Generalised Iterations of Gödel-sentence

Jonas Gjesvik
Master’s Thesis, Spring 2016
The front page depicts a section of the root system of the exceptional Lie group $E_8$, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.
Acknowledgements

To Dag who helped me forward and patiently explained things I should have understood already.

To my family who are always there for me.

To my friends who keep me sharp and blunt in equal measure.

And to Ros who believes in me.
# Contents

1 Introduction ............................................................................................................................ 7  
   1.1 Overview ............................................................................................................................. 7  
   1.2 Notations ............................................................................................................................. 9  
   1.3 Trees .................................................................................................................................. 10  
   1.4 Language and logic ............................................................................................................. 12  
      1.4.1 The Language .............................................................................................................. 12  
      1.4.2 The Logic ..................................................................................................................... 13  
      1.4.3 Set Theory .................................................................................................................. 14  
   1.5 Kleene's $\mathcal{O}$ and $\omega^1_{CK}$ ..................................................................................... 15  
   1.6 Hyperarithmetical sets ....................................................................................................... 16  

2 Normal Iteration ......................................................................................................................... 19  
   2.1 Basic Iteration ..................................................................................................................... 19  
      2.1.1 Incompleteness ............................................................................................................ 19  
      2.1.2 Iteration ...................................................................................................................... 20  
   2.2 Generalisations .................................................................................................................... 22  
      2.2.1 Ordinals ...................................................................................................................... 22  
      2.2.2 Kleene's $\mathcal{O}$ ......................................................................................................... 25  
   2.3 Iteration for $\mathcal{O}^*$ ......................................................................................................... 26  
      2.3.1 Introducing $\mathcal{O}^*$ ................................................................................................. 26  
      2.3.2 Exploring $\mathcal{O}^*$ .................................................................................................... 28  
      2.3.3 Iteration through $\mathcal{O}^*$ ....................................................................................... 30  

3 Meta-recursive Iteration ............................................................................................................. 33  
   3.1 Overview ............................................................................................................................. 33  
   3.2 Definitions and Explanation ............................................................................................... 33  
      3.2.1 The Language .............................................................................................................. 33  
      3.2.2 The Logic .................................................................................................................... 35  
      3.2.3 The Model ................................................................................................................... 36  
   3.3 Gödel-numbering ................................................................................................................. 38  
      3.3.1 Formulas ...................................................................................................................... 38  

5
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2</td>
<td>Proofs</td>
<td>41</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Incompleteness</td>
<td>44</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Iterations</td>
<td>46</td>
</tr>
<tr>
<td>3.4</td>
<td>Iteration for $O^{HYP}$</td>
<td>47</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Definition and Iteration</td>
<td>47</td>
</tr>
<tr>
<td>3.4.2</td>
<td>A completeness result</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>$\alpha$-recursion on admissible ordinals</td>
<td>53</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
<td>57</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Overview

Theorem 1 (Proposition VI (Gödel)) To every ω-consistent recursive class \( c \) of formulae there correspond recursive class-signs \( r \), such that neither \( vGenr \) nor \( \text{Neg}(vGenr) \) belongs to \( Flg(c) \) (where \( v \) is the free variable of \( r \)).

This is the wording of Gödel’s famous Incompleteness Theorem, showing that any recursive set of axioms cannot be complete in the sense that all sentences of first order arithmetic will be provable. The search for such a foundation was headed by the famous mathematician David Hilbert who in 1900 formulated 23 problems facing the mathematical society at the time. One of these was to prove the consistency of the axioms of arithmetic. A consequence of Gödel’s result was that no consistent axiom system can prove its own consistency (this is often referred to as Gödel’s Second Incompleteness Theorem). So to prove that the axioms of arithmetic are consistent one would need a strictly stronger axiom system, thus raising the question of the consistency of this new axiom system and so on. Gerhard Gentzen, for instance, showed that the consistency of the axioms of arithmetic follows from the well-foundedness of \( \varepsilon_0 \) the least ordinal closed under exponentiation, it is said "Gentzen is the man who proved the consistency of induction up to \( \omega \), by induction up to \( \varepsilon_0 \)". This is belittling the work of Gentzen as his result is quite interesting, but it shows the absurdity of any definite consistency proof. For an introduction to the history of Hilbert’s formalism and its supporters and adversaries see Mancosu [13].

Gödel’s result gives rise to an interesting notion which is the subject of this paper. If one has a recursive (or even recursively enumerable) set of axioms
for which the natural numbers is a model, assuming this system is consistent, one can then find a sentence that is unprovable yet true for the natural numbers. Such a sentence is often called a Gödel-sentence. It is a fact that these Gödel-sentences are equivalent to the sentence saying that the set of axioms are consistent. This new sentence can then safely be added as an axiom, this will not create an inconsistency since the sentence is true for the natural numbers. One then has a new set of axioms strictly stronger than the last. And again the Gödel’s result shows that a new sentence can be found that is unprovable yet true. One can in such a way continue to iterate this process for all natural numbers, getting stronger and stronger axiom sets each time.

Taking the limit supremum of these sets of axioms gives a new set of axioms strictly stronger than all the ones gained from finite iteration. This new set will be consistent given the consistency of all the others, which again only hinges on the consistency of the initial set of axioms. One can associate this new set of axioms with the ordinal $\omega$ as this ordinal is defined as the supremum of all finite ordinals in the same way the axiom set is the supremum of all finite extensions of the axiom set. Then again, as this set will be recursively enumerable, a new Gödel-sentence can be found and the iteration continues into the transfinite. Turing [7] wrote in his text about transfinite iterations in this manner, and I will also do so in this text. Turing expressed the hope that at some point in the iteration, constructible in a way defined later in the text, all sentences will be provable, thus making the union of all these iterations a complete axiom set. This turned out not to be true as any progression of axiom sets, with only one added sentence at each iteration, will be incomplete in the same way, but the complexity of the Gödel-sentence may increase as the iteration goes past computable. Gödel showed that any computable/recursive set of axioms has a $\Pi_1$ Gödel-sentence, but if one goes far enough in a non-recursive ordering, for instance certain branches in Kleene’s $O$ which I will come back to, then any true $\Pi_1$-sentence can be shown to be provable. So one can in a computable way prove any $\Pi_1$-sentence, but one cannot have a computable set of axioms proving all $\Pi_1$-sentences.

The rest of this section will be given to introducing notation and basic ideas and results needed for the study later in the text. In the second section the ideas expressed here will be given more consideration. I will show not only that this iteration is possible through many different ordinals and other relations, but I will use recursion theory to show that the iteration process will be computable. Computable in the sense that a program on a computer could, given finite time, do this process and come up with sets of axioms and Gödel-sentences. The actual mathematical value of a Gödel sentence is limited, a
1.2. NOTATIONS

Gödel-sentence only expresses its own unprovability and can not be used to for instance prove the Riemann Hypothesis. But the theoretical implications are interesting, as it shows the extent of the Incompleteness Theorem. I will end the section with Proposition 3 showing that for any initial set of axioms of arithmetic there is an amount of extensions of this set that is in one-one correspondence with the rational numbers, i.e. a countably dense set of sets of axioms!

In the third section I will introduce a new setting for the same kind of considerations. I will introduce an infinitary extension of the language $\mathcal{L}$, called $\mathcal{L}^{HY_\omega}$. For this language I will need an extension of the logic used in section two, this extension will be natural in that it follows the intuition about what the extended language expresses. I will use meta-recursion in place of recursion. Meta-recursion is an extension of recursion that allows for infinite computations, therefore the finite numbers are no longer a natural model for the language and I will have to extend the model as well. This will be done by considering an initial segment of Gödel’s constructible universe. Then in this new setting I will prove an incompleteness result similar to the one in the classical recursion theory setting, and I will use meta-recursion to provide a iteration of the same kind as in Chapter 2. This meta-recursive iteration will naturally be able to go much further than in the recursive case, as I will show, yet many results can be lifted from normal recursion, as is often the case with meta-recursion.

In the fourth section I will write a short explanation of how and why these results can be generalized to any $\Sigma_1$ admissible ordinal. This will not be discussed in much detail, as I think this is too great a generalization to give really interesting results, as one generalizes results one often dilutes the usability of the results.

1.2 Notations

In this text I will use a lot of notation from different sources and different parts of logic. Most of my notation is taken from Sacks [1], Normann [3], Leary and Kristiansen [4] and Girard [6].

I will use the symbols $\neg$, $\land$ and $\lor$ to denote the basic logical building blocks of not, and, or respectively. The arrows $\iff$, $\rightarrow$ and $\leftrightarrow$ will stand for implication and equivalence as normal. $\equiv$ will be used as a symbol for definition, that means that the mathematical object on the left of $\equiv$ will be defined as
the expression on the right.

Formulas will most often be represented by the Greek letters $\phi$, $\psi$ and $\theta$. Sometimes I will use a different name for essential formulas, but then I will specify the naming in the text. I will also use indexing on formulas, like $\theta_3$, when it seems natural. Greek letters will also be used as notation for ordinals, but mostly different letters, $\alpha$, $\beta$ and $\lambda$ mostly. I believe the context will make the over-reliance on the Greek alphabet untroublesome.

I will assume the reader has some knowledge of basic Gödel numbering of formulas. $\overline{\phi}$ will stand for the numbering of $\phi$. In later chapters I will consider a different numbering when extending the language, but the same notation will be used.

