \mapsto 2^j w - r] = 2^{k_\ell} w_\ell + q_\ell[w_i \mapsto 2^j w - r] = 2^{k_\ell} w_\ell + q'_\ell$ where $q'_\ell = q_\ell[w_i \mapsto 2^j w - r]$. Thus $k'_\ell = k_\ell$. Now:
  - If $\ell < i$ then $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ and since $2^j w - r \in \mathbb{Z}_m[w, w_{i+1}, \ldots, w_d]$ it follows $q'_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_{i-1}, w, w_{i+1}, \ldots, w_d] = \mathbb{Z}_m[w'_{\ell+1}, \ldots, w'_d]$.
  - If $\ell > i$ then $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ thus $w_i \notin \mathsf{vars}(q_\ell)$. Thus $q'_\ell = q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d] = \mathbb{Z}_m[w'_{\ell+1}, \ldots, w'_d]$.

  Thus, in either case $q'_\ell \in \mathbb{Z}_m[w'_{\ell+1}, \ldots, w'_d]$, as required.
- Suppose $\ell = i$. Then $W'_i = W_i[w_i \mapsto 2^j w - r] = (2^{k_i} w_i + q_i)[w_i \mapsto 2^j w - r] = 2^{k_i}(2^j w - r) + q_i = 2^{k_i + j} w + q'_i$ where $q'_i = -2^{k_i} r + q_i \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d] = \mathbb{Z}_m[w'_{i+1}, \ldots, w'_d]$. Now if $k_i + j \geq \omega$ then $2^{k_i + j} = 0 = 2^\omega$ hence $2^{k_i + j} = 2^{\min(k_i + j, \omega)} = 2^{k'_i}$. Since $w = w'_i$ it follows $W'_i = 2^{k'_i} w'_i + q'_i$ with $q'_i \in \mathbb{Z}_m[w'_{i+1}, \ldots, w'_d]$, as required.

To prove the second property, let $\vec{v} \in \gamma_{\vec{w}'}(F')$. Then, $v_i : \pi_i(\vec{v}) \in \gamma_{w : \pi_i(\vec{w})}(F')$. Thus, from Lemma 10 there exists $\vec{v}' \in \mathbb{Z}_m^d$ such that $v_i : \vec{v}' \in \gamma_{w:\vec{w}}(F \cup \{w_i - 2^j w + r\})$ and $\pi_i(\vec{v}') = \pi_i(\vec{v})$. In particular, it follows $\vec{v}' \in \gamma_{\vec{w}}(F)$ and $v_i : \vec{v}' \in \gamma_{w:\vec{w}}(w_i - 2^j w + r)$, hence $v'_i - 2^j v_i + [\![r]\!]_{\langle w_{i+1}, \ldots, w_d \rangle}(\langle v'_{i+1}, \ldots, v'_d \rangle) = v'_i - 2^j v_i + [\![r]\!]_{w:\vec{w}}(v_i : \vec{v}') = [\![w_i - 2^j w + r]\!]_{w:\vec{w}}(v_i : \vec{v}') = 0$. Now:

- Suppose $\ell < i$ and let $\vec{v}'' = \langle v'_1, \ldots, v'_{\ell-1}, v'_\ell + 2^{\omega-k_\ell}, v'_{\ell+1}, \ldots, v'_d\rangle$. Since $\vec{v}' \in \gamma_{\vec{w}}(F)$ it follows from the inductive hypothesis that $\vec{v}'' \in \gamma_{\vec{w}}(F)$, hence $v_i : \vec{v}'' \in \gamma_{w:\vec{w}}(F)$. Moreover, $[\![r]\!]_{w:\vec{w}}(v_i : \vec{v}'') = [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v''_{i+1}, \ldots, v''_d\rangle) = [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_d\rangle)$, since $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, hence $[\![w_i - 2^j w + r]\!]_{w:\vec{w}}(v_i : \vec{v}'') = v''_i - 2^j v_i + [\![r]\!]_{w:\vec{w}}(v_i : \vec{v}'') = v'_i - 2^j v_i + [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_d\rangle) = 0$. It follows $v_i : \vec{v}'' \in \gamma_{w:\vec{w}}(F \cup \{w_i - 2^j w + r\})$. Thus, by Lemma 10 it follows $v_i : \pi_i(\vec{v}'') \in \gamma_{w:\pi_i(\vec{w})}(F')$. But, $v_i : \pi_i(\vec{v}'') = \langle v_i, v'_1, \ldots, v'_{\ell-1}, v'_\ell + 2^{\omega-k_\ell}, v'_{\ell+1}, \ldots, v'_{i-1}, v'_{i+1}, \ldots, v'_d\rangle = \langle v_i, v_1, \ldots, v_{\ell-1}, v_\ell + 2^{\omega-k'_\ell}, v_{\ell+1}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_d\rangle$, since $k_\ell = k'_\ell$ and $\pi_i(\vec{v}') = \pi_i(\vec{v})$. Reordering variables yields $\langle v_1, \ldots, v_\ell + 2^{\omega-k'_\ell}, \ldots, v_d\rangle \in \gamma_{\vec{w}'}(F')$, as required.

- Suppose $\ell > i$ and let $\vec{v}'' = \langle v'_1, \ldots, v'_{i-1}, v''_i, v'_{i+1}, \ldots, v'_\ell + 2^{\omega-k_\ell}, \ldots, v'_d\rangle$ where $v''_i = 2^j v_i - [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_{\ell-1}, v'_\ell + 2^{\omega-k_\ell}, v'_{\ell+1}, \ldots, v'_d\rangle)$. It will be shown $v''_i = v'_i + c2^{\omega-k_i}$ for some $c \in \mathbb{Z}_m$.

  First, note since $v'_i - 2^j v_i + [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_d\rangle) = 0$ it follows

$$\begin{aligned} v''_i = v'_i &+ [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_\ell, \ldots, v'_d\rangle) \\ &- [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_\ell + c2^{\omega-k_\ell}, \ldots, v'_d\rangle) \end{aligned} \tag{1}$$

  Now, let t be a term in r. Since $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$ it follows $t = a w_{i+1}^{\alpha_1} \cdots w_d^{\alpha_{d-i}}$ for some a. It will be shown

$$\begin{aligned} &[\![t]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_\ell, \ldots, v'_d\rangle) \\ &\quad - [\![t]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_\ell + 2^{\omega-k_\ell}, \ldots, v'_d\rangle) = c2^{\omega-k_i} \end{aligned} \tag{2}$$

  for some $c \in \mathbb{Z}_m$. Note this holds iff $v'^{\alpha_1}_{i+1} \cdots [a(v'^{\alpha_{\ell-i}}_\ell - (v'_\ell + 2^{\omega-k_\ell})^{\alpha_{\ell-i}})] \cdots v'^{\alpha_{d-i}}_d = c2^{\omega-k_i}$. Hence, if it can be shown that $a(v'^{\alpha_{\ell-i}}_\ell - (v'_\ell + 2^{\omega-k_\ell})^{\alpha_{\ell-i}}) = c2^{\omega-k_i}$ for some c, the result will follow. First, note if $\alpha_{\ell-i} = 0$ then $a(v'^{\alpha_{\ell-i}}_\ell - (v'_\ell + 2^{\omega-k_\ell})^{\alpha_{\ell-i}}) = a(1-1) = 0 = c2^{\omega-k_i}$ where $c = 0$, as required. Thus, assume $\alpha_{\ell-i} > 0$.

  Now, let $\alpha > 0$ and $v, e \in \mathbb{Z}_m$ be arbitrary. Then, letting $b_s = \binom{\alpha}{s}$ for each

$0 \leq s \leq \alpha$ it follows

$$\begin{aligned}(v+e)^\alpha &= \textstyle\sum_{s=0}^{\alpha} b_s v^s e^{\alpha-s} \\ &= b_\alpha v^\alpha + \textstyle\sum_{s=0}^{\alpha-1} b_s v^s e^{\alpha-s} \\ &= v^\alpha + e\textstyle\sum_{s=0}^{\alpha-1} b_s v^s e^{(\alpha-1)-s} \\ &= v^\alpha + c'e\end{aligned}$$

where $c' = \sum_{s=0}^{\alpha-1} b_s v^s e^{\alpha-1-s}$. In particular, setting $\alpha = \alpha_{\ell-i}$, $v = v'_\ell$ and $e = 2^{\omega-k'_\ell}$ and rearranging yields

$$(v'_\ell + 2^{\omega-k'_\ell})^{\alpha_{\ell-i}} - {v'_\ell}^{\alpha_{\ell-i}} = c'2^{\omega-k'_\ell} \tag{3}$$

Now, since $\alpha_{\ell-i} > 0$ it follows $w_\ell \in \mathsf{vars}(t)$. Thus, since $\mathbf{safe}(\vec{W}, w_i, r)$ holds it follows $k_i + \mathsf{rank}(a) \geq k_\ell$, hence $\omega - k'_\ell + \mathsf{rank}(a) = \omega - k_\ell + \mathsf{rank}(a) \geq \omega - k_i$. In particular, $\mathsf{rank}(a2^{\omega-k'_\ell}) = \omega - k'_\ell + \mathsf{rank}(a) \geq \omega - k_i = \mathsf{rank}(2^{\omega-k_i})$, hence $a2^{\omega-k'_\ell} = c''2^{\omega-k_i}$ for some $c'' \in \mathbb{Z}_m$. Combining this with equation (3) yields that $a({v'_\ell}^{\alpha_{\ell-i}} - (v'_\ell + 2^{\omega-k'_\ell})^{\alpha_{\ell-i}}) = -ac'2^{\omega-k'_\ell} = -c'c''2^{\omega-k_i} = c2^{\omega-k_i}$ where $c = -c'c''$. It follows equation (2) holds for each term $t \in r$.

Now, since equation (2) holds for each term $t \in r$ it follows by linearity of polynomial evaluation that

$$\begin{aligned}&[\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1},\ldots,v'_\ell,\ldots,v'_d\rangle) \\ &\quad - [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1},\ldots,v'_\ell + 2^{\omega-k'_\ell},\ldots,v'_d\rangle) = c2^{\omega-k_i}\end{aligned}$$

for some $c \in \mathbb{Z}_m$. In particular, it follows from equation (1) that $v''_i = v'_i + c2^{\omega-k_i}$ for some $c \in \mathbb{Z}_m$, as required.

Substituting this equation into $\vec{v}''$ yields $\vec{v}'' = \langle v'_1,\ldots,v'_{i-1}, v'_i + c2^{\omega-k_i}, v'_{i+1},\ldots,v'_{\ell-1}, v'_\ell + 2^{\omega-k_\ell}, v'_{\ell+1},\ldots,v'_d\rangle$ and since $\vec{v}' \in \gamma_{\vec{w}}(F)$, c applications of the inductive hypothesis for component i and a further application for component ℓ yields $\vec{v}'' \in \gamma_{w:\vec{w}}(F)$. Moreover, since $[\![r]\!]_{w:\vec{w}}(v_i : \vec{v}'') = [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v''_{i+1},\ldots,v''_d\rangle) = [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1},\ldots,v'_\ell + 2^{\omega-k_\ell},\ldots,v'_d\rangle)$, it follows $[\![w_i - 2^j w + r]\!]_{w:\vec{w}}(v_i : \vec{v}'') = v''_i - 2^j v_i + [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1},\ldots,v'_{\ell-1}, v'_\ell + 2^{\omega-k_\ell}, v'_{\ell+1},\ldots,v'_d\rangle) = 0$. Thus $v_i : \vec{v}'' \in \gamma_{w:\vec{w}}(F \cup \{w_i - 2^j w + r\})$. Now, by Lemma 10, it follows $v_i : \pi_i(\vec{v}'') \in \gamma_{w:\pi_i(\vec{w})}(F')$. But, $v_i : \pi_i(\vec{v}'') = \langle v_i, v'_1,\ldots,v'_{i-1}, v'_{i+1},\ldots,v'_\ell +$

$2^{\omega-k_\ell}, \ldots, v'_d\rangle = \langle v_i, v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_\ell + 2^{\omega-k'_\ell}, \ldots, v_d\rangle$ since $k_\ell = k'_\ell$ and $\pi_i(\vec{v}') = \pi_i(\vec{v})$. Reordering variables yields $\langle v_1, \ldots, v_\ell + 2^{\omega-k'_\ell}, \ldots, v_d\rangle \in \gamma_{\vec{w}'}(F')$, as required.

