For example, if L is reducible to L' and $L' \in \text{P}$ then $L \in \text{P}$. A decision problem is NP-*hard* if every problem in NP is polynomial time reducible to it. A decision problem is NP-*complete* if it is NP-hard and in NP. Actually, the first NP-complete problem [8] has been propositional satisfiability (SAT). Chapter 2 is completely devoted to solving SAT.

1.3.4 Word Grammars

When Gödel presented his undecidability proof on the basis of arithmetic, many people still believed that the construction is so artificial that such problems will never arise in practice. This didn't change with Turing's invention of the Turing machine and the undecidable halting problem of such a machine. However, then Post presented his correspondence problem in 1946 [27] it became obvious that undecidability is not an artificial concept.

Definition 1.3.3 (Finite Word)**.** Given a nonempty alphabet Σ the set Σ^* of *finite words* over Σ is defined by

1. the empty word $\epsilon \in \Sigma^*$
2. for each letter $a \in \Sigma$ also $a \in \Sigma^*$
3. if $u, v \in \Sigma^*$ so $uv \in \Sigma^*$ where uv denotes the concatenation of u and v.

Definition 1.3.4 (Length of a Finite Word)**.** The length $|u|$ of a word $u \in \Sigma^*$ is defined by

1. $|\epsilon| := 0$,
2. $|a| := 1$ for any $a \in \Sigma$ and
3. $|uv| := |u| + |v|$ for any $u, v \in \Sigma^*$.

Definition 1.3.5 (Word Embedding)**.** Given two words u, v, then u is *embedded* in v written $u \sqsubseteq v$ if for $u = a_1 \ldots a_n$ there are words $v_0, \ldots, v_n$ such that $v = v_0 a_1 v_1 a_2 \ldots a_n v_n$.

Reformulating the above definition, a word u is embedded in v if u can be obtained from v by erasing letters. For example, *higman* is embedded in *highmountain*.

Definition 1.3.6 (PCP)**.** Given two finite lists of words $(u_1, \ldots, u_n)$ and $(v_1, \ldots, v_n)$ the *Post Correspondence Problem* (PCP) is to find a finite index list $(i_1, \ldots, i_k)$, $1 \leq i_j \leq n$, so that $u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$.

Take for example the two lists (a, b, bb) and (ab, ab, b) over alphabet $\Sigma = \{a, b\}$. Then the index list $(1, 3)$ is a solution to the PCP with common word abb.

Theorem 1.3.7 (Post 1942)**.** PCP is undecidable.

Lemma 1.3.8 (Higman's Lemma 1952)**.** For any infinite sequence of words $u_1, u_2, \ldots$ over a finite alphabet there are two words u_k, u_{k+l} such that $u_k \sqsubseteq u_{k+l}$.

Proof. By contradiction. Assume an infinite sequence $u_1, u_2, \ldots$ such that for any two words u_k, u_{k+l} they are not embedded, i.e., $u_k \not\sqsubseteq u_{k+l}$. Furthermore, I assume that the sequence is minimal at any word with respect to length, i.e., considering any u_k, there is no infinite sequence with the above property that shares the words up to u_{k-1} and then continues with a word of smaller length than u_k. Next, the alphabet is finite, so there must be a letter, say a that occurs infinitely often as the first letter of the words of the sequence. The words starting with a form an infinite subsequence $au'_{k_1}, au'_{k_2}, \ldots$ where $u_{k_i} = au'_{k_i}$. This infinite subsequence itself has the non-embedding property, because it is a subsequence of the originial sequence. Now consider the infinite sequence $u_1, u_2, \ldots, u_{k_1-1}, u'_{k_1}, u'_{k_2}, \ldots$. Also this sequence has the non-embedding property: if some $u_i \sqsubseteq u'_{k_j}$ then $u_i \sqsubseteq au'_{k_j}$ contradicting that the starting sequence is non-embedding. But then the constructed sequence contradicts the minimality assumption with respect to length, finishing the proof. ☐

Definition 1.3.9 (Context-Free Grammar)**.** A context-free grammar $G = (N_G, T_G, R_G, S_G)$ consists of:

1. a set of non-terminal symbols N_G
2. a set of terminal symbols T_G
3. a set R_G of rules $A \Rightarrow w$ where $A \in N_G$ and $w \in (N_G \cup T_G)^*$
4. a start symbol S_G where $S_G \in N$

For rules $A \Rightarrow w_1$, $A \Rightarrow w_2$ we write $A \Rightarrow w_1 \mid w_2$.

Given a context free grammar G and two words $u, v \in (N_G \cup T_G)^*$ I write $u \Rightarrow v$ if $u = u_1\, A\, u_2$ and $v = u_1\, w\, u_2$ and there is a rule $A \Rightarrow w$ in R_G. The *language* generated by G is $L(G) = \{w \in T_G^* \mid S \Rightarrow^* w\}$, where $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.

A context free grammar G is in *Chomsky Normal Form* [7] if all rules are if the form $A \Rightarrow B_1 B_2$ with $B_i \in N_G$ or $A \Rightarrow w$ with $w \in T_G^*$. It is said to be in *Greibach Normal Form* [15] if all rules are of the form $A \Rightarrow aw$ with $a \in T_G$ and $w \in N_G^*$.

1.4 Orderings

An ordering R is a binary relation on some set M. Depending on particular properties such as

(reflexivity)	$\forall\, x \in M\, R(x,x)$
(irreflexivity)	$\forall\, x \in M\, \neg R(x,x)$
(antisymmetry)	$\forall\, x,y \in M\, (R(x,y) \land R(y,x) \rightarrow x = y)$
(transitivity)	$\forall\, x,y,z \in M\, (R(x,y) \land R(y,z) \rightarrow R(x,z))$
(totality)	$\forall\, x,y \in M\, (R(x,y) \lor R(y,x))$

there are different types of orderings. The relation $=$ is the identity relation on M. The quantifier $\forall$ reads "for all", and the boolean connectives $\land$, $\lor$, and $\rightarrow$ read "and", "or", and "implies", respectively. For example, the above formula stating reflexivity $\forall\, x \in M\, R(x,x)$ is a shorthand for "for all $x \in M$ the relation $R(x,x)$ holds".

Actually, the definition of the above properties is informal in the sense that I rely on the meaning of certain symbols such as $\in$ or $\rightarrow$. While the former is assumed to be known from school math, the latter is "explained" above. So, strictly speaking this book is neither self contained, nor overall formal. For the concrete logics developed in subsequent chapters, I will formally define $\rightarrow$ but here, where it is used to state properties needed to eventually define the notion of an ordering, it remains informal. Although it is possible to develop the overall content of this book in a completely formal style, such an approach is typically impossible to read and comprehend. Since this book is about teaching a general framework to eventually generate automated reasoning procedures this would not be the right way to go. In particular, being informal starts already with the use of natural language. In order to support this "mixed" style, examples and exercises deepen the understanding and rule out potential misconceptions.