From recursion theory I will use the notation $\{e\}$ for the computable function with $e$ as its numbering in some enumeration of the computable function. $\{e\}^f$ will mean the function with numbering $e$ computable from the function (or set) $f$. $\mathcal{W}_e$ will denote the range of the computable function $\{e\}$. $T(e, x, y)$ is Kleene's T-predicate, $T(e, x, y)$ will hold true if and only if $\{e\}(x) \downarrow$, i.e. terminates, and $y$ is a code for a computation-tree of $\{e\}(x)$. Computation-trees will be explained further in the next section. If $y$ is a code for a computation-tree then there exists a primitive recursive function, say $P$, such that $P(y) = \{e\}(x)$. Kleene's T-predicate can be defined for any number of inputs, but it is well known that any finite input function can be viewed as a single input function using a pairing trick. The predicate will be computable, in fact primitively recursive.

The notation $(x)_i$ will stand for the $i$'th element of $x$, so if $x = <x_1, x_2, ..., x_i>$ then $(x)_3 = x_3$. If $x$ is not on this form then the notation is meaningless and $x_i = \emptyset$.

1.3 Trees

Most people with some knowledge of mathematics have come across trees in some form; be it choice trees in probability or as an undirected graph in graph theory. Trees are a very basic and intuitive concept in this part of mathematics so I will rely heavily on intuition when it comes to using them in proofs, but I will nonetheless give a short introduction. Here I will regard
trees as a certain ordering on sets. The point is that the top node of the tree will be the greatest element of the ordering, and any element that is further down a branch of the tree will be lesser in the ordering.

To be precise:

**Definition 1** A tree will be a non-empty ordering \( \leq \) with a greatest element \( 0 \). Such that the ordering is transitive and anti-symmetric. And

\[ \forall x, y [y \leq x \lor x \leq y \lor \neg \exists z(z \leq x \land z \leq y)] \]

A mathematical tree as an ordering is best visualised as a tree-like structure with the greatest element, \( 0 \), at the bottom and vertices as branches and vertexes as branch-points. In this paper I will allow infinite branching at any vertex, which will lead to trees with branches of arbitrary finite length. Later I will also consider trees with infinite length, for instance Kleene’s \( \mathcal{O} \).

A tree is only interesting when there are incomparable nodes, if not the tree will be a linear and thus a sequence as well. I will also use trees where the natural ordering will have \( 0 \) as its least element rather than greatest element, this will do no harm as the ordering will just be reversed. In that case the important property of being a tree-ordering will be the fact that any segment of the ordering bounded by an element will be totally ordered. In layman’s terms this just means that when two branches split apart, they cannot grow back together, which is an essential property of a tree.

A decorated tree is a tree with a value given to each vertex, or node. This will be useful when considering proof-trees later in the paper and is also used in recursion theory. In computation-trees the value on each node is the computation that is being called on, and the nodes below (or above depending on which way you view it) are the immediate sub-computations needed. For instance in a computation there might at some point be need to call for several (maybe even infinite) sub-computations when calling a function. This can be viewed as a branching in the computation-tree, where every sub-computation gets its own sub-tree starting at the node with value corresponding to the original call. The same kind of structure is applied in a proof-tree. Each step in a proof has some hypothesis-formulas and a consequent-formula, the last consequent, i.e. the formula being proved, will be the top node, and the hypothesis-formulas of the last inference will be the nodes of the top-node in the immediate sub-trees, i.e. the nodes directly below the top-node. In this way it is easy to see the connection between computations and proofs. A computation will terminate if its computation-tree is well-founded and in the same way a proof will be deemed acceptable if it has a well-founded
CHAPTER 1. INTRODUCTION

proof-tree. In Section 3 I will allow for some restricted class of infinite computations and inferences which will lead to infinite branching in the respective trees, this will however not remove the well-foundedness property of the trees.

In computations (and proofs) it is important that each computation is finite since a computation is supposed to be able to give an output in finite time, in a computation-tree this is the same as demanding that any branch is of finite length. In section 2 this means that any tree is finite, but in section 3 the allowance of infinite branching will allow for the possibility of infinite trees where any branch is of finite length. I will also use an ordering, Kleene's $\mathcal{O}$, that can be seen as a tree, but here the branching will not be finite.

1.4 Language and logic

1.4.1 The Language

In this text I will use the first order language $L$. $L$ will have countably many variables, I will not necessarily be very strict in using the correct notation for the variables, but it will always be clear from context what is and what is not a variable. The language will have a constant, 0, one unary function symbol, $S$, and one binary relation symbol, $=$, all with their usual meanings.

The terms of the language will be 0 and $S(t)$ for $t$ a term, so each natural number has a canonical term representing it. The atomic formulas of the language will be: $t_1 = t_2$ for terms $t_1$ and $t_2$. Any other formula will be built from the atomic formulas by use of the logical symbols $\neg$, $\lor$ and $\land$ and also the quantifier symbols; $\forall$ and $\exists$, all with their usual meaning. Implication, equivalence and bounded quantification ($\rightarrow$, $\leftrightarrow$ and $\forall x \in u$ and $\exists x \in u$ respectively) will be used as if they are basic symbols in the language, but for simplicity of induction I will assume they are defined from the others in the usual way. This "cheating" will sometimes be extended to $\land$ or $\lor$ as well as one can be defined using the other and negation. Generally, when using induction I will try to limit the amount of clauses I will have to prove, this is to not let the text be too technical and let the natural intuition of the arguments shine through.

It is usual to distinguish between variables ranging over elements and variables ranging over sets. Lower-case letters ($x, y, z$...) are often used for
variables of elements and upper-case (X, Y, Z...) for variables of sets. This is done in order to have a measure of the complexity of a given formula. Firstly one calls a formula $\Delta_0$ if all its quantifiers are bounded, a formula is $\Sigma_1$ if it is of the form $\exists x \phi$ where $\phi$ is a $\Delta_0$-formula and $x$ is a variable ranging over elements, similarly defined are $\Pi_1$-formulas with universal quantifiers over elements. Then a formula is defined to be $\Sigma_{n+1}$ if it is of the form $\exists x \psi$ and $\psi$ is a $\Pi_n$-formula, a symmetrical definition gives the $\Pi_{n+1}$ formulas. A formula is $\Delta_n$ if it is both $\Sigma_n$ and $\Pi_n$. Any formula using only variables ranging over elements and not over sets can be written as a $\Sigma_n$ or $\Pi_n$ formula for some $n$. This is called the normal form of the formula, all quantifiers are pushed to the left of the formula and are of varying form. An example:

$$\phi \equiv \forall x \exists y [y = S(x)]$$

$\phi$ is a $\Pi_2$ formula saying that any element has a successor.

In some part of this text I will make use of an extension of the language using set quantifiers. Introducing set quantifiers creates a new hierarchy of formulas, the second-order formulas. A formula is said to be arithmetic if it can be written using only element quantifiers and is analytical if it also uses function quantifiers. One gives a superscript to indicate if it is arithmetical or analytical, so $\Sigma_3^0$ formulas are of the form $\exists x \forall y \exists z \psi$ with $\psi$ a $\Delta_0$-formula, while a $\Sigma_3^1$ formula is of the form $\exists X \forall Y \exists Z \psi$ where $\psi$ here is an arithmetic formula.

A last note on complexities of formulas, I will often use function quantifiers instead of set quantification. This is equivalent as any set can be defined by its characteristic function, and any function can be defined explicitly by a set of pairs. This characterisation of formulas only makes sense if you have a preferred structure in mind, as the distinction of elements and sets only makes sense in this case. The intended model for these formulas will be the natural numbers, so $x, y, z...$ ranges over $\mathbb{N}$ and $X, Y, Z...$ ranges over $P(\mathbb{N}) = 2^\mathbb{N}$.

### 1.4.2 The Logic

The logic I will use in connection with this language is ordinary first order logic. All predicate tautologies are assumed true. Tautologies are sentences that are true for any valuation of the predicates involved. In propositional
logic predicates are atomic formulas or anything within the scope of a quantifier. For instance $\neg[p \land (\neg p \lor q)] \lor q$ is a tautology. It can be written as $[p \land (p \rightarrow q)] \rightarrow q$ also known as modus ponens.

Other rules in this logic will concern themselves with the quantifiers, the rules will be:

$$\phi(t) \vdash \exists x \phi(x)$$
$$\forall x \phi(x) \vdash \phi(t)$$
$$[\psi \rightarrow \phi] \vdash [\psi \rightarrow \forall x \phi(x)] \text{ when } x \text{ is not free in } \psi$$
$$[\phi \rightarrow \psi] \vdash [\exists x \phi(x) \rightarrow \psi] \text{ when } x \text{ is not free in } \psi$$

This is the normal way to introduce quantifiers. The two first clauses are just the intuitive way of using quantifiers; if you have an example where $\phi$ holds, then $\exists x \phi(x)$ holds, and if $\forall x \phi(x)$ holds then certainly $\phi$ holds for a specific term. The two other are to be thought of as proving without making assumptions on $x$, this is captured in that $x$ is not free in $\psi$. If $x$ is not free in $\psi$ in the third clause, then for any term, $t$, the implication will be the same as $t$ is not substituted in for $x$ in $\psi$. The same type of argument shows the validity of the fourth clause. I will not go into more detail on types of logic, and assume the reader has some knowledge of this system or an equivalent system like Peano Arithmetic (PA).

