



SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics,

Verification of selected NP-hard Problems

Zixuan Fan





SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics,

Verification of selected NP-hard Problems

**Verifikation der ausgewählten NP-schweren
Probleme**

Author:	Zixuan Fan
Supervisor:	Tobias Nipkow PhD.
Advisor:	Katharina Kreuzer
Submission Date:	Submission date



I confirm that this bachelor's thesis in informatics, is my own work and I have documented all sources and material used.

Munich, Submission date

Zixuan Fan

Acknowledgments

Abstract

NP-hardness is a widely discussed topic in the theoretical computer science. Since the publishing of the Cook-Levin-theorem in the early 1970s, the researchers have proved the NP-hardness of many problems. Among many NP-hard problems, the most famous are the 21 NP-complete problems given by Karp in the paper *Reducibility of Combinatorial Problems*, for it covers a considerable amount of NP-hard problems from various mathematical disciplines. The proofs of the NP-hardness, however, were limited on-paper. With the existence of the interactive theorem provers, it is possible to reproduce and verify the proofs with the aid of computers. In order to demonstrate the capability of interactive theorem provers in verifying the NP-hardness, we formalized and verified a few NP-hard problems using the well-known interactive theorem prover, Isabelle.

Contents

Acknowledgments	iii
Abstract	iv
1. Introduction	1
1.1. Motivations	1
1.2. Contributions	1
1.3. Outline	2
2. Preliminaries	3
2.1. Isabelle and Dependencies	3
2.2. NP-Hardness and polynomial reductions	4
2.2.1. Asymptotic Notation	4
2.2.2. Decision problems	5
2.2.3. Polynomial reductions	5
2.2.4. NP-Hardness and Satisfiability	6
2.3. Application of NREST and paradigm	6
3. Set Covering Problems	8
3.1. Exact Cover	8
3.1.1. Choice of reduction	8
3.1.2. Reduction Details	8
3.1.3. Implementation Details	10
3.2. Exact Hitting Set	13
3.2.1. Reduction Details	14
3.2.2. Implementation Details	14
4. Weighted Sum Problems	17
4.1. Subset sum	17
4.1.1. Reduction Details	17
4.1.2. Implementation Details	19
4.2. Subset sum in list and number partition	20
4.2.1. Implementation Details of Subset Sum in Sequence	22
4.2.2. Implementation Details of Partition	24
4.3. Knapsack and Zero-One Integer Programming	25
4.3.1. Knapsack	25
4.3.2. Zero-one Integer Programming	25

5. Chapter	27
A. Examples for reductions	29
A.1. Example for polynomial reduction from Satisfiability To Exact Cover	29
A.2. Example for polynomial reduction from Exact Cover To Subset Sum	29
A.3. Example for polynomial reduction from Subset Sum To Number Partition . .	30
List of Figures	31
List of Tables	32

1. Introduction

1.1. Motivations

We may encounter many real-life problems that require a decision process to find a solution. When shopping at a supermarket, for example, we always want to choose the shortest queue. Another example is board games like go and chess. These problems are formally defined as decision problems. One of the most famous decision problem classes is the **NP-Hard**. **NP-Hard** problems have been a fundamental research topic in theoretical computer science since the 1970s, when Cook and Levin showed that the **Satisfiability** problem is **NP-Complete** and Karp gave a list of 21 **NP-Hard** problems. In the next few decades, many attempts were made to show that $P = NP$ and to develop algorithms that computes **NP** problems efficiently. Among many fields related to **NP-Hardness**, we focus on the polynomial reductions, which show the **NP-Hardness** of decision problems.

All the existing proof of **NP-Hardness** were on-paper proofs, which lack the automated verification by a computer. With the existence of powerful interactive theorem provers, it is meaningful to formalize and verify the classical results of **NP-Hardness** in a computer, contributing to the theoretical basis of many existing formalisation results, e.g. cryptography, approximation algorithms etc. There has been an attempt to formalise **NP-Hard** problems that were given in Karp's paper in 1972. Our work benefits from this attempt and continues to formalise the rest of the 21 **NP-Hard** problems in the interactive theorem prover Isabelle.

1.2. Contributions

Our work contains 2 categories of problems.

1. Set Covering problems: **Exact Cover**, **Exact Hitting Set**
2. Weighted sum problems: **Subset Sum**, **Number Partition**, **Integer Programming**, **Knapsack**

For each listed problem, we present a polynomial reduction either from **Satisfiability** or from another problem that is listed above. Thus, a reduction trace from **Satisfiability** is witnessed. Furthermore, a proof for the soundness, completeness, and the polynomial complexity of each polynomial reduction is also presented.

1.3. Outline

In Chapter 2, we introduce the Isabelle dependencies and mathematical backgrounds of our work.

Chapter 3 and Chapter 4 follow with the formalisation and verification of the listed problems. For each decision problem, we present a definition of the problem and the reduction. Then, we sketch the proof of the correctness of the reduction and the polynomial bound. Finally, we present a few concrete implementation details. In Chapter 3, we discuss the polynomial reduction of the set cover problems, while Chapter 4 consists of that of the weighted sum problems.

To finish, we conclude the current status of the project of formalisation of Karp's 21 **NP-Hard** problems and present a few possibilities for the verification of the rest of the problems in Chapter 5.

2. Preliminaries

2.1. Isabelle and Dependencies

Isabelle/HOL

Isabelle is a generic interactive theorem prover. HOL is the Isabelle's formalization of Higher-Order Logic, a logical system with inductive sets, types, well-founded recursion etc. Our implementation requires the introduction of new datatypes, formalisation of natural numbers and integers. Thus, this type system is necessary.

HOL-Real_Asymp and Laudau_Symbols

TODO

NREST

TODO

The Karp21 Project

The project aims to formalise all of the 21 **NP-Hard** problems in Karp's paper in 1972. Up till now, there are **TOCOUNT** problems of them finished, with a few other **NP-Hard** problems that are related but not in Karp's list. Our work also contributes to this project, formalising six of the remaining problems. Though dependent on this project, our work only reuses a few definitions by the predecessors, while the most formalisation and verification is original. An overview of the project is given in the following graph.

DigitsInBase

This entry of Archive of Formal Proofs shows the uniqueness of representation of natural numbers given an arbitrary base. In other words, it proves the well-definedness of the n -ary counting systems. Our implementation benefits from this repository in showing the correctness of the polynomial reduction from **Exact Cover** to **Number Partition**.

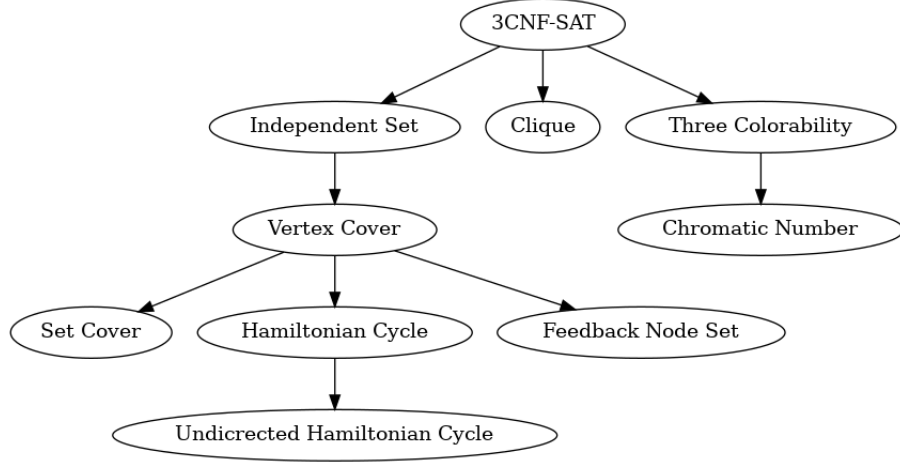


Figure 2.1.: The reduction graph of the Karp21 project

2.2. NP-Hardness and polynomial reductions

2.2.1. Asymptotic Notation

Conventionally, the asymptotic notation is used for defining complexity classes and for performing the algorithm analysis. We follow this convention and choose the big \mathcal{O} notation for algorithm analysis. To begin with, we present a brief introduction to the asymptotic notation.