Now, based on the above defined properties of a relation, the usual notions with respect to orderings are stated below.

Definition 1.4.1 (Orderings)**.** A *(partial) ordering* $\succeq$ (or simply ordering) on a set M, denoted $(M, \succeq)$, is a reflexive, antisymmetric, and transitive binary relation on M. It is a *total ordering* if it also satisfies the totality property. A *strict (partial) ordering* $\succ$ is a transitive and irreflexive binary relation on M. A strict ordering is *well-founded*, if there is no infinite descending chain $m_0 \succ m_1 \succ m_2 \succ \ldots$ where $m_i \in M$.

Given a strict partial order $\succ$ on some set M, its respective partial order $\succeq$ is constructed by adding the identities $(\succ \cup =)$. If the partial order $\succeq$ extension of some strict partial order $\succ$ is total, then we call also $\succ$ total. As an alternative, a strict partial order $\succ$ is total if it satisfies the strict totality axiom $\forall\, x,y \in M\, (x \neq y \rightarrow (R(x,y) \lor R(y,x)))$. Given some ordering $\succ$ the respective ordering $\prec$ is defined by $a \prec b$ iff $b \succ a$.

Example 1.4.2. The well-known relation $\leq$ on $\mathbb{N}$, where $k \leq l$ if there is a j so that $k + j = l$ for $k, l, j \in \mathbb{N}$, is a total ordering on the naturals. Its strict subrelation $<$ is well-founded on the naturals. However, $<$ is not well-founded on $\mathbb{Z}$.

Definition 1.4.3 (Minimal and Smallest Elements)**.** Given a strict ordering $(M, \succ)$, an element $m \in M$ is called *minimal*, if there is no element $m' \in M$ so that $m \succ m'$. An element $m \in M$ is called *smallest*, if $m' \succ m$ for all $m' \in M$ different from m.

Note the subtle difference between minimal and smallest. There may be several minimal elements in a set M but only one smallest element. Furthermore, in order for an element being smallest in M it needs to be comparable to all other elements from M.

Example 1.4.4. In $\mathbb{N}$ the number 0 is smallest and minimal with respect to $<$. For the set $M = \{q \in \mathbb{Q} \mid 5 \leq q\}$ the ordering $<$ on M is total, has the minimal and smallest element 5 but is not well-founded.

If $<$ is the ancestor relation on the members of a human family, then $<$ typically will have several minimal elements, the currently youngest children of the family, but no smallest element, as long as there is a couple with more than one child. Furthermore, $<$ is not total, but well-founded.

Well-founded orderings can be combined to more complex well-founded orderings by lexicographic or multiset extensions.

Definition 1.4.5 (Lexicographic and Multiset Ordering Extensions)**.** Let $(M_1, \succ_1)$ and $(M_2, \succ_2)$ be two strict orderings. Their *lexicographic combination* $\succ_{\text{lex}} = (\succ_1, \succ_2)$ on $M_1 \times M_2$ is defined as $(m_1, m_2) \succ (m_1', m_2')$ iff $m_1 \succ_1 m_1'$ or $m_1 = m_1'$ and $m_2 \succ_2 m_2'$.

Let $(M, \succ)$ be a strict ordering. The *multiset extension* $\succ_{\text{mul}}$ to multisets over M is defined by $S_1 \succ_{\text{mul}} S_2$ iff $S_1 \neq S_2$ and $\forall m \in M\, [S_2(m) > S_1(m) \rightarrow \exists m' \in M\, (m' \succ m \wedge S_1(m') > S_2(m'))]$.

The definition of the lexicographic ordering extensions can be exapanded to n-tuples in the obvious way. So it is also the basis for the standard lexicographic ordering on words as used, e.g., in dictionaries. In this case the M_i are alphabets, say *a-z*, where $a \prec b \prec \ldots \prec z$. Then according to the above definition *tiger* $\prec$ *tree*.

Example 1.4.6 (Multiset Ordering)**.** Consider the multiset extension of $(\mathbb{N}, >)$. Then $\{2\} >_{\text{mul}} \{1, 1, 1\}$ because there is no element in $\{1, 1, 1\}$ that is larger than 2. As a border case, $\{2, 1\} >_{\text{mul}} \{2\}$ because there is no element that has more occurrences in $\{2\}$ compared to $\{2, 1\}$. The other way round, 1 has more occurrences in $\{2, 1\}$ than in $\{2\}$ and there is no larger element to compensate for it, so $\{2\} \not>_{\text{mul}} \{2, 1\}$.

Proposition 1.4.7 (Properties of Lexicographic and Multiset Ordering Extensions)**.** Let $(M, \succ)$, $(M_1, \succ_1)$, and $(M_2, \succ_2)$ be orderings. Then

1. $\succ_{\text{lex}}$ is an ordering on $M_1 \times M_2$.
2. if $(M_1, \succ_1)$ and $(M_2, \succ_2)$ are well-founded so is $\succ_{\text{lex}}$.
3. if $(M_1, \succ_1)$ and $(M_2, \succ_2)$ are total so is $\succ_{\text{lex}}$.

4. $\succ_{\text{mul}}$ is an ordering on multisets over M.
5. if $(M, \succ)$ is well-founded so is $\succ_{\text{mul}}$.
6. if $(M, \succ)$ is total so is $\succ_{\text{mul}}$.

Please recall that multisets are finite.

The lexicographic ordering on words is not well-founded if words of arbitrary length are considered. Starting from the standard ordering on the alphabet, e.g., the following infinite descending sequence can be constructed: $b \succ ab \succ aab \succ \ldots$. It becomes well-founded if it is lexicographically combined with the length ordering, see Exercise **??**.

Lemma 1.4.8 (König's Lemma)**.** Every finitely branching tree with infinitely many nodes contains an infinite path.

1.5 Induction

More or less all sets of objects in computer science or logic are defined *inductively*. Typically, this is done in a bottom-up way, where starting with some definite set, it is closed under a given set of operations.

Example 1.5.1 (Inductive Sets)**.** In the following, some examples for inductively defined sets are presented:

1. The set of all Sudoku problem states, see Section 1.1, consists of the set of start states $(N; \top; \top)$ for consistent assignments N plus all states that can be derived from the start states by the rules Deduce, Conflict, Backtrack, and Fail. This is a finite set.
2. The set $\mathbb{N}$ of the natural numbers, consists of 0 plus all numbers that can be computed from 0 by adding 1. This is an infinite set.
3. The set of all strings Σ^* over a finite alphabet Σ. All letters of Σ are contained in Σ^* and if u and v are words out of Σ^* so is the word uv, see Section 1.2. This is an infinite set.

