Proving properties of a minimal covering algorithm

Ioannis Filippidis and Richard M. Murray

Control and Dynamical Systems
California Institute of Technology *

May 26, 2018

1 Overview

This work concerns the specification and proof using TLA$^+$ of properties of an algorithm for solving the minimal covering problem [1], which we have implemented in Python [3]. Minimal covering is the problem of choosing a minimal number of elements from a given set Y to "cover" all the elements from another given set X, as defined by a given function f. An element $y \in Y$ covers an element in $x \in X$ if $f[x, y]$, as illustrated in Fig. 1.

A more detailed description of the algorithm and its properties [2, Chapter 7], and the relevant TLA$^+$ modules [3] are available. The proofs of safety properties in these modules have been checked with the proof assistant TLAPS (version 1.4.3) [4], in the presence of the tools Zenon, CVC3, Isabelle, and LS4. We have been motivated to study and implement this algorithm for converting binary decision diagrams to formulas of small size, so that humans can read the results of symbolic computations [2].

2 Minimal covering algorithm

This algorithm was originally proposed for computing small implementations of circuits [1], and requires that the covering problem be defined within a complete lattice, i.e., that a function $Leq \in [Z \times Z \to \text{BOOLEAN}\ ]$ defines a complete lattice, and the sets $X, Y \in \text{SUBSET}\ Z$, with Leq used as the function f. If the given covering problem is not defined within a lattice, then it is first mapped to one within a suitable lattice, as Fig. 2 illustrates for the problem shown in Fig. 1. We refer to a cover of the set X that comprises of elements from the set Y as a cover "from Y".

The algorithm operates by repeating three steps:

1. Computing essential and ambiguous elements (cyclic core);
2. branching by picking an ambiguous element as candidate; and
3. pruning subtrees of the search tree created by branching, based on upper and lower bounds.

*{ifilippi,murray}@caltech.edu

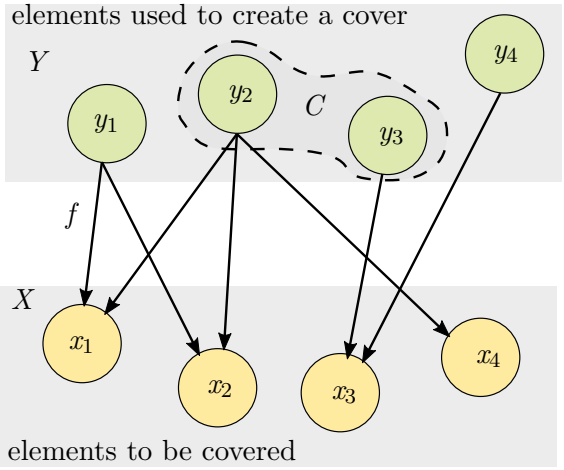


Figure 1: Example of a minimal covering problem, defined by the sets X, Y, and the function $f \in [X \times Y \to \text{BOOLEAN}]$. A set $C \in \text{SUBSET } Y$ is a cover of X if for each element $x \in X$, there exists a $y \in C$ such that $f[x, y]$. A cover C is minimal if no other cover from Y has cardinality smaller than C.

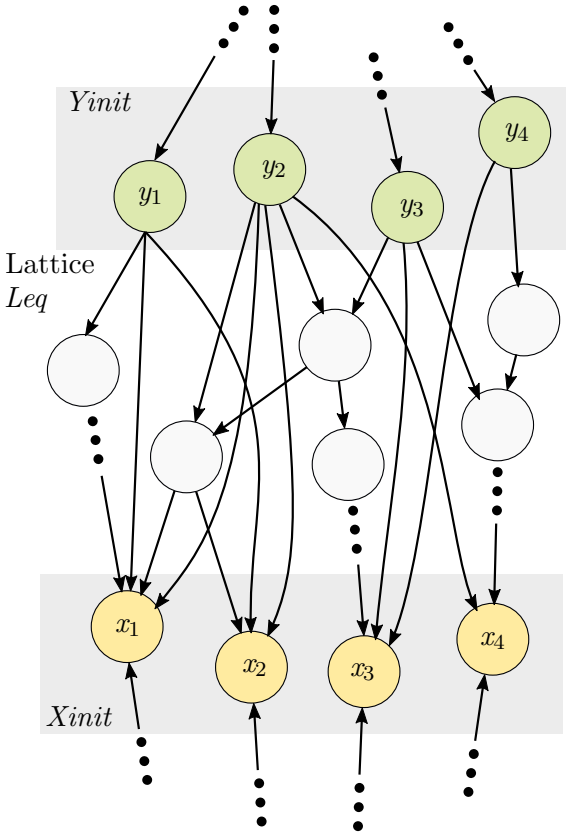


Figure 2: The covering problem of Fig. 1 in the presence of a (complete) lattice *Leq*, with *Xinit*, *Yinit* $\in Z$, and DOMAIN *Leq* $= Z \times Z$.

These steps are illustrated in Fig. 4. Given a covering problem, first an iterative computation finds which elements of Y are necessary (called essential) to cover X. This computation transforms the covering problem, with the resulting covering problem X_c, Y_c called the cyclic core. After finding the cyclic core, the computation proceeds by selecting an element $y_b \in Y_c$, and creating two covering problems, depending on whether y_b is included as part of the cover under construction or not.

We formalized the computation of essential elements and cyclic core (step 1), and checked the proofs of safety properties using TLAPS. These proofs are based on [1]. The cyclic core computation is described by the algorithm in Fig. 5 and illustrated in Fig. 3. Each step transforms the covering problem X, Y to a new covering problem. The steps are repeated until X and Y stop changing. Each step has the following effect:

- The step $X_1 = \mathit{Ceilings}(X, Y, \mathit{Leq})$ creates equivalence classes of the set X. For example, two elements $u_1, u_2 \in X$ belong to the same equivalence class if they are below the same set of elements from Y, i.e., if $\mathit{ThoseOver}(Y, u_1, \mathit{Leq}) = \mathit{ThoseOver}(Y, u_2, \mathit{Leq})$. More precisely, this transformation maps each $u \in X$ to $\mathit{Ceil}(u, Y, \mathit{Leq})$, which represents the equivalence class of u.