Definition 1 Let $f : \mathbb{R} \rightarrow \mathbb{R}$, $g : \mathbb{R} \rightarrow \mathbb{R}$ be two real valued functions with the same domain. $f(x)$ is big \mathcal{O} of $g(x)$, which writes

$$f(x) \in \mathcal{O}(g(x))$$

if there exists a real $M \geq 0$ and a real x_0 s.t.

$$|f(x)| \leq M|g(x)|, \forall x \geq x_0$$

Thus, f is above bounded by g . In other words, f is utmost as complex as g . Following this definition we can derive many complexity classes by g . A short list of commonly encountered complexity classes is given in table.

Name	Big \mathcal{O} Notation	Algorithmic Examples
Constant	$\mathcal{O}(1)$	Parity check
Logarithmic	$\mathcal{O}(\log n)$	Binary search in a sorted array
Linear	$\mathcal{O}(n)$	Addition of integers
Quasilinear	$\mathcal{O}(n \log n)$	Merge-sort and heap-sort
Polynomial	$\mathcal{O}(n^c), c \in \mathbb{N}$	Matrix multiplication
Factorial	$\mathcal{O}(n!)$	Enumeration of partitions of a set

Table 2.1.: List of commonly encountered complexity classes.

In most cases of this paper, we only consider the polynomial class, which contains the most classes listed above as subclasses except for the factorial class. For the purpose of simplicity, we did not formalise the theory of asymptotic classes and big \mathcal{O} notation, but used the available ISABELLE dependencies of **HOL-Real_Asymp** and **Landau_Symbols**.

2.2.2. Decision problems

Definition 2 *A decision problem is a yes-no question on an infinite set of fixed type of inputs.*

Generally, if we refer to a decision problem A , we are referring to the set of all inputs for which the answer to the yes-no question is yes. The handling of a decision problem usually involves two questions:

1. Is there an algorithm, which computes the solution to this problem, terminating on all inputs?
2. If the answer to the first question is yes, is this algorithm efficient?

If the answer to the first question is yes for a problem, it is a decidable problem, otherwise it is non-decidable. We do not expect a yes or no answer for the second question, but would like to find the optimal complexity for the algorithm. While some problems are possible to computed in an optimal upperbound by a deterministic algorithm, there are also a few problems, for which no deterministic polynomial algorithm is found. We define them formally as **NP**.

Definition 3 *If there is a non-deterministic algorithm that decides the solution to the problem in polynomial time, it is in the complexity class **NP**.*

Definition 4 *If a problem is at least as complex as the most complex problems in **NP**. It is in the complexity class **NP-Hard**.*

Although many attempts have been made to prove or reject the existence of a non-deterministic algorithm for the **NP** problems, our work focuses on the NP-Hardness. We would like to formally prove that many classical decision problems are **NP-Hard**. For this reason, we have to introduce polynomial reduction.

2.2.3. Polynomial reductions

Given two decision problems A and B , a reduction is a function $f : A \rightarrow B$, which maps the inputs of the question of the first problem to that of the second problem. A reduction is polynomial if and only if the reduction function has a polynomial bound. For a polynomial reduction from A to B , we write $A \leq_p B$.

Let M and N denote the domains of A and B respectively. A function $g : M \rightarrow N$ is a polynomial reduction if and only if the following conditions are fulfilled.

$$x \in A \iff g(x) \in B \tag{2.1}$$

$$\exists k \in \mathbb{N}. f \in \mathcal{O}(n^k) \tag{2.2}$$

For the convenience reason, we usually separate (2.1) into the soundness and completeness of the reduction.

$$\text{soundness} : x \in A \implies g(x) \in B \quad (2.3)$$

$$\text{completeness} : g(x) \in B \implies x \in A \quad (2.4)$$

2.2.4. NP-Hardness and Satisfiability

To show a decision problem B is **NP-Hard**, we have to find a **NP-Hard** problem and polynomial reduction s.t. $A \leq_p B$. A first proven **NP-Hard** problem is **Satisfiability**, which was independently proven by Cook in 1971 and Levin in 1973. The **Satisfiability** problem is denoted by

Definition 5 Satisfiability

Input: A propositional logical formula in conjunctive normal form.

Output: Is there a valid assignment for this formula?

In the previous implementation of the project, the **Satisfiability** is defined by a list of clauses, with the clauses as the sets of variables. There have been many attempts to solve **Satisfiability** problem in a polynomial bound, as well as many approaches to solve **Satisfiability** problem efficiently in certain scenarios. Thus, **Satisfiability** is one of the most studies **NP-Hard** problems, from which there are also many **NP-Hard** problems reduced. Our first reduction also stems from **Satisfiability**, while all the other reductions are constructed upon novel introduced problems. More details on the reduction and implementation are given in Chapter 3 and Chapter 4. A glimpse of the available implementation is given by

```
datatype 'a lit = Pos 'a | Neg 'a

type_synonym 'a three_sat = "'a lit set list"

definition lift :: "('a ⇒ bool) ⇒ 'a lit ⇒ bool" ("↑" 60) where
  "lift σ ≡ λl. case l of Pos x ⇒ σ x | Neg x ⇒ ¬ σ x"

definition models :: "('a ⇒ bool) ⇒ 'a three_sat ⇒ bool" (infixl "⊨" 55) where
  "σ ⊨ F ≡ ∀cls ∈ set F. ∃l ∈ cls. (σ↑) l"

definition sat :: "'a three_sat ⇒ bool" where
  "sat F ≡ ∃σ. σ ⊨ F"

definition "cnf_sat ≡ {F. sat F ∧ (∀cls ∈ set F. finite cls)}"
```

2.3. Application of NREST and paradigm

The NREST package offers an approach for approximating the complexity of non-deterministic processes. This is especially useful when iterating a set, a collection or any other unordered

data structures. Thus, we use this package throughout this work. In our complexity analysis, the following commands are used.

- *RETURN* **res**. A command that returns the result **res**. It costs exactly one time unit.
- *SPECT* [**cond** \rightarrow **cost**]. A command used for checking a condition. Checking the condition **cond** take **cost** time units.
- *SPEC* $P \ Q$. A command used for assignment. Should $P \ x$ hold for an object x , it is a valid object after the assignment, which takes $Q \ x$ time units.

To apply the NREST approach in the complexity analysis, we convert the algorithm into the NREST commands. During the conversion, we follow the following principles for counting the complexity.

1. Checking the condition always costs only time unit.
2. Per iteration it costs 1 time unit each for iteration, modification and insertion.
3. All other operations costs should cost one time unit, if not stated explicitly.

Then, it is possible show a few property of this approach. Let f denote a polynomial reduction from A to B , while f_{alg} denotes the NREST version of the reduction. Furthermore, we define sizing functions s_A and s_B as metrics for the asymptotic classes. To show that the reduction is polynomial, we show that the reduction is polynomially bounded in terms of time and space, which are respectively the *refines* and the *size* lemma.

$$refines : \quad f_{alg}(A) \leq \mathcal{O}((s_A(A))^k) \quad (2.5)$$

$$size : \quad s_B(f(A)) \leq \mathcal{O}((s_A(A))^k) \quad (2.6)$$

In the end, we can conclude the following implementation paradigm to show that a reduction is correct and polynomial.