All the previous examples have in common that there is an underlying well-founded ordering on the sets induced by the construction. The minimal elements for the Sudoku are the problem states $(N; \top; \top)$, for the natural numbers it is 0 and for the set of strings it is the empty word. Now in order to prove a property of an inductive set it is sufficient to prove it (i) for the minimal element(s) and (ii) assuming the property for an arbitrary set of elements, to prove that it holds for all elements that can be constructed "in one step" out those elements. This is the principle of *Noetherian Induction*.

Theorem 1.5.2 (Noetherian Induction)**.** Let $(M, \succ)$ be a well-founded ordering, and let Q be a predicate over elements of M. If for all $m \in M$ the implication

if $Q(m')$, for all $m' \in M$ so that $m \succ m'$, (induction hypothesis)
then $Q(m)$. (induction step)

is satisfied, then the property $Q(m)$ holds for all $m \in M$.

Proof. Let $X = \{m \in M \mid Q(m) \text{ does not hold}\}$. Suppose, $X \neq \emptyset$. Since $(M, \succ)$ is well-founded, X has a minimal element m_1. Hence for all $m' \in M$ with $m' \prec m_1$ the property $Q(m')$ holds. On the other hand, the implication which is presupposed for this theorem holds in particular also for m_1, hence $Q(m_1)$ must be true so that m_1 cannot be in X - a contradiction. □

Note that although the above implication sounds like a one step proof technique it is actually not. There are two cases. The first case concerns all elements that are minimal with respect to $\prec$ in M and for those the predicate Q needs to hold without any further assumption. The second case is then the induction step showing that by assuming Q for all elements strictly smaller than some m, Q holds for m.

Now for context free grammars. Let $G = (N, T, P, S)$ be a context-free grammar (possibly infinite) and let q be a property of T^* (the words over the alphabet T of terminal symbols of G).

q holds for *all* words $w \in L(G)$, whenever one can prove the following two properties:

1. (*base cases*)
   $q(w')$ holds for each $w' \in T^*$ so that $X ::= w'$ is a rule in P.

2. (*step cases*)
   If $X ::= w_0 X_0 w_1 \dots w_n X_n w_{n+1}$ is in P with $X_i \in N$, $w_i \in T^*$, $n \geq 0$, then for all $w'_i \in L(G, X_i)$, whenever $q(w'_i)$ holds for $0 \leq i \leq n$, then also $q(w_0 w'_0 w_1 \dots w_n w'_n w_{n+1})$ holds.

Here $L(G, X_i) \subseteq T^*$ denotes the language generated by the grammar G from the nonterminal X_i.

Let $G = (N, T, P, S)$ be an *unambiguous* (why?) context-free grammar. A function f is well-defined on $L(G)$ (that is, unambiguously defined) whenever these 2 properties are satisfied:

1. (base cases)
   f is well-defined on the words $w' \in T^*$ for each rule $X ::= w'$ in P.

2. (step cases)
   If $X ::= w_0 X_0 w_1 \dots w_n X_n w_{n+1}$ is a rule in P then $f(w_0 w'_0 w_1 \dots w_n w'_n w_{n+1})$ is well-defined, assuming that each of the $f(w'_i)$ is well-defined.

1.6 Rewrite Systems

The final ingredient to actually start the journey through different logical systems is rewrite systems. Here I define the needed computer science background for defining algorithms in the form of rule sets. In Section 1.1 the rewrite rules Deduce, Conflict, Backtrack, and Fail defined an algorithm for solving 4×4 Sudokus. The rules operate on the set of Sudoku problem states, starting with a set of initial states $(N;\top;\top)$ and finishing either in a solution state $(N;D;\top)$ or a fail state $(N;\top;\bot)$. The latter are called *normal forms* (see below) with respect to the above rules, because no more rule is applicable to a solution state $(N;D;\top)$ or a fail state $(N;\top;\bot)$.

Definition 1.6.1 (Rewrite System)**.** A *rewrite system* is a pair $(M,\rightarrow)$, where M is a non-empty set and $\rightarrow\ \subseteq M \times M$ is a binary relation on M. Figure 1.4 defines the needed notions for $\rightarrow$.

$$\begin{array}{lll}
\rightarrow^0 & = \{\,(a,a) \mid a \in M\,\} & \textit{identity} \\
\rightarrow^{i+1} & = \rightarrow^i \circ \rightarrow & i+1\textit{-fold composition} \\
\rightarrow^+ & = \bigcup_{i>0} \rightarrow^i & \textit{transitive closure} \\
\rightarrow^* & = \bigcup_{i\geq 0} \rightarrow^i \ = \ \rightarrow^+ \cup \rightarrow^0 & \textit{reflexive transitive closure} \\
\rightarrow^= & = \rightarrow \cup \rightarrow^0 & \textit{reflexive closure} \\
\rightarrow^{-1} & = \leftarrow \ = \{\,(b,c) \mid c \rightarrow b\,\} & \textit{inverse} \\
\leftrightarrow & = \rightarrow \cup \leftarrow & \textit{symmetric closure} \\
\leftrightarrow^+ & = (\leftrightarrow)^+ & \textit{transitive symmetric closure} \\
\leftrightarrow^* & = (\leftrightarrow)^* & \textit{refl. trans. symmetric closure}
\end{array}$$

Figure 1.4: Notation on $\rightarrow$

For a rewrite system $(M,\rightarrow)$ consider a sequence of elements a_i that are pairwise connected by the symmetric closure, i.e., $a_1 \leftrightarrow a_2 \leftrightarrow a_3 \ldots \leftrightarrow a_n$. Then a_i is called a *peak* in such a sequence, if actually $a_{i-1} \leftarrow a_i \rightarrow a_{i+1}$.

$\boxed{\texttt{C}}$

Actually, in Definition 1.6.1 I overload the symbol $\rightarrow$ that has already denoted logical implication, see Section 1.4, with a rewrite relation. This overloading will remain throughout this book. The rule symbol $\Rightarrow$ is only used on the meta level in this book, e.g., to define the Sudoku algorithm on problem states, Section 1.1. Nevertheless, these meta rule systems are also rewrite systems in the above sense. The rewrite symbol $\rightarrow$ is used on the formula level inside a problem state. This will become clear when I turn to more complex logics starting from Chapter 2.

Definition 1.6.2 (Reducible)**.** Let $(M,\rightarrow)$ be a rewrite system. An element $a \in M$ is *reducible*, if there is a $b \in M$ such that $a \rightarrow b$. An element $a \in M$ is *in normal form (irreducible)*, if it is not reducible. An element $c \in M$ is a *normal form* of b, if $b \rightarrow^* c$ and c is in normal form, denoted by $c = b{\downarrow}$. Two elements b and c are *joinable*, if there is an a so that $b \rightarrow^* a \ {}^*{\leftarrow}\ c$, denoted by $b \downarrow c$.

