Relational Solver Notes

Jules Jacobs

November 6, 2025

1 Roadmap

This document sketches the relational solver that accompanies the Mox type system notes. The goal is to develop a mode solver that is powerful enough to handle the mode constraints produced by the type checker.

At a high level, we will develop a solver for binary constraints between modes. We will first analyze a class of constraints that can be solved in polynomial time. Then we will develop a more powerful solver that can handle that class of constraints.

2 Constraint Language

As a first step, assume we have a finite domain V of values, and a set of binary relations $R_i \subseteq V \times V$. Given a set of variables and asserted contraints between them, we want to determine if there exists a valuation of the variables that satisfies all the constraints.

In general this is a NP-complete problem: consider $V = \{0, 1, 2, \ldots, k\}$ and $R_1 = \{(a, b) \mid a \neq b\} \subseteq V \times V$. Given a graph we can use this constraints to encode the k-coloring problem: we want to assign a color to each vertex such that no two adjacent vertices have the same color. The k-coloring problem is NP-complete, so this problem is also NP-complete.

However, certain classes of constraints can be solved in polynomial time. Consider the set of constraints $R_i = \{(a, b) \mid b \geq a + i\} \subseteq V \times V$. Given a graph of variables and constraints, we can solve this problem in polynomial time using the Floyd-Warshall algorithm.
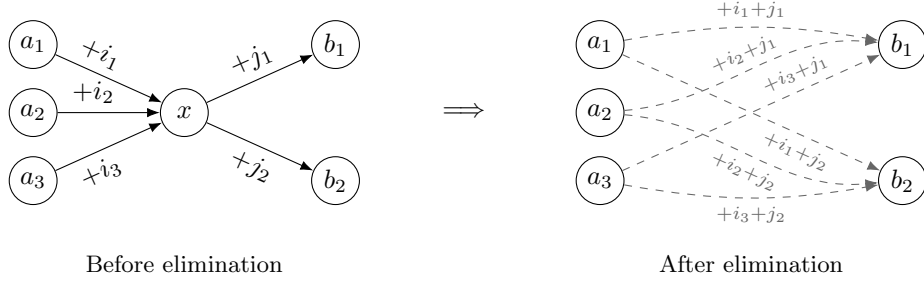
Equivalently, we can solve the problem by variable elimination: we take a variable x and all adjacent constraints on it. We assert all transitive constraints (where R_i composes with R_j to produce R_{i+j}) and repeat until all variables are eliminated. If, during this process, we ever see a constraint between a variable and itself with $i \neq 0$, then the constraints are unsatisfiable.

Why does this elimination strategy work for this class of constraints, but not for the general case, and in particular for the k-coloring problem with inequality constraints?
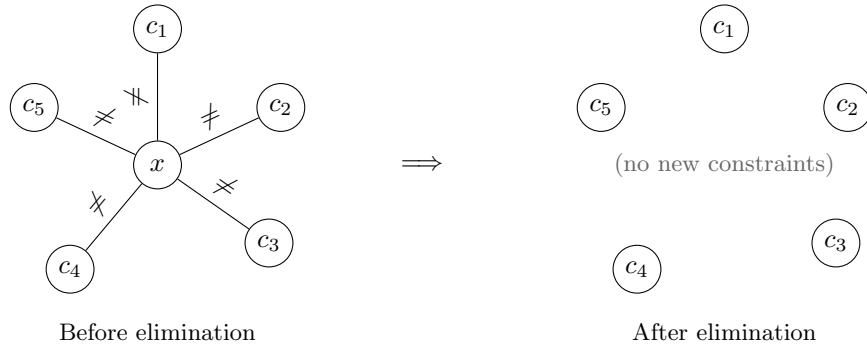
Consider a variable x with:

- a set of predecessors a_1, a_2, a_3 with $R_{i_1}(a_1, x), R_{i_2}(a_2, x), R_{i_3}(a_3, x)$ constraints on them,

- a set of successors b_1, b_2 with $R_{j_1}(x, b_1), R_{j_2}(x, b_2)$ constraints on them,

When eliminating x, we assert every transitive constraint $R_{i_p + j_q}(a_p, b_q)$ for $p \in \{1, 2, 3\}$ and $q \in \{1, 2\}$:

Before elimination $\qquad\qquad$ After elimination

I claim that if there is a solution for the neighboring variables that satisfies all of those transitive constraints, then there is a solution for x that satisfies the original constraints: we can simply set x to any value in the interval $\max(a_1 + i_1, a_2 + i_2, a_3 + i_3) \le x \le \min(b_1 + j_1, b_2 + j_2)$. This interval is guaranteed to be non-empty if the transitive constraints hold.