- Suppose $\ell = i$ and let $\vec{v}'' = \langle v'_1, \ldots, v'_i + 2^{(\omega-k'_i)+j}, \ldots, v'_d\rangle$. Note since $k'_i = \min(\omega, k_i+j)$ it follows $0 \leq (\omega-k'_i)+j \leq \omega$. Then, $[\![w_i - 2^j w + r]\!]_{w:\vec{w}}((v_i + 2^{\omega-k'_i}) : \vec{v}'') = v'_i + 2^{(\omega-k'_i)+j} - 2^j(v_i + 2^{\omega-k'_i}) + [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_d\rangle) = v'_i - 2^j v_i + [\![r]\!]_{\langle w_{i+1},\ldots,w_d\rangle}(\langle v'_{i+1}, \ldots, v'_d\rangle) = 0$. Since also $\vec{v}'' \in \gamma_{\vec{w}}(F)$ it follows from 2^j applications of the inductive hypothesis that $\vec{v}'' \in \gamma_{\vec{w}}(F)$, hence $(v_i + 2^{\omega-k'_i}) : \vec{v}'' \in \gamma_{w:\vec{w}}(F)$. It follows $(v_i + 2^{\omega-k'_i}) : \vec{v}'' \in \gamma_{w:\vec{w}}(F \cup \{w_i - 2^j w + r\})$, hence by Lemma 10, $(v_i + 2^{\omega-k'_i}) : \pi_i(\vec{v}'') \in \gamma_{w:\pi(\vec{w})}(F')$. But, $(v_i + 2^{\omega-k'_i}) : \pi_i(\vec{v}'') = \langle v_i + 2^{\omega-k'_i}, v'_1, \ldots, v'_{i-1}, v'_{i+1}, \ldots, v'_d\rangle = \langle v_i + 2^{\omega-k'_i}, v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_d\rangle$. Reordering variables yields $\langle v_1, \ldots, v_i + 2^{\omega-k'_i}, \ldots, v_d\rangle \in \gamma_{\vec{w}'}(F')$, as required.

□

Proof for Proposition 11. Since $\langle \vec{W}'', F''\rangle = \textbf{constrain}(\langle \vec{W}, F\rangle, j, w_i, r)$ it follows

- $\vec{W}'' = \vec{W}'[w_i \mapsto w]$ where $\vec{W}' = \vec{W}[w_i \mapsto 2^j w - r]$,
- There exists $F' \in \mathbb{Z}_m[\vec{w}']$ such that $F \cup \{w_i - 2^j w + r\} \rightarrow_{\mathsf{elim}[w_i]} F'$ and

$$F'' = \begin{cases} F'[w \mapsto 0] & \text{if } W'_i = 2^\omega w + q \wedge q \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d] \\ F'[w \mapsto w_i] & \text{otherwise} \end{cases}$$

where $w \notin \mathsf{vars}(\vec{w})$ and $\vec{w}' = \vec{w}[i \mapsto w]$. Then, from Lemma 11, for all $1 \leq \ell \leq d$,

- $W'_\ell = 2^{k'_\ell} w'_\ell + q''_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d\rangle \in \gamma_{\vec{w}'}(F')$ then $\langle v_1, \ldots, v_\ell + 2^{\omega-k'_\ell}, \ldots, v_d\rangle \in \gamma_{\vec{w}'}(F')$

where $q''_\ell \in \mathbb{Z}_m[w'_{\ell+1}, \ldots, w'_d]$. Now, let $1 \leq \ell \leq d$. For the first property:

- Suppose $\ell \neq i$. Then, $w'_\ell = w_\ell \neq w$ so $W' = 2^{k_\ell} w_\ell + q''_\ell$. It follows $W''_\ell = (2^{k_\ell} w_\ell + q'_\ell)[w \mapsto w_i] = 2^{k'_\ell} w_\ell + q'_\ell$ where $q'_\ell = q''_\ell[w \mapsto w_i]$. Since $q''_\ell \in \mathbb{Z}_m[w'_{\ell+1}, \ldots, w'_d]$ it follows $q'_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$, as required.

- Suppose $\ell = i$. Then, $w'_i = w$, hence $W'_i = 2^{k'_i}w + q''_i$. It follows $W''_i = (2^{k'_i}w + q''_i)[w \mapsto w_i] = 2^{k'_i}w_i + q''_i$. Since $q''_i \in \mathbb{Z}_m[w'_{i+1}, \ldots, w'_d] = \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, the result follows with $q'_i = q''_i$.

For the second property:

- Suppose $k'_i = \omega$. Then, $W'_i = 2^\omega w + q''_\ell$ where $q''_\ell \in \mathbb{Z}_m[w'_{i+1}, \ldots, w'_d] = \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, hence $F'' = F'[w \mapsto 0]$. Let $\vec{v} \in \gamma_{\vec{w}}(F'')$. Now,

  - Suppose $\ell \neq i$ and let $\vec{v} \in \gamma_{\vec{w}}(F'')$. Then, if $\vec{v'} \in \mathbb{Z}_m^d$ is defined such that $\pi_i(\vec{v'}) = \pi_i(\vec{v})$ and $v'_i = 0$ then $\vec{v'} \in \gamma_{\vec{w'}}(F')$. Thus, by the property above $\langle v'_1, \ldots, v'_\ell + 2^{\omega - k'_\ell}, \ldots, v'_d \rangle \in \gamma_{\vec{w'}}(F')$ and since $v'_i = 0$ it follows $\langle v'_1, \ldots, v'_\ell + 2^{\omega - k'_\ell}, \ldots, v'_d \rangle \in \gamma_{\vec{w}}(F'')$. But now, since $w_i \notin \mathsf{vars}(F'')$ it follows if $\vec{v''} \in \mathbb{Z}_m^d$ is defined such that $\pi_i(\vec{v''}) = \pi_i(\vec{v'})$ and $v''_i = v_i$ then $\vec{v''} \in \gamma_{\vec{w}}(F'')$. But, then $\vec{v''} = \langle v_1, \ldots, v_\ell + 2^{\omega - k'_\ell}, \ldots, v_d \rangle$ and the result follows.
  - Suppose $\ell = i$ and let $\vec{v} \in \gamma_{\vec{w}}(F'')$. Then, since $w_i \notin \mathsf{vars}(F'')$, $\langle v_1, \ldots, v_i + 2^{\omega - k'_i}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F'')$, as required.

- Suppose $k'_i < \omega$. Then, $W'_i = 2^{k'_i}w + q''_\ell \neq 2^\omega w + q$ with $q \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, hence $F'' = F'[w \mapsto w_i]$. Thus, if $\vec{v} \in \gamma_{\vec{w}}(F'')$ then $\vec{v} \in \gamma_{\vec{w'}}(F')$, hence by the inductive property $\langle v_1, \ldots, v_\ell + 2^{\omega - k'_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w'}}(F')$ and so $\langle v_1, \ldots, v_\ell + 2^{\omega - k'_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F'')$.

For the third property, note since $F \cup \{w_i - 2^j w + r\} \rightarrow_{\mathsf{elim}[w_i]} F'$ it follows $\mathsf{vars}(F') \subseteq (\mathsf{vars}(F) \cup \{w\} \cup \mathsf{vars}(r)) \setminus \{w_i\}$. But since $\mathsf{vars}(r) \subseteq \mathsf{vars}(F)$ it follows $\mathsf{vars}(F') \subseteq (\mathsf{vars}(F) \cup \{w\}) \setminus \{w_i\}$. Moreover, since $F'' = F'[w \mapsto 0]$ or $F'' = F'[w \mapsto w_i]$ it follows $\mathsf{vars}(F'') \subseteq (\mathsf{vars}(F') \setminus \{w\}) \cup \{w_i\} \subseteq \mathsf{vars}(F) \cup \{w_i\} = \mathsf{vars}(F)$, since $\{w_i\} \subseteq \mathsf{vars}(F)$. Now, suppose $k'_\ell = \omega$. Then,

- Suppose $\ell \neq i$. Then, $k_\ell = \omega$, hence by the inductive hypothesis, $w_\ell \notin \mathsf{vars}(F)$. But since $\mathsf{vars}(F'') \subseteq \mathsf{vars}(F)$ it follows $w_\ell \notin \mathsf{vars}(F'')$, as required.

- Suppose $\ell = i$. Then, $k'_i = \omega$ so, as shown above, $F'' = F'[w \mapsto 0]$. It follows $\mathsf{vars}(F'') \subseteq \mathsf{vars}(F') \setminus \{w\} \subseteq \mathsf{vars}(F) \setminus \{w_i\}$, so $w_i \notin \mathsf{vars}(F'')$, as required.

□

Corollary 8. Let $S = \langle \vec{W}, F \rangle \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ and suppose

- $W_\ell = 2^{k_\ell} w_\ell + q_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F)$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F)$
- If $k_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F)$

for some $0 \leq k_\ell \leq \omega$ and $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$. Suppose $\mathbf{simplify}(S) = \langle \vec{W}', F' \rangle$. Then,

- $W'_\ell = 2^{k'_\ell} w_\ell + q'_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F')$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k'_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F')$
- If $k'_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F')$

for some $k_\ell \leq k'_\ell \leq \omega$ and $q'_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$.

proof for Corollary 8. Let $\vec{W}'' = \vec{W}$ and $F'' = \mathbf{gb}_{\vec{w}}(F)$. It will be shown the hypotheses also hold for the pair $\langle \vec{W}'', F'' \rangle$. To that end:

- $W''_\ell = W_\ell = 2^{k_\ell} w_\ell + q_\ell$ where $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$,
- Note $\gamma_{\vec{w}}(F'') = \gamma_{\vec{w}}(\langle F'' \rangle_{\vec{w}}) = \gamma_{\vec{w}}(\langle F \rangle_{\vec{w}}) = \gamma_{\vec{w}}(F)$. Thus, if $\vec{v} \in \gamma_{\vec{w}}(F'')$ then $\vec{v} \in \gamma_{\vec{w}}(F)$, hence $\langle v_1, \ldots, v_\ell + 2^{\omega - k_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F) = \gamma_{\vec{w}}(F'')$.
- If $k_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F)$. But since $\mathsf{vars}(F'') = \mathsf{vars}(F)$ it follows $w_\ell \notin \mathsf{vars}(F'')$,

The result follows. Now, proceed by induction on the number n of calls to **simplify**:

- Suppose $n = 0$. Then, $\langle \vec{W}', F' \rangle = \langle \vec{W}'', F'' \rangle$ and by the argument above the result holds with $k'_\ell = k_\ell$.

- Suppose $n > 0$ and let $S' = \langle \vec{W}'', F'' \rangle$. Then, there exists $2^{\omega-j}(w_i + r) \in F''$ where $j > 0$, $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$ and $\mathbf{safe}(\vec{W}'', w_i, r)$ holds and, moreover, $\langle \vec{W}', F' \rangle = \mathbf{simplify}(S'')$ where $S'' = \mathbf{constrain}(S', j, w_i, r)$. Now, since $2^{\omega-j}(w_i + r) \in F''$ and $j > 0$ it follows $\{w_i\} \cup \mathsf{vars}(r) \subseteq \mathsf{vars}(F'') = \mathsf{vars}(F)$. The argument above shows the hypotheses hold for S', hence Lemma 11 implies the result holds for S'' and some k''_ℓ satisfying $k_\ell \leq k''_\ell \leq \omega$. But now, the call $\mathbf{simplify}(S'')$ requires $n - 1$ calls and returns $\langle \vec{W}', F' \rangle$ hence, by induction, the hypothesis holds for $\langle \vec{W}', F' \rangle$ for some k'_ℓ satisfying $k''_\ell \leq k'_\ell \leq \omega$. Since $k_\ell \leq k''_\ell$, the result follows.