Traditionally, $c = b{\downarrow}$ implies that the normal form of b is unique. However, when defining logical calculi as abstract rewrite systems on states in subsequent chapters, sometimes it is useful to write $c = b{\downarrow}$ even if c is not unique. In this case, c is an arbitrary irreducible element obtained from reducing b.

Definition 1.6.3 (Properties of $\rightarrow$)**.** A relation $\rightarrow$ is called

Church-Rosser	if $b \leftrightarrow^* c$ implies $b \downarrow c$
confluent	if $b \;{}^*{\leftarrow}\; a \rightarrow^* c$ implies $b \downarrow c$
locally confluent	if $b \leftarrow a \rightarrow c$ implies $b \downarrow c$
terminating	if there is no infinite descending chain $b_0 \rightarrow b_1 \ldots$
normalizing	if every $b \in A$ has a normal form
convergent	if it is confluent and terminating

Lemma 1.6.4. If $\rightarrow$ is terminating, then it is normalizing.

The reverse implication of Lemma 1.6.4 does not hold. Assuming this is a frequent mistake. Consider $M = \{a, b, c\}$ and the relation $a \rightarrow b$, $b \rightarrow a$, and $b \rightarrow c$. Then $(M, \rightarrow)$ is obviously not terminating, because we can cycle between a and b. However, $(M, \rightarrow)$ is normalizing. The normal form is c for all elements of M. Similarly, there are rewrite systems that are locally confluent, but not confluent, see Figure . In the context of termination the property holds, see Lemma 1.6.6.

Theorem 1.6.5. The following properties are equivalent for any rewrite system $(M, \rightarrow)$:

(i) $\rightarrow$ has the Church-Rosser property.

(ii) $\rightarrow$ is confluent.

Proof. (i) $\Rightarrow$ (ii): trivial.

(ii) $\Rightarrow$ (i): by induction on the number of peaks in the derivation $b \leftrightarrow^* c$. □

Lemma 1.6.6 (Newman's Lemma : Confluence versus Local Confluence)**.** Let $(M, \rightarrow)$ be a terminating rewrite system. Then the following properties are equivalent:

(i) $\rightarrow$ is confluent

(ii) $\rightarrow$ is locally confluent

Proof. (i) $\Rightarrow$ (ii): trivial.

(ii) $\Rightarrow$ (i): Since $\rightarrow$ is terminating, it is a well-founded ordering (see Exercise **??**). This justifies a proof by Noetherian induction where the property $Q(a)$ is "a is confluent". Applying Noetherian induction, confluence holds for all $a' \in M$ with $a \rightarrow^+ a'$ and needs to be shown for a. Consider the confluence property for a: $b \;{}^*{\leftarrow}\; a \rightarrow^* c$. If $b = a$ or $c = a$ the proof is done. For otherwise, the situation can be expanded to $b \;{}^*{\leftarrow}\; b' \leftarrow a \rightarrow c' \rightarrow^* c$ as shown in Figure 1.5. By local confluence there is an a' with $b' \rightarrow^* a' \;{}^*{\leftarrow}\; c'$. Now b', c' are strictly smaller than a, they are confluent and hence can be rewritten to a single a'', finishing the proof (see Figure 1.5). □

$$\begin{tikzcd}
a \arrow[r] \arrow[d] \arrow[dr, phantom, "\text{L.C.}"] & c' \arrow[r, "*"] \arrow[d, dotted, "*"] & c \arrow[dd, dotted, "*"] \\
b' \arrow[r, dotted, "*"] \arrow[d, "*"] \arrow[dr, phantom, "\text{I.H.}"] & a' \arrow[d, dotted, "*"] \arrow[r, phantom, "\text{I.H.}"] & \\
b \arrow[r, dotted, "*"] & d \arrow[r, dotted, "*"] & a''
\end{tikzcd}$$

Figure 1.5: Proof of (ii) $\Rightarrow$ (i) of Newman's Lemma 1.6.6

Lemma 1.6.7. If $\rightarrow$ is confluent, then every element has at most one normal form.

Proof. Suppose that some element $a \in A$ has normal forms b and c, then $b\ {}^*\!\!\leftarrow a \rightarrow^* c$. If $\rightarrow$ is confluent, then $b \rightarrow^* d\ {}^*\!\!\leftarrow c$ for some $d \in A$. Since b and c are normal forms, both derivations must be empty, hence $b \rightarrow^0 d\ {}^0\!\!\leftarrow c$, so b, c, and d must be identical. $\square$

Corollary 1.6.8. If $\rightarrow$ is normalizing and confluent, then every element b has a unique normal form.

Proposition 1.6.9. If $\rightarrow$ is normalizing and confluent, then $b \leftrightarrow^* c$ if and only if $b{\downarrow} = c{\downarrow}$.

Proof. Either using Theorem 1.6.5 or directly by induction on the length of the derivation of $b \leftrightarrow^* c$. $\square$

1.7 Calculi: Rewrite Systems on Logical States

The previous section introduced computational properties of rewrite systems. There, for a rewrite system $(M, \rightarrow)$, the elements of M are abstract. In this section I assume that the elements of M are states including formulas of some logic. If the elements of M are actually such states, then a rewrite system $(M, \rightarrow)$ is also called a *calculus*. In this case, in addition to properties like termination or confluence, properties such as soundness and completeness make sense as well. Although these properties were already mentioned in Section 1.1 they are presented here on a more abstract level.

Starting from Chapter 2 I will introduce various logics and calculi for these logics where the below properties make perfect sense. The Sudoku language is a (very particular) logic as well. It motivates only partly the below notions, because the boolean structure of a Sudoku formula is very simple. It is a conjunction N of equations $f(x, y) = z$ (see Section 1.1). Then a Sudoku formula N is called *satisfiable* if it can be extended to a formula $N \wedge N'$ such that all squares are defined exactly once in $N \wedge N'$ and $N \wedge N'$ represents a Sudoku solution. In this case the formula $N \wedge N'$ is also called a *model* of N. In case the Sudoku formula is not satisfiable the actual derivation $(N; \top; \top) \Rightarrow^* (N; \top; \bot)$ represents a *proof* of unsatisfiability. For example, the Sudoku formula $f(1, 1) = 1 \wedge f(1, 2) = 2 \wedge f(1, 3) = 3 \wedge f(2, 4) = 4$ is unsatisfiable. A Sudoku formula N is *valid* if for any extended formula $N \wedge N'$ such that all squares are defined exactly once in $N \wedge N'$ the formula $N \wedge N'$ represents a Sudoku solution. The Sudoku rewrite system investigates satisfiability.

With respect to the above definitions the only valid Sudoku formulas are actually formulas N where values for all squares are defined in N. For otherwise, for some undefined square an extension N' could just add a value that violates a Sudoku constraint.