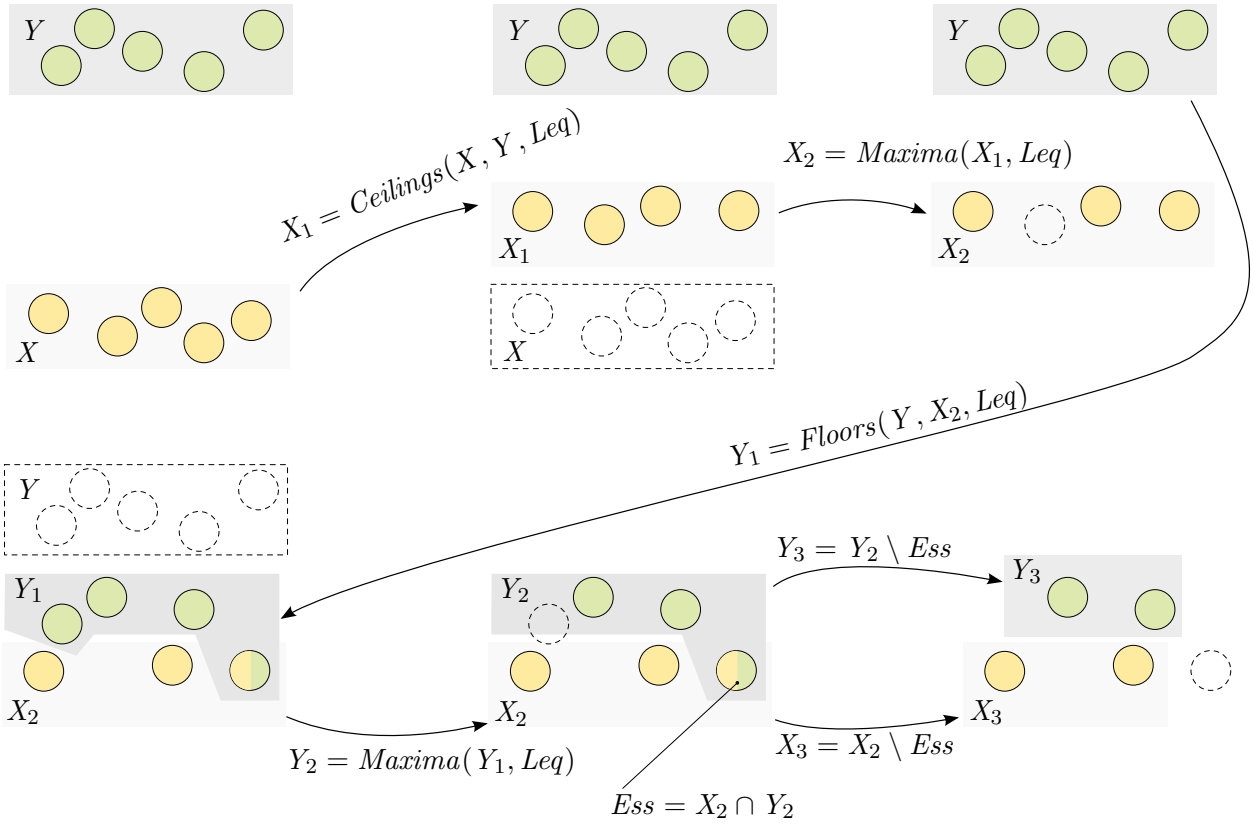


Figure 3: Transformations of minimal covering problem during an iteration of the fixpoint computation that computes the cyclic core, described by the algorithm of Fig. 5.

- The step $X_2 = Maxima(X_1, Leq)$ keeps those $u \in X_1$ that suffice to be covered.
- The step $Y_1 = Floors(Y, X_2, Leq)$ creates equivalence classes of the set Y. For example, two elements $y_1, y_2 \in Y$ belong to the same equivalence class if they are above the same set of elements from X_2, i.e., if $ThoseUnder(X_2, v_1, Leq) = ThoseUnder(X_2, v_2, Leq)$. More precisely, this transformation maps each $v \in Y$ to $Floor(v, X_2, Leq)$, which represents the equivalence class of v.
- The step $Y_2 = Maxima(Y_1, Leq)$ keeps those $v \in Y_1$ that suffice to cover X_2.
- The step $Ess = X_2 \cap Y_2$ removes from Y_2 elements that are necessary for covering X_2, in that each covers some element in X_2 uncovered by any other $y \in Y_2$. These elements are removed also from X_2.

The operators mentioned above are defined in the modules *Lattices* and *Optimization*.

The specification and proof in the module *CyclicCore* contain only the body of the loop of the algorithm in Fig. 5. Termination is proved by showing that X, Y eventually stop changing. In the presence of a loop condition as in Fig. 3, after X, Y stop changing, the procedure returns. See also [2, pp. 66–70].

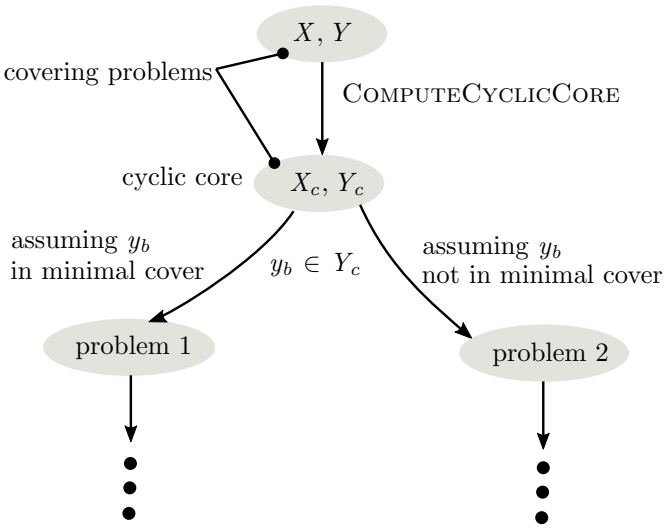


Figure 4: The covering algorithm transforms a minimal covering problem X, Y using a fixpoint computation that removes essential elements, and identifies elements that cover the same subset of X. If the resulting covering problem X_c, Y_c (cyclic core), is nonempty, then branching leads to a search over smaller problems.

def ComputeCyclicCore(*Xinit*, *Yinit*, *Leq*) :
  $X, Y := Xinit, Yinit$
  $Xold, Yold, E := \{\}, \{\}, \{\}$
  while $\langle X, Y \rangle \neq \langle Xold, Yold \rangle$:
    $Xold, Yold := X, Y$
    $Y := MaxFloors(Y, X, Leq)$
    $X := MaxCeilings(X, Y, Leq)$
    $Essential := X \cap Y$
    $X := X \setminus Essential$
    $Y := Y \setminus Essential$
    $E := E \cup Essential$
  return X, Y, E

Figure 5: The algorithm that computes the cyclic core, described in a syntax that is similar to PlusCal, with syntactic elements from Python.

3 TLA$^+$ modules

The theorems and proofs are organized into modules of general interest and modules specific to the cyclic core computation, similarly to the specification of Naiad [5]. The modules *FiniteSetFacts*, *Optimization*, and *MinCover* contain general results that are independent of the problem we study. The modules *Lattices*, *CyclicCore*, and *StrongReduction* are progressively more specific, with the last two specifying the cyclic core and cover reconstruction algorithms. The proofs and dependency of theorems in the module *Lattices* are deeper than those in the modules *CyclicCore* and *StrongReduction*, reflecting differences in their content [6, §5].

The module contents are as follows:

Table 1: TLA$^+$ modules. The checking time is using TLAPS with CVC3, Zenon, LS4, and Isabelle.