### 1.4.3 Set Theory

Set theory is in a unique position when it comes to languages of mathematics, as any other language and branch of mathematics can be defined inside set theory. Any model and set used in this paper can therefore be thought of as inside $V$, the universe of sets. I will have need for set theory at some points in this text, but I will not be too rigorous in its introduction here. An introduction to set theory will take to long and is not necessary for the limited use of set theory in this paper. Rather I will refer to Kunen for a thorough introduction. The language of set theory is the same as $\mathcal{L}$ only with the added relation $\in$ which should be known to most as the relation saying an object is an element of another. The axioms of set theory I will not list here, and will not need in any explicit way. That is if all mathematics is viewed inside set theory I will have implicit use for it, but it wont feature in my arguments.
1.5 Kleene's $\mathcal{O}$ and $\omega^C_1$

Kleene's $\mathcal{O}$ is a way to define $\omega^C_1$. $\omega^C_1$ is the first non-constructive ordinal. Constructive ordinals are defined through successor and recursive limits. So any ordinal less than $\omega^C_1$ can be reached in a computable way. This is the motivation for the definition of $\mathcal{O}$. $\mathcal{O}$ will be a hierarchy of notations for ordinals, using the enumeration of recursive functions and the fact that $\omega^C_1$ is countable all notations can be defined to be natural numbers. $\omega^C_1$ is a well-ordering so it can be seen as a sequence, $\mathcal{O}$ on the other hand will be better viewed as a tree, since different notations for the same ordinal will be incomparable.

The following definition is taken from Sacks [1]. (In the definition (u) is the universal quantification over the variable u)

**Definition 2 (Kleene's $\mathcal{O}$ (2.1 in Sacks))** The formula $x <_\mathcal{O} y$ is to be read: $x$ and $y$ are notations for constructive ordinals and $x$ is less than $y$ according to the ordering of notations. The ordering $<_\mathcal{O}$ is not linear, because the same ordinal may have two different notations.

The predicate $x <_\mathcal{O} y$, regarded as a set of ordered pairs, is the closure of a finite set under $\Sigma^1_1$ closure condition as described in the remarks following section 1.6.

The closure condition $A(X)$ has three clauses.

1. $(u)(v)[(u,v) \in X \rightarrow \langle v, 2^v \rangle \in X]$

2. $(n)[e(n) \text{ is defined} \& \langle \{e\}(n), \{e\}(n+1) \in X \rightarrow (n)[\{\{e\}(n), 3 \cdot 5^e \rangle \in X]$

3. $(u)(v)(w)[\langle u,v \rangle, \langle v,w \rangle \in X \rightarrow \langle u,w \rangle \in X]$

Breaking down the definition a little, Kleene's $\mathcal{O}$ is the closure of a finite set under three conditions. The set $X$ is the explicit definition of the ordering, that means that $\langle u,v \rangle \in X$ is the same as $u <_\mathcal{O} v$. In Sacks' definition the number one is associated with the ordinal zero, i.e. $\emptyset$, and $2^v$ is the immediate successor of $v$. It is sufficient to let the finite starting set be $\{\langle 1,2 \rangle\}$ which is to be associated with the fact that $0 <_\mathcal{O} 1$. Then the first closure condition makes sure that $\mathcal{O}$ is closed under succession, and therefore that $\omega \subset \mathcal{O}$. The second condition means that the ordering is closed under recursive limits. The recursiveness of the limits is an important distinction, if one
would take the closure over arbitrary limits one would get the ordering with
height the same as the ordinals, this would not be as useful as this ordering.
In addition it is this condition that makes Kleene’s \( \mathcal{O} \) well-founded. The
recursive limits will split the ordering into different paths as two different
limits will be incomparable, so \( \mathcal{O} \) is not a linear ordering, and hence not a
well-ordering. It might be instructive to look at \( \mathcal{O} \) as a three and each recursive
limit as the start of a new branch. Thus \( \omega \) has infinitely many notations
and in the same way, at the height of each limit ordinal the ordering will
branch out according to the different limits. The last condition is to secure
that the ordering is transitive.

\( \mathcal{O} \) will be \( \Pi^1_1 \). As stated in the definition \( \mathcal{O} \) is a \( \Sigma^1_1 \)-closure of a finite set.
What that means is that if I let \( A(x) \) be the disjunction of the clauses for \( \mathcal{O} \)
then the least set satisfying these clauses, called the closure of \( A \), will be \( \Pi^1_1 \)
because it will be equivalent to:

\[
x \in \mathcal{O} \iff \forall X [ A(X) \to x \in X]
\]

It can be showed, Sacks [1], that the height of \( \mathcal{O} \) is \( \omega^{CK}_1 \). Intuitively it
makes sense as the ordering is closed under recursive limits, and any ordinal
with a notation in \( \mathcal{O} \) can be reached recursively. \( \mathcal{O} \) will have branches that
are shorter than \( \omega^{CK}_1 \). In fact it will have branches as short as \( \omega^2 \), because
one can make a branch where any increasing computable function with in-
dices e will be a limit of \( \omega \cdot n \) for some \( n \). Then the limit will be \( \omega^2 \), but it
will not be computable. It is important to note that for any element, \( x \in \mathcal{O} \),
the restriction of the ordering to elements less than \( x \), i.e. \( \{ y | y <_\mathcal{O} x \} \), is a
recursive well-ordering.

1.6 Hyperarithmetical sets

A finite set of natural numbers can be viewed as the set enumerated at a
finite stage of a computation, i.e. the set \( \{ x | T(e, x, y) \land y < n \} \) is finite for
any \( n \). A hyperarithmetical set can in a similar fashion be viewed as the set
of natural numbers enumerated at some step in a transfinite computation of
length \( < \omega^{CK}_1 \), so hyperarithmetical sets is in some way an extension of the
finite sets. This analogy is imperfect, but is a useful for meta-recursive the-
ory. Hyperarithmetical sets are very important in general recursion theory,
and also in higher order logic.

The hyperarithmetical sets have three different equivalent definitions, and I
will make use of all of them. I will therefore explain here in enough detail
so that a person with a background in logic, but no introduction to hyper-
arithmetical, will be able to understand my further use of the notion. The
easiest and shortest way of defining a hyperarithmetical set is as a set defin-
able by a $\Delta_1^1$ formula of set theory (or any equivalent language). This gives
an idea of the complexity and range of the hyperarithmetical sets, but it does
not give much insight into exactly what it is and what it is useful for. I will
now give two other definitions, given in the texts Sacks [1] and Normann [3]
respectively.

Sacks first defines a family of set he calls H-sets. These are created by
iterating the Turing-jump through Kleenes $O$. The Turing jump is a way
to move through the Turing degrees of recursion theory. A function is said
to be recursive in another function (one can use sets or their characteristic
functions as well) then one defines one function to be of lesser or equal Tur-
ing degree than another if it is recursive in the other. The Turing jump of
a function is then defined to be the union of all the functions recursive in
that function (again if you prefer using sets it is essentially the same). It is
shown that the Turing jump of a function or set is of a higher Turing de-
gree than the function or set, i.e. $X <^T X'$ where $X'$ is the Turing jump of $X$.

Then Sacks defines the H-sets inductively by iterating the Turing jump
through Kleenes $O$ starting with the empty set and using the clauses written
in the definition above for successor and recursive limits. Finally the hyper-
arithmetical sets are defined to be the ones recursive in some H-set. So in a
way the hyperarithmetical sets can be seen as recursive set in some higher
form of recursiveness where some infinite computations are allowed in using
the given H-set in computations.

The other definition is from Normann’s text on recursion theory [3]. The
idea here is that the hyperarithmetical sets are an extension of the arith-
metical sets using a restricted search through all functions. Let
$^2E(f) = 1$ if $\exists n \ f(n) \neq 0$

$^2E(f) = 0$ if $\forall n \ f(n) = 0$

So $^2E$ is a functional that will be non-zero if and only if the input function is non-zero, notice that $^2E$ is undefined if the input function is zero for some values and undefined for others. Then a function is defined to be hyperarithmetical if it can be written as $\{e\}^2E(x)$ for some $e$. Again hyperarithmeticality can be seen as a form of limited infinite computation, using $^2E$ as a sort of restricted existential quantifier.

These are the three different definitions of hyperarithmetical I will use. I will not go into proving their equivalence, the two other definitions are shown equivalent to $\Delta_1^1$, but I will refer to Sacks [1] and Normann [3] for these proofs.
Chapter 2
Normal Iteration

2.1 Basic Iteration

2.1.1 Incompleteness

Theorem 2 (Incompleteness version III (Leary p. 296 [4])) There exists a primitive recursive function $\iota$ with the following property. If $A$ is a consistent set of $L_{NT}$-axioms and $e$ is an index such that $W_e = \{ \ulcorner \eta \urcorner \mid A \vdash \eta \}$ then $\iota(e)$ is the Gödel number of a $\Pi_1$-sentence $\theta$ such that $\mathfrak{N} \theta$ and $A \nvdash \theta$.

If one compares this with the original statement from Gödel’s paper cited at the beginning of the introduction there are several differences. The result has been refined several times over the years, but the most important for my argument in this section is the existence of the function $\iota$. I will hereby refer to this function as $\text{Ind}$, it is so called because it is a function manipulating indexes of functions and formulas. Let $e$ be an index for a computable function that gives as its output all provable sentences in the given set of axioms (hereby referred to as a theory). Then $\text{Ind}$ takes $e$ as input and gives as output the Gödel number of a $\Pi_1$ Gödel-sentence for the theory, i.e. a sentence true in the normal model $\mathfrak{N}$ yet unprovable in the theory. Roughly speaking $\text{Ind}$ takes as input all provable sentences of a set of axioms and outputs a unprovable sentence.

When one considers the goal of this function, to construct a computable function iterating Gödel-sentences, one sees the importance of $\text{Ind}$. It will be the heart of the machinery, it will generate new sentences to be added to the theory. I will not go into too much detail of the proof of this version of incompleteness, but I will dwell a bit on the thought behind the proof since
The idea is to use recursion theory to show incompleteness. One can easily define a strictly recursively enumerable set $K$, i.e. a recursively enumerable set that is not computable, by diagonalizing over all computable functions. Since $K$ is recursively enumerable but not computable its complement, $(K)^c$, will not be recursively enumerable since a set is computable if and only if both it and its complement is recursively enumerable. After that it is shown that any recursively enumerable set can be defined by a $\Sigma_1$-formula, with the immediate consequence that there is a $\Pi_1$-formula, $\psi(x)$ that defines $\bar{K}$. Since formal proofs are computable there will then be at least one $a \in (K)^c$ such that one cannot prove $\psi(a)$ since if not $(K)^c$ would be recursively enumerable.