As another example consider solving systems of linear equations over the rationals, e.g., solving a system like

$$\begin{aligned} 3x + 4y &= 4 \\ x - y &= 6. \end{aligned}$$

One standard method solving such a system is variable elimination. To this end, first two equations are normalized with respect to one variable, here I choose y:

$$\begin{aligned} y &= 1 - \tfrac{3}{4}x \\ y &= x - 6. \end{aligned}$$

Next the two equations are combined and normalized to an equation for the remaining variables, here x:

$$\tfrac{7}{4}x = 7$$

eventually yielding the solution $x = 4$ and $y = -2$. The below rewrite system describes the solution process via variable elimination. It operates on a set N of equations. The rule Eliminate eliminates one variable from two equations via a combination. The notion $\doteq$ includes the above exemplified normalizations on the equations, in particular, transforming the equations to isolate a variable, and transforming it into a unique form for comparison.

Eliminate $\{x \doteq s, x \doteq t\} \uplus N \;\Rightarrow_{\text{LAE}}\; \{x \doteq s, x \doteq t, s \doteq t\} \cup N$
provided $s \neq t$, and $s \doteq t \notin N$

Fail $\{q_1 \doteq q_2\} \uplus N \;\Rightarrow_{\text{LAE}}\; \emptyset$
provided $q_1, q_2 \in \mathbb{Q}$, $q_1 \neq q_2$

Executing the two rules on the above example with $N = \{3x+4y = 4, x-y = 6\}$ yields:

$$\begin{array}{ll} N & \\ \Rightarrow^{\text{Eliminate}}_{\text{LAE}} & N \cup \{\frac{7}{4}x = 7\}, \\ \Rightarrow^{\text{Eliminate}}_{\text{LAE}} & N \cup \{x = 4, y = -2\}, \end{array}$$

where Eliminate is first applied to y and then to x. Now no more rule is applicable. The rewrite system terminates. It is confluent, because no equations are eliminated from N except for rule Fail that immediately produces a normal form. The rules are sound, because variable elimination is sound and Fail is sound. Any solution after the application of a rule also solves the equations before the application of a rule. So, if the initial system of equations has a solution, the rules will identify the solution. Once the rule set terminates, either $N = \emptyset$ and there is no solution, or a solution is present in the final N. If the original system of equations is not under-determined, N contains an equation $x = q$ for each variable x where $q \in \mathbb{Q}$.

The LAE system is complete, because variable elimination does not rule out any solutions. In general, this can be shown by ensuring that any solution before the application of a rule solves also the equations after application of a rule.

For the system two normalized forms are needed. For the application of Eliminate the two equations are transformed such that the selected variable is isolated. For comparison, the equations are transformed in unique normal form, e.g., in a form $a_1x_1 + \ldots + a_nx_n = q$ where $a_i, q \in \mathbb{Q}$.

The LAE rewrite system can be further improved by adding a subsumption rule removing redundant equations. For example, the rule

Subsume $\{s \doteq t, s' \doteq t'\} \uplus N \ \Rightarrow_{\text{LAE}} \ \{s \doteq t\} \cup N$
provided $s \doteq t$ and $qs' \doteq qt'$ are identical for some $q \in \mathbb{Q}$

deletes an equation if it is a variant of an existing one that can be obtained by multiplication with a constant. Obviously, adding this rule improves the performance of the rewrite system, but now it is no longer obvious that the rewrite system consisting of the rules Eliminate, Subsume, and Fail is confluent, sound, terminating, and complete.

In general, a calculus consists of *inference* and *reduction* rewrite rules. While inference rules add formulas of the logic to a state, reduction rules remove formulas from a state or replace formulas by simpler ones.

A calculus or rewrite system on some state can be *sound*, *complete*, *strongly complete*, *refutationally complete* or *terminating*. Terminating means that it terminates on any input state, see the previous section. Now depending on whether the calculus investigates validity (unsatisfiability) or satisfiability of the formulas contained in the state the aforementioned notions have (slightly) different meanings.

	Validity	Satisfiability
Sound	If the calculus derives a proof of validity for the formula, it is valid.	If the calculus derives satisfiability of the formula, it has a model.
Complete	If the formula is valid, a proof of validity is derivable by the calculus.	If the formula has a model, the calculus derives satisfiability.
Strongly Complete	For any validity proof of the formula, there is a derivation in the calculus producing this proof.	For any model of the formula, there is a derivation in the calculus producing this model.

There are some assumptions underlying these informal definitions. First, the calculus actually produces a proof in case of investigating validity, and in case of investigating satisfiability it produces a model. This in fact requires the specific notion of a proof and a model. Then soundness means in both cases that the calculus has no bugs. The results it produces are correct. Completeness means that if there is a proof (model) for a formula, the calculus could eventually find it. Strong completeness requires in addition that any proof (model) can be found by the calculus. A variant of a complete calculus is a *refutationally complete* calculus: a calculus is refutationally complete, if for any unsatisfiable formula it derives a proof of contradiction. Many automated theorem procedures like resolution (see Section 2.6), or tableaux (see Section 2.4) are actually only refutationally complete.

With respect to the above notions, the Sudoku calculus is complete but not strongly complete for satisfiability.

Historic and Bibliographic Remarks

For context free languages see [2].

Chapter 2

Propositional Logic

A logic is a formal language with a mathematically precise semantics. A formal language is rigidly defined by a grammar and there are efficient algorithms that can decide whether a string of characters belongs to the language or not. The semantics is typically a notion of truth based on the notion of an abstract model. Propositional logic is concerned with the logic of propositions. In propositional logic from the propositions "Socrates is a man" and "If Socrates is a man then Socrates is mortal" the conclusion "Socrates is mortal" can be derived. The logic is expressive enough to talk about propositions, but not, e.g., about individuals. This will be possible in first-order logic (Chapter **??**), a proper extension of propositional logic.

Nevertheless, propositional logic is an interesting candidate for many applications. For example, our overall computer technology is based on propositions, i.e., bits that can either become true or false. The representation of numbers on a computer is based on fixed length bit-vectors rather than on the abstract concept of an arbitrarily large number as known from math. Hardware is designed on a "logical level" that meets to a large extend propositional logic and is, therefore, the currently most well-known application of propositional logic reasoning in computer science.

2.1 Syntax

Consider a finite, non-empty signature Σ of propositional variables, the "alphabet" of propositional logic. In addition to the alphabet "propositional connectives" are further building blocks composing the sentences (formulas) of the language. Auxiliary symbols such as parentheses enable disambiguation.