1. Prove that the reduction is correct.
2. Implement the reduction in NREST commands.
3. Prove that the reduction costs polynomial time.
4. Prove that the algorithm costs polynomial space.

3. Set Covering Problems

In this chapter, we discuss about the **NP-Hardness** of a few set covering problems. Covering problems ask whether a certain combinatorical structure A covers another structure B . Alternatively, it may also ask for the minimal size of A to cover B . We focus on a subclass of covering problems, the exact covering problem. In this subclass, A covers B exactly, i.e. no element in B is covered twice in A . In Karp's paper in 1972, the following covering problems were included: Exact Cover, Exact Hitting Set, 3-Dimensional Matching, and Steiner Tree. In our implementation, we reduced **Satisfiability** to **Exact Cover**, and then reduced **Exact Cover** to **Exact Hitting Set**.

3.1. Exact Cover

The exact cover problem is a special case of the set cover problem. Besides the covering property, it also requires the uniqueness of the elements.

Definition 6 *Exact Cover*

Input: A set X and a collection S of subsets of X .

Output: Is there a disjoint subset S' of S s.t. each element in X is contained in one of the elements of S' ?

definition "exact_cover $\equiv \{(X, S). \text{finite } X \wedge \bigcup S \subseteq X \wedge (\exists S' \subseteq S. \bigcup S' = X \wedge \text{disjoint } S')\}$ "

3.1.1. Choice of reduction

Since **Exact Cover** is a fundamental **NP-Hard** problem, there are many different reductions available. Karp's reduction based on the chromatic number problem. Although the chromatic number problem was formalized in Karp's 21 project, we did not choose this reduction because of the complexity of the graph traversal and the differences between Karp's definition and the available Isabelle's definition. Additionally, there is an easy reduction from **Satisfiability** to **Exact Cover**. This reduction does not involve graph traversal. The only technical barrier is the typeless set. While Isabelle only supports typed sets, we resolve this problem by creating a container type. More details follow in the section 3.1.3.

3.1.2. Reduction Details

Given a propositional logical formula F , we index the variables and the clauses and use the following notations.

1. x_i denotes the i -th variable in the formula with $x_i \in \text{vars } F$
2. c_i denotes the i -th clause in the formula with $c_i \in F$
3. p_{ij} denotes the j -th position/literal in the i -th clause with $p_{ij} \in c_i$

Then we construct a set X and which contains all 3 different kinds objects.

$$X = \text{vars } F \cup F \cup \bigcup_{c_i \in F} c_i$$

Furthermore, we construct S , a collection of subsets of X . We determine the following subsets

1. $\{p_{ij}\}$. The unary set of positions
2. $\{c_i, p_{ij}\}$. The binary set of a clause and a position in it.
3. $\text{pos}(x_i) := \{x_i\} \cup \{p_{ab} \mid p_{ab} = x_i\}$. The set of its positive occurrences as positions.
4. $\text{neg}(x_i) := \{x_i\} \cup \{p_{ab} \mid p_{ab} = \neg x_i\}$. The set of a variable with its negative occurrences as positions.

S contains all of the four types of subsets.

$$\begin{aligned} S = & \{p_{ij} \mid p_{ij} \in c_i, c_i \in F\} \cup \{\{c_i, p_{ij}\} \mid p_{ij} \in c_i, c_i \in F\} \\ & \cup \{\{x_i\} \cup \{p_{ij} \mid p_{ij} \in c_i, c_i \in F\} \mid x_i \in \text{vars } F, x_i = p_{ij}\} \\ & \cup \{\{x_i\} \cup \{p_{ij} \mid p_{ij} \in c_i, c_i \in F\} \mid x_i \in \text{vars } F, \neg x_i = p_{ij}\} \end{aligned}$$

The pair of (X, S) is the input for the **Exact Cover** problem.

Lemma 1 (Soundness) *Let F be satisfiable. The pair (X, S) is then an instance of the exact cover.*

Let $\sigma \models F$ be a valid assignment. We construct an exact cover $S' \subseteq S$ of X in the following steps.

1. For each $x_i \in \text{vars } F$, $\text{pos}(x)$ is included in S' if $\sigma(\text{pos}(x)) \equiv \top$. Otherwise we insert $\text{neg}(x)$ into S' .
2. For each $c_i \in F$, we choose the minimal j with $\sigma(p_{ij}) \equiv \top$, and insert $\{c_i, p_{ij}\}$ into S'
3. For each $p_{ij} \in c_i$, if $\sigma(p_{ij}) \equiv \top$ and $\{c_i, p_{ij}\}$ is not in S' , the unary set $\{p_{ij}\}$ is included.

Obviously, each position in $\text{pos}(x)$ and $\text{neg}(x)$ will be false under the assignment σ , while the positions in the other sets are all true. By design, the positions in the second and the third steps never duplicate. Thus, no same position occurs in two different sets in the collection S' . Furthermore, each clause exists in exactly one set in the second step, which is also the case for the variables in the first step. Hence S' is disjoint.

From this fact, we can also conclude that clauses and variables are covered in S' . Now we only have to show that all the positions are covered. If a position p_{ij} is false under σ , it is covered in the first step. Otherwise it is either covered in the second step or the third step. With the disjointness, we may conclude that S' covers X exactly and that the reduction is sound.

Lemma 2 (Completeness) *Let (X, S) be reduced from F . If (X, S) is an instance of the exact cover, F has to be satisfiable.*

Given an exact cover pair (X, S) reduced from F , it is easy to reconstruct the model σ with the same approach as in the proof of the soundness, showing that F is satisfiable. Thus, the completeness of the reduction is proven.

Lemma 3 (Polynomial Complexity) *The construction of (X, S) from F is polynomial.*

In the reduction, we have to iterate all of the variables, the clauses and the positions. Thus, we have to find a polynomial bound with regards to three metrics. Let n, m and k denotes the number of variables, clauses and positions respectively. To derive the three metrics, we have to iterate all of the clauses, resulting the complexity of $\mathcal{O}(m)$. Unexpectedly, this results in the set X , indicating a linear complexity for the construction.

Now the interesting part is the collection S . For each type of subsets, we give a polynomial bound

1. $\{p_{ij}\}$. It is sufficient to merely iterate the positions, which requires the complexity of k .
2. $\{c_i, p_{ij}\}$. Each clause is iterated for $|c_i|$ times. Since $|c_i|$ is a constant, there is a $c \in \mathbb{N}$ s.t. $|c_i| \leq c$. Obviously, the complexity is above bounded by $c \cdot m$.
3. $pos(x)$ and $neg(x)$. For each variable x , it is required to iterate all of the positions, which produces the complexity of $2 \cdot nk$ in total.

Thus, the construction costs the polynomial complexity of $k + cm + nk \in \mathcal{O}(nk + m)$. With the linear complexity of the construction of X , we conclude that the reduction has the polynomial complexity.