Module	Content	TLAPS time	Obligations
FiniteSetFacts	Addendum to *FiniteSetTheorems*	5.8 s	27
Optimization	Min/maxum/al elements, Antichains	31 s	311
MinCover	Def of minimal covers, and their properties	30 s	237
Lattices	Lattices, Floor, Ceiling, Essential elements	5 min	1334
CyclicCore	Spec of cyclic core fixpoint, safety properties	8 min	1561
StrongReduction	Spec of all mincovers below, correctness proofs	45 min	3038

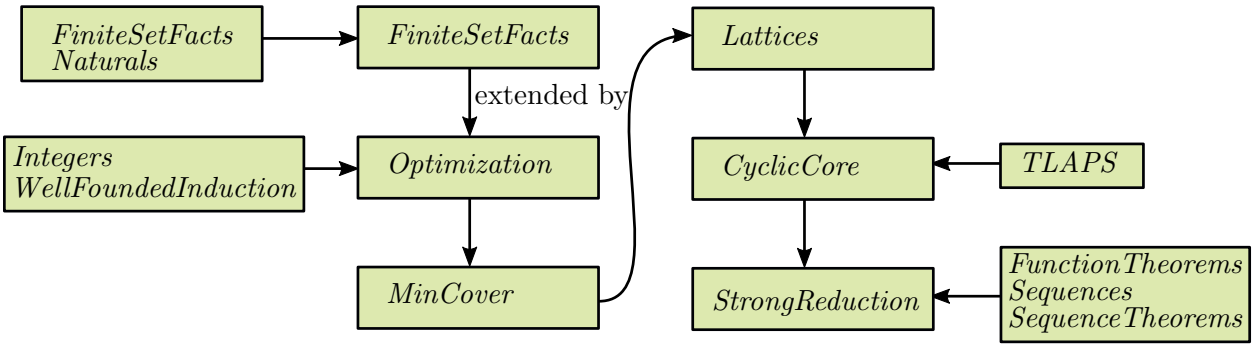


Figure 6: Module dependency relation.

- The module *FiniteSetFacts* contains results about finite sets (extending the TLAPS module *FiniteSetTheorems*).
- The module *Optimization* formalizes notions of minimum, minimal, maximum, and maximal elements, binary relations as functions, antichains, and their properties.
- The module *MinCover* defines what a minimal cover is, and includes theorems about minimal covers, with emphasis on the finite case.
- The module *Lattices* defines operations relevant to minimal covering in a lattice, and theorems about them. These operations are performed repeatedly during the cyclic core computation, until a fixpoint is reached. The *Lattices* module proves that transformations due to these operations allow us to reconstruct minimal solutions of the input problem, and that for finite sets as input, every iteration decreases the problem size or reaches a fixpoint. These results are used in the modules *CyclicCore* and *StrongReduction* to prove correctness of the algorithm.
- The module *CyclicCore* specifies the cyclic core fixpoint algorithm, and proves relevant safety properties (checked by TLAPS), and its termination (checked by human).
- The module *StrongReduction* specifies an algorithm that we designed for constructing all the minimal covers of the input problem from the minimal covers of the cyclic core, and proves its safety properties of interest (checked by TLAPS).

Checking these proofs with TLAPS takes about an hour (for 6508 obligations) on a 3.1 GHz CPU, as shown in Table 1. Fig. 6 shows how the above modules depend on each other, on standard modules, and on modules from the TLAPS library.

4 An algorithm for enumerating all minimal covers

The cyclic core and branch-and-bound computations transform the covering problem, until an empty problem is obtained. The path of transformations determines the cardinality of the minimal cover to the input problem *Xinit*, *Yinit*. The branch-and-bound search terminates when the path is known to yield a minimal cover, as detected using the lower and upper bounds. A minimal cover for the input problem can be constructed by taking the union of essential elements along the path, and finding for each one an element from *Yinit* that is above it, as described in [1]. In addition to

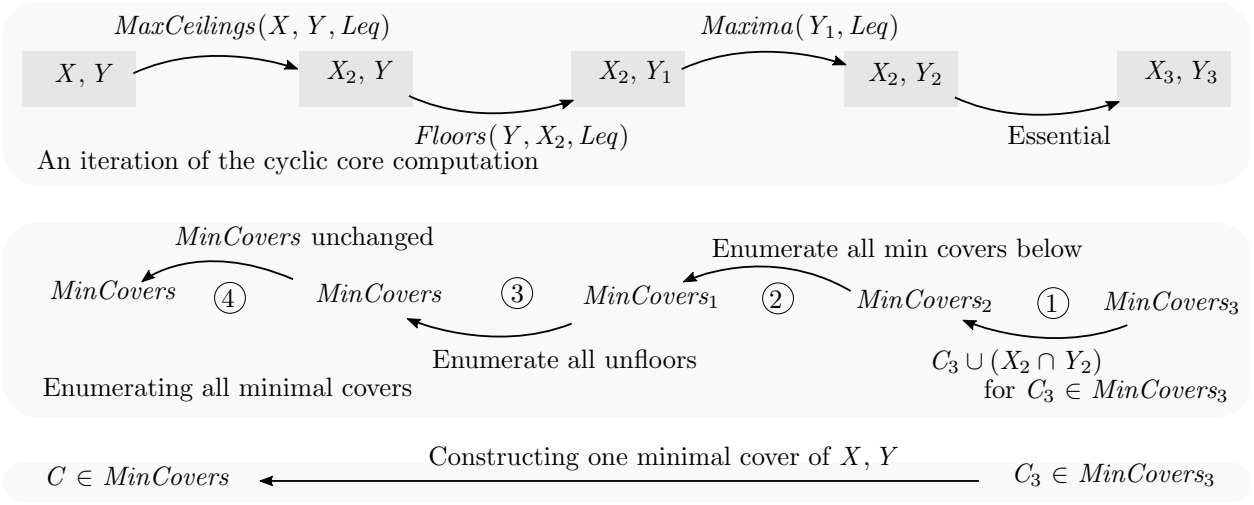


Figure 7: Obtaining solutions of the problem X, Y from solutions of the problem X_3, Y_3 at the end of an iteration. A single minimal cover C_3 of X_3 from Y_3 can be transformed in one step to a minimal cover of X from Y. The set of minimal covers of X_3 from Y_3 can be transformed to the set of minimal covers of X from Y in multiple steps.

computing one solution, the set of all minimal covers may also be of interest. Fig. 7 illustrates a single iteration of the cyclic core computation, the enumeration of all minimal covers of the problem at the start of that iteration, and the construction of one minimal cover, which is simpler.

While developing this TLA$^+$ proof, we noticed that the original cyclic core algorithm yields some minimal covers, but not necessarily the entire set of minimal covers. Some minimal covers can be lost with that approach, as witnessed by a counterexample shown in Fig. 16. We modified the original algorithm to produce all the minimal covers of the input problem. The algorithm specification and proof are in the module *StrongReduction*, and the algorithm is described in Fig. 17. So formalization helped locate a bug in the original algorithm, and prepared us for amending it.

Fig. 13 illustrates the operation of the algorithm. This algorithm corresponds to step No. 2 in Fig. 7. It enumerates all minimal covers from the set Y that refine a minimal cover from the set $Maxima(Y, Leq)$ (where $Refines(A, B, Leq) \triangleq \forall u \in A : \exists v \in B : Leq[u, v]$). This enumeration is performed starting with a minimal cover $C_m \in \text{SUBSET } Maxima(Y, Leq)$, and iteratively replacing each element with an element from Y that is below it and also above all elements of X uniquely covered by the element that is replaced, as illustrated in Figs. 12 and 13.