This is the way to prove incompleteness with recursion theory and with one more lemma the version of incompleteness stated above can be proved:

**Lemma 1 (Lemma 7.7.6 (Leary))** Let $\theta(x)$ be a $L_{NT}$-formula, and let $A$ be a set of $L_{NT}$-axioms. There is a primitive recursive function $j$ with the following property: If $W_e = \{\lfloor \eta \rfloor | A \vdash \eta \}$, then $W_{j(e)} = \{a | A \vdash \theta(a)\}$

Then one lets the $\theta$ in the lemma be the defining formula of $\bar{K}$. This will be a sufficient explanation of the incompleteness theorem for my purposes, but for a full (and humorous) introduction to the proof I will refer the reader to Leary [4].

### 2.1.2 Iteration

Now back to the construction! I have the $Ind$ function, but I will need some other small functions to put everything together. $Ind$ is the function that takes the next step in the iteration, so I just have to make sure that $Ind$ gets the right input. The next piece I need is a function that takes as input $e_1$ an index for the set of axioms from $A$, i.e. $W_{e_1} = \{\lfloor \theta \rfloor | \theta \in A \}$ and outputs the index, $e_2$, for $W_{e_2} = \{\lfloor \eta \rfloor | A \vdash \eta \}$. I will call it $Prov$ because it is essentially a proof checking machine in the spirit of Hilbert. It takes in axioms and outputs all the theorems that can be proved from these axioms. The idea
that formal proofs is computable are easy to grasp (it is kind of the point of formal proofs), but I will show it in a bit more detail, relying heavily on the introduction I got to recursion theory in [3] and [4].

The important fact is that for the right Gödel numbering provability will be \( \Sigma_1 \)-definable, i.e. there will exist a \( \Sigma_1 \)-relation on natural numbers \( \text{Prv}(x) \) that will be true exactly when \( x \) is the Gödel number of a provable sentence for the given theory. Since a formula is recursively enumerable exactly when it is \( \Sigma_1 \) this will make provability recursively enumerable. Now let \( e_p \) be the index of the computable function that represents \( \text{Prv}(x) \) in the theory \( W_{e_1} \), so the domain of \( \{ e_p \} \) is the indexes of all the provable sentences in \( A \). Let \( \pi_1(x) \) and \( \pi_2(x) \) be two functions that split natural numbers into unique pairs (these functions can be shown to be computable see for instance Normann [3]), and let \( a \in A \). Now let \( e_2(x) = \pi_1(x) \) if \( T(e_p, \pi_1(x), \pi_2(x)) \) and \( e_2(x) = a \) if not. This function will have as its range all the provable sentences in \( A \), hence \( W_{e_2} = \{ \langle \eta \rangle \mid A \vdash \eta \} \). It will also be computable since all its components are so. So if I let \( \text{Prov}(e_1) = e_2 \) I get the function desired for this paragraph. \( \text{Prov}(x) \) can been shown to be computable since it only uses computable functions and operations.

I will need another function I will call \( \text{Add} \), but it will not require much explanation as I feel its computability is intuitive. \( \text{Add} \), as the name suggests, will add one more sentence to a set of sentences, I will use it to add the newly created Gödel-sentences to the theory. More formally \( \text{Add}(e, \langle \theta \rangle) = e \) where \( W_e = \{ \langle \theta \rangle \mid \theta \in A \} \) and \( W_{e+} = \{ \langle \theta \rangle \mid \theta \in A + \psi \} \).

I will use this version of the recursion theorem for the basic iteration:

**Theorem 3 (Theorem 3.2.39 in [Leary [4]])** Let \( f(e, \bar{x}) \) be a partial, computable function.

Then there is an index \( e_0 \) such that for all \( \bar{x} \)

\[
\phi_{e_0}(\bar{x}) \simeq f(e_0, \bar{x})
\]

Note that the theorem does not imply that the promised function will be non-empty, this will usually have to be shown by an inductive argument.

Now it is time to add all these parts (\( \text{Ind} \), \( \text{Prov} \) and \( \text{Add} \)) together and use the recursion theorem stated above to tie it all up. First let \( \text{It}(e, 0) = e_A \), where
$e_A$ is the index of the theory $A$, and let $It(e, a + 1) = Add(It(e, a), \phi_e(a))$. $It(e, a)$ is a function that outputs a code for a set of sentences $A \cup \{\phi_e(b) \mid b < a\}$. If I now get the input $e$ to be the function outputting Gödel-sentences I will be done. Now let $f(e, a) = Ind(Prov(It(e, a)))$, then its easy to see that $f(e, 0)$ is a code for a Gödel-sentence of $A$ for any input $e$, while $f(e, a)$ is a Gödel-sentence for $A \cup \{\phi_e(b) \mid b < a\}$ so using the recursion theorem the input will be code for the function itself so $A \cup \{\phi_e(b) \mid b < a\}$ will in fact be the set of axioms of $A$ plus all the previously iterated Gödel-sentences which will give the desired result, henceforth called $A_a$ in all circumstances. $f$ is the composition of three computable functions it is itself computable so I can use the recursion theorem stated above. So I let

$$Göd(a) = \phi_{e_0}(a) = f(e_0, a) = Ind(Prov(It(e_0, a)))$$

and the function is created. Now lets test it on the first few iterations. $Göd(0) = Ind(Prov(It(e_0, 0)))$ Now $It(e_0, 0)$ will output $e_A$ and $Prov$ and $Ind$ will together create an index for a Gödel-sentence for $A$, as wanted. The next iteration becomes: $Göd(1) = Ind(Prov(It(e_0, 1)))$. Now $It(e_0, 1) = Add(It(e_0, 0), \phi_{e_0}(0))) = Add(It(e_0, 0), Göd(0)) = e_A$. Which is exactly what is needed for $Prov$ and $Ind$ to produce the next Gödel-sentence, so the construction works!

This also shows that the function is non-empty and in fact total. The reason is that for any input the successful running of the function is only dependent on computations of inputs of a lesser value (lesser in the natural ordering of natural numbers), and for input 0 the function is not calling on itself at all. This is what I mean by showing the totality of the function by an inductive argument. I get this proposition:

**Proposition 1** Given a recursively enumerable set of axioms, $A$, with the natural numbers as a model, and a computable way of coding formulas. Then there exists a total computable function, $f$, with the following properties. $f(n)$ will be the code of a $\Pi_1$-sentence that is a true sentence for the natural numbers and unprovable for $A_n$

## 2.2 Generalisations

### 2.2.1 Ordinals

$Göd$ from the previous section iterates Gödel-sentences using the natural numbers with their natural ordering. In this section I will discuss other or-
2.2. GENERALISATIONS

Derivings one can use for iterations. I will here use terminology and results from ordinal theory some of which is covered in the introduction, but for a complete introduction to ordinals see Kunen [8] and for recursive ordinals see Sacks [1]. The construction of Göd can be made with iteration through a whole host of different orderings. I will in this section show why this is possible for many of them and pick out a few examples where I show more explicitly how the construction works. I will start with ordinals up to $\omega_1^{CK}$. Then I will show that I can do the same for a restricted class of well-founded orderings. Finally I will introduce a more complex ordering $O^*$, which will not be well-founded, but sufficiently restricted so that the construction will work in this case as well (and for other orderings like it). All this will show the great complexity one can find in expanding theories with $\mathcal{N}$ as a model.

Iterating through ordinals should be pretty intuitive, successor ordinals are dealt with normally, and limit ordinals are taken as the union of all previous ordinals. And this is indeed the idea of it. Since I want to make a computable function I am forced to only consider recursive ordinals, that is ordinals, $\alpha < \omega_1^{CK}$. I also need to use Effective Transfinite Recursion.

**Theorem 4 (Effective Transfinite Recursion Theorem 3.2 (Sacks [1]))**

Let $<_R$ be a wellfounded relation whose field is a subset of $\omega$, and $I : \omega \to \omega$ a recursive function. For all $e < \omega$ and $x$ in the field of $<_R$, $e(x,y)$ defined for all $y < _R x$ implies $I(e)(x)$ defined. Then for some $c$, $c(x)$ is defined for all $x$ in the field of $<_R$, and $\{c\} \simeq \{I(c)\}$.

Effective Transfinite Recursion (ETR) is a way of using recursion on more complex orderings than on $\omega$. If one looks at the description of ETR one sees that the conditions on the ordering are: it is well-founded, and that its field is a subset of $\omega$ (as a set not an ordering). The last requirement can be seen as the requirement that the ordering is countable, because if its field is a subset of $\omega$ it is obviously countable, and if it is countable one can map it into $\omega$. So any recursive, countable ordinal will satisfy the requirements and it will be possible to iterate through such an ordinal. In fact the use of ETR is not restricted to recursive orderings, as seen in the definition. This may seem surprising, but ETR has very general requirements and I will show how complex the orderings can be when using ETR. The iteration process does not necessarily proceed in the way I showed when testing Göd in the last section, rather it computes using indexes of previous iterations, this makes
the infinite iterations computable.

One requirement not mentioned in the theorem, but which will always be the case in my examples, and which seem necessary for the construction is that for any element, $x$, in the domain of the ordering the restriction of the ordering to elements less than $x$ is recursive. I call this restricted ordering an initial segment below $x$. This is tied up to the fact that the iteration function, $I$, must be recursive. As I will use elements below $x$, to compute at $x$ the need for the initial segment below $x$ to be recursive follows.