The key blocker for k-coloring is that this property does not hold for inequality constraints. Suppose for example we have a vertex x and $k+1$ neighbors with inequality constraints. The transitive constraints are trivial if $k \ge 3$, because if we have $a \ne x \ne b$, then for a given value of a, all values of b are still possible, by choosing a particular value for x. Thus, the strategy of variable elimination does not work for k-coloring, for $k \ge 3$: for $\ne$ constraints, eliminating x produces no useful transitives; every neighbour pair remains unconstrained:

Before elimination $\qquad\qquad$ After elimination

The OxCaml mode solver has constraints of the form $x \le G(y)$ where G are modalities with left adjoints. Like the interval constraints we considered above, these constraints can be solved in polynomial time using variable elimination, and for the same reason.

2.1 Which constraints can be solved by elimination?

For a set of constraints to be solvable by elimination, we need that the original problem has a solution iff the transformed problem has a solution. We will now analyze precisely when this property holds, and provide an easily checkable criterion for it.

Let's first analyze the problem more precisely. Given a domain V and a family of relations $\mathsf{Rel} \subseteq \mathcal{P}(V \times V)$, the elimination algorithm may construct new relations in the following four ways:

- It composes two relations $R, S \in \mathsf{Rel}$ to produce

$$RS = \{(a, b) \mid \exists c.\ (a, c) \in R \land (c, b) \in S\}$$

- It intersects two relations $R, S \in \mathsf{Rel}$ when it merges two parallel edges

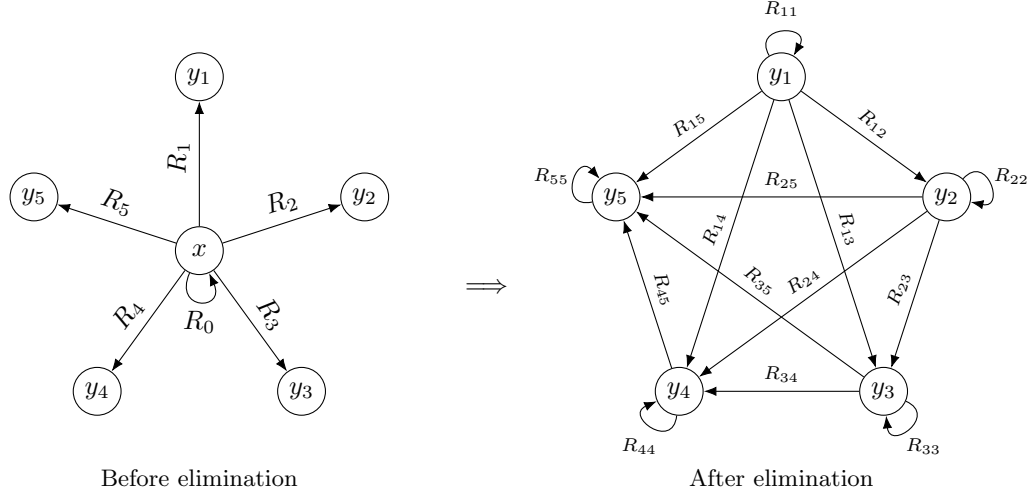$$R \cap S = \{(a, b) \mid (a, b) \in R \land (a, b) \in S\}$$

- It reverses the direction of a relation $R \in \mathsf{Rel}$ to produce

$$R^T = \{(b, a) \mid (a, b) \in R\}$$

- We strengthen a relation $R \in \mathsf{Rel}$ asserted between a variable and itself

$$R^* = \{(a, a) \mid (a, a) \in R\}$$

We therefore first close Rel under these three operations. If the domain V is finite, the resulting set of relations is finite. We can picture the relations before and after elimination of a variable as follows:

Before elimination $\implies$ After elimination

Eliminating x yields edges $R_{ij} = R_i^\top R_0^* R_j$ directed from y_j to y_i for all $i \geq j$, including self-loops R_{ii}. If there were existing relations between the y_i then we intersect them with these.