Definition 2.1.1 (Propositional Formula)**.** The set PROP(Σ) of *propositional formulas* over a signature Σ is inductively defined by:

PROP(Σ)	Comment
$\bot$	connective $\bot$ denotes "false"
$\top$	connective $\top$ denotes "true"
P	for any propositional variable $P \in \Sigma$
$(\neg\phi)$	connective $\neg$ denotes "negation"
$(\phi \land \psi)$	connective $\land$ denotes "conjunction"
$(\phi \lor \psi)$	connective $\lor$ denotes "disjunction"
$(\phi \rightarrow \psi)$	connective $\rightarrow$ denotes "implication"
$(\phi \leftrightarrow \psi)$	connective $\leftrightarrow$ denotes "equivalence"

where $\phi, \psi \in \text{PROP}(\Sigma)$.

The above definition is an abbreviation for setting PROP(Σ) to be the language of a context free grammar $\text{PROP}(\Sigma) = L((N, T, P, S))$ (see Definition 1.3.9) where $N = \{\phi, \psi\}$, $T = \Sigma \cup \{(,)\} \cup \{\bot, \top, \neg, \land, \lor, \rightarrow, \leftrightarrow\}$ with start symbol rules $S \Rightarrow \phi \mid \psi$, $\phi \Rightarrow \bot \mid \top \mid (\neg\phi) \mid (\phi\land\psi) \mid (\phi\lor\psi) \mid (\phi \rightarrow \psi) \mid (\phi \leftrightarrow \psi)$, $\psi \Rightarrow \phi$, and $\phi \Rightarrow P$, for every $P \in \Sigma$.

As a notational convention we assume that $\neg$ binds strongest and we omit outermost parenthesis. So $\neg P \lor Q$ is actually a shorthand for $((\neg P) \lor Q)$. For all other logical connectives parenthesis are explicitly shown if needed. The connectives $\land$ and $\lor$ are actually associative and commutative, see the next Section 2.2. Therefore, the formula $((P \land Q) \land R)$ can be written $P \land Q \land R$ without causing confusion.

The connectives $\land$ and $\lor$ are introduced as binary connectives. They are associative and commutative as already mentioned above. When implementing formulas both connectives are typically considered to be of variable arity. This saves both space and enables more efficient algorithms for formula manipulation.

Definition 2.1.2 (Atom, Literal, Clause)**.** A propositional variable P is called an *atom*. It is also called a *(positive) literal* and its negation $\neg P$ is called a *(negative) literal*. The functions comp and atom map a literal to its complement, or atom, respectively: if $\text{comp}(\neg P) = P$ and $\text{comp}(P) = \neg P$, $\text{atom}(\neg P) = P$ and $\text{atom}(P) = P$ for all $P \in \Sigma$. Literals are denoted by letters L, K. Two literals P and $\neg P$ are called *complementary*. A disjunction of literals $L_1 \lor \ldots \lor L_n$ is called a *clause*. A clause is identified with the multiset of its literals.

The length of a clause C, i.e., the number of literals, is denoted by $|C|$ according to the cardinality of its multiset interpretation. Automated reasoning is very much formula manipulation. In order to precisely represent the manipulation of a formula, we introduce positions.

Definition 2.1.3 (Position)**.** A *position* is a word over $\mathbb{N}$. The set of positions of a formula ϕ is inductively defined by

$$\begin{array}{rcl}
\text{pos}(\phi) & := & \{\epsilon\} \text{ if } \phi \in \{\top, \bot\} \text{ or } \phi \in \Sigma \\
\text{pos}(\neg\phi) & := & \{\epsilon\} \cup \{1p \mid p \in \text{pos}(\phi)\} \\
\text{pos}(\phi \circ \psi) & := & \{\epsilon\} \cup \{1p \mid p \in pos(\phi)\} \cup \{2p \mid p \in \text{pos}(\psi)\}
\end{array}$$

where $\circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}$.

The prefix order $\leq$ on positions is defined by $p \leq q$ if there is some p' such that $pp' = q$. Note that the prefix order is partial, e.g., the positions 12 and 21 are not comparable, they are "parallel", see below. The relation $<$ is the strict part of $\leq$, i.e., $p < q$ if $p \leq q$ but not $q \leq p$. The relation $\parallel$ denotes incomparable, also called parallel positions, i.e., $p \parallel q$ if neither $p \leq q$, nor $q \leq p$. A position p is *above* q if $p \leq q$, p is *strictly above* q if $p < q$, and p and q are *parallel* if $p \parallel q$.

The *size* of a formula ϕ is given by the cardinality of $\operatorname{pos}(\phi)$: $|\phi| := |\operatorname{pos}(\phi)|$. The *subformula* of ϕ at position $p \in \operatorname{pos}(\phi)$ is inductively defined by $\phi|_\epsilon := \phi$, $\neg\phi|_{1p} := \phi|_p$, and $(\phi_1 \circ \phi_2)|_{ip} := \phi_i|_p$ where $i \in \{1,2\}$, $\circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}$. Finally, the *replacement* of a subformula at position $p \in \operatorname{pos}(\phi)$ by a formula ψ is inductively defined by $\phi[\psi]_\epsilon := \psi$, $(\neg\phi)[\psi]_{1p} := \neg\phi[\psi]_p$, and $(\phi_1 \circ \phi_2)[\psi]_{1p} := (\phi_1[\psi]_p \circ \phi_2)$, $(\phi_1 \circ \phi_2)[\psi]_{2p} := (\phi_1 \circ \phi_2[\psi]_p)$, where $\circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}$.

Example 2.1.4. The set of positions for the formula $\phi = (P \land Q) \rightarrow (P \lor Q)$ is $\operatorname{pos}(\phi) = \{\epsilon, 1, 11, 12, 2, 21, 22\}$. The subformula at position 22 is Q, $\phi|_{22} = Q$ and replacing this formula by $P \leftrightarrow Q$ results in $\phi[P \leftrightarrow Q]_{22} = (P \land Q) \rightarrow (P \lor (P \leftrightarrow Q))$.

A further prerequisite for efficient formula manipulation is the notion of the *polarity* of the subformula $\phi|_p$ of ϕ at position p. The polarity considers the number of "negations" starting from ϕ at position ϵ down to p. It is 1 for an even number of explicit or implicit negation symbols along the path, -1 for an odd number and 0 if there is at least one equivalence connective along the path.