Iteration through any recursive ordinal is now easily seen to be recursive, as the ordering of any recursive ordinal satisfies all the criterion of ETR. In fact it will work for $\omega_1^{CK}$ as well, since any initial segment of $\omega_1^{CK}$ will be recursive. I will use a little time to make sure the construction works. When I look at the construction I have made:

$$G^\odot(a) = \phi_{e_0}(a) = f(e_0, a) = \text{Ind}(\text{Prov}(It(e_0, a)))$$

I will need to translate this into the language of Sacks, since the definition in the ETR-theorem is different from the one I used for basic iteration. The difference is that in the version I used the function has two inputs, with the intention that one of the inputs will be an index for the function itself. Sacks version is about using the function $I$ as an iterator on the index of the function, i.e. $e$ is a function and $I(e)$ is a different function using the number $e$ as a input, then $e_0$ is a fix-point of the iterator so $\{I(e_0)\} = \{e_0\}$. These two definitions are equal as $\{I(e)\}$ can be seen as the function, $f(e, x)$ where $x$ is a free variable. I let $\{I(e)\}(a) = \text{Ind}(\text{Prov}(It(e, a)))$ and get an $e_0$ such that $\{e_0\}(a) = \{I(e_0)\}(a) = \text{Ind}(\text{Prov}(It(e_0, a))) = G^\odot(a)$, so I assume that $\{e\}(b)$ gives a Gödel-sentence for all $b < \omega a$ and then $\{I(e)\}(a)$ gives a new Gödel-sentence.

By the ETR-theorem I generally have to prove that for all computable functions, $e$, if $e(y)$ is defined for all $y <_R x$, where $<_R$ is the ordering in question, then $I(e)(x)$ is defined. This implication is hard to prove as I would have to consider how every computable function defined on all $y <_R x$ would react when put in to my function. And that would require a more technically detailed definition than I feel necessary. Fortunately the fix-point is already guaranteed without this, the requirement is only there to make sure, through an induction argument, that the fix-point function is non-empty. When one
looks at my function, f, it is easily seen that the function will be non-empty as the case where a = 0 is independent of the other input, i.e. f(e,0) is the same for any e. So when e_0 is a fix-point f(e_0,0) will be defined, but then f(e_0,S(0)) (the immediate successor of 0, assuming any random ordering) will be defined as

\[ f(e_0, S(0)) = Ind(Prov(Add(It(e_0,0), \phi_{e_0}(0)))) \]

\[ = Ind(Prov(Add(e_A, \phi_{e_0}(0)))) \]

And \( \phi_{e_0}(0) \) is defined by the previous argument. And so one can see that for any element, a, this goes through as long as it works for all previous elements in the ordering and the initial segment below a is recursive. Especially it works for ordinals. Note for this argument I have to assume the existence of a least element, 0, of the ordering.

2.2.2 Kleene’s \( \mathcal{O} \)

I have already explained why the construction will work for ordinals up to \( \omega_1^{CK} \) using Effective Transfinite Recursion, and it is easily seen that the same argument works in the case of \( \mathcal{O} \). From the definition used here it is clear that \( \mathcal{O} \) is a subset of \( \omega \), and it is well-founded as every branch is the order of an ordinal \( \leq \omega_1^{CK} \). Therefore the construction will work in the case of Kleene’s \( \mathcal{O} \) as well. I have this proposition:

**Proposition 2** For a given \( \Sigma_1 \)-set of axioms, A, and a computable coding of formulas there exists a partial computable function, \( \text{Göd}_\mathcal{O} \) with the following properties. If \( x \in \mathcal{O} \) then \( \text{Göd}_\mathcal{O}(x) \) gives the code of a Gödel-sentence for \( A_x \).

The domain of the function \( \text{Göd}_\mathcal{O} \) is bigger than \( \mathcal{O} \), this follows since the domain of any computable function is recursively enumerable and \( \mathcal{O} \) is certainly not. I will come back to this fact in the next section. Turing showed that for any \( \Pi_1 \) sentence an iteration process through \( \mathcal{O} \) will eventually arrive at a set of axioms that can prove the sentence, and in fact the process only needs to go to \( \omega + 1 \). Feferman later proved that for any element, a, of \( \mathcal{O} \) if \( A_a \) is the axiom set associated with a, then an element, b, of \( \mathcal{O} \) with \( b <_\mathcal{O} a \) such that the sentence is provable for \( A_b \) and \( |b| = |a| + \omega + 1 \). So for any element of \( \mathcal{O} \) a point can be found on a branch extending a where the given \( \Pi_1 \) sentence can be proved. See Feferman [11] for more results of the
sort. He also shows that even for \( \bigcup_{a \in \mathcal{O}} A_a \) one can find a \( \Pi_2 \) sentence that is true for the natural numbers and unprovable from the axioms. \( \bigcup_{a \in \mathcal{O}} A_a \) is of course not recursively enumerable. Feferman also considers adding many more sentences so that in fact the resulting iteration process through \( \mathcal{O} \) becomes complete in the sense that any first order sentence will have a proof at some point in the process, but to get this result Feferman has to consider a much stronger iteration process than adding a single sentence at each point. This in turn makes the seemingly ground breaking result more trivial, but still interesting.

### 2.3 Iteration for \( \mathcal{O}^* \)

#### 2.3.1 Introducing \( \mathcal{O}^* \)

So far I have only considered well-founded orderings as the ordering for iterating Gödel-sentences. In this section I will describe a non-well-founded ordering that I will show can be used for an iterative progression of Gödel-sentences. It is an ordering that in a way is locally a copy of \( \mathcal{O} \). The idea is that as \( \mathcal{O} \) is the smallest ordering closed under succession, transitivity and recursive limits, this ordering is in some way the largest such ordering with the property that any initial segment below any element will be recursively enumerable. This is the motivation for calling it \( \mathcal{O}^* \).

I will describe \( \mathcal{O}^* \) in two steps; first I will use countably infinite steps to create \( \mathcal{O}^+ \), and then I will remove the bits I do not want from \( \mathcal{O}^+ \) to make \( \mathcal{O}^* \). I start with all the natural numbers, and define an ordering on all numbers like this:

1. \( x <_{\mathcal{O}^*} 2^x \)

2. \( \exists m [ x <_{\mathcal{O}^*} \{e\}(m) ] \rightarrow x <_{\mathcal{O}^*} 3 \cdot 5^e \)

3. \( x <_{\mathcal{O}^*} y \land y <_{\mathcal{O}^*} \rightarrow x <_{\mathcal{O}^*} z \)

Note the similarities with the ordering of \( \mathcal{O} \), in fact on \( \mathcal{O} \) this ordering be the same as \( <_{\mathcal{O}} \). The point is that I now have an ordering extending \( <_{\mathcal{O}} \) to more elements, then the idea is then to remove enough numbers to create an
2.3. Iteration for $O^*$

Ordering similar to $O$ yet bigger and more complex. I do this first by using
infinitely many steps removing numbers, like this:

1. $O_0^+ = \mathbb{N}$

2. $x \in O_{n+1}^+ \iff x \in O_n^+ \land [x = 1 \lor (x = 2^y \land y \in O_n^+) \lor (x = 3 \cdot 5^e \land \{e\} \text{ is total} \land \forall m \in \mathbb{N}(\{e\}(m) < O^* \{e\}(m+1) \land \{e\}(m) \in O_n^+)]$

Then I let:

$x \in O^+ \iff \forall m (x \in O_n^+)$

So I start with all natural numbers and then for each step I remove all
numbers that are not successors or recursive limits of elements from the pre-
vious step, then $O^+$ will be the intersection of all $O_n^+$. This is in contrast to
the building of $O$ where I started with a finite set and then added numbers
according to the clauses. This makes sure that $O \subseteq O^+$ since an upward clo-
sure of positive properties will be the least set closed under these properties
containing the starting set. $O^+$ will also be closed under the same properties,
and since the starting set for $O$ is a finite set this is contained in $O^+$. What
is also clear from the definition is that $O^*$ is arithmetic, which is needed to
show the complexity of $O^*$ later.

Then I want to remove more from $O^+$. Specifically, I want to remove any
hyperarithmetical infinite downward sequence, the logic behind this require-
mint is to make the infinite downward sequences that are left in $O^*$ be so
complex that a failure to compute G"{o}d cannot occur, since such a failure will
necessitate a hyperarithmetical downward sequence. Using the definitions of a
hyperarithmetical function from Normann [3], I let:

$e \in A \iff \forall n(\{e\}(2^E,n) \downarrow) \land \forall n ((\{e\}(2^E,n) > O^* \{e\}(2^E,n+1))$

Which is to say; in $A$ lies all the numbers that will be indexes of total
hyperarithmetical downward sequences. $A$ will be $\Pi^1_1$ since the sub-formula
$\{e\}(2^E,n) \downarrow$ is $\Pi^1_1$, this by the definition of a meta-computable function.
Then I can create $O^*$ by removing all elements from $O^+$ that is a part of
such a infinite downward hyperarithmetical sequence. I also make sure that
for any element the restriction of the ordering to that element is total. This
is important since it makes it much easier to talk about branches when $O^*$
is viewed as a tree. Remember a tree should only branch downwards not
upwards ($rng(\{e\})$ is here defined as the range of the function with index $e$,
i.e. $x \in \text{rng}(x) \iff \exists y(\{e\}(y) = x)$. Formally the definition for $O^*$ will then be:

$$x \in O^* \leftrightarrow x \in O^+ \land (\{<u,v> | u <_{O^*} v <_{O^*} x\} \text{ is a total ordering})$$

$$\land \forall e \in A \rightarrow x \notin \text{rng}(e)$$

The set given in this formula, $\{<u,v> | u <_{O^*} v <_{O^*} x\}$, is the initial segment of $O^*$ below $x$. It is essential that this restriction gives a total-ordering for any element of $O^*$, this will make $O^*$ have some of the same properties as $O$, and will make the use of recursion possible as any element will be uniquely determined by its restriction, as is seen crucial in the use of ETR. Since $e \in A$ is $\Pi^1_1$, $O^*$ will be $\Sigma^1_1$. $O$ is obviously a subset of $O^*$, by the construction of these orderings. $O^*$ is closed under succession and recursive limits, and $O$ is the smallest such ordering. The fact that $O^*$ is $\Sigma^1_1$ also proves that $O \subset O^*$. This follows from the fact that any $\Pi^1_1$ set can be mapped into $O$, i.e. $O$ is a complete $\Pi^1_1$ set. Thus if $O = O^*$ every $\Pi^1_1$ set would also be $\Sigma^1_1$, which is impossible. $O$ is defined as the closure of all sets closed under transitivity, successor and recursive limits, $O^*$ can similarly be shown to be the closure of all hyperarithmetic sets under the same closure conditions [14].