The key question is: for which families of relations Rel does this algorithm work?

3

Answer. Intuitively, the elimination step is sound if every time we eliminate a variable x, the constraints we generate between its neighbors are "complete": any solution of the projected constraints on the neighbors can be extended to a value of x. The obstruction is exactly the phenomenon we saw with $\neq$-constraints: locally consistent projections need not extend.

We now characterize when elimination is complete.

2.2 Closure and slices

Fix a finite domain V and a finite family of binary relations $\mathsf{Rel} \subseteq \mathcal{P}(V \times V)$. The elimination algorithm only ever uses relations obtained from Rel by:

- composition: $RS = \{(a,c) \mid \exists b.\ (a,b) \in R \wedge (b,c) \in S\}$,

- intersection: $R \cap S$,

- converse: $R^{\top} = \{(b,a) \mid (a,b) \in R\}$,

- restriction to the diagonal (from self-loops): $R^* = \{(a,a) \mid (a,a) \in R\}$.

Let $\mathsf{Cl}(\mathsf{Rel})$ be the smallest set of relations containing Rel and closed under these four operations.

For a relation $R \in \mathsf{Cl}(\mathsf{Rel})$ and an element $b \in V$ we write:

$$R[\_,b] \;=\; \{a \in V \mid (a,b) \in R\} \qquad \text{and} \qquad R[a,\_] \;=\; \{b \in V \mid (a,b) \in R\}$$

for its *column* and *row* slices. These are exactly the unary constraints on x induced by fixing the other endpoint.

Let $\mathcal{F}$ be the family of all such slices:

$$\mathcal{F} \;=\; \{\, R[\_,b], R[a,\_] \mid R \in \mathsf{Cl}(\mathsf{Rel}),\ a,b \in V \,\} \;\subseteq\; \mathcal{P}(V).$$

Because $\mathsf{Cl}(\mathsf{Rel})$ is closed under intersection, so is $\mathcal{F}$: for any $A, B \in \mathcal{F}$ there exists $C \in \mathcal{F}$ with $C = A \cap B$. This intersection-closure is crucial below.

2.3 A Helly-style condition

The key combinatorial property is a Helly-type condition on $\mathcal{F}$.

Definition 1 (Helly-2 for slices)**.** *We say that* Rel *has the* Helly-2 slice property *if the following holds:*

For every finite subfamily $\{F_1, \ldots, F_k\} \subseteq \mathcal{F}$, *if all pairwise intersections are non-empty,*
$$F_i \cap F_j \neq \varnothing \quad \text{for all } i \neq j,$$
then the total intersection is non-empty,

$$\bigcap_{i=1}^{k} F_i \neq \varnothing.$$

Because $\mathcal{F}$ is intersection-closed, this is equivalent to the absence of a *bad triple*:

Lemma 2 (Triples suffice for intersection-closed families). *Assume $\mathcal{F}$ is closed under intersection. Then $\mathcal{F}$ has the Helly-2 slice property if and only if there do not exist $A, B, C \in \mathcal{F}$ such that*

$$A \cap B \neq \varnothing, \quad B \cap C \neq \varnothing, \quad C \cap A \neq \varnothing, \quad A \cap B \cap C = \varnothing.$$

Proof. ($\Rightarrow$) Immediate: such a triple would violate Helly-2.

($\Leftarrow$) Suppose Helly-2 fails. Pick a counterexample $\{F_1, \ldots, F_m\} \subseteq \mathcal{F}$ of minimal size: all pairs intersect, but $\bigcap_i F_i = \varnothing$. Since $\mathcal{F}$ is intersection-closed, for each $i \geq 2$ the set $A_i := F_1 \cap F_i$ lies in $\mathcal{F}$, is non-empty, and $\bigcap_{i=2}^m A_i = \bigcap_{i=1}^m F_i = \varnothing$. If some $A_i \cap A_j$ were empty, then F_1, F_i, F_j would form a bad triple. Otherwise $\{A_2, \ldots, A_m\}$ is a smaller counterexample, contradicting minimality. $\square$

Thus in our setting we can detect failure of the Helly-2 slice property by a finite search for such bad triples among slices in $\mathsf{Cl}(\mathsf{Rel})$.