Definition 2.1.5 (Polarity)**.** The *polarity* of the subformula $\phi|_p$ of ϕ at position $p \in \operatorname{pos}(\phi)$ is inductively defined by

$$\begin{array}{rcl}
\operatorname{pol}(\phi, \epsilon) & := & 1 \\
\operatorname{pol}(\neg\phi, 1p) & := & -\operatorname{pol}(\phi, p) \\
\operatorname{pol}(\phi_1 \circ \phi_2, ip) & := & \operatorname{pol}(\phi_i, p) \quad \text{if } \circ \in \{\land, \lor\},\ i \in \{1,2\} \\
\operatorname{pol}(\phi_1 \rightarrow \phi_2, 1p) & := & -\operatorname{pol}(\phi_1, p) \\
\operatorname{pol}(\phi_1 \rightarrow \phi_2, 2p) & := & \operatorname{pol}(\phi_2, p) \\
\operatorname{pol}(\phi_1 \leftrightarrow \phi_2, ip) & := & 0 \quad \text{if } i \in \{1,2\}
\end{array}$$

Example 2.1.6. Reconsider the formula $\phi = (A \land B) \rightarrow (A \lor B)$ of Example 2.1.4. Then $\operatorname{pol}(\phi, 1) = \operatorname{pol}(\phi, 11) = -1$ and $\operatorname{pol}(\phi, 2) = \operatorname{pol}(\phi, 22) = 1$. For the formula $\phi' = (A \land B) \leftrightarrow (A \lor B)$ we get $\operatorname{pol}(\phi', \epsilon) = 1$ and $\operatorname{pol}(\phi', p) = 0$ for all other $p \in \operatorname{pos}(\phi')$, $p \neq \epsilon$.

2.2 Semantics

In *classical logic* there are two truth values "true" and "false" which we shall denote, respectively, by 1 and 0. There are *many-valued logics* [31] having more than two truth values and in fact, as we will see later on, for the definition of some propositional logic calculi, we will need an implicit third truth value called "undefined".

Definition 2.2.1 ((Partial) Valuation)**.** A Σ-*valuation* is a map

$$\mathcal{A} : \Sigma \to \{0,1\}.$$

where $\{0,1\}$ is the set of *truth values.* A *partial* Σ-*valuation* is a map $\mathcal{A}' : \Sigma' \to \{0,1\}$ where $\Sigma' \subseteq \Sigma$.

Definition 2.2.2 (Semantics)**.** A Σ-valuation $\mathcal{A}$ is inductively extended from propositional variables to propositional formulas $\phi, \psi \in \mathrm{PROP}(\Sigma)$ by

$$\begin{array}{rcl}
\mathcal{A}(\bot) & := & 0 \\
\mathcal{A}(\top) & := & 1 \\
\mathcal{A}(\neg\phi) & := & 1 - \mathcal{A}(\phi) \\
\mathcal{A}(\phi \land \psi) & := & \min(\{\mathcal{A}(\phi), \mathcal{A}(\psi)\}) \\
\mathcal{A}(\phi \lor \psi) & := & \max(\{\mathcal{A}(\phi), \mathcal{A}(\psi)\}) \\
\mathcal{A}(\phi \to \psi) & := & \max(\{1 - \mathcal{A}(\phi), \mathcal{A}(\psi)\}) \\
\mathcal{A}(\phi \leftrightarrow \psi) & := & \text{if } \mathcal{A}(\phi) = \mathcal{A}(\psi) \text{ then } 1 \text{ else } 0
\end{array}$$

If $\mathcal{A}(\phi) = 1$ for some Σ-valuation $\mathcal{A}$ of a formula ϕ then ϕ is *satisfiable* and we write $\mathcal{A} \models \phi$. In this case $\mathcal{A}$ is a *model* of ϕ. If $\mathcal{A}(\phi) = 1$ for all Σ-valuations $\mathcal{A}$ of a formula ϕ then ϕ is *valid* and we write $\models \phi$. If there is no Σ-valuation $\mathcal{A}$ for a formula ϕ where $\mathcal{A}(\phi) = 1$ we say ϕ is *unsatisfiable.* A formula ϕ *entails* ψ, written $\phi \models \psi$, if for all Σ-valuations $\mathcal{A}$ whenever $\mathcal{A} \models \phi$ then $\mathcal{A} \models \psi$.

Accordingly, a formula ϕ is satisfiable, valid, unsatisfiable, respectively, with respect to a partial valuation $\mathcal{A}'$ with domain Σ', if for any valuation $\mathcal{A}$ with $\mathcal{A}(P) = \mathcal{A}'(P)$ for all $P \in \Sigma'$ the formula ϕ is satisfiable, valid, unsatisfiable, respectively, with respect to a $\mathcal{A}$.

I call the fact that some formula ϕ is satisfiable, unsatisfiable, or valid, the *status* of ϕ. Note that if ϕ is valid it is also satisfiable, but not the other way round.

Valuations of propositional logic collapse with *interpretations.* Given a formula ϕ and an interpretation (valuation) $\mathcal{A}$ such that $\mathcal{A}(\phi) = 1$ then the interpretation $\mathcal{A}$ is also called a *model* for ϕ.

Valuations can be nicely represented by sets or sequences of literals that do not contain complementary literals nor duplicates. If $\mathcal{A}$ is a (partial) valuation of domain Σ then it can be represented by the set $\{P \mid P \in \Sigma \text{ and } \mathcal{A}(P) = 1\} \cup \{\neg P \mid P \in \Sigma \text{ and } \mathcal{A}(P) = 0\}$. Another, equivalent representation are *Herbrand* interpretations that are sets of positive literals, where all atoms not contained in an Herbrand interpretation are false. If $\mathcal{A}$ is a total valuation of domain Σ then it corresponds to the Herbrand interpretation $\{P \mid P \in \Sigma \text{ and } \mathcal{A}(P) = 1\}$.

T

Please note the subtle difference between an Herbrand interpretation and a valuation represented by a set of literals. The latter can be partial with respect to a formula whereas the former is always total by definition. For example, the empty Herbrand interpretation assigns false to all propositional variables.

For example, for the valuation $\mathcal{A} = \{P, \neg Q\}$ the truth value of $P \lor Q$ is $\mathcal{A}(P \lor Q) = 1$, for $P \lor R$ it is $\mathcal{A}(P \lor R) = 1$, for $\neg P \land R$ it is $\mathcal{A}(\neg P \land R) = 0$, and the status of $\neg P \lor R$ cannot be established by $\mathcal{A}$. In particular, $\mathcal{A}$ is a partial valuation for $\Sigma = \{P, Q, R\}$. A literal L is *defined* with respect to a partial valuation $\mathcal{A}$ if $L \in \mathcal{A}$ or $\operatorname{comp}(L) \in \mathcal{A}$.

Example 2.2.3. The formula $\phi \lor \neg\phi$ is valid, independently of ϕ. According to Definition 2.2.2 we need to prove that for all Σ-valuations $\mathcal{A}$ of ϕ we have $\mathcal{A}(\phi \lor \neg\phi) = 1$. So let $\mathcal{A}$ be an arbitrary valuation. There are two cases to consider. If $\mathcal{A}(\phi) = 1$ then $\mathcal{A}(\phi \lor \neg\phi) = 1$ because the valuation function takes the maximum if distributed over $\lor$. If $\mathcal{A}(\phi) = 0$ then $\mathcal{A}(\neg\phi) = 1$ and again by the before argument $\mathcal{A}(\phi \lor \neg\phi) = 1$. This finishes the proof that $\models \phi \lor \neg\phi$.