□

Proof for corollary 6. The result follows by induction on the depth n at which a call arises:

- Suppose $n = 0$. Only $\mathbf{cover}(S)$ is called at depth 0. Then $\vec{W} = \langle w_1, \ldots, w_d \rangle$ hence $W_\ell = 2^k_\ell w_\ell + q_\ell$ where $k_\ell = 0$ and $q_\ell = 0 \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$. Moreover, since $2^{\omega-k_\ell} = 2^{\omega-0} = 0$, it follows $\langle v_1, \ldots, v_\ell + 2^{\omega-k_\ell}, \ldots, v_d \rangle = \langle v_1, \ldots, v_\ell, \ldots, v_d \rangle$, hence the second property is satisfied vacuously. Finally, since $k_\ell = 0 < \omega$ for all ℓ, the third property also holds vacuously.

- Suppose **constrain** is called at depth $n > 0$.

  - Suppose $\mathbf{constrain}(S', j, w_i, r)$ is called from $\mathbf{cover}(S)$. Then, the call $\mathbf{cover}(S)$ occurs at depth $n - 1$, hence by the inductive hypothesis the conclusions hold for S. Moreover, since $\mathbf{cover}(S)$ makes a recursive call, it follows $S' = \langle \vec{W}', F' \rangle$. Thus, by Corollary 8 it follows the conclusions also hold for S'.
  - Suppose $\mathbf{constrain}(S', j, w_i, r)$ is called from $\mathbf{simplify}(S)$. Then, the call $\mathbf{simplify}(S)$ occurs at depth $n - 1$, hence by the inductive hypothesis the conclusions hold for S. Thus, the argument in the proof for Corollary 8 shows the conclusions hold for S'.

- Suppose **simplify** is called at depth $n > 0$.

  - Suppose **simplify**(S) is called from **cover**(S). Since **cover**(S) is called at depth $n-1$, the conclusions hold for S, hence the call **simplify**(S).
  - Suppose **simplify**(S'') is called from **simplify**(S). Then, the call **simplify**(S) occurs at depth $n-1$, hence by the inductive hypothesis the conclusions hold for S. Thus, the argument in the proof for Corollary 8 shows the conclusions also hold for S''.

- Suppose **cover**(S'_ℓ) is called from **cover**(S) at depth $n > 0$. The call **cover**(S) occurs at depth $n-1$, hence the conclusions hold for S. Let $S' =$ **simplify**(S). Now, since $S'_\ell =$ **constrain**$(S', 1, w_i, r)$ where $w_i \in \mathsf{vars}(F')$ and $r = 0$ or $r = 1$, it follows $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$ and $\{w_i\} \cup \mathsf{vars}(r) \subseteq \mathsf{vars}(F')$. Moreover, since $\mathsf{vars}(r) = \emptyset$ it follows that **safe**$(\vec{W}', w_i, r)$ holds. Thus, from Proposition 11, the conclusions also hold for S'_ℓ.

□

proof for Theorem 3. Let $\mathsf{level}_i(2^k w_i + q) = k$ and define $\mathsf{level} : \mathbb{Z}_m[\vec{w}]^d \to \{0, \ldots, \omega\}^d$ by $\mathsf{level}(W_1, \ldots, W_d) = \langle \mathsf{level}_1(W_1), \ldots, \mathsf{level}_d(W_d) \rangle$. Consider a call **constrain**(S, j, w_i, r) occuring during the execution of **cover**. Note first in any such call $j > 0$, $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, $\{w_i\} \cup \mathsf{vars}(r) \subseteq \mathsf{vars}(F)$ and **safe**$(\vec{W}, w_i, r)$ holds. Now, suppose $S = \langle \vec{W}, F \rangle$ and the call returns $\langle \vec{W}', F' \rangle$. Then, by Corollary 6, there exist $0 \leq k_\ell \leq \omega$ and $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ such that $W_\ell = 2^{k_\ell} w_\ell + q_\ell$ and if $k_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F)$. Thus, by Proposition 11, there exist $q'_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ such that

$$W'_\ell = 2^{k'_\ell} w_\ell + q'_\ell \qquad \text{where} \qquad k'_\ell = \begin{cases} \min(k_i + j, \omega) & \text{if } \ell = i \\ k_\ell & \text{otherwise} \end{cases}$$

and if $k'_\ell = \omega$ then $x_\ell \notin \mathsf{vars}(F')$, for each $1 \leq \ell \leq d$. Now, since $w_i \in \mathsf{vars}(F)$ it follows $k_i < \omega$. Thus, $k'_i = \min(k_i + j, \omega) > k_i$. Since $k'_\ell = k_\ell$ for each $\ell \neq i$ it thus follows $\mathsf{level}(\vec{W}') = \langle k'_1, \ldots, k'_d \rangle > \langle k_1, \ldots, k_d \rangle = \mathsf{level}(\vec{W})$, where the comparison is pointwise.

Now, consider a call **simplify**(S) occuring during the execution of **cover**, where $S = \langle \vec{W}, F \rangle$, and suppose **simplify**(S) makes the recursive call **simplify**(S''), where $S'' = \langle \vec{W}', F' \rangle$. Then, $\langle \vec{W}', F' \rangle =$ **constrain**(S', j, x_i, r). By the previous

argument it follows that $\mathsf{level}(\vec{W}') > \mathsf{level}(\vec{W})$. In particular, it follows any sequence of recursive calls to **simplify** determines a strictly increasing sequence of levels, bounded above by $\langle \omega, \ldots, \omega \rangle$ pointwise. Termination of **simplify** follows immediately.

Finally, consider a call $\mathbf{cover}(S)$ occuring during the execution of **cover**, where $S = \langle \vec{W}, F \rangle$, and suppose $\mathbf{cover}(S)$ makes the recursive call $\mathbf{cover}(S'_\ell)$, where $S'_\ell = \langle \vec{W}'', F'' \rangle$. Then, $\langle \vec{W}'', F'' \rangle = \mathbf{constrain}(S', j, x_i, r)$ where $S' = \mathbf{simplify}(S) = \langle \vec{W}', F' \rangle$. Then, by Corollary 8, it follows $\mathsf{level}(\vec{W}') \geq \mathsf{level}(\vec{W})$. Moreover, by the previous argument, $\mathsf{level}(\vec{W}'') > \mathsf{level}(\vec{W}')$, hence $\mathsf{level}(\vec{W}'') > \mathsf{level}(\vec{W})$. In particular, it follows any sequence of recursive calls to **cover** determines a strictly increasing sequence of levels, bounded above by $\langle \omega, \ldots, \omega \rangle$ pointwise. Termination of **cover** thus follows.

□

proof for Theorem 3. Let $\mathsf{level}_i(2^k w_i + q) = k$ and define $\mathsf{level} : \mathbb{Z}_m[\vec{w}]^d \to \{0, \ldots, \omega\}^d$ by $\mathsf{level}(W_1, \ldots, W_d) = \langle \mathsf{level}_1(W_1), \ldots, \mathsf{level}_d(W_d) \rangle$. Consider a call $\mathbf{constrain}(S, j, w_i, r)$ occuring during the execution of **cover**. Note first in any such call $j > 0$, $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, $\{w_i\} \cup \mathsf{vars}(r) \subseteq \mathsf{vars}(F)$ and $\mathbf{safe}(\vec{W}, w_i, r)$ holds. Now, suppose $S = \langle \vec{W}, F \rangle$ and the call returns $\langle \vec{W}', F' \rangle$. Then, by Corollary 6, there exist $0 \leq k_\ell \leq \omega$ and $q_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ such that $W_\ell = 2^{k_\ell} w_\ell + q_\ell$ and if $k_\ell = \omega$ then $w_\ell \notin \mathsf{vars}(F)$. Thus, by Proposition 11, there exist $q'_\ell \in \mathbb{Z}_m[w_{\ell+1}, \ldots, w_d]$ such that

$$W'_\ell = 2^{k'_\ell} w_\ell + q'_\ell \qquad \text{where} \qquad k'_\ell = \begin{cases} \min(k_i + j, \omega) & \text{if } \ell = i \\ k_\ell & \text{otherwise} \end{cases}$$

and if $k'_\ell = \omega$ then $x_\ell \notin \mathsf{vars}(F')$, for each $1 \leq \ell \leq d$. Now, since $w_i \in \mathsf{vars}(F)$ it follows $k_i < \omega$. Thus, $k'_i = \min(k_i + j, \omega) > k_i$. Since $k'_\ell = k_\ell$ for each $\ell \neq i$ it thus follows $\mathsf{level}(\vec{W}') = \langle k'_1, \ldots, k'_d \rangle > \langle k_1, \ldots, k_d \rangle = \mathsf{level}(\vec{W})$, where the comparison is pointwise.

Now, consider a call $\mathbf{simplify}(S)$ occuring during the execution of **cover**, where $S = \langle \vec{W}, F \rangle$, and suppose $\mathbf{simplify}(S)$ makes the recursive call $\mathbf{simplify}(S'')$, where $S'' = \langle \vec{W}', F' \rangle$. Then, $\langle \vec{W}', F' \rangle = \mathbf{constrain}(S', j, x_i, r)$. By the previous argument it follows that $\mathsf{level}(\vec{W}') > \mathsf{level}(\vec{W})$. In particular, any sequence

of recursive calls to **simplify** determines a strictly increasing sequence of levels, bounded above by $\langle \omega, \ldots, \omega \rangle$. Termination of **simplify** follows immediately.

Finally, consider a call **cover**(S) occuring during the execution of **cover**, where $S = \langle \vec{W}, F \rangle$, and suppose **cover**$(S)$ makes the recursive call **cover**(S'_ℓ), where $S'_\ell = \langle \vec{W}'', F'' \rangle$. Then, $\langle \vec{W}'', F'' \rangle = \textbf{constrain}(S', j, x_i, r)$ where $S' = \textbf{simplify}(S) = \langle \vec{W}', F' \rangle$. Then, by Corollary 8, it follows $\mathsf{level}(\vec{W}') \geq \mathsf{level}(\vec{W})$. Moreover, by the previous argument, $\mathsf{level}(\vec{W}'') > \mathsf{level}(\vec{W}')$, therefore $\mathsf{level}(\vec{W}'') > \mathsf{level}(\vec{W})$. In particular, any sequence of recursive calls to **cover** determines a strictly increasing sequence of levels, bounded above by $\langle \omega, \ldots, \omega \rangle$. Termination of **cover** thus follows.