3.1.3. Implementation Details

Reduction and Proof

Intuitively, the variables, positions and clause can be represented by indices or tuples of indices. However, it is not easy to unify the representation s.t. they are of the same type. One possible solution is introducing a new dimension in the tuple, and use it to classify the represented objects. However, we might still have to introduce a new enumerate type for this purpose. On the contrary, a container type is exempted from the tedious handling of tuples, although it still introduces a new type. Thus, we implement this container type with a few auxiliary functions `xc_element` as follows.

```
datatype 'a xc_element = V 'a | C "'a lit set" | L "'a lit" "'a lit set"

fun var :: "'a lit  $\Rightarrow$  'a" where
  "var (Neg a) = a" | "var (Pos a) = a"
```

definition "vars F \equiv (var) ' \bigcup (set F)"

definition vars_of_sat :: "'a three_sat \Rightarrow 'a xc_element set" **where**
 "vars_of_sat F = {V v | v. v \in vars F}"

definition clauses_of_sat :: "'a three_sat \Rightarrow 'a xc_element set" **where**
 "clauses_of_sat F = {C c | c. c \in set F}"

definition literals_of_sat :: "'a three_sat \Rightarrow 'a xc_element set" **where**
 "literals_of_sat F = {L l c | l c. c \in set F \wedge l \in c}"

Then, it is possible to define the reduction function.

definition literal_sets
 :: "'a three_sat \Rightarrow 'a xc_element set set" **where**
 "literal_sets F = {{l} | l. l \in (literals_of_sat F)}"

definition clauses_with_literals
 :: "'a three_sat \Rightarrow 'a xc_element set set" **where**
 "clauses_with_literals F = {{C c, l} | c l. C c \in (clauses_of_sat F) \wedge l \in (literals_of_sat F)
 \wedge l \in {L a c | a. a \in c}}"

definition var_true_literals
 :: "'a three_sat \Rightarrow 'a xc_element set set" **where**
 "var_true_literals F =
 {{V v} \cup {l. l \in (literals_of_sat F)
 \wedge (\exists c. C c \in (clauses_of_sat F) \wedge L (Neg v) c = 1)} | v. V v \in (vars_of_sat F)}"

definition var_false_literals
 :: "'a three_sat \Rightarrow 'a xc_element set set" **where**
 "var_false_literals F =
 {{V v} \cup {l. l \in (literals_of_sat F)
 \wedge (\exists c. C c \in (clauses_of_sat F) \wedge L (Pos v) c = 1)} | v. V v \in (vars_of_sat F)}"

abbreviation "comp_X F \equiv
 vars_of_sat F \cup clauses_of_sat F \cup literals_of_sat F"

abbreviation "comp_S F \equiv
 literal_sets F \cup clauses_with_literals F
 \cup var_true_literals F \cup var_false_literals F"

definition sat_xc :: "'a three_sat \Rightarrow 'a xc_element set * 'a xc_element set set"
where
 "sat_xc F = (comp_X F, comp_S F)"

Proof of the correctness of the reduction argues about the disjointness and covering property by showing that a specific type of element is existent or non-existent in the constructed sets.

Though the majority of the proof is lengthy, it does not require too many techniques and is overall trivial after applying the corresponding lemmas and definitions.

The only exception is the construction of $\{c_i, p_{ij}\}$ in the second step. p_{ij} is chosen as the position with the least index. However, in our definition, there is no such indexing property in the **xc_element** datatype, meaning that such construction is not possible. Fortunately, the choice of the p_{ij} does not necessarily require a minimal index. Any arbitrary p_{ij} that is true under the assignment σ is a valid choice. For this reason, we chose to use the **SOME** predicate in Isabelle, which obtains an arbitrary instance that fulfills the required property.

```
definition constr_cover_clause
  :: "'a lit set  $\Rightarrow$  ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a xc_element set set" where
  "constr_cover_clause c  $\sigma$  =
    (SOME s.  $\exists p \in c. (\sigma \uparrow) p \wedge s = \{\{C\ c, L\ p\ c\}\} \cup \{\{L\ q\ c\} \mid q. q \in c \wedge q \neq p \wedge (\sigma \uparrow) q\}$ )"
```

```
lemma constr_cover_clause_unfold:
assumes " $\sigma \models F$ " "c  $\in$  set F"
shows " $\exists p \in c. (\sigma \uparrow) p \wedge \text{constr\_cover\_clause } c\ \sigma = \{\{C\ c, L\ p\ c\}\} \cup \{\{L\ q\ c\} \mid q. q \in c \wedge q \neq p \wedge (\sigma \uparrow) q\}$ "
```

To end with, we show that the reduction is sound, complete, and consequently correct.

```
theorem is_reduction_sat_xc:
  "is_reduction sat_xc cnf_sat exact_cover"
```

Polynomial Complexity

To begin with, it is necessary to determine the metrics, on which the complexity is dependent. For a logical formula F , we will iterate all of the variables, clauses and positions. Hence all of them are needed as metrics. Nevertheless, the NREST implementation does not support a complexity bound with different metrics. As a result, it is necessary to choose the maximum of all metrics and use it as our sole metrics¹. Similarly, we define $\max|X||S|$ as the metrics for the exact cover instance (X, S) . Then we can define the NREST algorithm.

```
definition "sat_to_xc_alg  $\equiv$  ( $\lambda F.$ 
  do
    {
      VS  $\leftarrow$  mop_vars_of_sat F;
      CS  $\leftarrow$  mop_clauses_of_sat F;
      LS  $\leftarrow$  mop_literals_of_sat F;
      s1  $\leftarrow$  mop_literal_sets F;
      s2  $\leftarrow$  mop_clauses_with_literals F;
      s3  $\leftarrow$  mop_var_true_literals F;
      s4  $\leftarrow$  mop_var_false_literals F;
```

¹A few previous reductions from **Satisfiability** uses the number of clauses as a metrics. It was also correct for those reductions iterated only the clauses.

```

    X ← mop_union_x CS VS LS;
    S ← mop_union_s s1 s2 s3 s4;
    RETURN (X, S)
  }
)"

```

```

definition "sat_to_xc_time_aux l m n =
  2 * m + 3 * n + 1 + 1
+ 3 * l + 3 * m + m + 3 * n
+ (3 + 1) * n + (3 + 1 + 1) * n + m
+ (3 + 1) * l + (3 + 1 + 1) * n + (1 + 1) * m + 1
+ (3 + 1) * l + (3 + 1 + 1) * n + (1 + 1) * m + 1
+ 2 + 3"

```

```

definition "sat_to_xc_space_aux l m n =
  l + m + n
+ n + n
+ l + l"

```

The proof of the polynomial bound is, however, hardly automated. Part of the reason is that Isabelle fails to find and apply the monotonicity of the space function, time function and the cardinality. For this reason, it is necessary to show such relationships. For example, one of them is about the cardinality

```

lemma card_Un_le_4:
  "card (a ∪ b ∪ c ∪ d) ≤ card a + card b + card c + card d"

```

Finally, we show the *refines* and *size* lemma as well as that the reduction is polynomial.

```

lemma sat_to_xc_size:
  "size_XC (sat_xc F) ≤ sat_to_xc_space (size_SAT_max F)"

lemma sat_to_xc_refines:
  "sat_to_xc_alg F ≤
    SPEC (λy. y = sat_xc F) (λ_. sat_to_xc_time (size_SAT_max F))"

theorem is_reduction_sat_xc:
  "is_reduction sat_xc cnf_sat exact_cover"

```

3.2. Exact Hitting Set

The hitting set problems are variants of the set covering problems. Essentially, they are two different ways of viewing the same problem. Just as the hitting set² is a variant of the set covering, the exact hitting set is a variant of the exact cover.

²Note that the exact hitting set problem was referred to as the hitting set problem in Karp's work, whereas it is generalized to be another problem nowadays.