2.3.2 Exploring $O^*$

To get it started I will explain more thoroughly and formally what I mean when I say that $O^*$ locally looks like $O$. When looking at the definition of $O^*$, any element has a successor and $O^*$ is closed under recursive limits just like $O$. So from any element of $O^*$, there will be a local ordering isomorphic with $O$. What I will also show is that if I follow a branch of $O^*$ out of the local copy of $O$ I will come upon an infinite downward sequence. Now to show the order type of branches in $O^*$. Let $a$ be any element of $O^*$:

$$x \sim_a y \leftrightarrow (x, y <_{O^*} a \land \text{there are no infinite descending sequence between } x \text{ and } y)$$
2.3. **Iteration for $O^*$**

$\sim_a$ is easily seen to be reflexive, symmetric and transitive for any $a \in O^*$ and is thus an equivalence relation on the initial segment below $a$. Then let: $[x] = \{y | x \sim_a y\}$. $[x]$ will be a total-ordering around $x$ and below $a$. I call this the interval around $x$ (under $a$).

**Claim 1** $\forall x \ | \ [x] = \omega_1^{CK}$ unless $x \sim_a a$.

**Proof:** If $x \sim_a a$ then obviously the well-ordered interval around $x$ can be extended by considering a $b \in O^*$ greater than $a$, when I have proved that any interval that is not the last indeed has the order type $\omega_1^{CK}$ then any last interval must have order type $< \omega_1^{CK}$.

Assume the whole interval lies beneath $a$. Then it is totally ordered as the ordering, $<_{O^*}$, restricted to any element of $O^*$, is a total ordering. First, I will show that it contains a least element. Now a first thought might be: since $x \sim y$ if and only if there are no infinite descending sequences, then $[x]$ cannot contain any infinite descending sequences. But this argument is false, as even if there are no infinite descending sequence between $x$ and $y$ if $x \sim y$, there is always the possibility that I can find a new element, $z$, less than both $x$ and $y$ with $z \sim x$ and $z \sim y$, and so continue like this to infinity.

I will show that this leads to a contradiction with the essential property of $O^*$, namely that there are no hyperarithmetical downward infinite sequences.

First I claim that any interval is $\Pi^1_1$. This is easily seen because $O^*$ restricted to an initial segment is recursively enumerable. To check that there is no infinite downward sequence between $x$ and $y$, one must use one universal quantifier over all functions. And it follows since $\Pi^1_1$ is closed under universal quantification over functions.

Then I can use a uniformization principle discussed in Sacks [1]. A uniformization principle is a way of choosing an unique next element in an ordering. The principle says that if $Q(x,y)$ is a $\Pi^1_1$ ordering, then one can find a $\Pi^1_1$ ordering $P(x,y)$ that uniforms $Q(x,y)$. In other words, if I have a $\Pi^1_1$ ordering then choosing a unique next element in that ordering is $\Pi^1_1$. This is exactly what I need, because I can then from an element, $a_0$ of the interval, $[x]$ (remember it is $\Pi^1_1$), choose an unique element less than $a_0$, say $a_1$ in a $\Pi^1_1$ way. Then continuing in such a way using the recursion theorem I get a $\Pi^1_1$-function which is an infinite downward sequence in $<_{O^*}$. And any $\Pi^1_1$-function is in fact $\Delta^1_1$ i.e. hyperarithmetical. This is a contradiction, hence there must be a least element of any interval.
Now, since there is a least element, it is clear that each interval is in fact well ordered, so there is some ordinal that has an isomorphic ordering to any interval. If \( |x| < \omega^C_{CK} \) then \([x]\) will be \( \Delta_1 \) since it will be recursive in some ordinal less than \( \omega^C_{CK} \). Then anything under a and over the interval will also be \( \Delta_1 \), since it can be defined as \( y \notin [x] \land x < \_ \). This set will not have a least element by the construction of \([x]\). Then I can construct a hyperarithmetical downwards sequence of elements over \([x]\) in the same way as in the last paragraph, which leads to a contradiction. By the way \( \_ \) is built it is clear that \( \omega^C_{CK} < |x| \), since it is closed under successors and recursive limits. I can then conclude that \( |x| = \omega^C_{CK} \). ■

Now, consider two different intervals, \([x]\) and \([y]\), on the same branch of \( \_ \). If these two intervals are not the same there has to be an infinite descending sequence between them, it follows that there is an interval, \([z]\), between these two intervals. Thus the intervals are densely packed, and they are obviously countable, thus the ordering of intervals are of the same order as the non-negative rationals, \( \mathbb{Q}^+ \cup \{0\} \).

Then any branch in \( \_ \) either terminates as a recursive ordinal, or has the order type; \( \omega^C_{CK} \cdot \eta + \gamma \) where \( \eta \) is the ordering of the non-negative rationales and \( \gamma \) is a recursive ordinal. The fact that I can make a computable function, that will iterate theories through this ordering, as I will show next, is quite remarkable. Not only is it not recursive, it is not even meta-recursive.

### 2.3.3 Iteration through \( \_ \)

To show that the construction of \( \text{Göd}_\_ \) works on \( \_ \) requires an extra argument. \( \_ \) is not well-founded so the argument in the proof of ETR from Sacks does not hold. But as I have previously noted, the function is independent of the ordering at the first step, and the fixed point from the theorem is guaranteed for any ordering where the iteration function, \( I \), is recursive. The further restrictions on the ordering is only needed to show that the fixed point is total. So I can use a sort of induction argument to show that a failure of the function at a point leads to a contradiction.

A word about the initial segments of \( \_ \) is in order. A prerequisite for the construction of \( \text{Göd}_\_ \) is that the partial function, \( f(e,x) \), or the iterative function, \( I \), in the other definition, is recursive. As previously mentioned this would require the set of axioms of previously iterated Gödel-sentences
plus the starting set of axioms, $A_a$, to be $\Sigma_1$ for any $a \in O^*$ and a way to ensure this is if every initial segment is $\Sigma_1$ itself. Let $a$ be an element in $O^*$, and let $O^*_a$ be the initial segment of $O^*$ below $a$. Let immediate predecessors be defined in the intuitive way: $d$ is the immediate predecessor of $2^d$ and \{e\}(n) is the immediate predecessor of $3 \cdot 5^e$ for any $n$. Now by iterating predecessors from $a$ and down, I will get the initial segment below $a$, i.e. $O^*_a$. This is easily seen to be $\Sigma_1$.

So assume that $\text{Göd}_{O^*}(x) \uparrow$ for some $x \in O^*$. By construction of $\text{Göd}_{O^*}$, this cannot be the least element for which the computation does not terminate, so there is an $x_1$ with $x_1 <_{O^*} x$ such that $\text{Göd}_{O^*}(x_1) \uparrow$. $\text{Göd}_{O^*}(x) \uparrow$ is $\Pi_1$ since it is the same as $\forall y[-T(e, x, y)]$ for some index $e$. $\text{Göd}_{O^*}(x)$ has an index since it is a computable function (it is computable even if we do not know if it has domain all of $O^*$ yet). So if there is a failure of $\text{Göd}_{O^*}(x)$, then I can create a downward infinite sequence in $O^*$ that will be arithmetic and thus hyperarithmetic, which will be the contradiction.

The way to create this sequence is to look for the next element by going through the natural numbers in the usual ordering (i.e. $\omega$). So the formula will be:

$$b = a_n \iff b <_{O^*} a_{n-1} \land \text{Göd}_{O^*}(b) \uparrow \land \forall c < b \left[ \text{Göd}_{O^*}(c) \uparrow \implies \exists m < n(c = a_m) \right]$$

(2.1)

This way the formula picks out an unique next number in the sequence, and it is obviously arithmetic so any failure of $\text{Göd}_{O^*}(x)$ on $O^*$ would lead to a contradiction. I conclude that one can find a computable function that iterates adding new unprovable true sentences to an axiom system through any branch of $O^*$. This iteration will be massive, and as an easy consequence there exists a dense set of theories all consistent and all being extensions of PA with the natural numbers as a model. Now this function is essentially defined in the same way as $\text{Göd}_O$ in the previous section, and it is easily seen that $\text{Göd}_{O^*}$, restricted to $O$, will be $\text{Göd}_O$ or at least have exactly the same qualities. Again the domain of $\text{Göd}_{O^*}$ is bigger than $O^*$ since $O^*$ is not recursively enumerable.