□

Proof for proposition 12. First note since **constrain**(S, j, w_i, r) is a call made during the execution of **cover**, it follows $j > 0$, $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$, $\{w_i\} \cup \mathsf{vars}(r) \subseteq \mathsf{vars}(F)$ and **safe**$(\vec{W}, w_i, r)$ holds. Now, since $\langle \vec{W}'', F'' \rangle = \textbf{constrain}(\langle \vec{W}, F \rangle, j, w_i, r)$ it follows

- $\vec{W}'' = \vec{W}'[w_i \mapsto w]$ where $\vec{W}' = \vec{W}[w_i \mapsto 2^j w - r]$,
- There exists $F' \in \mathbb{Z}_m[\vec{w}']$ such that $F \cup \{w_i - 2^j w + r\} \rightarrow_{\mathsf{elim}[w_i]} F'$ and

$$F'' = \begin{cases} F'[w \mapsto 0] & \text{if } W'_i = 2^\omega w + q \wedge q \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d] \\ F'[w \mapsto w_i] & \text{otherwise} \end{cases}$$

where $w \notin \mathsf{vars}(\vec{w})$ and $\vec{w}' = \vec{w}[i \mapsto w]$. Then, from Lemma 11, for all $1 \leq \ell \leq d$,

- $W'_\ell = 2^{k'_\ell} w'_\ell + q''_\ell$
- If $\langle v_1, \ldots, v_\ell, \ldots, v_d \rangle \in \gamma_{\vec{w}'}(F')$ then $\langle v_1, \ldots, v_\ell + 2^{\omega - k'_\ell}, \ldots, v_d \rangle \in \gamma_{\vec{w}'}(F')$

where $q''_\ell \in \mathbb{Z}_m[w'_{\ell+1}, \ldots, w'_d]$. It will first be shown

$$\gamma_{\vec{w}}(\langle \vec{W}'', F'' \rangle) = \{ [\![\vec{W}']\!]_{\vec{w}'}(\vec{a}'') \mid \vec{a}'' \in \gamma_{\vec{w}'}(F') \} \tag{4}$$

Note, since $\vec{W}'' = \vec{W}'[w \mapsto w_i]$ it follows for any $1 \leq \ell \leq d$ that $[\![W''_\ell]\!]_{\vec{w}}(\vec{a}'') = [\![W'_\ell]\!]_{\vec{w}'}(\vec{a}'')$ for all $\vec{a}'' \in \mathbb{Z}_m^d$. Thus, the result holds if $\gamma_{\vec{w}}(F'') = \gamma_{\vec{w}'}(F')$. To that end:

- Suppose $k'_i < \omega$ so $F'' = F'[w \mapsto w_i]$. Then, $f \in F'$ iff $f[w \mapsto w_i] \in F''$. Moreover, for any $\vec{a}'' \in \mathbb{Z}_m^d$, $[\![f]\!]_{\vec{w}'}(\vec{a}'') = [\![f[w \mapsto w_i]]\!]_{\vec{w}}(\vec{a}'')$. In particular, $[\![f]\!]_{\vec{w}'}(\vec{a}'') = 0$ iff $[\![f[w \mapsto w_i]]\!]_{\vec{w}}(\vec{a}'') = 0$. The result follows.

- Suppose $k'_i = \omega$ so $F'' = F'[w \mapsto 0]$. Then, since $2^{\omega - k'_i} = 2^0 = 1$, it follows from above if $\vec{v} \in \gamma_{\vec{w}'}(F')$ then $\langle v_1, \ldots, v_i + 1, \ldots, v_d \rangle \in \gamma_{\vec{w}'}(F')$. Repeatedly applying this result yields $\vec{v} \in \gamma_{\vec{w}'}(F')$ iff $\langle v_1, \ldots, v_{i-1}, v, v_{i+1}, \ldots, v_d \rangle \in \gamma_{\vec{w}'}(F')$ for all $v \in \mathbb{Z}_m$. Moreover, since $w_i \notin \mathsf{vars}(F'')$ it also follows $\vec{v} \in \gamma_{\vec{w}}(F'')$ iff $\langle v_1, \ldots, v_{i-1}, v, v_{i+1}, \ldots, v_d \rangle \in \gamma_{\vec{w}}(F'')$ for all $v \in \mathbb{Z}_m$. Finally, $f \in F'$ iff $f[w \mapsto 0] \in F''$ and for any $\vec{a}'' \in \mathbb{Z}_m^d$ with $a''_i = 0$, $[\![f]\!]_{\vec{w}'}(\vec{a}'') = [\![f[w \mapsto w_i]]\!]_{\vec{w}}(\vec{a}'')$. It follows:

$$
\begin{aligned}
\vec{a}'' \in \gamma_{\vec{w}'}(F') \text{ iff } & \langle a''_1, \ldots, a''_{i-1}, 0, a''_{i+1}, \ldots, a''_d \rangle \in \gamma_{\vec{w}'}(F') \\
\text{iff } & [\![f]\!]_{\vec{w}'}(\langle a''_1, \ldots, a''_{i-1}, 0, a''_{i+1}, \ldots, a''_d \rangle) = 0 \text{ for all } f \in F' \\
\text{iff } & [\![f[w \mapsto 0]]\!]_{\vec{w}}(\langle a''_1, \ldots, a''_{i-1}, 0, a''_{i+1}, \ldots, a''_d \rangle) = 0 \text{ for all } f \in F' \\
\text{iff } & [\![f]\!]_{\vec{w}}(\langle a''_1, \ldots, a''_{i-1}, 0, a''_{i+1}, \ldots, a''_d \rangle) = 0 \text{ for all } f \in F'' \\
\text{iff } & \langle a''_1, \ldots, a''_{i-1}, 0, a''_{i+1}, \ldots, a''_d \rangle \in \gamma_{\vec{w}}(F'') \\
\text{iff } & \vec{a}'' \in \gamma_{\vec{w}}(F'')
\end{aligned}
$$

Thus, $\gamma_{\vec{w}}(F'') = \gamma_{\vec{w}'}(F')$ and so equation (4) holds. To complete the argument it will be shown

$$\gamma_{\vec{w}}(\langle \vec{W}, F \cup \{2^{\omega - j}(w_i + r)\} \rangle) = \{[\![\vec{W}']\!]_{\vec{w}'}(\vec{a}'') \mid \vec{a}'' \in \gamma_{\vec{w}'}(F')\} \tag{5}$$

To that end:

- Let $\vec{a} \in \gamma_{\vec{w}}(F \cup \{2^{\omega - j}(w_i + r)\})$. Then, $\vec{a} \in \gamma_{\vec{w}}(F)$ and $[\![2^{\omega - j}(w_i + r)]\!]_{\vec{w}}(\vec{a}) = 2^{\omega - j}(a_i + [\![r]\!]_{\vec{w}}(\vec{a})) = 0$. But this implies there exists $b \in \mathbb{Z}_m$ such that $a_i - 2^j b + [\![r]\!]_{\vec{w}}(\vec{a}) = 0$. Thus, $[\![w_i - 2^j w + r]\!]_{w:\vec{w}}(b : \vec{a}) = 0$ and since $\vec{a} \in \gamma_{\vec{w}}(F)$ then $b : \vec{a} \in \gamma_{w:\vec{w}}(F)$, and so $b : \vec{a} \in \gamma_{w:\vec{w}}(F \cup \{w_i - 2^j + r)\})$. But now, by Lemma 10 it follows $b : \pi_i(\vec{a}) \in \gamma_{w:\pi_i(\vec{w})}(F')$, hence $\vec{a}[i \mapsto b] \in \gamma_{\vec{w}'}(F')$ by reordering variables. Since $\vec{W}' = \vec{W}[w_i \mapsto 2^j w - r]$ it follows $[\![W'_\ell]\!]_{\vec{w}'}(\vec{a}[i \mapsto b]) = [\![W_\ell]\!]_{\vec{w}}(\vec{a}[i \mapsto 2^j b - [\![r]\!]_{\vec{w}}(\vec{a})]) = [\![W_\ell]\!]_{\vec{w}}(\vec{a})$. Hence $[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) = [\![\vec{W}']\!]_{\vec{w}'}(\vec{a}[i \mapsto b]) \in \{[\![\vec{W}']\!]_{\vec{w}'}(\vec{a}'') \mid \vec{a}'' \in \gamma_{\vec{w}'}(F')\}$ as required.

- Let $\vec{a}'' \in \gamma_{\vec{w}'}(F')$. Then, by reordering variables, $a_i'' : \pi_i(\vec{a}'') \in \gamma_{w:\pi_i(\vec{w})}(F')$. Thus, but Lemma 10, there exists $\vec{a}$ such that $a_i'' : \vec{a} \in \gamma_{w:\vec{w}}(F \cup \{w_i - 2^j w + r\})$ and $\pi_i(\vec{a}) = \pi_i(\vec{a}'')$. It follows $[\![w_i - 2^j w + r]\!]_{w:\vec{w}}(a_i'' : \vec{a}) = a_i - 2^j a_i'' + [\![r]\!]_{\vec{w}}(\vec{a}) = 0$, hence $[\![2^{\omega-j}(w_i + r)]\!]_{\vec{w}}(\vec{a}) = 2^{\omega-j}(a_i + [\![r]\!]_{\vec{w}}(\vec{a})) = 2^{\omega-j}(a_i - 2^j a_i'' + [\![r]\!]_{\vec{w}}(\vec{a})) = 0$. Since also $\vec{a} \in \gamma_{\vec{w}}(F)$ it follows $\vec{a} \in \gamma_{\vec{w}}(F \cup \{2^{\omega-j}(w_i + r)\})$. Now, since $\vec{W}' = \vec{W}[w_i \mapsto 2^j - r]$ it follows $[\![W_\ell']\!]_{\vec{w}'}(\vec{a}'') = [\![W_\ell]\!]_{\vec{w}}(\vec{a''}[i \mapsto 2^j a_i'' - [\![r]\!]_{\vec{w}}(\vec{a})]) = [\![W_\ell]\!]_{\vec{w}}(\vec{a})$. Hence $[\![\vec{W}']\!]_{\vec{w}}(\vec{a}'') = [\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \in \gamma_{\vec{w}}(\langle \vec{W}, F \cup \{2^{\omega-j}(w_i + r)\}\rangle)$ as required.

□

Proof of Corollary 7. Let $F' = \mathbf{gb}_{\prec_{\vec{w}}}(F)$ and $S'' = \langle \vec{W}, F' \rangle$. Note $\gamma_{\vec{w}}(F) = \gamma_{\vec{w}}(\langle F \rangle_{\vec{w}}) = \gamma_{\vec{w}}(\langle F' \rangle_{\vec{w}}) = \gamma_{\vec{w}}(F')$, hence $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S'')$. Now, proceed by induction on the number n of recursive calls to **simplify**:

- Suppose $n = 0$. Then,
  - Suppose $S' \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$. Since $n = 0$, it follows $S'' = S'$, hence $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S'') = \gamma_{\vec{w}}(S')$.
  - Suppose $S' = \mathbf{nil}$. Then, there exists $c \in F'$ such that $c \in \mathbb{Z}_m \setminus \{0\}$. But then, $\gamma_{\vec{w}}(F) = \gamma_{\vec{w}}(F') \subseteq \gamma_{\vec{w}}(c) = \emptyset$. It follows $\gamma_{\vec{w}}(S) = \emptyset$, as required.
- Suppose $n > 0$. Then, there exists $2^{\omega-j}(w_i + r) \in F'$ where $j > 0$, $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$ and $\mathbf{safe}(\vec{W}, w_i, r)$ holds. Moreover, $S' = \mathbf{simplify}(S''')$ where $S''' = \mathbf{constrain}(S'', j, w_i, r)$. Now, by Proposition 12 it holds that $\gamma_{\vec{w}}(\langle \vec{W}, F' \cup \{2^{\omega-j}(w_i + r)\}\rangle) = \gamma_{\vec{w}}(S''')$. But since $2^{\omega-j}(w_i + r) \in F'$ it holds that $\gamma_{\vec{w}}(F' \cup \{2^{\omega-j}(w_i + r)\}) = \gamma_{\vec{w}}(F')$, hence $\gamma_{\vec{w}}(S'') = \gamma_{\vec{w}}(S''')$. Thus it follows $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S''')$. Now, since $\mathbf{simplify}(S''')$ makes $n - 1$ recursive calls, it follows by the inductive hypothesis:
  - If $S' \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ then $\gamma_{\vec{w}}(S''') = \gamma_{\vec{w}}(S')$, hence $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S')$, as required.
  - If $S' = \mathbf{nil}$ then $\gamma_{\vec{w}}(S''') = \emptyset$, hence $\gamma_{\vec{w}}(S) = \emptyset$, as required.