Definition 7 Exact Hitting Set

Input: A collection of sets S

Output: Is there a finite set W s.t. the intersection of W and each element $s \in S$ contains exactly one element?

$$\text{Exact Hitting Set} := \{S \mid \exists W. \forall s \in S. |W \cap s| = 1\}$$

3.2.1. Reduction Details

Given an exact cover pair (X, S) , the hitting set input C is constructed by

$$C = \{\{s \mid u \in s, s \in S\} \mid u \in X\}$$

Thus, C is the set of sub-collections denoted by c_u . All sets in c_u share the same element u .

Lemma 4 (Soundness) *Let (X, S) be an exact cover instance. A collection C reduced from (X, S) is then an instance of the exact hitting set.*

The soundness of this reduction is straightforwardly proven with the existence of S' that covers X exactly. For the soundness of the reduction, it suffices to show

$$\exists W. \forall c_u \in C. |W \cap c_u| = 1$$

Let $W = S'$ and $c_u = \{s \mid u \in s, s \in S\} \in C$ be an arbitrary element. Since S' covers X exactly, there is exactly one $s \in S'$ that contains u . Moreover, this s is also included in the c_u , for $s \in S'$ and $S' \subseteq S$. Thus, it holds that $W \cap c_u = S' \cap c_u = \{s\}$ and consequently $|W \cap c_u| = 1$.

Lemma 5 (Completeness) *Let the collection C be a collection reduced from a pair (X, S) . If C is an instance of the exact hitting set, (X, S) has to be an instance of the exact cover.*

The proof of the completeness shares a similar construction. The only difference is that W is not necessarily a subset of S . Nevertheless, there exists a subset $W' \subseteq W$ s.t. it is not only a subset of S' , but it also fulfills the same property as W . Let $S' = W'$, the completeness is then proven analogously as the soundness.

Lemma 6 (Polynomial Complexity) *The construction of C from (X, S) is polynomial.*

Finally, we show that the reduction is polynomial. In our reduction, it is necessary to iterate the set X and the collection S in a nested loop. With the cardinality $|X|$ and $|S|$ as the metrics, it is obvious that the reduction costs the complexity of $\mathcal{O}(|X||S|)$.

3.2.2. Implementation Details
Reduction and Proof

Since the exact cover problem is defined over a finite set X and a finite collection S , we have to check if the X and S are finite and if S is a collection of X . Thus, a condition statement, which checks this requirement, is added to the implementation. Furthermore, the proof of the correctness is implemented as described above. Following is a snippet of implemented definitions and lemmas.

definition "exact_hitting_set $\equiv \{S. \exists W. \text{finite } W \wedge (\forall s \in S. \text{card } (W \cap s) = 1)\}$ "

definition "xc_to_ehs $\equiv \lambda(X, S). (\text{if finite } X \wedge \bigcup S \subseteq X \text{ then } \{\{s. u \in s \wedge s \in S\} \mid u. u \in X\} \text{ else } \{\{\}\})$ "

theorem is_reduction_xc_to_ehs:
 "is_reduction xc_to_ehs exact_cover exact_hitting_set"

Polynomial Complexity

We determine the size of the exact hitting set entry C as $|C|$. According to the paradigm, we define the NREST algorithm and show the *refines* and *size* lemma as follows

definition "xc_to_ehs_alg $\equiv \lambda(X, S).$
 do {
 $b \leftarrow \text{mop_check_finiteness_and_is_collection } (X, S);$
 if b
 then do {
 $S' \leftarrow \text{mop_construct_sets } (X, S);$
 RETURN S'
 }
 else do {
 RETURN $\{\{\}\}$
 }
 }
 "

definition "xc_to_ehs_space $n \equiv 1 + n * n$ "

definition "xc_to_ehs_time $n \equiv 1 + 3 * n * n$ "

lemma xc_to_ehs_size:
 "size_ehs (xc_to_ehs xc) \leq xc_to_ehs_space (size_XC xc)"

lemma xc_to_ehs_refines:
 "xc_to_ehs_alg xc \leq SPEC ($\lambda y. y = \text{xc_to_ehs } xc$) ($\lambda_. \text{xc_to_ehs_time } (\text{size_XC } xc)$)"

The proof of is mostly automated after unfolding the necessary definitions. However, an additional step is required for indicating the relationship between the sizing functions. While it holds $|C| = |X|$, the size of the exact cover is defined by $\max |X| |S|$ instead of $|X|$. Hence we can only conclude that the size of the exact hitting set is less equal than the size of the exact cover. The proof automation will then fails in showing $|C| \leq |S| \cdot |S| + 1$ when $|S| \geq |X|$. For this reason, we have to prove one additional lemma about the cardinality of the exact hitting set.

lemma card_ehs_le:
assumes "finite X " "card $X \leq Y$ "
shows "card $\{\{s. u \in s \wedge s \in S\} \mid u. u \in X\} \leq \text{Suc } (Y * Y)$ "

Finally, we show that the reduction is correct and polynomial.

theorem xc_to_ehs_is_polyred:
"ispolyred xc_to_ehs_alg exact_cover exact_hitting_set size_XC size_ehs"

4. Weighted Sum Problems

In this chapter, we discuss the **NP-Hardness** of a few weighted problems. In weighted sum problems, there is a weighting function w over a set S . The problems ask about whether the weight of this elements in S satisfies a equality or inequality relationship. The weighted sum problems are also mathematical programming problems, which is another large discipline of **NP-Hard** problems. In Karp's paper, there were **Exact Cover**, **Subset Sum**¹, and **Knapsack** introduced. We present reductions from **Exact Cover** to **Subset Sum**, from **Subset Sum** to **Number Partition**, **Knapsack** and **Integer Programming**.

4.1. Subset sum

We define the subset sum problem with a set and a weighting function. There are also a few alternative definitions using a multiset or a list without a weighting function, which is useful for our other reductions. More details follow in the number partition section.

Definition 8 *Subset Sum*

Input: A finite set S , a weighting function w , and an integer B

Output: Is there a subset $S' \subseteq S$ s.t.

$$\sum_{x \in S'} w(x) = B \tag{1}$$

4.1.1. Reduction Details

We reduce the exact cover problem to the subset sum. We start with the exact cover problem over the sets of natural numbers. Given (X, S) an exact cover instance over natural numbers, let S be the set in the entry of the subset sum problem. Then we define the weighting function w and the sum B by

$$w(s) = \sum_{x \in s} p^x, B = w(X)$$

Where p is a natural that is no less than $|S|$ The triple (S, w, B) is then an input for the subset sum problem.

Lemma 7 (Soundness) *If (X, S) is an instance of the exact cover. The reduced (S, w, B) is then an instance of the subset sum problem.*

¹What Karp presented was referred to as Knapsack, although the definition is closer to the subset sum nowadays

Apparently, the reduction is sound. Let the $S' \subseteq S$ be an exact cover of X . It then holds that

$$\sum_{s \in S'} w(s) = \sum_{s \in S'} \left(\sum_{x \in s} p^x \right) = \sum_{x \in \bigcup S'} p^x = \sum_{x \in X} p^x = B$$

Thus, (S, w, B) is an instance of the subset sum problem.

Lemma 8 (Completeness) *Let (S, w, B) be reduced from (X, S) . If (S, w, B) is a subset sum instance, (X, S) has to be an exact cover instance.*

Obtain $S' \subseteq S$ for which **(1)** holds. We show the disjointness of S' with contradiction. Assume that S' is not disjoint. Then there exist $s_1, s_2 \in S'$ s.t. $s_1 \cap s_2 \neq \emptyset$. Let $x \in s_1 \cap s_2$ be arbitrary. Then there are two cases for the coefficient c_x of p^x in the polynomial $\sum_{x \in S'} w(x)$ of p .