It might be natural to assume that the iteration through a branch in $O^*$ will possibly be strictly stronger than iteration through any branch in $O$. As previously discussed, there is a branch in $O$ such that any true $\Pi_1$ sentence is provable with the sentences generated through the branch. As it turns out, this is as strong as the branches in $O^*$ can get as well. The point is
that it is shown that any branch in $O$ satisfying the properties above has to be $\Sigma_1$, i.e. not recursively enumerable, but then it will not be extended in $O^*$ as any initial segment of a branch will be recursively enumerable. Since $\text{Göd}_{O^*}$ produces new $\Pi_1$ sentences for any $a \in O^*$ it is shown that any branch in $O$ that provides a $\Pi_1$-complete iteration will not be extended in $O^*$. So the same limitations hold; there is a $\Pi_2$ sentence that is true for the natural numbers and not provable in $\bigcup_{a \in O^*} A_a$. So the extension from $O$ to $O^*$ is not one of strength, but one of complexity. The amount of different theories for the natural numbers will increase massively by iterating through $O^*$, as shown by this proposition:

**Proposition 3** Let $A_0$ be a consistent recursively enumerable set of axioms with the natural numbers as a model. For any positive rational number $q \in \mathbb{Q}^+$ there exists a consistent recursively enumerable extension, $A_q$, of $A_0$ such that for any $r < q$, $A_r \subset A_q$. 
Chapter 3

Meta-recursive Iteration

3.1 Overview

Now it is time to consider a different sort of recursion than in Chapter 2. Specifically I will introduce a new concept of computation and then use this concept to define a similar kind of iteration as in the previous section. The new type of computation has been heavily studied and I will use several results from other books. The extended power of this computation allows me to also extend the language, logic and model in question. The new concept of computation will allow some infinite computations, this will allow me to consider a language with infinite formulas and logic with infinite proofs. The extension of computations is natural in a way. It will allow for infinite computations, but only for recursive infinities. This naturally binds this new computation, meta-computation, to Kleene's $\mathbb{O}$ and hyper arithmetical sets. In fact in this new set the hyperarithmetical sets are the natural replacement of the natural numbers and $\mathbb{O}$ is meta-computable. The model will be a set of hyperarithmetical sets and will be the natural one for these types of computations, just as the finite numbers are the natural setting for ordinary recursion.

3.2 Definitions and Explanation

3.2.1 The Language

First I will introduce the new language $L_{\omega_1 \omega}$. $L_{\omega_1 \omega}$ is an extension of the language $L$. This will not be my intended language as I will need to restrict it to fit the recursion theory. The sub-script $\omega_1$ and $\omega$ is there to show the extent of the extension, the first sub-script, $\omega_1$ in this case, is to show that
the language accepts conjunctions and disjunctions of any set of formulas of cardinality less than \( \omega_1 \), so countably infinite disjunctions and conjunctions are allowed. The second sub-script is to indicate how many variables one can capture in an existential or universal quantifier. Everything less than the sub-script is allowed, in this case only a finite number of variable are allowed as in the "normal" language \( L \). So \( L_{\omega_1\omega} \) has as its formulas all formulas of the language \( L \) together with all formulas of the form:

\[
\psi \equiv \bigwedge_{i \in \mathbb{N}} \phi_i \quad \text{and} \quad \theta \equiv \bigvee_{i \in \mathbb{N}} \phi_i
\]

were all \( \phi_i \) are sentences of \( L_{\omega_1\omega} \) strictly simpler than \( \psi \) and \( \theta \).

Some examples may prove insightful. The sentence:

\[
\forall x (x = 0 \lor x = S(0) \lor x = S(S(0)) \lor ...)
\]

which says that the model is exactly the natural numbers \( \mathbb{N} \) is a sentence in \( L_{\omega_1\omega} \). This shows the power of \( L_{\omega_1\omega} \) as it can define the natural numbers in one (infinite) sentence, something \( L \) could not do for any consistent set of sentences. It also indicates a failure of both the compactness theorem and the Upward Löwenheim-Skolem theorem as any model with this sentence as an axiom must have only \( \mathbb{N} \) as its model and thus be countable.

Another interesting example:

\[
\neg(\exists x_0, x_1, x_2, ...) \bigwedge (x_{n+1} < x_n)
\]

This is not a sentence of \( L_{\omega_1\omega} \), since it has infinite variables in an existential quantifier. Rather it is a sentence of \( L_{\omega_1\omega} \) (and, of course, more complicated languages). This sentence expresses the defining quality of a well-ordering and it can be shown (for instance in Keisler \([3]\)) that the class of well-orderings can not be classified in \( L_{\omega_1\omega} \). This only shows that there are even more complicated languages that actually expands on \( L_{\omega_1\omega} \) in a meaningful way, but I will focus my attention on \( L_{\omega_1\omega} \) because it is considered to be the extension with the most practical gain in ratio to its added complexity, and because it neatly synchronizes with the concepts I am going to introduce later in this text.

It will be necessary to restrict the language \( L_{\omega_1\omega} \) to \( L_{\omega_1\omega}^{HYP} \) which will be only the formulas from \( L_{\omega_1\omega} \) with a hyperarithmetic code. This is because to make
any sentence in the language, and by extension any proof, meta-computable. This will neatly synchronize with the model I will introduce later as well, it will ensure that all the formulas can be coded into the model, an essential condition for any work on Gödel-numbering. The language $L^{HYP}_{\omega_1 \omega}$ will also contain a constant for every element of the intended standard model. I will need this result from Keisler [9], a fragment of a language is just a subset of the formulas closed under simple operations. $L^{HYP}_{\omega_1 \omega}$ obviously satisfies these closure properties.

**Theorem 5 (Completeness Theorem for $L_A$ Page 18 Keisler [9])** If $\varphi$ is a sentence of a countable fragment $L_A$ of $L_{\omega_1 \omega}$ then $\vdash_{L_A} \varphi$ if and only if $\models \varphi$

### 3.2.2 The Logic

The logic will need to be different than the one used for $L$. There I used ordinary first order logic. First order logic is not equipped to deal with infinite connectives. I will use a simple extension of this logic with just three new rules of inference. These rules will be natural in the sense that they capture the intuitive notion of the infinite connectives. The new rules are:

\[
\phi_1, \phi_2, \phi_3, \ldots \vdash \bigwedge_{i \in \mathbb{N}} \phi_i \\
\bigwedge_{i \in \mathbb{N}} \phi_i \vdash \phi_j \text{ (for any } j \in \mathbb{N}) \\
\phi_j \vdash \bigvee_{i \in \mathbb{N}} \phi_i \text{ (for any } j \in \mathbb{N})
\]

Again after extending some part of the iteration, I am forced to restrict it again. The concern is the same, proofs will have to be meta-computable. The proofs I will consider as legitimate will be ones build up inductively such that the code of the proof is hyperarithmetical, as this will ensure that the fact of being a proof and of seeking for a proof will be metarecursive (as will be shown later). The restriction mostly has to do with the first infinite rule, as it will require the set of proofs of the $\phi_i$ to be meta-recursive. Proper exploration of the proofs that will be allowed in this setting will wait until a later section, were I will introduce the Gödel-numbering of formulas and proofs. Then the restriction will be the natural one.
CHAPTER 3. META-RECURSIVE ITERATION

Next I will introduce meta-recursion and meta-recursively enumerable sets. The definition of metarecursion is not as intuitive and neat as that of regular recursion. There are no basic operations working as closure conditions in this case. I have already hinted that in some way the hyperarithmetical sets of numbers are analogous to finite sets in some transfinite computable way. Meta-recursion is this transfinite computation. Hyperarithmetical sets will be the finite numbers, finite sets will be hyperarithmetical sets of hyperarithmetical sets. So every meta-recursive function is a function from HYP to HYP.

Here is the formal definition from Sacks [1]

**Definition 3 (1.1 Definition of Metarecursive (Page 116 in Sacks))**

Let $Q$ be a $\Pi^1_1$ set of unique notations of recursive ordinals (Theorem 2.4 II). Let $n : \omega^1_{CK} \to Q$ take each recursive ordinal to its unique notation. Thus $|n(\beta)| = \beta$.

Assume $A \subseteq \omega^1_{CK}$. $A$ is called metarecursively enumerable if $n[A]$ is $\Pi^1_1$. ($n[A] = \{n(a) | a \in A\}$.) $A$ is called metarecursive if $A$ and $\omega^1_{CK} \setminus A$ are metarecursively enumerable. $A$ is said to be metafinite if $n[A]$ is hyperarithmetical.

So it is that metafinite is hyperarithmetic, metarecursive is a $\Delta^1_1$ set of metafinite (hyperarithmetical) sets and metarecursively enumerable is a $\Pi^1_1$ set of metafinite sets.

### 3.2.3 The Model

The model I will use in place of the natural numbers have been carefully selected for its purpose. It should be a set of hyperarithmetical sets, since the hyperarithmetical sets are the metafinite ones in metarecursion theory. In addition it should be rich enough so that the new extended language, $L_{\omega_1^1}$, can be coded into it. The model I will use has two equivalent definitions, and it would be smart to get acquainted with both of them. One is through the famous constructible hierarchy of Gödel. I assume the reader will have some knowledge of this hierarchy. The model will be $L(\omega^1_{CK})$, i.e. those sets definable by Zermelo-Fraenkel set theory using only constructible ordinals.
3.2. DEFINITIONS AND EXPLANATION

The other definition will be useful when I in the next section will use Gödel-numbering in this setting. First I inductively define a subset of $\omega$ to be a code, $a \in K$, if:

1. $a = \{0\}$
2. $\forall n \in \omega \ [a_n = \{m | m < n, m \in a\} \in K]$  

(In this part to make sure every $a \in K$ is a subset of $\omega$ I let $<m,n> = 2^m \cdot 3^n$)

The idea is that the codes code sets from the Hereditary Countable sets (HC), and that each set has at least one code. Then one can decipher the codes by an inductive process, $i$:

1. $i(\{0\}) = \emptyset$
2. $i(a) = \{i(a_m) | m \in \omega\}$

Then if I only consider the sets with codes that can be defined by a $\Delta_1^1$ formula, i.e. the hyperarithmetical codes, then I get the model I want, namely HC(HYP). As previously hinted HC(HYP) = $L(\omega_1^{CK})$ and I will use this fact without proving it, but the intuition is that a set definable in an ordinal less than $\omega_1^{CK}$ will be hyperarithmetical and thus be in HC(HYP).