4.1 Proof structure

The proof of safety properties of the enumeration algorithm is structured as follows. Given the set of minimal covers from $Max \triangleq Maxima(Y, Leq)$, we want to find the set of minimal covers from Y. Each minimal cover C from Y refines some minimal cover C_m from Max, as illustrated in Fig. 8. Therefore, it suffices to find, for each C_m, those minimal covers from Y that refine C_m (Fig. 9), i.e., the set

$$AllMinCoversBelow(C_m, X, Y, Leq) \triangleq \{C \in \text{SUBSET } Y : \wedge IsAMinCover(C, X, Y, Leq) \\ \wedge Refines(C, C_m, Leq)\}$$

The proof has two directions (Fig. 10):

1. Soundness: Given a minimal cover C_m from Max, enumeration yields only minimal covers from Y.

2. Completeness: No minimal covers from Y that refine the cover C_m are missing from the set *MinCoversBelow* returned by the algorithm of Fig. 17.

Given a minimal cover $C_m \in \text{SUBSET } Maxima(Y, Leq)$, the enumeration algorithm successively replaces each element y_{max} with an element $y_k \in BelowAndSuff(y_{max}, Q, Y)$ (Line 3347 in the module *StrongReduction*). The elements y_k are those that cover the elements x that only y_{max} covers (Figs. 11 and 12).

4.1.1 Soundness proof

Successive replacements are illustrated in Fig. 13. Let $Q \triangleq Partial \cup Patch(k)$, from the definition of the invariant *PartialCoversInStack* (*Partial* corresponds to *PartialCover* in Fig. 17). The invariant *PartialCoversInStack* implies the following two invariance properties, which ensure that each *PartialCover* augmented with the set $Patch(k)$ (equal to $Image(Lm, k..N)$) is a minimal cover:

1. Cover: If Q is a cover, then elementwise replacement (of y_{max} by y_k) yields a *QNext* that is a cover (Line 3535 in the module *StrongReduction*).

2. Minimality: Elementwise replacement preserves cardinality, so $Cardinality(QNext) = N$ (Line 3607 in the module *StrongReduction*).

Thus, each Q is a minimal cover from Y. So any *PartialCover* added to *MinCoversBelow* is a minimal cover (these additions occur in steps that satisfy the action *Collect*).

The assertion that the intermediate sets Q are covers (Line 3535 in the module *StrongReduction*) is proved by considering two cases:

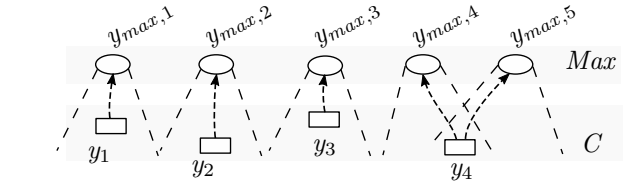
1. y_k covers those x that are covered by only y_{max} (i.e., the set $Only(y_{max}, Q)$) (Line 3578 in the module *StrongReduction*).

2. Elements $x \notin Only(y_{max}, Q)$ are covered by elements in $Q \setminus \{y_{max}\}$, which is a subset of *QNext* (Line 3540 in the module *StrongReduction*).

The theorem *StackContainsPartialCovers* asserts the above properties. This theorem is used to prove the theorem *StrongReductionSoundness*.

4.1.2 Completeness proof

Completeness is established by proving that there are no minimal covers that refine C_m but are not returned by the enumeration algorithm (such covers would correspond to the area with the question mark in Fig. 10).



Each $y_i \in C$ can be mapped to some $y_{max,j} \in Max \triangleq Maxima(Y, Leq)$, with $Leq[y_i, y_{max,j}]$, thus each cover C to some cover of maximal elements.

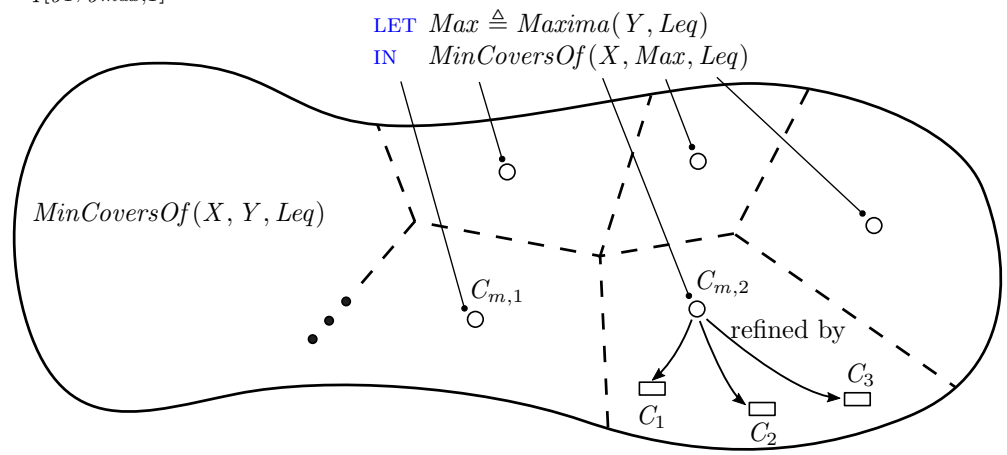


Figure 8: Each minimal cover containing elements from Y refines some minimal cover comprised of elements from $Maxima(Y, Leq)$. So the set of minimal covers of X from Y can be partitioned into subsets, each subset containing covers that refine the same minimal cover from $Maxima(Y, Leq)$. The operator *MinCoversOf* is defined in the module *StrongReduction*.

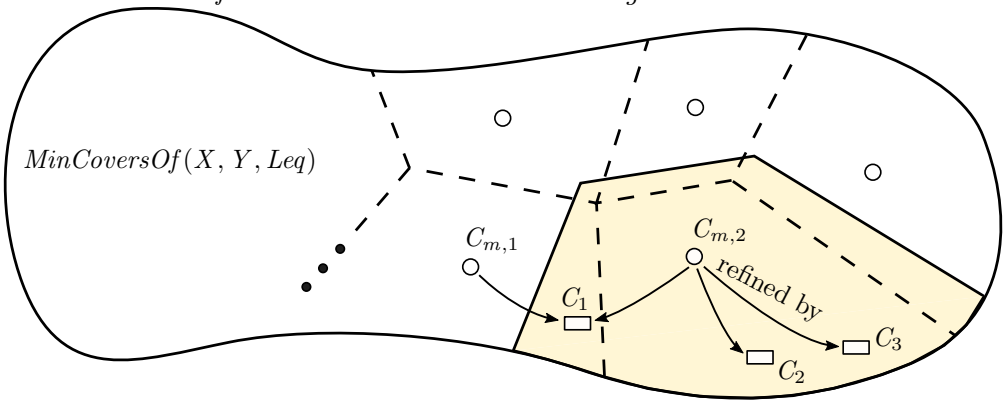


Figure 9: A cover C may refine two covers $C_{m,1}, C_{m,2}$ from $Max \triangleq Maxima(Y, Leq)$. The algorithm described in the module *StrongReduction* enumerates all the minimal covers from Y that refine a given minimal cover C_m from Max. Applying this algorithm to all minimal covers from Max, the resulting set contains all minimal covers from Y, as asserted by the theorem *MinCoversSubseteqUnionCandidatesBelow*.