1. $c_x \geq 2$. The corresponding coefficient c'_x in the polynomial $w(X)$ is one. From **(1)** it is obvious that this case is not valid.
2. $c_x = 1$ or $c_x = 0$. Though **(1)** is satisfied, there are still at least two p^x in the polynomial $\sum_{x \in S'} w(x)$. Hence the number of p^x in this polynomial is at least p . However, there are utmost $|S|$ elements in S' , meaning that there are utmost $|S|$ such p^x . From the fact the $p > |S|$, it is not possible that the number of p^x is greater equal p . As a result, this case is also not valid.

In conclusion, the assumption is false and S' is thus disjoint. Then, it follows directly from the disjointness that S' covers X , otherwise **(1)** is not satisfied.

This reduction is, however, limited to the exact cover over natural numbers. It is still necessary to generalize the reduction to any arbitrary type. For this reason, we need to construct a mapping.

Lemma 9 *Let S be an arbitrary finite S . Then there exists a bijective function $f, S \rightarrow N$ s.t. the image of f is empty when S is empty, and is $\{1, 2, \dots, |S| - 1\}$ otherwise.*

The proof for this lemma is trivial. With this approach, we can covert each exact cover problem to an exact cover problem over the natural numbers and then reduce it to the subset sum problem.

Lemma 10 (Polynomial Complexity) *The construction of (S, w, B) from (X, S) is polynomial.*

When we map an arbitrary set into a natural number set as presented in Lemma 9, we have to iterate over the set X , which costs the complexity of $|X|$. Furthermore, we have to iterate over S to construct w and B , resulting in the complexity of $2|S|$. In total, it only costs $|X| + 2|S| \in \mathcal{O}(|X| + |S|)$, i.e. the linear complexity.

4.1.2. Implementantation Deatails

Reduction and Proof

In implementation, we started with the construction as presented in Lemma 9. The function **map_to_nat** returns a function that maps X to a corresponding set of natural numbers. Following this, we define the weighting function and reduction function. Similar to what happened with the exact hitting set, we also check if X is finite and if S is a collection before we perform a reduction, for these conditions are a requirement under our definition.

```

definition "is_subset_sum SS  $\equiv$ 
  (case SS of (S, w, B) => (sum w S = B))"

definition "subset_sum  $\equiv$ 
  {(S, w, B) | S w B. finite S  $\wedge$  ( $\exists S' \subseteq S$ . is_subset_sum (S', w, B))}"
definition "map_to_nat X  $\equiv$  (SOME f. (if X = {} then bij_betw f X {} else bij_betw
  f X {1..card X}))"
definition "weight p X  $\equiv$  (sum ( $\lambda x$ . p ^ x) X)"

definition
"xc_to_ss XC  $\equiv$ 
  (case XC of (X, S) =>
    (if infinite X  $\vee$  ( $\neg \bigcup S \subseteq X$ ) then ({}, card, 1::nat)
    else
      (let f = map_to_nat X;
        p = max 2 (card S + 1)
      in
        (S,
          ( $\lambda A$ . (weight p (f ' A))),
          (weight p (f ' X)))
        )
      )
  )"

```

Apart from the definition, we need a lemma that converts the sum to a polynomial. To guarantee that p^k is unique for an arbitrary k , we require that $p \geq 2$ as in the lemma **weight_eq_poly**.

```

lemma weight_eq_poly:
fixes X:: "nat set" and p::nat
assumes "p  $\geq$  2"
shows "weight p X =  $\sum \{p ^ x \mid x . x \in X\}$ "

```

The main part of the proof is implemented as discussed in the reduction details. A problem occurred when showing the completeness of the reduction. Besides the proof presented in the implementation details, it is necessary to show that the representation of a natural number in the form of the polynomials with the base p is unique. To achieve this, we imported the Archive of Formal Proofs entry **DigitInBase** and applied the theorem **seq_uniqueness**.

```

theorem seq_uniqueness:
  fixes m j :: nat and D :: "nat  $\Rightarrow$  nat"
  assumes "eventually_zero D" and "m = ( $\sum$ i. D i * bi)" and " $\wedge$ i. D i < b"
  shows "D j = ith_digit m j"

```

Then, we can show that the reduction is correct.

```

theorem is_reduction_xc_to_ss:
  "is_reduction xc_to_ss exact_cover subset_sum"

```

Polynomial Complexity

The implementation of the proof for polynomial complexity is identical to what is introduced in reductio details. Fortunately, there is not additional step to be stated, for all of the proof is automated.

```

definition "xc_to_ss_alg  $\equiv$  ( $\lambda$ (X, S).
  do {
    b  $\leftarrow$  mop_check_finite_collection (X, S);
    if b
    then do {
      RETURNT ({}, card, 1)
    }
    else do {
      f  $\leftarrow$  mop_constr_bij_mapping (X, S);
      p  $\leftarrow$  mop_constr_base (X, S);
      w  $\leftarrow$  mop_constr_weight p f;
      B  $\leftarrow$  mop_constr_B p f X;
      RETURNT (S, w, B)
    }
  }
)"

definition "xc_to_ss_space n = 1 + n + 1 + n + 1"
definition "xc_to_ss_time n = 1 + (3 * n + 1) + (n + 3) + 1 + (3 * n + 1)"
lemma xc_to_ss_size:
  "size_SS (xc_to_ss (X, S))  $\leq$  xc_to_ss_space (size_XC (X, S))"

lemma xc_to_ss_refines:
  "xc_to_ss_alg (X, S)  $\leq$  SPEC ( $\lambda$ y. y = xc_to_ss (X, S)) ( $\lambda$ _. xc_to_ss_time (size_XC (X, S)))"

theorem xc_to_ss_ispolyred:
  "ispolyred xc_to_ss_alg exact_cover subset_sum size_XC size_SS"

```

4.2. Subset sum in list and number partition

The next problem that we want to reduce to is partition.

Definition 9 Number Partition

Input: A finite sequence as of natural numbers.

Output: Is there a sub-sequence $as' \subset as$ s.t.

$$\sum_{x \in as'} x = \sum_{x \in as - as'} x \quad (2)$$

For the modeling of the sequence, we use the list. Although it is possible to define the partition problem using the set and weighting function as in the subset sum problem, choose the presented definition for two reasons.

1. Showing that the definition of the problem is not an important factor in reduction, i.e. both definition are valid and reducible under Isabelle.
2. Staying consistent with the available Archive of Formal Proof instance *Hardness of Lattice Problems*, which is also the purpose of this work, i.e. providing theoretical bases for a few other verification projects.

Thus, we need to perform the reduction from the list version of the subset sum problem. We give the definition of the subset sum problem using a sequence.

Definition 10 Subset Sum in Sequence

Input: A finite sequence as of natural numbers, a natural number s

Output: Is there a sub-sequence $as' \subset as$ s.t.

$$\sum_{x \in as'} x = B \quad (3)$$

Given the subset sum instance (S, w, B) , it is obvious that we can obtain a sequence as by converting S into a sequence and map the sequence with the weighting function w . Let $s = B$. The resulting pair (as, s) is then an instance of the subset sum problem in sequence representation. Then we reduce (as, s) to a partition instance bs .

$$bs = (1 - s + \sum_{x \in as} x) \# (s + 1) \# as$$

Lemma 11 (Soundness) *If there exists an as' s.t. the equation (3) holds for (as, B) , (2) should hold for the reduced bs .*

We construct a bs' from as' by

$$\begin{aligned} bs' &= (1 - s + \sum_{x \in as} x) \# as' \\ bs - bs' &= (s + 1) \# (as - as') \end{aligned}$$

Where the sums of the sequences satisfy the equation

$$\sum_{x \in bs'} x = (1 - s + \sum_{x \in as} x) + \sum_{x \in as'} x = (1 - s + \sum_{x \in as} x) + s = (s + 1) + (\sum_{x \in as} x - s) = \sum_{x \in bs - bs'} x$$

Thus, the reduction is sound.