□

Proof of Corollary 7. Let $F' = \mathbf{gb}_{\prec_{\vec{w}}}(F)$ and $S'' = \langle \vec{W}, F' \rangle$. Note $\gamma_{\vec{w}}(F) = \gamma_{\vec{w}}(\langle F \rangle_{\vec{w}}) = \gamma_{\vec{w}}(\langle F' \rangle_{\vec{w}}) = \gamma_{\vec{w}}(F')$, hence $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S'')$. Now, proceed by induction on the number n of recursive calls to **simplify**:

- Suppose $n = 0$. Then,
  - Suppose $S' \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$. Since $n = 0$, it follows $S'' = S'$, hence $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S'') = \gamma_{\vec{w}}(S')$.
  - Suppose $S' = \mathbf{nil}$. Then, there exists $c \in F'$ such that $c \in \mathbb{Z}_m \setminus \{0\}$. But then, $\gamma_{\vec{w}}(F) = \gamma_{\vec{w}}(F') \subseteq \gamma_{\vec{w}}(c) = \emptyset$. It follows $\gamma_{\vec{w}}(S) = \emptyset$, as required.
- Suppose $n > 0$. Then, there exists $2^{\omega-j}(w_i + r) \in F'$ where $j > 0$, $r \in \mathbb{Z}_m[w_{i+1}, \ldots, w_d]$ and $\mathbf{safe}(\vec{W}, w_i, r)$ holds. Moreover, $S' = \mathbf{simplify}(S''')$ where $S''' = \mathbf{constrain}(S'', j, w_i, r)$. Now, by Proposition 12 it holds that $\gamma_{\vec{w}}(\langle \vec{W}, F' \cup \{2^{\omega-j}(w_i + r)\} \rangle) = \gamma_{\vec{w}}(S''')$. But since $2^{\omega-j}(w_i + r) \in F'$ it holds that $\gamma_{\vec{w}}(F' \cup \{2^{\omega-j}(w_i + r)\}) = \gamma_{\vec{w}}(F')$, hence $\gamma_{\vec{w}}(S'') = \gamma_{\vec{w}}(S''')$. Thus it follows $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S''')$. Now, since $\mathbf{simplify}(S''')$ makes $n - 1$ recursive calls, it follows by the inductive hypothesis:
  - If $S' \in \mathbb{Z}_m[\vec{w}]^d \times \wp(\mathbb{Z}_m[\vec{w}])$ then $\gamma_{\vec{w}}(S''') = \gamma_{\vec{w}}(S')$, hence $\gamma_{\vec{w}}(S) = \gamma_{\vec{w}}(S')$, as required.
  - If $S' = \mathbf{nil}$ then $\gamma_{\vec{w}}(S''') = \emptyset$, hence $\gamma_{\vec{w}}(S) = \emptyset$, as required.

□

Proof of Theorem 4. It will first be shown if $S = \langle \vec{W}, F \rangle$ and $\mathbf{cover}(S) = \mathcal{W}$ then $\mathcal{W}$ is a cover of $\gamma_{\vec{w}}(S)$. To that end, let $S' = \mathbf{simplify}(S)$. Then:

- Suppose first $S' = \mathbf{nil}$. Then, by Corollary 7, $\gamma_{\vec{w}}(S) = \emptyset$. But, in this case $\mathcal{W} = \emptyset$, which is a cover of $\emptyset$ over $\vec{w}$.
- Otherwise, $S' = \langle \vec{W}', F' \rangle$. Now, proceed by induction on the maximum depth n of a recursive call to **cover**:

– Suppose $n = 0$, so $\mathbf{cover}(S)$ makes no recursive calls. Then, $F = \emptyset$ and $\mathcal{W} = \{\vec{W}\}$. But then, $\gamma_{\vec{w}}(S) = \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \gamma_{\vec{w}}(F)\} = \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\} = \{[\![\vec{W}']\!]_{\vec{w}}(\vec{a}) \mid \vec{W}' \in \mathcal{W} \wedge \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\}$, and so $\mathcal{W}$ is a cover of $\gamma_{\vec{w}}(S)$ over $\vec{w}$.

– Suppose $n > 0$. Then, letting $S'_\ell = \mathbf{constrain}(S', 1, w_i, \ell)$ and $\mathcal{W}_\ell = \mathbf{cover}(S'_\ell)$ for each $\ell \in \{0, 1\}$ it follows $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2$. Now, by Proposition 12, $\gamma_{\vec{w}}(S'_\ell) = \gamma_{\vec{w}}(\langle \vec{W}', F' \cup \{2^{\omega-1}(w_i + \ell)\}\rangle) = \{[\![\vec{W}']\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \gamma_{\vec{w}}(F' \cup \{2^{\omega-1}(w_i + \ell)\})$. Now, since $\gamma_{\vec{w}}(F') = \gamma_{\vec{w}}(F' \cup \{2^{\omega-1}(w_i + 0)\}) \cup \gamma_{\vec{w}}(F' \cup \{2^{\omega-1}(w_i + 1)\})$ it thus follows $\gamma_{\vec{w}}(S') = \gamma_{\vec{w}}(S'_0) \cup \gamma_{\vec{w}}(S'_1)$. But now, since the maximum depth of a recursive call in $\mathbf{cover}(S'_\ell)$ is $n - 1$, it follows from the inductive hypothesis $\mathcal{W}_\ell$ is a cover of $\gamma_{\vec{w}}(S'_\ell)$. Since $\gamma_{\vec{w}}(S') = \gamma_{\vec{w}}(S'_0) \cup \gamma_{\vec{w}}(S'_1)$ and $\mathcal{W} = \mathcal{W}_0 \cup \mathcal{W}_1$ it follows immediately that $\mathcal{W}$ is a cover of $\gamma_{\vec{w}}(S')$.

It thus follows if $\mathbf{cover}(S) = \mathcal{W}$ then $\mathcal{W}$ is a cover of $\gamma_{\vec{w}}(S)$ over $\vec{w}$. To conclude the proof note that $\mathbf{cover}(F_0) = \mathbf{cover}(\langle \vec{W}, F\rangle)$ where $\vec{W} = \langle w_1, \ldots, w_d\rangle$ and $F = F_0[x_1 \mapsto w_1, \ldots, x_d \mapsto w_d]$. Thus, by the result above, $\mathcal{W}$ is a cover of $\gamma_{\vec{w}}(\langle \vec{W}, F\rangle)$. But $\gamma_{\vec{x}}(F_0) = \gamma_{\vec{w}}(F) = \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \gamma_{\vec{w}}(F)\} = \gamma_{\vec{w}}(\langle \vec{W}, F\rangle)$. Thus, $\mathcal{W}$ is a cover of $\gamma_{\vec{x}}(F_0)$ and hence of F_0, as required.

□

Proof of Lemma 7. Let $p \in \mathbb{Z}_m[\vec{x}]$ and suppose $p = q_1(x_1 - W_1) + \cdots + q_d(x_d - W_d) + r$ where each $q_1, \ldots, q_d, r \in \mathbb{Z}_m[\vec{x} : \vec{w}]$. Now, suppose $\vec{b} = [\![\vec{W}]\!]_{\vec{w}}(\vec{a})$ where $\vec{a} \in \mathbb{Z}_m^{|\vec{w}|}$. Then,

$$\begin{aligned}
[\![p]\!]_{\vec{x}}(\vec{b}) &= [\![p]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a}) \\
&= \left(\textstyle\sum_{\ell=1}^{d} [\![q_\ell]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a})[\![x_\ell - W_\ell]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a})\right) + [\![r]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a}) \\
&= \left(\textstyle\sum_{\ell=1}^{d} [\![q_\ell]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a})(b_\ell - [\![W_\ell]\!]_{\vec{w}}(\vec{a}))\right) + [\![r]\!]_{\vec{a}:\vec{w}}(\vec{b} : \vec{a}) \\
&= [\![r]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a})
\end{aligned}$$

where the final step follows since $b_\ell - [\![W_\ell]\!]_{\vec{w}}(\vec{a}) = 0$ for all $1 \leq \ell \leq d$. In particular, it follows $[\![p]\!]_{\vec{x}}(\vec{b}) = 0$ iff $[\![r]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a}) = 0$. Now:

- Suppose $p \in \langle B\rangle_{\vec{x}}$. By assumption, $\mathsf{elim}[\vec{w}](\langle\{x_1 - W_1, \ldots, x_d - W_d\} \cup B_{\mathsf{Null}_m[\vec{w}]}\rangle_{\vec{x}:\vec{w}}) = \langle B\rangle_{\vec{x}}$, hence $p = q_1(x_1 - W_1) + \cdots + q_d(x_d - W_d) + r$ where

each $q_\ell \in \mathbb{Z}_m[\vec{x} : \vec{w}]$ and $r \in \langle B_{\mathsf{Null}_m[\vec{w}]}\rangle_{\vec{x}:\vec{w}} \subseteq \mathbb{Z}_m[\vec{x} : \vec{w}]$. Then, $[\![r]\!]_{\vec{w}}(\vec{a}) = 0$ for all $\vec{a} \in \mathbb{Z}_m^{|\vec{w}|}$. Thus, from the equality derived above it follows $[\![p]\!]_{\vec{x}}(\vec{b}) = 0$ for all $\vec{b} \in \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\}$. Therefore, $p \in \alpha_{\vec{x}}(\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\})$.

- Suppose $p \in \alpha_{\vec{x}}(\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\})$. Then, $[\![p]\!]_{\vec{x}}(\vec{b}) = 0$ for all $\vec{b} \in \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\}$. Moreover, from Lemma 3, it follows $p = q_1(x_1 - W_1) + \cdots + q_d(x_d - W_d) + r$ where each $q_\ell \in \mathbb{Z}_m[\vec{x} : \vec{w}]$ and $r \in \mathbb{Z}_m[\vec{w}] \subseteq \mathbb{Z}_m[\vec{x} : \vec{w}]$. Thus, from the equality derived above, $[\![r]\!]_{\vec{w}}(\vec{a}) = [\![r]\!]_{\vec{x}:\vec{w}}(\vec{b} : \vec{a}) = 0$ for all $\vec{a} \in \mathbb{Z}_m^{|\vec{w}|}$, where $\vec{b} = [\![\vec{W}]\!]_{\vec{w}}(\vec{a})$. It follows $r \in \langle B_{\mathsf{Null}_m[\vec{w}]}\rangle_{\vec{w}}$, hence $p \in \langle\{x_1 - W_1, \ldots, x_d - W_d\} \cup B_{\mathsf{Null}_m[\vec{w}]}\rangle_{\vec{x}:\vec{w}}$. Since $p \in \mathbb{Z}_m[\vec{x}]$ it thus follows $p \in \mathsf{elim}[\vec{w}](\langle\{x_1 - W_1, \ldots, x_d - W_d\} \cup B_{\mathsf{Null}_m[\vec{w}]}\rangle_{\vec{x}:\vec{w}}) = \langle B\rangle_{\vec{x}}$.

Therefore $\langle B\rangle_{\vec{x}} = \alpha_{\vec{x}}(\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\})$, as required.

□

Proof of Theorem 5. It must be shown $\uparrow_{\vec{x}} \langle B\rangle_{\vec{x}} = \langle B'\rangle_{\vec{x}}$. First note since $\mathcal{W}$ is a cover of B over $\vec{w}$ it follows $\gamma_{\vec{x}}(\langle B\rangle_{\vec{x}}) = \gamma_{\vec{x}}(B) = \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{W} \in \mathcal{W} \wedge \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\}$. Now, if $\gamma_{\vec{x}}(B) = \emptyset$ then $\uparrow_{\vec{x}} \langle B\rangle_{\vec{x}} = \langle 1\rangle_{\vec{x}}$. But, from the previous equality it also follows $\mathcal{W} = \emptyset$, hence $\bigsqcup_{\vec{W}\in\mathcal{W}}\langle B_{\vec{W}}\rangle_{\vec{x}} = \langle B'\rangle_{\vec{x}}$ is degenerate and equals $\mathbb{Z}_m[\vec{x}] = \langle 1\rangle_{\vec{x}} = \uparrow_{\vec{x}} \langle B\rangle_{\vec{x}}$ as required.

Thus, suppose $\gamma_{\vec{x}}(B) \neq \emptyset$, so $\mathcal{W} \neq \emptyset$. Then, if $\vec{W} \in \mathcal{W}$ it follows $\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\} \subseteq \gamma_{\vec{x}}(\langle B\rangle_{\vec{x}})$. Therefore, from Proposition 6, $\alpha_{\vec{x}}(\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\}) \sqsubseteq \langle B\rangle_{\vec{x}}$. It thus follows from Lemma 7 that $\langle B_{\vec{W}}\rangle_{\vec{x}} \sqsubseteq \langle B\rangle_{\vec{x}}$, hence by Proposition 5, $\langle B'\rangle_{\vec{x}} = \bigsqcup_{\vec{W}\in\mathcal{W}}\langle B_{\vec{W}}\rangle_{\vec{x}} \sqsubseteq \langle B\rangle_{\vec{x}}$ and so $\gamma_{\vec{x}}(\langle B'\rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(\langle B\rangle_{\vec{x}})$.