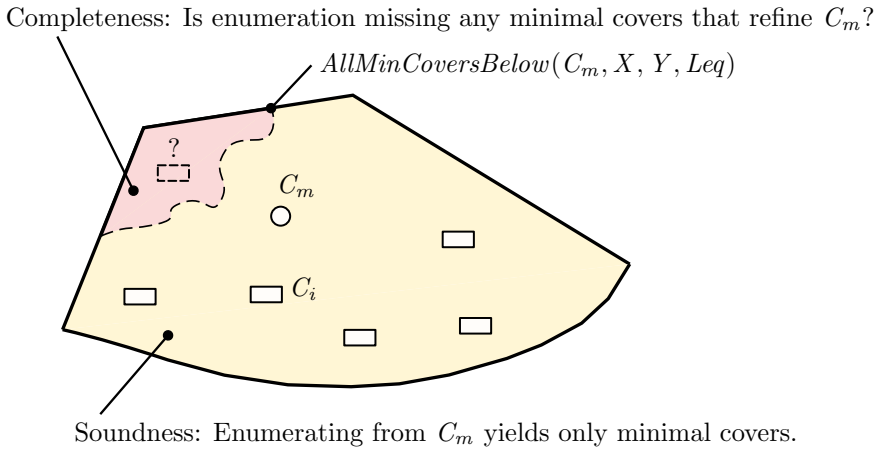


Figure 10: A single run of the enumeration algorithm generates all the minimal covers C_i that refine a given minimal cover $C_m \in Maxima(Y, Leq)$. The theorem *StrongReductionSoundness* asserts that the algorithms enumerates only minimal covers. The theorem *StrongReductionCompleteness* asserts that no minimal covers that refine C_m are missed by the algorithm. These two theorems are combined in the theorem *StrongReductionSafety*. The module *StrongReduction* contains these theorems.

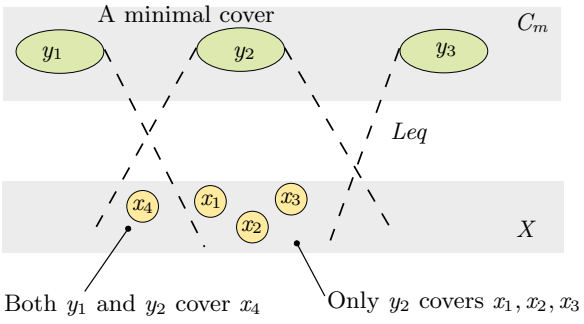


Figure 11: A minimal cover contains only essential elements, as established by the theorem *MinimalHasAllEssential*. Each essential element $y_i \in C_m$ covers an element $x_j \in X$ that is uncovered by other $y_j \in C_m \setminus \{y_i\}$.

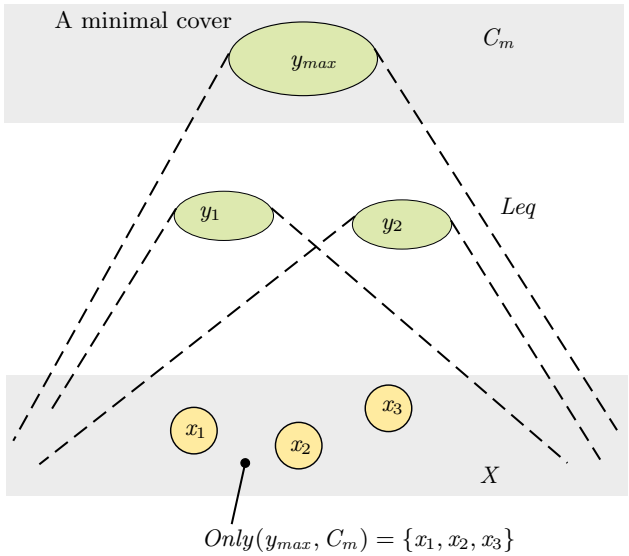


Figure 12: The set $BelowAndSuff(y_{max}, C_m, Y)$ contains those $y \in Y$ that are below y_{max} (i.e., $Leq[y, y_{max}]$), and that cover all x that within C_m are uniquely covered by y_{max}.