Theorem 2.2.4 (Deduction Theorem)**.** $\phi \models \psi$ iff $\models \phi \to \psi$

Proof. $(\Rightarrow)$ Suppose that ϕ entails ψ and let $\mathcal{A}$ be an arbitrary Σ-valuation. We need to show $\mathcal{A} \models \phi \to \psi$. If $\mathcal{A}(\phi) = 1$, then $\mathcal{A}(\psi) = 1$, because ϕ entails ψ, and therefore $\mathcal{A} \models \phi \to \psi$. For otherwise, if $\mathcal{A}(\phi) = 0$, then $\mathcal{A}(\phi \to \psi) = \max(\{(1 - \mathcal{A}(\phi)), \mathcal{A}(\psi)\}) = \max(\{(1, \mathcal{A}(\psi)\}) = 1$, independently of the value of $\mathcal{A}(\psi)$. In both cases $\mathcal{A} \models \phi \to \psi$.
$(\Leftarrow)$ By contraposition. Suppose that ϕ does not entail ψ. Then there exists a Σ-valuation $\mathcal{A}$ such that $\mathcal{A} \models \phi$, $\mathcal{A}(\phi) = 1$ but $\mathcal{A} \not\models \psi$, i.e., $\mathcal{A}(\psi) = 0$. By definition, $\mathcal{A}(\phi \to \psi) = \max(\{(1 - \mathcal{A}(\phi)), \mathcal{A}(\psi)\}) = \max(\{(1 - 1), 0\}) = 0$, hence $\phi \to \psi$ does not hold in $\mathcal{A}$. $\square$

So both writings $\phi \models \psi$ and $\models \phi \to \psi$ are actually equivalent. I extend the former notion to sets or sequences on the left denoting conjunction. For example, $\chi, \phi \models \psi$ is short for $\chi \land \phi \models \psi$.

Proposition 2.2.5. The equivalences of Figure 2.1 are valid for all formulas ϕ, ψ, χ.

T

Note that the formulas $\phi \land \psi$ and $\psi \land \phi$ are equivalent. Nevertheless, recalling the problem state definition for Sudokus in Section 1.1 the two states $(N; f(2,3) = 1 \land f(2,4) = 4; \top)$ and $(N; f(2,4) = 4 \land f(2,3) = 1; \top)$ are significantly different. For example, it can be that the first state can lead to a solution by the rules of the algorithm where the latter cannot, because the latter implicitly means that the square $(2,4)$ has already been checked for all values smaller than 4. This reveals the important point that arguing by logical equivalence in the context of a rule set manipulating formulas, a calculus, can lead to wrong results.

Lemma 2.2.6 (Formula Replacement)**.** Let ϕ be a propositional formula containing a subformula ψ at position p, i.e., $\phi|_p = \psi$. Furthermore, assume $\models \psi \leftrightarrow \chi$. Then $\models \phi \leftrightarrow \phi[\chi]_p$.

Proof. By induction on $|p|$ and structural induction on ϕ. For the base step let $p = \epsilon$ and $\mathcal{A}$ be an arbitrary valuation.

$$\begin{aligned}
\mathcal{A}(\phi) &= \mathcal{A}(\psi) && \text{(by definition of position)}\\
&= \mathcal{A}(\chi) && \text{(because } \mathcal{A} \models \psi \leftrightarrow \chi\text{)}\\
&= \mathcal{A}(\phi[\chi]_\epsilon) && \text{(by definition of replacement)}
\end{aligned}$$

For the induction step the lemma holds for all positions p and has to be shown for all positions ip. By structural induction on ϕ, I show the cases where $\phi = \neg\phi_1$ and $\phi = \phi_1 \to \phi_2$ in detail. All other cases are analogous.

If $\phi = \neg\phi_1$ then showing the lemma amounts to proving $\models \neg\phi_1 \leftrightarrow \neg\phi_1[\chi]_{1p}$. Let $\mathcal{A}$ be an arbitrary valuation.

$$\begin{aligned}
\mathcal{A}(\neg\phi_1) &= 1 - \mathcal{A}(\phi_1) && \text{(expanding semantics)}\\
&= 1 - \mathcal{A}(\phi_1[\chi]_p) && \text{(by induction hypothesis)}\\
&= \mathcal{A}(\neg\phi[\chi]_{1p}) && \text{(contracting semantics)}
\end{aligned}$$

If $\phi = \phi_1 \to \phi_2$ then showing the lemma amounts to proving the two cases $\models (\phi_1 \to \phi_2) \leftrightarrow (\phi_1 \to \phi_2)[\chi]_{1p}$ and $\models (\phi_1 \to \phi_2) \leftrightarrow (\phi_1 \to \phi_2)[\chi]_{2p}$. Both cases are similar so I show only the first case. Let $\mathcal{A}$ be an arbitrary valuation.

$$\begin{aligned}
\mathcal{A}(\phi_1 \to \phi_2) &= \max(\{(1 - \mathcal{A}(\phi_1)), \mathcal{A}(\phi_2)\}) && \text{(expanding semantics)}\\
&= \max(\{(1 - \mathcal{A}(\phi_1[\chi]_p)), \mathcal{A}(\phi_2)\}) && \text{(by induction hypothesis)}\\
&= \mathcal{A}((\phi_1 \to \phi_2)[\chi]_{1p}) && \text{(applying semantics)}
\end{aligned}$$

□

Lemma 2.2.7 (Polarity Dependent Replacement)**.** Consider a formula ϕ, position $p \in \operatorname{pos}(\phi)$, $\operatorname{pol}(\phi, p) = 1$ and (partial) valuation $\mathcal{A}$ with $\mathcal{A}(\phi) = 1$. If for some formula ψ, $\mathcal{A}(\psi) = 1$ then $\mathcal{A}(\phi[\psi]_p) = 1$. Symmetrically, if $\operatorname{pol}(\phi, p) = -1$ and $\mathcal{A}(\psi) = 0$ then $\mathcal{A}(\phi[\psi]_p) = 1$. If $\operatorname{pol}(\phi, p) = 1$ and $\mathcal{A}(\psi) = 1$ then $\mathcal{A}(\phi) = \mathcal{A}(\phi[\psi]_p)$.

Proof. Exercise **??**: by induction on the length of p. □

Note that the case for the above lemma where $\operatorname{pol}(\phi, p) = 0$ is actually Lemma 2.2.6.

C

The equivalences of Figure 2.1 show that the propositional language introduced in Definition 2.1.1 is redundant in the sense that certain connectives can be expressed by others. For example, the equivalence Eliminate $\to$ expresses implication by means of disjunction and negation. So for any propositional formula ϕ there exists an equivalent formula ϕ' such that ϕ' does not contain the implication connective. In order to prove this proposition the above replacement lemma is key.