But also, since $\langle B'\rangle_{\vec{x}} = \bigsqcup_{\vec{W}\in\mathcal{W}}\langle B_{\vec{W}}\rangle_{\vec{x}}$ it follows $\gamma_{\vec{x}}(\langle B'\rangle_{\vec{x}}) \supseteq \bigcup_{\vec{W}\in\mathcal{W}} \gamma_{\vec{x}}(\langle B_{\vec{W}}\rangle_{\vec{x}})$. By Lemma 7 it follows $\langle B_{\vec{W}}\rangle_{\vec{x}} = \alpha_{\vec{x}}(\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\})$, hence $\gamma_{\vec{x}}(\langle B_{\vec{W}}\rangle_{\vec{x}}) \supseteq \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\}$ and so $\gamma_{\vec{x}}(\langle B'\rangle_{\vec{x}}) \supseteq \bigcup_{\vec{W}\in\mathcal{W}}\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\} = \{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{W} \in \mathcal{W} \wedge \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\} = \gamma_{\vec{x}}(\langle B\rangle_{\vec{x}})$.

Thus, $\gamma_{\vec{x}}(\langle B\rangle_{\vec{x}}) = \gamma_{\vec{x}}(\langle B'\rangle_{\vec{x}})$. But then Theorem **??** implies $\uparrow_{\vec{x}} \langle B\rangle_{\vec{x}} = \uparrow_{\vec{x}} \langle B'\rangle_{\vec{x}}$. To conclude the proof, note since $\langle B_{\vec{W}}\rangle_{\vec{x}} = \alpha_{\vec{x}}(\{[\![\vec{W}]\!]_{\vec{w}}(\vec{a}) \mid \vec{a} \in \mathbb{Z}_m^{|\vec{w}|}\})$ it follows $\langle B_{\vec{W}}\rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$ for each $\vec{W} \in \mathcal{W}$. Thus, since $\langle B'\rangle_{\vec{x}} = \bigsqcup_{\vec{W}\in\mathcal{W}}\langle B_{\vec{W}}\rangle_{\vec{x}}$ it follows from Proposition 5 that $\langle B'\rangle_{\vec{x}} \in \mathsf{MPAD}_m[\vec{x}]$. In particular, $\uparrow_{\vec{x}} \langle B'\rangle_{\vec{x}} = \langle B'\rangle_{\vec{x}}$, hence $\uparrow_{\vec{x}} \langle B\rangle_{\vec{x}} = \langle B'\rangle_{\vec{x}}$, as required.

□

Proof of Proposition 13. The assumption means $\uparrow_{\vec{x}} \langle B_1 \cup B_2 \rangle_{\vec{x}} = \langle B \rangle_{\vec{x}}$ and it must be shown $\uparrow_{\vec{x}} (\langle B_1 \rangle_{\vec{x}} \cup \langle B_2 \rangle_{\vec{x}}) = \langle B \rangle_{\vec{x}}$, which follows iff $\uparrow_{\vec{x}} \langle B_1 \cup B_2 \rangle_{\vec{x}} = \uparrow_{\vec{x}} (\langle B_1 \rangle_{\vec{x}} \cup \langle B_2 \rangle_{\vec{x}})$. Thus, by Proposition 4, the result holds iff $\gamma_{\vec{x}}(\langle B_1 \cup B_2 \rangle_{\vec{x}}) = \gamma_{\vec{x}}(\langle B_1 \rangle_{\vec{x}} \cup \langle B_2 \rangle_{\vec{x}})$. To show this:

- Since $B_1 \subseteq \langle B_1 \rangle_{\vec{x}}$ and $B_2 \subseteq \langle B_2 \rangle_{\vec{x}}$ it follows $B_1 \cup B_2 \subseteq \langle B_1 \rangle_{\vec{x}} \cup \langle B_2 \rangle_{\vec{x}}$, hence $\gamma_{\vec{x}}(\langle B_1 \cup B_2 \rangle_{\vec{x}}) = \gamma_{\vec{x}}(B_1 \cup B_2) \supseteq \gamma_{\vec{x}}(\langle B_1 \rangle_{\vec{x}} \cup \langle B_2 \rangle_{\vec{x}})$.
- Since $\langle B_1 \rangle_{\vec{x}} \subseteq \langle B_1 \cup B_2 \rangle_{\vec{x}}$ and $\langle B_2 \rangle_{\vec{x}} \subseteq \langle B_1 \cup B_2 \rangle_{\vec{x}}$ it follows $\langle B_1 \rangle_{\vec{x}} \cup \langle B_2 \rangle_{\vec{x}} \subseteq \langle B_1 \cup B_2 \rangle_{\vec{x}}$. Thus, $\gamma_{\vec{x}}(\langle B_1 \cup B_2 \rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(\langle B_1 \rangle_{\vec{x}} \cup \langle B_2 \rangle_{\vec{x}})$.

The result follows.

□

A.6 Proofs for abstract transfer functions

Proof of Proposition 14. Let $A' = [\![\textbf{assume}\ (p = 0)]\!](A)$. By assumption, $\uparrow_{\vec{x}} \langle B \cup \{p\}\rangle_{\vec{x}} = \langle B'\rangle_{\vec{x}}$, hence it must be shown $\alpha_{\vec{x}}(A') = \uparrow_{\vec{x}} \langle B \cup \{p\}\rangle_{\vec{x}}$. To that end:

- Suppose $q \in \alpha_{\vec{x}}(A')$, so $A' \subseteq \gamma_{\vec{x}}(q)$, and let $\vec{a} \in \gamma_{\vec{x}}(\langle B \cup \{p\}\rangle_{\vec{x}})$. Then, $\vec{a} \in \gamma_{\vec{x}}(B) = A$ and $[\![p]\!]_{\vec{x}}(\vec{a}) = 0$, hence $\vec{a} \in A' \subseteq \gamma_{\vec{x}}(q)$. It follows $\gamma_{\vec{x}}(\langle B \cup \{p\}\rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(q)$, hence $q \in \uparrow_{\vec{x}} \langle B \cup \{p\}\rangle_{\vec{x}}$.
- Suppose $q \in \uparrow_{\vec{x}} \langle B \cup \{p\}\rangle_{\vec{x}}$, so $\gamma_{\vec{x}}(\langle B \cup \{p\}\rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(q)$, and let $\vec{a} \in A'$. Then, $\vec{a} \in A = \gamma_{\vec{x}}(B)$ and $[\![p]\!]_{\vec{x}}(\vec{a}) = 0$, hence $\vec{a} \in \gamma_{\vec{x}}(\langle B \cup \{p\}\rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(q)$. It follows $A' \subseteq \gamma_{\vec{x}}(q)$, hence $q \in \alpha_{\vec{x}}(A')$.

Thus $\uparrow_{\vec{x}} \langle B \cup \{p\}\rangle_{\vec{x}} = \alpha_{\vec{x}}(A')$, as required.

□

Proof of Proposition 15. Let $A' = [\![\textbf{assume}\ (p \neq 0)]\!](A)$. By assumption, for each $1 \leq k \leq \omega$ it holds $\uparrow_{\vec{x}} \langle B \cup \{2^{\omega-k}p + 2^{\omega-1}\}\rangle_{\vec{x}} = \langle B_k\rangle_{\vec{x}}$. Now:

- Let $q \in \alpha_{\vec{x}}(A')$ and $\vec{a} \in \gamma_{\vec{x}}(\langle B \cup \{2^{\omega-k}p + 2^{\omega-1}\}\rangle_{\vec{x}})$. Then, $\vec{a} \in \gamma_{\vec{x}}(B) = A$ and $[\![2^{\omega-k}p + 2^{\omega-1}]\!]_{\vec{x}}(\vec{a}) = 2^{\omega-k}[\![p]\!]_{\vec{x}}(\vec{a}) + 2^{\omega-1} = 0$. In particular, $[\![p]\!]_{\vec{x}}(\vec{a}) \neq 0$, so $\vec{a} \in A'$. But then, $[\![q]\!]_{\vec{x}}(\vec{a}) = 0$, hence $\gamma_{\vec{x}}(\langle B \cup \{2^{\omega-k}p + 2^{\omega-1}\}\rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(q)$. It follows $q \in \uparrow_{\vec{x}} \langle B \cup \{2^{\omega-k}p + 2^{\omega-1}\}\rangle_{\vec{x}} = \langle B_k\rangle_{\vec{x}}$, hence $q \in \bigcap_{k=1}^{\omega} \langle B_k\rangle_{\vec{x}} = \bigsqcup_{k=1}^{\omega} \langle B_k\rangle_{\vec{x}}$.
- Let $q \in \bigsqcup_{k=1}^{\omega} \langle B_k\rangle_{\vec{x}} = \bigcap_{k=1}^{\omega} \langle B_k\rangle_{\vec{x}}$, so $q \in \langle B_k\rangle_{\vec{x}}$ for each $1 \leq k \leq \omega$, and $\vec{a} \in A'$. Then, $\vec{a} \in A = \gamma_{\vec{x}}(B)$ and $[\![p]\!]_{\vec{x}}(\vec{a}) \neq 0$. Let $[\![p]\!]_{\vec{x}}(\vec{a}) = \sum_{\ell=1}^{\omega} b_\ell 2^{\ell-1}$ where each $b_\ell \in \{0, 1\}$. Since $[\![p]\!]_{\vec{x}}(\vec{a}) \neq 0$, there exists $1 \leq k \leq \omega$ such that $b_k = 1$ and $b_\ell = 0$ for all $\ell < k$. Then, $2^{\omega-k}[\![p]\!]_{\vec{x}}(\vec{a}) = \sum_{\ell=1}^{\omega} b_\ell 2^{\omega-k+\ell-1} = \sum_{\ell=1}^{k} b_\ell 2^{\omega-k+\ell-1} = 2^{\omega-1}$. In particular $[\![2^{\omega-k}p + 2^{\omega-1}]\!]_{\vec{x}}(\vec{a}) = 2^{\omega-k}[\![p]\!]_{\vec{x}}(\vec{a}) + 2^{\omega-1} = 2^{\omega-1} + 2^{\omega-1} = 0$, hence $\vec{a} \in \gamma_{\vec{x}}(\langle B \cup \{2^{\omega-k}p + 2^{\omega-1}\}\rangle_{\vec{x}})$. But since $q \in \langle B_k\rangle_{\vec{x}} = \uparrow_{\vec{x}} \langle B \cup \{2^{\omega-k}p + 2^{\omega-1}\}\rangle_{\vec{x}}$ it follows $\gamma_{\vec{x}}(\langle B \cup \{2^{\omega-k}p + 2^{\omega-1}\}\rangle_{\vec{x}}) \subseteq \gamma_{\vec{x}}(q)$, hence $\vec{a} \in \gamma_{\vec{x}}(q)$. Therefore, $A' \subseteq \gamma_{\vec{x}}(q)$ and so $q \in \alpha_{\vec{x}}(A')$.

It follows $\bigsqcup_{k=1}^{\omega} \langle B_k\rangle_{\vec{x}} = \alpha_{\vec{x}}(A')$, as required.

□

Proof of Lemma 8. Put $w = y_j$. First, if $\vec{b} \in A'$ then $\vec{b} = \langle a_1, \ldots, a_{j-1}, [\![p]\!]_{\vec{x}}(\vec{a}), a_j, \ldots, a_d\rangle$ for some $\vec{a} \in A$. Now, if $q \in B$ then $[\![q]\!]_{\vec{y}}(\vec{b}) = [\![q]\!]_{\vec{x}}(\vec{a}) = 0$. Moreover, $[\![w-p]\!]_{\vec{y}}(\vec{b}) = [\![p]\!]_{\vec{x}}(\vec{a}) - [\![p]\!]_{\vec{x}}(\vec{a}) = 0$. It follows $\vec{b} \in \gamma_{\vec{y}}(B \cup \{w-p\}) = \gamma_{\vec{y}}(\langle B \cup \{w-p\}\rangle_{\vec{y}})$, thus $A' \subseteq \gamma_{\vec{y}}(\langle B \cup \{w-p\}\rangle_{\vec{y}})$. Therefore, by Proposition 6 it follows $\alpha_{\vec{y}}(A') \sqsubseteq \langle B \cup \{w-p\}\rangle_{\vec{y}}$, or equivalently $\langle B \cup \{w-p\}\rangle_{\vec{y}} \subseteq \alpha_{\vec{y}}(A')$.