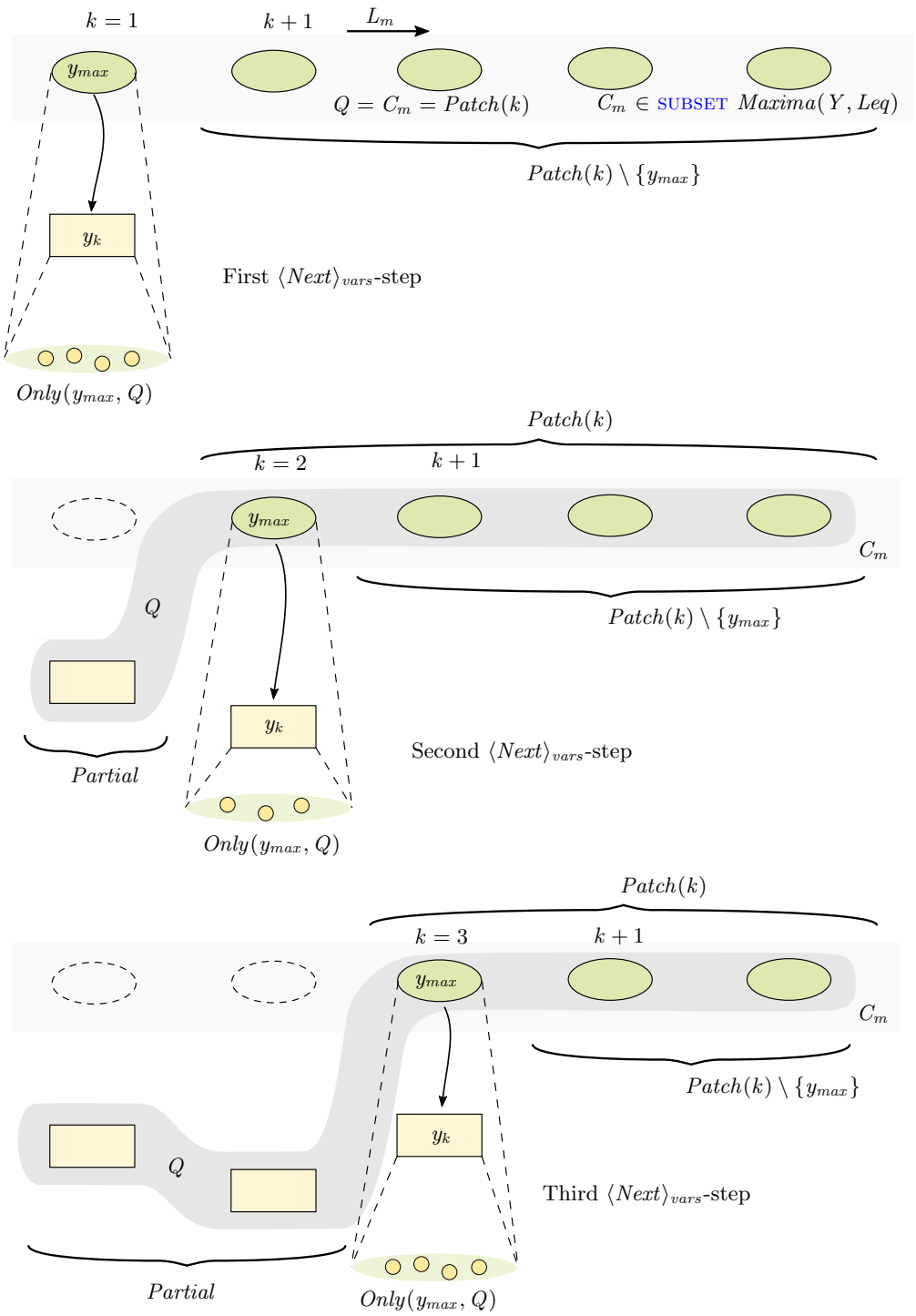


Figure 13: Given a minimal cover $C_m \in$ SUBSET $Maxima(Y, Leq)$, the enumeration algorithm successively replaces each element y_{max} with an element $y_k \in BelowAndSuff(y_{max}, Q, Y)$. Each replacement $y_k \in BelowAndSuff(y_{max}, Q, Y)$ yields a different partial cover $Partial$, thus leading to different minimal covers from Y.

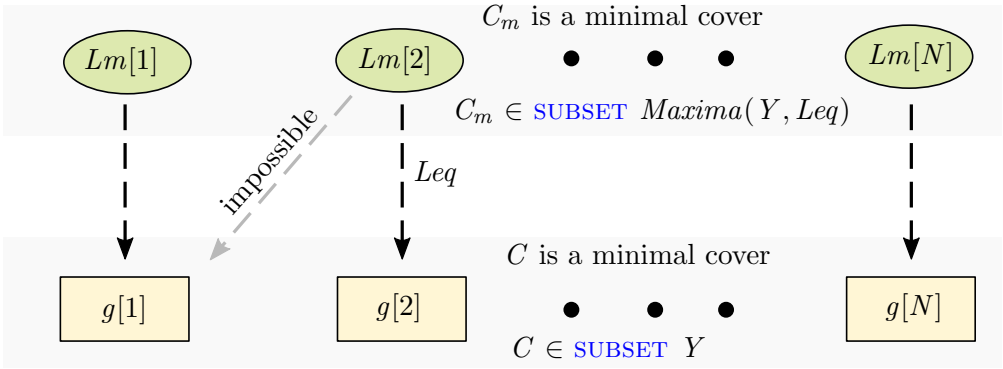


Figure 14: The lattice function *Leq* induces a bijection between a minimal cover $C_m \in$ SUBSET $Maxima(Y, Leq)$, and a minimal cover $C \in$ SUBSET Y that refines C_m, i.e., $C \in AllMinCoversBelow(C_m, X, Y, Leq)$. Each element of C_m covers exactly one element of C, so $\neg Leq[g[i], Lm[j]$ for $i \neq j$ and $i, j \in 1..N$.

Establishing a bijection The proof is by contradiction, assuming that there exists a minimal cover C that refines C_m. For any such cover C, there exists a bijection between C and C_m, as asserted by the theorem *MinCoverRefinementInducesBijection* in the module *StrongReduction*, which is illustrated in Fig. 14. The proof of this theorem depends on three other theorems:

- The theorem *AtMostOneBelow*: A $ym \in C_m$ cannot cover more than one $y \in C$. Otherwise, $N - 1$ elements from C_m would suffice to cover C, because by minimality $Cardinality(C) = Cardinality(C_m)$ and $Cardinality(C_m) = N$. C is a cover, so $N - 1$ elements would suffice to cover X, contradicting that C_m is minimal.
- The theorem *AtMostOneAbove*: A $y \in C$ cannot be below more than one $y_m \in C_m$. If so, then by the theorem *AtMostOneBelow*, the two elements of C_m that cover y would not cover any other elements of C, and one of them would suffice to cover y. The cover C refines C_m, so $N - 1$ elements of C_m would suffice to cover C, and thus X, contradicting that C_m is minimal.
- The theorem *MinCoverRefinementHasBelow*: If the cover C refines a minimal cover C_m (with respect to *Leq*), then each $y_m \in C_m$ is above some $y \in C$.

Proving these theorems relies on finiteness of the sets C and C_m.

Completeness by contradiction The theorem *StrongReductionCompleteness* uses the bijection between C and C_m to show that if any element $y_{max} \in C_m$ is replaced by an element $y \in (Y \setminus BelowAndSuff(y_{max}, Q, Y))$, then some $x \in Only(y_{max}, Q)$ remains uncovered, so C cannot be a cover. This is proved in two steps:

1. Any element $yc \in Partial$ is in $Q \setminus \{y_{max}\}$, so yc cannot cover x, by the definition of *Only* (Line 2502 in the module *StrongReduction*).
2. Any element $yc \in After \triangleq Image(g, (k+1)..N)$ is below some $yt \in Patch(k+1)$, by the bijection between C and C_m. Thus, if yc covered x, then yt would cover x (by transitivity of *Leq*), but $yt \in Q \setminus \{y_{max}\}$, a contradiction (Line 2562 in the module *StrongReduction*).

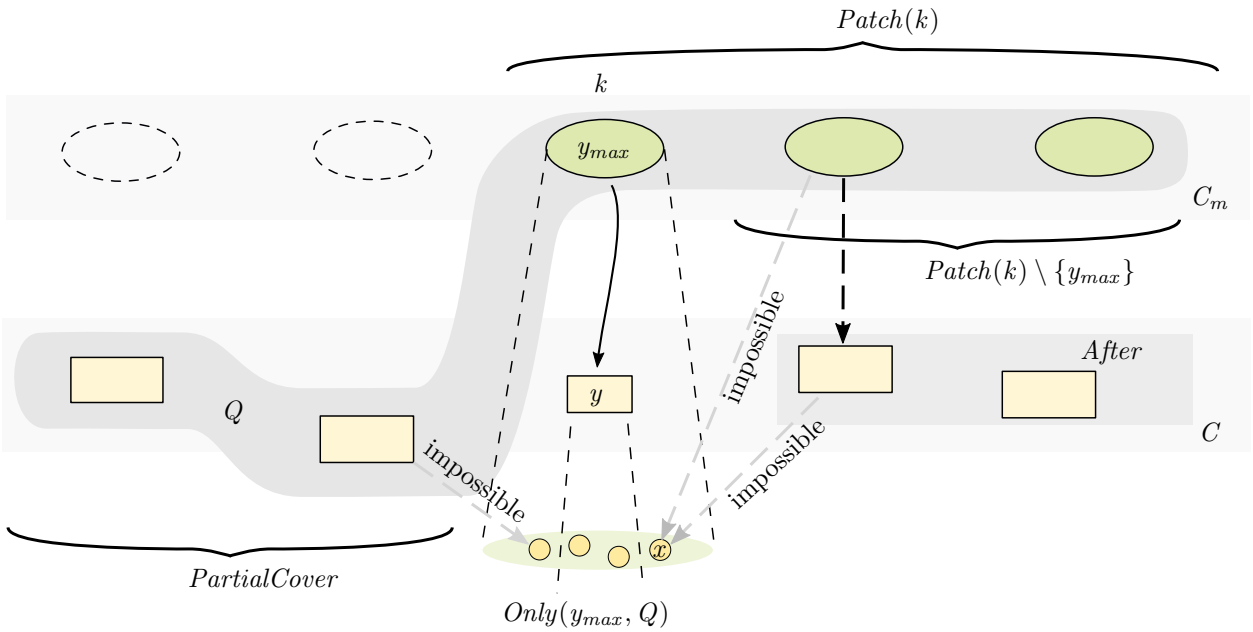


Figure 15: Completeness is proved by assuming that a minimal cover $C \in \text{SUBSET } Y$ refines C_m, and that C is not found by the enumeration algorithm. Thus, some replacement step introduces a $y \in C$ that leaves uncovered some x that cannot be covered by any elements from $C \setminus \{y\}$, contradicting the assumption that C is a cover.