2.4 Correctness of elimination

We now state the main correctness criterion.

Theorem 3 (Characterization of admissible constraint families). *Let V be finite and $\mathsf{Rel} \subseteq \mathcal{P}(V \times V)$. Consider the following variable-elimination algorithm: at each step, pick a variable x, and for every two neighbors $y_i \xleftarrow{R_i} x \xrightarrow{R_j} y_j$ insert the composed edge $y_i \xleftarrow{R_i R_0^* R_j} y_j$ (where R_0^* is the current self-loop on x, possibly* Id*), intersecting with any existing edge between y_i and y_j, then delete x. If at any point a self-loop on some variable becomes empty on the diagonal (i.e. forbids all (a, a)), report unsatisfiable.*

Then the following are equivalent:

1. *For every finite constraint graph over* Rel, *this elimination algorithm decides satisfiability correctly: the final instance is satisfiable iff the original instance is satisfiable.*

2. Rel *has the Helly-2 slice property.*

Proof sketch. (2) $\Rightarrow$ (1) (*completeness of elimination*). It is enough to show that a *single* elimination step is complete, and then argue by induction on the number of variables.

Consider eliminating x from its star. Fix an assignment to the neighbors $\{y_i\}$ that satisfies all constraints generated by the algorithm between the y_i. For each incident edge, this assignment induces a slice $S_i \in \mathcal{F}$ of admissible values for x (e.g., for $y_i \xrightarrow{R_i} x$ with y_i fixed to b_i we get $S_i = R_i[\_, b_i]$). By construction, every binary constraint added between y_i and y_j enforces that S_i and S_j intersect:

if they did not, the corresponding composed relation would have removed the chosen pair (b_i, b_j). Hence the family $\{S_i\}$ is pairwise-intersecting. By the Helly-2 slice property, $\bigcap_i S_i \neq \varnothing$. Picking any a in this intersection and setting $x := a$ extends the neighbor assignment to a full solution of the original constraints around x. Inductively, every model of the final instance lifts to a model of the original one.

Soundness of the algorithm (it never introduces spurious solutions) holds because every new edge relation is logically implied by the existence of an intermediate x satisfying its incident constraints.

$(1) \Rightarrow (2)$ (*necessity*). Assume the Helly-2 slice property fails. By Lemma 2 and intersection-closure, there exist $S_1, S_2, S_3 \in \mathcal{F}$ with all pairwise intersections non-empty but empty triple intersection. By definition of $\mathcal{F}$, we can realize each S_i as the admissible values for some variable x given a fixed neighbor y_i and a relation $R_i \in \mathsf{Cl}(\mathsf{Rel})$. We construct a constraint star with center x and leaves y_1, y_2, y_3 so that, for the chosen values of the y_i, the corresponding slices are precisely S_i. By pairwise intersection, every pair of leaves admits a value of x; hence the elimination algorithm, which only asserts pairwise compositions, accepts this partial assignment to y_1, y_2, y_3. But by emptiness of $S_1 \cap S_2 \cap S_3$, no value of x satisfies all three constraints simultaneously, so the original instance is unsatisfiable. Thus elimination is not complete, contradicting (1). $\qquad\square$

Corollary 4 (Finite check). *For finite V, the Helly-2 slice property for* Rel *is decidable:*

1. *compute* $\mathsf{Cl}(\mathsf{Rel})$ *(finite);*

2. *form all slices* $R[\_, b], R[a, \_]$;

3. *check that no triple of slices has non-empty pairwise intersections and empty triple intersection.*

If no such triple exists, variable elimination as above is sound and complete for all constraint graphs over Rel.

In particular, difference constraints and the OxCaml-style modal constraints $x \leq G(y)$ with adjoints fall into this framework: their induced slices are (discrete) intervals, which satisfy Helly-2, so elimination is complete. By contrast, $\neq$-constraints generate non-convex slices, violate Helly-2, and are correctly rejected by this criterion.

3 Solver Architecture

4 Open Questions