To show the opposite inclusion, let $q \in \alpha_{\vec{y}}(A')$. Then, by Lemma 3, $q = u(w-p) + r$ for some $u \in \mathbb{Z}_m[\vec{y}]$ and $r \in \mathbb{Z}_m[\vec{x}]$. Let $\vec{a} \in A$. Then, $\vec{b} = \langle a_1, \ldots, a_{j-1}, [\![p]\!]_{\vec{x}}(\vec{a}), a_j, \ldots, a_d\rangle \in A'$ and $[\![r]\!]_{\vec{x}}(\vec{a}) = [\![r]\!]_{\vec{y}}(\vec{b}) = [\![q-u(w-p)]\!]_{\vec{y}}(\vec{b}) = [\![q]\!]_{\vec{y}}(\vec{b}) - [\![u]\!]_{\vec{y}}(\vec{b})[\![w-p]\!]_{\vec{y}}(\vec{b}) = 0 - [\![u]\!]_{\vec{y}}(\vec{b})0 = 0$. It follows $A \subseteq \gamma_{\vec{x}}(r)$, thus $r \in \langle B\rangle_{\vec{x}}$. Since $q = u(w-p)+r$ where $r \in \langle B\rangle_{\vec{x}}$ it thus follows $q \in \langle B \cup \{w-p\}\rangle_{\vec{y}}$. Therefore, $\alpha_{\vec{y}}(A') \subseteq \langle B \cup \{w-p\}\rangle_{\vec{y}}$ as required. □

Proof for Proposition 16. Let $A' = \mathbb{Z}_m \times A$. Observe $b : \vec{a} \in \gamma_{w:\vec{x}}(B)$ iff $\vec{a} \in \gamma_{\vec{x}}(B) = A$, hence $\gamma_{w:\vec{x}}(\langle B\rangle_{w:\vec{x}}) = \gamma_{w:\vec{x}}(B) = A'$. Thus, for $p \in \mathbb{Z}_m[w : \vec{x}]$ it holds that $\gamma_{w:\vec{x}}(\langle B\rangle_{w:\vec{x}}) \subseteq \gamma_{w:\vec{x}}(p)$ iff $A' \subseteq \gamma_{w:\vec{x}}(p)$, hence $\uparrow_{w:\vec{x}} \langle B\rangle_{w:\vec{x}} = \alpha_{w:\vec{x}}(A')$. By assumption $\uparrow_{w:\vec{x}} \langle B\rangle_{w:\vec{x}} = \langle B''\rangle_{w:\vec{x}}$, hence $\langle B''\rangle_{w:\vec{x}} = \alpha_{w:\vec{x}}(A')$. By Proposition 8, $\mathsf{elim}[x_j](\langle B''\rangle_{w:\pi_j(\vec{x})}) = \alpha_{w:\pi_j(\vec{x})}(\pi_{j+1}(A'))$. Since $\mathsf{elim}[x_j](\langle B''\rangle_{w:\vec{x}}) = \langle B'''\rangle_{w:\pi_j(\vec{x})}$ holds by assumption, it follows $\langle B'''\rangle_{w:\pi_j(\vec{x})} = \alpha_{w:\pi_j(\vec{x})}(\pi_{j+1}(A'))$.

Now, let $A'' = \{\langle b_1, \ldots, b_j, [\![w]\!]_{w:\pi_j(\vec{x})}(\vec{b}), b_{j+1}, \ldots, b_d\rangle \mid \vec{b} \in \pi_{j+1}(A')\}$. Then, Lemma 8 implies $\alpha_{w:\vec{x}}(A'') = \langle B'' \cup \{x_j - w\}\rangle_{w:\vec{x}}$. From Proposition 8 it follows $\mathsf{elim}[w](\alpha_{w:\vec{x}}(A'')) = \alpha_{\vec{x}}(\pi_1(A''))$, hence $\mathsf{elim}[w](\langle B'' \cup \{x_j - w\}\rangle_{w:\vec{x}}) = \alpha_{\vec{x}}(\pi_1(A''))$. But, by assumption, $\mathsf{elim}[w](\langle B'' \cup \{x_j - w\}\rangle_{w:\vec{x}}) = \langle B'\rangle_{\vec{x}}$, hence $\langle B'\rangle_{\vec{x}} = \alpha_{\vec{x}}(\pi_1(A''))$.

To conclude, note that $\pi_{j+1}(A') = \pi_{j+1}(\mathbb{Z}_m \times A) = \mathbb{Z}_m \times \pi_j(A)$, and so

$$\begin{aligned} A'' &= \{\langle b_1, \ldots, b_j, [\![w]\!]_{w:\pi_j(\vec{x})}(\vec{b}), b_{j+1}, \ldots, b_d\rangle \mid \vec{b} \in \pi_{j+1}(A')\} \\ &= \{\langle b_1, \ldots, b_j, b_1, b_{j+1}, \ldots, b_d\rangle \mid \vec{b} \in \mathbb{Z}_m \times \pi_j(A)\} \\ &= \{\langle b, a_1, \ldots, a_{j-1}, b, a_{j+1}, \ldots, a_d\rangle \mid \vec{a} \in A \wedge b \in \mathbb{Z}_m\} \end{aligned}$$

Thus, $\pi_1(A'') = \{\langle a_1, \ldots, a_{j-1}, b, a_{j+1}, \ldots, a_d\rangle \mid \vec{a} \in A \wedge b \in \mathbb{Z}_m\} = [\![x_j \coloneqq *]\!](A)$, hence $\langle B'\rangle_{\vec{x}} = \alpha_{\vec{x}}([\![x_j \coloneqq *]\!](A))$, as required.

□

Proof of Proposition 17. Let $A' = \{\langle [\![p]\!]_{\vec{x}}(\vec{a}), a_1, \ldots, a_d\rangle \mid \vec{a} \in A\}$. Then, $\alpha_{w:\vec{x}}(A') = \langle B \cup \{w-p\}\rangle_{w:\vec{x}}$ by Lemma 8. Thus, from Proposition 8 it follows $\mathsf{elim}[x_j](\alpha_{w:\vec{x}}(A')) =$

$\alpha_{w:\pi_j(\vec{x})}(\pi_{j+1}(A'))$, and therefore $\mathsf{elim}[x_j](\langle B \cup \{w - p\}\rangle_{w:\vec{x}}) = \alpha_{w:\pi_j(\vec{x})}(\pi_{j+1}(A'))$. But, by assumption, $\mathsf{elim}[x_j](\langle B \cup \{w - p\}\rangle_{w:\vec{x}}) = \langle B''\rangle_{w:\pi_j(\vec{x})}$, hence $\langle B''\rangle_{w:\pi_j(\vec{x})} = \alpha_{w:\pi_j(\vec{x})}(\pi_{j+1}(A'))$.

Now, let $A'' = \{\langle b_1, \ldots, b_j, [\![w]\!]_{w:\pi_j(\vec{x})}(\vec{b}), b_{j+1}, \ldots, b_d\rangle \mid \vec{b} \in \pi_{j+1}(A')\}$. Then, Lemma 8 implies $\alpha_{w:\vec{x}}(A'') = \langle B'' \cup \{x_j - w\}\rangle_{w:\vec{x}}$. From Proposition 8 it follows $\mathsf{elim}[w](\alpha_{w:\vec{x}}(A'')) = \alpha_{\vec{x}}(\pi_1(A''))$, hence $\mathsf{elim}[w](\langle B'' \cup \{x_j - w\}\rangle_{w:\vec{x}}) = \alpha_{\vec{x}}(\pi_1(A''))$. But, by assumption, $\mathsf{elim}[w](\langle B'' \cup \{x_j - w\}\rangle_{w:\vec{x}}) = \langle B'\rangle_{\vec{x}}$, hence $\langle B'\rangle_{\vec{x}} = \alpha_{\vec{x}}(\pi_1(A''))$.

To conclude, note that

$$\begin{aligned} A'' &= \{\langle b_1, \ldots, b_j, [\![w]\!]_{w:\pi_j(\vec{x})}(\vec{b}), b_{j+1}, \ldots, b_d\rangle \mid \vec{b} \in \pi_{j+1}(A')\} \\ &= \{\langle b_1, \ldots, b_j, b_1, b_{j+1}, \ldots, b_d\rangle \mid \vec{b} \in \pi_{j+1}(A')\} \\ &= \{\langle [\![p]\!]_{\vec{x}}(\vec{a}), a_1, \ldots, a_{j-1}, [\![p]\!]_{\vec{x}}(\vec{a}), a_{j+1}, \ldots, a_d\rangle \mid \vec{a} \in A\} \end{aligned}$$

Thus, $\pi_1(A'') = \{\langle a_1, \ldots, a_{j-1}, [\![p]\!]_{\vec{x}}(\vec{a}), a_{j+1}, \ldots, a_d\rangle \mid \vec{a} \in A\} = [\![x_j \coloneqq p]\!](A)$, hence $\langle B'\rangle_{\vec{x}} = \alpha_{\vec{x}}([\![x_j \coloneqq p]\!](A))$, as required. □

Bibliography

[1] Ábráham, E. (2015). Building Bridges between Symbolic Computation and Satisfiability Checking. In *International Symposium on Symbolic and Algebraic Computation*, ACM Press, pp. 1–6.

[2] Adams, W. and Loustaunau, P. (1994). *An Introduction to Gröbner Bases*. American Mathematical Society.

[3] Backeman, P., Rümmer, P. and Zeljic, A. (2018). Bit-Vector Interpolation and Quantifier Elimination by Lazy Reduction. In *Formal Methods in Computer Aided Design*, IEEE, pp. 1–10.

[4] Bosma, W. et al. (2010). *Handbook of Magma Functions*. University of Sydney, 2nd edn., `https://magma.maths.usyd.edu.au/magma/handbook/`.

[5] Brickenstein, M. et al. (2009). New Developments in the Theory of Gröbner Bases and Applications to Formal Verification. *Journal of Pure and Applied Algebra*, 213, pp. 1612–1635.

[6] Brown, C. (2016). Bridging Two Communities to Solve Real Problems. In *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, IEEE Press, pp. 11–14.

[7] Bryant, R., German, S. and Velev, M. (2001). Exploiting Positive Equality in a Logic of Equality with Uninterpreted Functions. *ACM Transactions on Computational Logic*, 2(1), pp. 93–134.

[8] Buchberger, B. (2006). Bruno Buchberger's PhD thesis 1965: An Algorithm for Finding the Basis Elements of the Residue Class ring of a Zero Dimensional Polynomial Ideal. *Journal of Symbolic Computation*, 41, pp. 475–511.

[9] Cacher, D. et al. (2014). Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases. *Science of Computer Programming*, 93, pp. 89–109.

[10] Codish, M. et al. (1995). Improving Abstract Interpretations by Combining Domains. *ACM Transactions on Programming Languages and Systems*, 17(1).

[11] Collins, G. and Hong, H. (1991). Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *Journal of Symbolic Computation*, 12, pp. 299–328.

[12] Collins, G. E. (1975). Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. In *Automata Theory and Formal Languages*, Springer, pp. 134–183.

[13] Colón, M. A. (2004). Approximating the Algebraic Relational Semantics of Imperative Programs. In *Static Analysis Symposium*, *Lecture Notes in Computer Science*, vol. 3148, Springer, pp. 296–311.

[14] Cousot, P. and Cousot, R. (1977). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages*, ACM Press, pp. 238–252.