Lemma 12 (Completeness) *Let bs be reduced from (as, B) . If there exists a bs' s.t. (2) holds for bs , (3) should then hold for (as, B) .*

It holds that

$$(1 - s + \sum_{x \in as} x) + (s + 1) = \sum_{x \in as} x + 2 > \sum_{x \in as} x$$

As a result, $(1 - s + \sum_{x \in as} x)$ and $s + 1$ are not supposed to be simultaneously existent in bs' . After separating the first two elements of bs into different subsequences, the as' is constructed by obtaining the tail of bs' , with which the completeness is proven.

Lemma 13 (Polynomial Complexity) *The reduction from the subset sum to partition is polynomial.*

The conversion between the definitions of the subset sum problem costs linear complexity w.r.t. the cardinality of the set, for it is necessary to iterate the set S and map the sequence with the weighting function. More specifically it costs the complexity of $|S| + 1$. Furthermore, we iterate the sequence similarly when reducing the subset sum to partition, costing the complexity of $|as| + 2$. In total, the complexity is $\mathcal{O}((|S| + 1) + (|as| + 2)) \in \mathcal{O}(|S|)$ because of $|as| = |S|$.

4.2.1. Implementation Details of Subset Sum in Sequence

Reduction and Proof

Although the reduction is more straightforward compared to the previously introduced ones, the implementation is even lengthier. The reason is that conversion of the set to a list also is a reduction, which we have to verify. The intermediate step is then defined by

definition "subset_sum_indices $\equiv \{(S, w, B). \text{finite } S \wedge S = (\text{if } S = \{\} \text{ then } \{\} \text{ else } \{1.. \text{card } S\}) \wedge (\exists S' \subseteq S. \text{sum } w \ S' = B)\}$ "

Apparently, to map S to an set of integers from 1 to $|S|$, we need to apply **Lemma 9** again. Additionally, we have to check if S is finite, for finiteness is a requirement of the reduction. Thus, the reduction and the proof is given by

definition "generate_func $S \equiv (\text{SOME } f. (\text{if } S = \{\} \text{ then } \text{bij_betw } f \ S \ \{\} \text{ else } \text{bij_betw } f \ S \ \{1.. \text{card } S\}))"$

definition "ss_to_ss_indices $\equiv (\lambda(S, w, B). \text{if finite } S \text{ then } ((\text{generate_func } S) \ ' \ S, (\lambda x. w \ (\text{inv_into } S \ (\text{generate_func } S) \ x)), B) \text{ else } (\{\}, \text{id}, 1))"$

theorem is_reduction_ss_to_ss_indices:
"is_reduction ss_to_ss_indices subset_sum subset_sum_indices"

Then, it suffices to convert the set into a list and perform the map, in which we used the function **sorted_list_of_set**, converting a set of ordered type to a sorted list. Similarly, we check if S is finite and if S is of form $\{1, 2, \dots, |S|\}$ as a requirement. The proof for the correctness is then mostly straightforward after unfolding the necessary definitions and using a few available lemmas in the list library, such as **nth_equalityI** etc.

definition "subset_sum_indices $\equiv \{(S, w, B). \text{finite } S \wedge S = (\text{if } S = \{\} \text{ then } \{\} \text{ else } \{1..\text{card } S\}) \wedge (\exists S' \subseteq S. \text{sum } w \text{ } S' = B)\}$ "

definition ss_indices_to_ss_list :: "nat set \times (nat \Rightarrow nat) \times nat \Rightarrow nat list \times nat" **where**
"ss_indices_to_ss_list $\equiv (\lambda(S, w, B). \text{if } (\text{finite } S \wedge S = \{1..\text{card } S\}) \text{ then } (\text{map } w \text{ } (\text{sorted_list_of_set } S), B) \text{ else } ([], 1))$ "

theorem is_reduction_ss_indices_to_ss_list:
"is_reduction ss_indices_to_ss_list subset_sum_indices subset_sum_list"

Polynomial Complexity

Similar to the reduction from exact cover to subset sum, the proof for the polynomial complexity for two reductions is rather trivial.

definition "ss_to_ss_indices_alg $\equiv \lambda(S, w, B).$
do {
 b \leftarrow mop_check_finiteness (S, w, B);
 if b
 then do {
 S' \leftarrow mop_mapping_of_set (S, w, B);
 w' \leftarrow mop_updating_the_weighting (S, w, B);
 RETURN (S', w', B)
 }
 else do {
 RETURN ({}, id, 1)
 }
}"

definition "ss_to_ss_indices_space n = 1 + n + n + 1"

definition "ss_to_ss_indices_time n = 1 + (3 * n + 3 * n + 1) + (3 * n + 3 * n + 1) + 1"

theorem ss_to_ss_indices_is_polyred:
"ispolyred ss_to_ss_indices_alg subset_sum subset_sum_indices size_SS size_ss_indices"

definition "ss_indices_to_ss_list_alg $\equiv \lambda(S, w, B).$
do {
 b \leftarrow mop_check_finiteness_set (S, w, B);
 if b
 then do {
 as \leftarrow mop_mapping_to_list (S, w, B);
 s \leftarrow mop_nat_to_int B;
 RETURN (as, s)
 }
 else do {

```

    RETURN ( [], 1 )
  }
}"

```

definition "ss_indices_to_ss_list_space n = 1 + 2 * n"

definition "ss_indices_to_ss_list_time n = 1 + 3 * n + 3 * n + 1"

theorem ss_indices_to_ss_list_is_polyred:

"ispolyred ss_indices_to_ss_list_alg subset_sum_indices subset_sum_list size_ss_indices size_ss_list"

4.2.2. Implementantion Details of Partition

Instead of the original definition, where the sum a sub-sequence is equal to another, we use a different definition in the implementation, where the twice of the sum of a sub-sequence is equal to to the sum of the sequence. We have also shown that this definition is equivalent to the original definition, i.e. **part_alter** in implementation.

definition "part \equiv {as::nat list. \exists xs. ($\forall i < \text{length } xs. xs[i] \in \{0, 1\}\}) \wedge \text{length } as = \text{length } xs$
 $\wedge 2 * (\sum i < \text{length } as. as[i] * xs[i]) = (\sum i < \text{length } as. as[i])$ }"

definition "part_alter \equiv {as::nat list. \exists xs. ($\forall i < \text{length } xs. xs[i] \in \{0, 1\}\}) \wedge \text{length } as = \text{length } xs$
 $\wedge (\sum i < \text{length } as. as[i] * xs[i]) = (\sum i < \text{length } as. as[i] * (1 - xs[i]))$ }"

theorem part_eq_part_alter: "part = part_alter"

theorem is_reduction_ss_list_to_part:

"is_reduction ss_list_to_part subset_sum_list part"

The reason was initially for the convenience of the proof. In the original definition, it is necessary to consider the sum of the sub-sequence ($as - as'$) when showing the soundness lemma. This is, unfortunately, rather complex under our definition, for we have to flip the list xs , the zero-one list that is used for multiplication. For this flipping operation, we have show the lemma **sum__part**. If we use the new definition, this is avoidable.

lemma sum_binary_part:

assumes "($\forall i < \text{length } xs. xs[i] = (0::nat) \vee xs[i] = 1$)" "length as = length xs"

shows " $(\sum i < \text{length } as. as[i] * xs[i]) + (\sum i < \text{length } as. as[i] * (1 - xs[i])) = (\sum i < \text{length } as. as[i])$ "

However, when showing the completeness lemma, we found out that we have to show the same statement for the new definition, too. Thus, it is not a absolutely better definition.