This reasoning is illustrated in Fig. 15.

4.2 Stack-based and set-based variants

The module *CyclicCore* specifies and proves properties of the cover reconstruction algorithm of Fig. 17, which uses a stack. This choice has little effect on the algorithm's size, but makes the proof more complicated, due to the presence of sequences (order). We observed this consequence while developing the proof, and decided to complete the stack-based proof, so that we can compare with a set-based specification and proof, which we plan to develop. A variant of the algorithm that uses a set instead of a stack is listed in Fig. 18.

5 Practical observations

During development we collected observations about practical details and utility features that we would find useful if available. For example, declaring a constant in a module and stating assumptions about its properties in other modules, instead of using instantiation and substitution by a given expression, reduces expression nesting. Automated reporting of the maximal step number used in a proof level would aid writing of a proof before renumbering. Theorem names as directives for what lines to prove would also be convenient when directly calling the proof manager. More extensive memoization of proof results would help accelerate development.

In the context of this work we developed scripts for common tasks, including creating "header" files by removing proofs, typesetting a collection of TLA$^+$ modules as a single document, and a

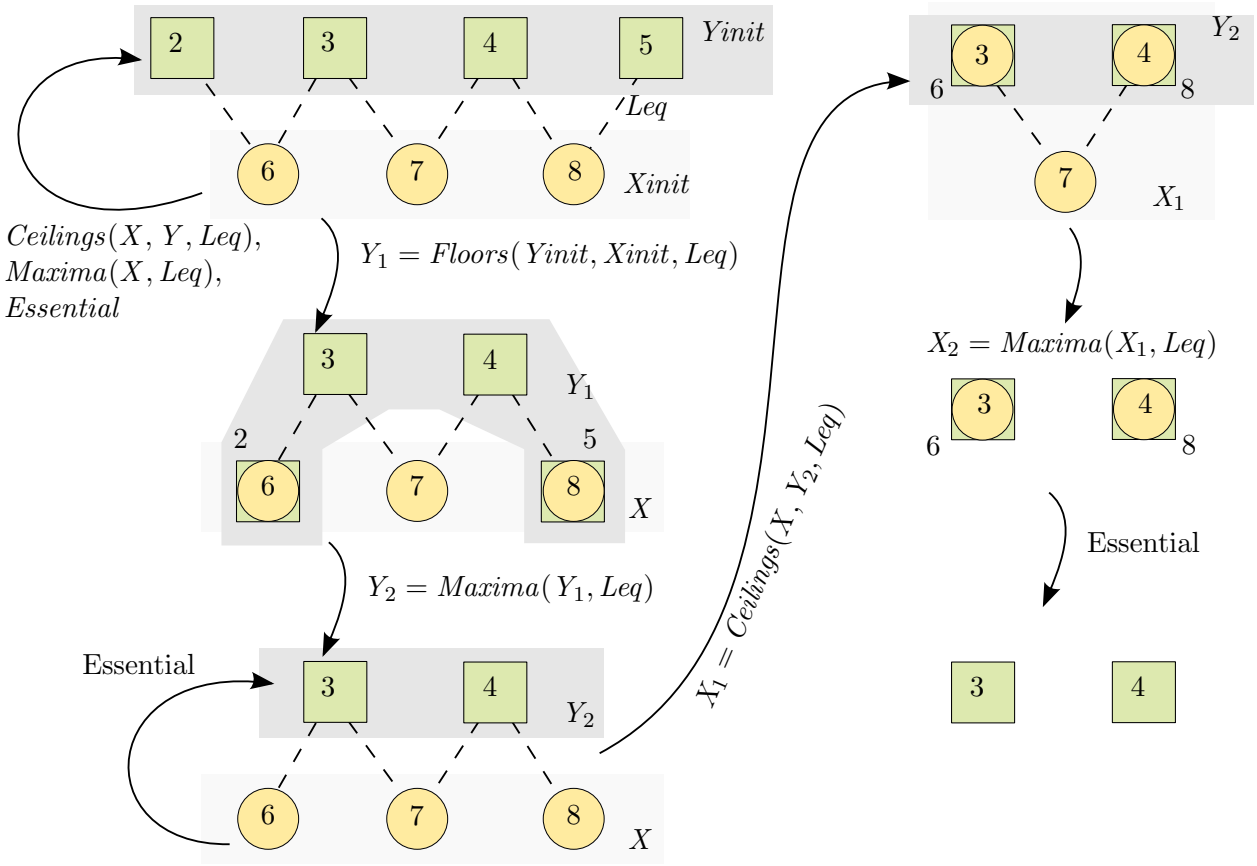


Figure 16: An example of computing a minimal cover. One iteration of transformations suffices to reach a fixpoint, and the cyclic core is empty ($X_3 = \{\}$ and $Y_3 = \{\}$ in Fig. 3). Lattice elements above and below *Xinit*, *Yinit* are not shown. See also [1, Figs. 13–15 on pp. 119–120] and [2, Figs. 7.13–7.14 on p. 79].

```
def EnumerateMincoversBelow(Cm) :
  stack := ⟨{}⟩
  Lm := Enumerate(Cm)
  N := Cardinality(Cm)
  MinCoversBelow := {}
  while stack ≠ ⟨⟩ :
    end := Len(stack)
    PartialCover := stack[end]
    i := Cardinality(PartialCover)
    stack := SubSeq(stack, 1, end − 1)
    if i = N :
      MinCoversBelow := MinCoversBelow ∪ {PartialCover}
      continue
    k := i + 1
    succ := BelowAndSuff(Lm[k], PartialCover ∪ Image(Lm, k..N), Y)
    for z ∈ succ :
      NewCover := PartialCover ∪ {z}
      stack := stack ∘ ⟨NewCover⟩
  return MinCoversBelow
```

Figure 17: Stack-based variant of the enumeration algorithm. Proofs of safety properties for this algorithm are in the module *StrongReduction*, where the operators *Enumerate* and *BelowAndSuff* are defined. The operators *Len* and *SubSeq* are defined in the standard module *Sequences*, and the operator *Cardinality* in the standard module *FiniteSets* [7, p. 341].

```
def EnumerateMincoversBelow(Cm) :
  Partials := {{}}
  Lm := Enumerate(Cm)
  N := Cardinality(Cm)
  MinCoversBelow := {}
  while Partials ≠ {} :
    PartialCover := CHOOSE p ∈ Partials : TRUE
    Partials := Partials \ {PartialCover}
    i := Cardinality(PartialCover)
    if i = N :
      MinCoversBelow := MinCoversBelow ∪ {PartialCover}
      continue
    k := i + 1
    succ := BelowAndSuff(Lm[k], PartialCover ∪ Image(Lm, k..N), Y)
    Partials := Partials ∪ {PartialCover ∪ {z} : z ∈ succ}
  return MinCoversBelow
```

Figure 18: Set-based variant of the enumeration algorithm. Compare to Fig. 17.