[15] Cousot, P. and Cousot, R. (1992). Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In *International Symposium on Programming Language Implementation and Logic Programming*, Springer, pp. 269–295.

[16] Cousot, P. and N, H. (1978). Automatic Discovery of Linear Restraints among Variables of a Program. In *Principles of Programming Languages*, ACM Press, p. 8496.

[17] Cousot, P. et al. (2005). The ASTRÉE Analyzer. In *European Symposium on Programming*, *Lecture Notes in Computer Science*, vol. 3444, Springer, pp. 21–30.

[18] Cox, D., Little, J. and O'Shea, D. (1992). *Ideals, Varieties and Algorithms.* Undergraduate Texts in Mathematics, Springer.

[19] Davenport, J. H. et al. (2019). Symbolic Computation and Satisfiability Checking. *Journal of Symbolic Computation*, `https://doi.org/10.1016/j.jsc.2019.07.017`.

[20] Davis, M., Logemann, G. and Loveland, D. (1962). A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7), p. 394?397.

[21] de Oliveira, S., Bensalem, S. and Prevosto, V. (2016). Polynomial invariants by linear algebra. In *Automated Technology for Verification and Analysis*, vol. 9938, Springer, pp. 479–494.

[22] Dill, D. (1989). Timing Assumptions and Verification of Finite-state Concurrent Systems. In *CAV*, *Lecture Notes in Computer Science*, vol. 407, Springer, pp. 197–212.

[23] Eder, C. and Faugàre, J. C. (2017). A Survey on Signature-Based Algorithms for Computing Gröbner Bases. *Journal of Symbolic Computation*, 80, pp. 719–784.

[24] Eder, C., Pfister, G. and Popescu, A. (2017). On Signature-Based Gröbner Bases over Euclidean Rings. In *International Symposium on Symbolic and Algebraic Computation*, ACM Press, pp. 141–148.

[25] Eder, C., Pfister, G. and Popescu, A. (2021). Standard bases over Euclidean domains. *Journal of Symbolic Computation*, 102, pp. 21–36.

[26] Elder, M. et al. (2014). Abstract Domains of Affine Relations. *ACM Transactions on Programming Languages and Systems*, 36(4), pp. 1–73.

[27] Faugére, J. (1999). A New Efficient Algorithm for Computing Gröbner Bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3), pp. 61–88.

[28] Faugère, J. C. (2002). A New Efficient Algorithm for Computing GröbnerBases without Reduction to Zero (F5). In *International Symposium on Symbolic and Algebraic Computation*, ACM Press, pp. 75–83.

[29] Fekete, Y. and Codish, M. (2014). Simplifying Pseudo-Boolean Constraints in Residual Number Systems. In *SAT*, *Lecture Notes in Computer Science*, vol. 8561, Springer, pp. 351–366.

[30] Francis, M. and Verron, T. (2020). A Signature-Based Algorithm for Computing Gröbner Bases over Principal Ideal Domains. *Mathematics in Computer Science*, 14, pp. 515–530.

[31] Ganesh, V. and Dill, D. (2007). A Decision Procedure for Bit-vectors and Arrays. In *Computer-Aided Verification*, Springer, no. 4590 in Lecture Notes in Computer Science, pp. 519–531.

[32] Gange, G. et al. (2013). Solving Difference Constraints over Modular Arithmetic. In *CADE*, *Lecture Notes in Computer Science*, vol. 7898, Springer, pp. 215–230.

[33] Gange, G. et al. (2014). Interval Analysis and Machine Arithmetic: Why Signedness Ignorance Is Bliss. *ACM Transactions on Programming Languages and Systems*, 37(1), pp. 1–35.

[34] Granger, P. (1991). Static Analysis of Linear Congruence Equalities Among Variables of a Program. In *Theory and Practice of Software*, *Lecture Notes in Computer Science*, vol. 493, Springer, pp. 169–192.

[35] Greuel, G.-M., Seelisch, F. and Wienand, O. (2011). The Gröbner Basis of the Ideal of Vanishing Polynomials. *Journal of Symbolic Computation*, 46(5), pp. 561–570.

[36] Griggio, A. (2011). Effective Word-Level Interpolation for Software Verification. In *Formal Methods in Computer-Aided Design*, IEEE, pp. 28–36.

[37] Harrison, W. (1977). Compiler Analysis of the Value Ranges for Variables. *IEEE Transactions on Software Engineering*, 3(3), pp. 243–250.

[38] Hilbert, D. (1890). Über die Theorie der Algebraischen Formen. *Mathematische Annalen*, 36(4), pp. 473–534.

[39] Horácek, J. et al. (2017). Integrating Algebraic and SAT Solvers. In *Mathematical Aspects of Computer and Information Sciences*, vol. 10693, Springer, pp. 147–162.

[40] Howe, J., King, A. and Simon, A. (2019). Incremental Closure for Systems of Two Variables Per Equality. *Theoretical Computer Science*, 768, pp. 1–64.

[41] Hrushovski, E. et al. (2018). Polynomial Invariants for Affine Programs. In *Logic in Computer Science*, ACM Press, pp. 530–539.

[42] Humenberger, A., Jaroschek, M. and Kovács, L. (2018). Invariant Generation for Multi-Path Loops with Polynomial Assignments. In *Verification, Model Checking and Abstract Interpretation*, *Lecture Notes in Computer Science*, vol. 10747, Springer, pp. 226–246.

[43] Hungerbühler, N. and Specker, E. (2006). A Generalization of the Smarandache Function to Several Variables. *Integers: Electronic Journal Combinatorial Number Theory*, 6, pp. 1–11.

[44] Jovanović, D. and de Moura, L. (2012). Solving Non-linear Arithmetic. In *International Joint Conference on Automated Reasoning*, vol. 7364, Springer, pp. 339–354.

[45] Kandri-Rody, A. and Kapur, D. (1988). Computing a Gröbner Basis of a Polynomial Ideal over a Euclidean Domain. *Journal of Symbolic Computation*, 6(1), pp. 37–57.

[46] Karr, M. (1976). Affine Relationships Among Variables of a Program. *Acta Informatica*, 6(2), pp. 133–151.

[47] Kempner, A. J. (1921). Polynomials and their Residue Systems. *Transactions of the American Mathematical Society*, 22(2), pp. 240–266.

[48] King, A. and Søndergaard, H. (2008). Inferring Congruence Equations using SAT. In *Computer-Aided Verification*, *Lecture Notes in Computer Science*, vol. 5123, Springer, pp. 281–293.

[49] Kovács, L. (2008). Reasoning Algebraically About P-Solvable Loops. In *Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes in Computer Science*, vol. 4963, Springer, pp. 249–264.

[50] Kovács, L. (2010). A Complete Invariant Generation Approach for P-Solvable Loops. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, *Lecture Notes in Computer Science*, vol. 5947, Springer, pp. 242–256.

[51] Kroening, D. and Strichman, O. (2016). *Decision Procedures*. Springer.

[52] Lang, S. (2002). *Graduate Texts in Mathematics: Algebra*. Springer.

[53] Manquinhoand, V. M. and Marques-Silva, J. (2006). Using Cutting Planes in Pseudo-Boolean Optimization. *Journal on Satisability, Boolean Modeling and Computation*, 2, pp. 199–208.

[54] Marques-Silva, J. and Sakallah, K. (1999). GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48, pp. 506–521.

[55] Marques-Silva, J. P., Lynce, I. and Malik, S. (2009). Conflict-Driven Clause Learning SAT Solvers. In A. Biere, M. Heule, H. Van Maaren and T. Walsh, eds., *Handbook of Satisfiability*, IOS Press, pp. 131–153.

[56] Marques-Silva, J. P. and Sakallah, K. A. (1999). GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5), pp. 506–521.

[57] Mayr, E. (1989). Membership in Polynomial Ideals over Q is Exponential Space Complete. In *Symposium on Theoretical Aspects of Computer Science*, *Lecture Notes in Computer Science*, vol. 349, Springer, pp. 400–406.

[58] Michel, L. and Van Hentenryck, P. (2012). Constraint Satisfaction over Bit-vectors. In *Constraint Programming*, *Lecture Notes in Computer Science*, vol. 7514, Springer, pp. 527–543.

[59] Miné, A. (2006). The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1), pp. 31–100.

[60] Möller, H. M. (1988). On the Construction of Gröbner Bases Using Syzygies. *Journal of Symbolic Computation*, 6(2–3), pp. 345–359.

[61] Moskewicz, M. W. et al. (2001). Chaff: Engineering an Efficient SAT Solver. In *Design Automation Conference*, pp. 530–535.

[62] Müller-Olm, M. and Seidl, H. (2004). A Note on Karr's Algorithm. In *International Colloquium on Automata, Languages, and Programming*, *Lecture Notes in Computer Science*, vol. 3142, Springer, pp. 1016–1028.

[63] Müller-Olm, M. and Seidl, H. (2004). Computing Polynomial Program Invariants. *Information Processing Letters*, 91, pp. 233–244.

[64] Müller-Olm, M. and Seidl, H. (2005). Analysis of Modular Arithmetic. In *European Symposium on Programming*, *Lecture Notes in Computer Science*, vol. 3444, Springer, pp. 46–60.

[65] Müller-Olm, M. and Seidl, H. (2007). Analysis of Modular Arithmetic. *ACM Transactions on Programming Languages and Systems*, 29(5), pp. 1–26.

[66] Nelson, G. and Oppen, D. (1979). Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems*, 1(2), pp. 245–257.

[67] Nieuwenhuis, R. and Oliveras, A. (2005). DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic. In *Computer-Aided Verification*, *Lecture Notes in Computer Science*, vol. 3576, Springer, pp. 321–334.

[68] Nieuwenhuis, R., Oliveras, A. and Tinelli, C. (2006). Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6), pp. 937–977.

[69] Reps, T., Sagiv, M. and Yorsh, G. (2004). Symbolic Implementation of the Best Transformer. In *Verification, Model Checking and Abstract Interpretation*, *Lecture Notes in Computer Science*, vol. 2937, Springer, pp. 252–266.

[70] Rodríguez-Carbonell, E. and Kapur, D. (2007). Generating all Polynomial Invariants in Simple Loops. *Journal of Symbolic Computation*, 42, pp. 443–476.

[71] Sankaranarayanan, S., Sipma, H. B. and Manna, Z. (2004). Non-linear Loop Invariant Generation using Gröbner Bases. In *Principles of Programming Languages*, ACM Press, pp. 318–329.

[72] Seed, T., King, A. and Evans, N. (2020). Reducing Bit-Vector Polynomials to SAT using Gröbner Bases. In *Theory and Applications of Satisfiability Testing*, *Lecture Notes in Computer Science*, vol. 12178, Springer, pp. 361–377.

[73] Shekhar, N. et al. (2005). Equivalence Verification of Polynomial Datapaths with Fixed-Size Bit-Vectors using Finite Ring Algebra. In *International Conference on Computer-Aided Design*, IEEE Computer Society, pp. 291–296.

[74] Simon, A. and King, A. (2010). The Two Variable Per Inequality Abstract Domain. *Higher-Order and Symbolic Computation*, 23(1), pp. 87–143.

[75] Singmaster, D. (1974). On Polynomial Functions (mod m). *Journal of Number Theory*, 6(5), pp. 345–352.

[76] T. Viehmann and G. Kremer and E. Ábráham (2017). Comparing Different Projection Operators in the Cylindrical Algebraic Decomposition for SMT Solving. In *International Workshop on Satisfiability Checking and Symbolic Computation*, `http://www.sc-square.org/CSA/workshop2-papers/RP2-FinalVersion.pdf`.

[77] Wang, W., Søndergaard, H. and Stuckey, P. (2019). Wombit: A Portfolio Bit-Vector Solver Using Word-Level Propagation. *Journal of Automated Reasoning*, 63(3), pp. 723–762.

[78] Warren, H. (2012). *Hacker's Delight*. Addison-Wesley.

[79] Zhang, L. and Malik, S. (2002). The Quest for Efficient Boolean Satisfiability Solvers. In *International Conference on Computer Aided Verification*, Springer, pp. 17–36.