The proof of the polynomial complexity is also trivial.

Should I include the codes without much description, or leave them unincluded?
Possible description: choice of the complexity and what is done in each step, but would be too lengthy.

4.3. Knapsack and Zero-One Integer Programming

Knapsack and zero-one integer programming are another two classical weighted sum problems. While subset sum was referred to as knapsack in Karp's paper, its definition is nowadays different. Additionally, zero-one integer programming was originally reduced from **Satisfiability**. Nevertheless, there exists a trivial reduction from subset sum to both of the problems. Thus, we include them in this chapter and present a reduction for them each. Since the reduction is short and trivial, the implementation does not have many meaningful details and is thus omitted.

4.3.1. Knapsack

Definition 11 *Knapsack*

Input: A finite set S , a weighting function w , a limiting function b , a upperbound W , a lowerbound B

Output: Is there a subset $S' \subseteq S$ s.t.

$$\begin{aligned} \sum_{x \in S'} w(x) &\leq W \\ \sum_{x \in S'} b(x) &\geq B \end{aligned}$$

We reduce to **Knapsack**. With (S, w, B) as an instance of the subset sum, (S, w, w, B, B) is then an instance of knapsack with

$$\begin{aligned} \sum_{x \in S'} w(x) &= W \\ \sum_{x \in S'} b(x) &= B \end{aligned}$$

where $b = w$ and $W = B$. Apparently, the reduction is constant, for no iteration is necessary.

4.3.2. Zero-one Integer Programming

Definition 12 *Integer Programming*

Input: A finite set X of pairs (x, b) , where x is an m -tuple of integers and b is an integer, an m -tuple x and an integer B

Output: Is there an m -tuple y of integers s.t.

$$\begin{aligned} x^T \cdot y &\leq b \\ c^T \cdot y &\geq B, \forall (x, b) \in X \end{aligned}$$

Although most researchers tend to use matrix for the definition of the zero-one integer programming, we follow the definition from the intractability book(to cite), because it is convenient for our definition and consequently requires less effort. Given an instance of subset sum problem in sequence, (as, s) , let

$$X = \{(as, s)\}, c = as, B = s$$

(X, c, B) is then an instance of the zero-one integer programming problem, for there exists an xs s.t. $xs^T \cdot as = s$. Since there is no iteration, the reduction is constant, too.

5. Chapter

We have successfully formalized the **NP-Hardness** of a few selected classical decision problem. The whole work consists of 3834 lines of codes, with which we add six new problems into the Karp21 project. A general overview of the progress of the list is given in the following graph.

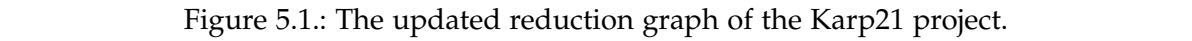
The new reductions from this work are given as red arrows. It is also noticed that a reduction from **Satisfiability** to **3CNF-Satisfiability** is not formalized yet, and is an interesting work in the future. With this work, we manage to show the possibility of formalising and verifying the polynomial reductions and to provide a theoretical basis for other works related to the complexity theory, especially the NP-Hardness.

Future Work

As a future work, it is necessary to add a few new reductions and complete Karp's list of twenty-one NP-hard problems. We summarize the remaining problems as follows,

1. 3cnf-satisfiability, feedback arc set, clique cover. Reductions to these problems are not related to this work. Reductions are strongly dependent on the previous works.
2. 3-dimensional match, steiner tree, job sequencing, max cut. The reductions presented by Karp are dependent on this work. Unfortunately, the definition of these problems are significantly different from the existing problems. Thus, the reductions from the existing problems may not be straightforward and can be replaced by a better one.

Furthermore, a few other classical NP-Hard problems that are not in Karp's list can also be added to Karp21 project. Traveling salesman problem, for example, can be reduced from Hamilton's circuit, while the bin packing problem is also reducible from the partition.



A. Examples for reductions

In this part we present examples for polynomial reductions to assist with the understanding.

A.1. Example for polynomial reduction from Satisfiability To Exact Cover

Input: A logical formula in conjunctive normal form

$$F := (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3)$$

Output: The constructed set is

$$X := \{x_1, x_2, x_3\} \cup \{c_1, c_2, c_3\} \cup \{p_{11}, p_{12}, p_{13}, p_{21}, p_{22}, p_{31}, p_{32}\}$$

The constructed collection is

$$\begin{aligned} S := & \{\{p_{11}\}, \{p_{12}\}, \{p_{13}\}, \{p_{21}\}, \{p_{22}\}, \{p_{31}\}, \{p_{32}\}\} \\ & \cup \{\{c_1, p_{11}\}, \{c_1, p_{12}\}, \{c_1, p_{13}\}, \{c_2, p_{21}\}, \{c_3, p_{31}\}, \{c_3, p_{32}\}\} \\ & \cup \{\{x_1, p_{11}p_{31}\}, \{x_1, p_{21}\}, \{x_2, p_{12}, p_{22}\}, \{x_2\}, \{x_3, p_{32}\}, \{x_3, p_{13}\}\} \end{aligned}$$

Validity: Apparently, the only valid assignment σ of F is given by

$$\sigma = \{x_1 \equiv \perp, x_2 \equiv \top, x_3 \equiv \top\}$$

Wir construct an exact cover S' by

$$S' = \{\{c_1, p_{22}\}, \{c_2, p_{21}\}, \{c_3, p_{32}\}, \{x_1, p_{11}, p_{31}\}, \{x_2\}, \{x_3, p_{31}\}, \{p_{22}\}\}$$

A.2. Example for polynomial reduction from Exact Cover To Subset Sum

Input: The instance of exact cover is given by

$$\begin{aligned} X &:= \{1, 2, 3, 4\} \\ S &:= \{\{1\}, \{2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}\} \end{aligned}$$

Output: While S is not changed, the weighting function w and the sum B is given by

$$w(s) = \sum_{x \in s} 4^x, B = w(X) = 4 + 4^2 + 4^3 + 4^4 = 340$$

Validity: An exact cover S' is given by

$$S' = \{\{1\}, \{2\}, \{3, 4\}\}$$

Hence it holds that

$$w(\{1\}) + w(\{2\}) + w(\{3, 4\}) = 4 + 4^2 + (4^3 + 4^4) = 340 = B$$

A.3. Example for polynomial reduction from Subset Sum To Number Partition

Input: We use the same subset sum entry as in the previous example. (S, w, B) be then converted to

$$\begin{aligned} as &:= [4, 16, 80, 272, 320, 72] \\ s &:= 340 \end{aligned}$$

Output: The reduced bs is then

$$bs := [425, 341, 4, 16, 80, 272, 320, 72]$$

Validity: With $as' = [4, 16, 320]$, the corresponding bs' is

$$\begin{aligned} bs' &= [425, 4, 16, 320] \\ bs - bs' &= [341, 80, 272, 72] \end{aligned}$$

with the equality

$$425 + 4 + 16 + 320 = 765 = 341 + 80 + 272 + 72$$

The other reductions are easier to understand, hence no example is provided here.

List of Figures

2.1. The reduction graph of the Karp21 project	4
5.1. The updated reduction graph of the Karp21 project.	28

List of Tables

2.1. List of commonly encountered complexity classes. 4