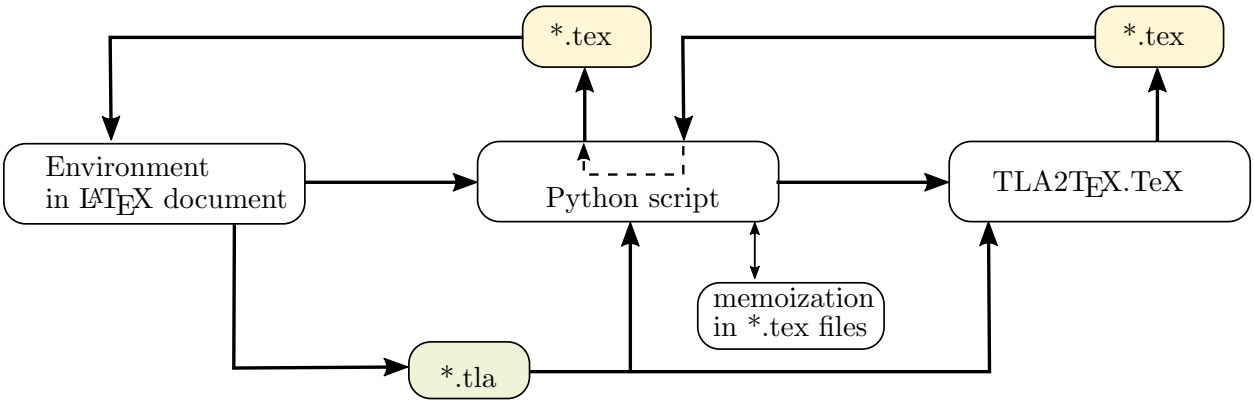


Figure 19: Call graph and file operations for calling TLA2TEX from within a LATEX environment via a Python script.

LATEX environment that calls TLA2TEX via Python to typeset the environment's content, memoizing the results on disk, and updating only when the content's hash changes. The operation of this script is shown in Fig. 19, where a LATEX environment that contains TLA$^+$ source dumps its contents to a `*.tla` file, and calls a Python script, which calls TLA2TEX to convert the `*.tla` file to a `*.tex` file. The Python script reads the `*.tex` output of TLA2TEX, and extracts the LATEX source, placing it in a `tlatex` environment. To avoid typesetting an unchanged TLA$^+$ environment twice, the Python script memoizes the LATEX source code generated by TLA2TEX for each environment, using `*.tex` files. Memoized environments that are obsolete because their contents have changed are identified on each run, and garbage collected on the next run. In the presence of an `\includeonly` statement, garbage collection is active for only the included files, so that memoized environments that correspond to files unmentioned by the `\includeonly` statement be omitted from garbage collection.

References

[1] O. Coudert, "Two-level logic minimization: An overview," *Integration, the VLSI Journal*, vol. 17, no. 2, pp. 97–140, 1994.

[2] I. Filippidis, "Decomposing formal specifications into assume-guarantee contracts for hierarchical system design," Ph.D. dissertation, California Institute of Technology, 2017. [Online]. Available: http://resolver.caltech.edu/CaltechTHESIS:12172017-171304566

[3] ——, "Cyclic core computation specification and proofs," California Institute of Technology, Tech. Rep., 2017, PDF available at: https://thesis.library.caltech.edu/10611/6/mincover.pdf TLA$^+$ source available at: https://github.com/johnyf/omega/tree/master/spec/mincover Python implementation at: https://github.com/johnyf/omega/blob/master/omega/symbolic/cover.py .

[4] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz, "A TLA$^+$ proof system," in *Proc. LPAR Work., Knowledge Exchange: Automated Provers and Proof Assistants Work.*, vol. 418, 2008, pp. 17–37.

MODULE *OptimizationSummary*

Definitions of binary function properties (ir/reflexive, transitive, anti/symmetric, support), antichains, minimal, maximal, minimum, maximum elements.
Theorems about maxima: Idempotence, identity for antichains, the set of maxima is an antichain, existence of smaller element if non-maximal.
Theorems about support of function with domain of the form $S \times S$.
Properties of *Lt* ("less than"), existence of a minimal element in finite set with *Leq* transitive and antisymmetric, existence of a minimal element below any given element, if *Leq* is also reflexive.

MODULE *MinCoverSummary*

Definitions of cover, cover from a set Y, set of covers, refinement.
Definitions of minimal cover, minimal cost.
Theorems: Minimal cover properties (generic and with cardinality as cost), all minimal covers have same cardinality, a minimal cover has minimal cardinality, adding elements to and removing elements from a cover.

MODULE *LatticesSummary*

Definitions
Theorems about lattices, existence and uniqueness of *Inf*, *Sup*, monotonicity, domain symmetry.
Theorems about *Floor* and *Floors*, existence, monotonicity, shrinking, idempotence, cardinality.
Theorems about $Geq \triangleq UpsideDown(Leq)$.
Theorems about *Ceil* and *Ceilings*.
Theorems about the variant: if $Cardinality(X)$, $Cardinality(Y)$ are unchanged in one cyclic core iteration, then X, Y are unchanged.
(1) Theorems about *MinCovers* transformation when adding or removing essential elements.
Theorems about *Hat* and *MaxHat*.
(4) Theorems about *MaxCeilings* leaving the set of covers unchanged.
(2) Theorems about $Maxima(Y, Leq)$ and minimal covers.
(3) Theorems about the effect of $Floors(Y, X, Leq)$ (and *Unfloors*) on minimal covers.

MODULE *CyclicCoreSummary*

Assumptions, Algorithm specification, Definitions
Auxiliary invariance theorems.
Proofs that minimal covers can be constructed from minimal covers of the covering problem in the current step, together with the set of essential elements.
Proof that cyclic core steps yield feasible covering problems.
Termination theorem (checked by hand).

MODULE *StrongReductionSummary*

Assumptions, Algorithm specification, Definitions
Auxiliary theorems about intermediate results during the enumeration.
Minimal cover properties: essentiality, partitioning, refinement.
Leq induces bijection between minimal covers in case one refines the other.
Proofs of safety properties.

Figure 20: Summaries of the modules *Optimization*, *MinCover*, *Lattices*, *CyclicCore*, and *StrongReduction*. The circled numbers correspond to steps in Fig. 7.

[5] T. L. Rodeheffer, "The Naiad clock protocol: Specification, model checking, and correctness proof," Microsoft Research, Tech. Rep. MSR-TR-2013-20, Feb 2013.

[6] L. Lamport, "How to write a 21st century proof," *Journal of fixed point theory and applications*, vol. 11, no. 1, pp. 43–63, 2012.

[7] ——, *Specifying systems*. Addison-Wesley, 2002.