There are other reasons why HC(HYP) is the natural model for this extended iteration. The first reason is that HC(HYP) is $\Sigma_1$ admissible. A set is $\Sigma_1$ admissible if it is transitive, closed under pairing and union and satisfies $\Delta_1$ separation and $\Sigma_1$ bounding. A set satisfies $\Delta_1$ separation if for any element, $a$, and $\Delta_1$ sentence $\phi(y)$

$$\exists x (\forall y)(y \in x \leftrightarrow y \in a \land \phi(y))$$

And a set satisfies $\Sigma_1$ bounding if for any element, $a$, and $\Sigma_1$ sentence $\psi(x,y)$

$$\forall x \in a (\exists y) \psi(x,y) \rightarrow (\exists z)(\forall x \in a)(\exists y \in z) \psi(x,y)$$

The other reason is this result from Sacks which follows from $\Sigma_1$ admissibility and the definition of meta-recursion.

**Proposition 4 (1.3 Proposition page 153 Sacks [1])** Let $A \subset \omega_1^{CK}$. Then $A$ is metarecursively enumerable iff $A$ is $\Sigma_1$ definable over HC(HYP).

This makes HC(HYP) have a similar relationship with metarecursiveness and $L_{\omega_1^{CK}}^{HYP}$ provability as the natural numbers has with recursiveness and $L$ provability, this makes it possible to prove a similar incompleteness result in this setting.
CHAPTER 3. META-RECURSIVE ITERATION

3.3 Gödel-numbering

3.3.1 Formulas

In the normal case, with the natural numbers as the model, Gödel numbering uses the fact that each number can be factored into unique primes of differing multiplicity to ensure a unique coding for every term, formula and sentence. Even though one can consider the natural numbers as a subset of HC(HYP) and therefore also consider some sets as representations of prime numbers, this will not be adequate for the purpose of coding $L_{\omega_1^1}^{HYP}$. The problem with this idea is the infinite disjunctions and conjunctions that is the extension of $L$, there is no natural number with infinite prime factors. Theoretically one could code this language into the natural numbers as well, since it is countable. In a way using notations for the recursive ordinals and a numbering for the language of set theory every element of $L(\omega_1^{CK})$ can be seen as a natural number, but this is not very easily manageable and a new numbering inside $L(\omega_1^{CK})$ is preferable. Fortunately, as previously noted, HC(HYP) was deliberately chosen to deal with this problem. Here one can find infinite subcodes in a unique way to ensure the same sort of properties that makes Gödel numbering such an effective tool in proving incompleteness. I will in this section go through the Gödel numbering in some detail and look at analogous relations and results as in the "regular" case. The most important result will be to show I can define a $\Sigma_1$ formula, $Prv(x)$, in the normal logic that will hold true if and only if x is the Gödel number of a formula in $L_{\omega_1^1}^{HYP}$ provable in $PA_\omega$.

In HC(HYP) everything is sets of sets (of sets, of sets etc.) so I will use the notation of ordered pairs with this in mind. $<a, b>$ will mean \{a, \{a, b\}\}. I will also use numerals for natural numbers in the numbering. This is possible because the set of natural numbers is an element of HC(HYP) where natural numbers are defined in the classical set theoretic fashion (i.e. the set of all previous natural numbers, with zero the empty set). A consequence of this is that any search through the natural numbers will be bounded (and hence $\Delta_1$). This will be useful later. Here is the formal numbering I have chosen:

1. $\bar{0} \Rightarrow <1, 1>$

2. $\bar{i} \Rightarrow <2, i>$
3.3. GÖDEL-NUMBERING

3. $x \Rightarrow < 3, x >$

4. $t_1 = t_2 \Rightarrow < 4, t_1, t_2 >$

5. $t_1 \in t_2 \Rightarrow < 5, t_1, t_2 >$

6. $\phi \land \psi \Rightarrow < 6, \phi, \psi >$

7. $\neg \phi \Rightarrow < 7, \phi >$

8. $\exists x \phi \Rightarrow < 8, \phi >$

9. $\bigwedge_{i \in \mathbb{N}} \phi_i \Rightarrow < 9, \{< 10, i, \phi >\} >$

There really is not much new here, only the last clause dealing with the infinite connective. Its meaning should be clear, 9 is an index for the connective, 10 is an index for being a part of the connective and $i$ is an index for which number it is in the enumeration of the formulas that are connected. Now I can go on to define some relations on HC(HYP) that will pick out when an element is a coding for variables, constants, formulas etc. Most of these should be easy to imagine, at least after seeing a few examples, but some will require me to go into a little more detail. As a light start here is a relation that holds true if and only if $x$ is the code for a constant in the language $\mathcal{L}^{\text{HYP}}_{\omega 1 \omega}$:

$$\text{Con}(x) \equiv \exists y \in TC(x) [x = < 3, y >]$$

$TC(x)$ is the transitive closure of $x$, easily seen to be in HC(HYP) and also to be $\Delta_1$. $TC(x)$ is a unnecessary large set for finding $y$, but the point is that the quantifier is bounded so that the formula is $\Delta_1$. This is analogous to the bounds on quantifiers in normal Gödel numbering being unnecessary large. The formula is not written in the language of set theory as $< x, y >$ is not a symbol, but writing this out is of no difficulty and of no use as it will only obfuscate the intended meaning. As already explained the formula is easily seen to be $\Delta_1$. Another easy example:
\[ \text{Var}(x) \iff \exists i \in \omega [x = < 2, i>] \]

Here Var(x) says that x is a coding for a variable \(v_i\). The formula contains an existential quantifier, but since \(\omega\) is an element of HC(HYP) this does not complicate the formula and it is also \(\Delta_1\).

Now by combining these two, along with a formula for when \(x = 0\), I can make a relation,

\[ \text{Term}(x) \iff \text{Zero}(x) \lor \text{Var}(x) \lor \text{Con}(x) \]

that holds if and only if \(x\) is a code for a term in the language \(\mathcal{L}_{\omega_1 \omega}^{HYP}\), since the only terms in this language are the variables and the constants. Then I can just as easily make a \(\Delta_1\)-relation \(\text{AtFor}(x)\) that will hold true if \(x\) is the code of an atomic formula. These relations will be a cornerstone when I continue to create more complicated relations, as terms make up the base case in the inductive building of formulas.

Next is the relation \(\text{Form}(x)\) which states that \(x\) is a code for a formula in \(\mathcal{L}_{\omega_1 \omega}^{HYP}\). This definition will be by recursion on the building and coding of formulas in \(\mathcal{L}_{\omega_1 \omega}^{HYP}\). The definition I write will use some abbreviations and short-cuts to make it more understandable, a fully written out version would only lead to confusion. The important thing to note is that the definition is by induction. Naturally since formulas are defined inductively:

\[ \text{Form}(x) \iff \exists y, z \in \text{TC}(x)[\text{AtFor}(x) \lor [x = < 6, y, z > \land \text{Form}(y) \land \text{Form}(z)] \lor [x = < 7, y > \land \text{Form}(y)] \lor [x = < 8, y > \land \text{Form}(y)] \lor [x = < 9, y > \land (\forall z \in y)(\exists i \in \mathbb{N})(z = < 10, i, v > \land \text{Form}(v))]] \]

The structure is built on the definition of numbering showed above, and the formula is just checking if each case is true. The point is that in every case of \(\text{Form}(y)\) on the right side of the equation, the code being checked is a subcode of the original code. The definition is by recursion on a \(\Delta_1\)-formula and it is a fact that such a recursive definition will itself be \(\Delta_1\). This is the recursion theorem for meta-recursion, see Theorem 6 below. Thus the relation for being a code for a formula is \(\Delta_1\).
3.3. Gödel-numbering

3.3.2 Proofs

Now I have to come back to what I mentioned earlier about proofs, and coding of them. The main point is that if I consider all possible proofs that can arise with this logic and language, then there will be no reasonable way of checking what is a proof and what is not. The problem arises because of the infinite conjunctions and disjunctions and the way one proves $\bigwedge \phi_i$ which requires infinite assumptions. This gives rise to uncountably many possible proofs, and this situation will be impossible to check in any realistic way, not to mention code into HC(HYP), which itself is countable.

The restriction will be that the admissible proofs will be the ones with an hyperarithmetical proof-tree, the point being that any proof-tree with a hyperarithmetical code will be in HC(HYP). A proof-tree will be a tree with nodes, each node will have the form $<i,⌜\phi⌝>$ where $i$ is the index used for a specific rule of inference and $\phi$ is a sentence that follows from the lower branches by that rule of inference. The indexation does not really matter as there are only finitely many rules of inference and infinitely many indexes to go from. Atomic formulas and axioms will constitute the end nodes of the tree as they do not require any proof (for instance $1 = 1$ do not require a proof) It is common to look at proofs as strings of sentences where each sentence follows from any number of previous ones using a rule of inference, but taking into account the infinite rule used with the infinite connector this method is unsatisfactory. The proof-tree created will be manageable in the way that it will be well-founded. Any branch of the proof-tree will be finite, only its width will be allowed to be countable, as it will be when dealing with infinite connectives. This will allow proof-trees with infinite length as the infinite branching could lead to branches of arbitrary finite lengths.

The point of all this is to be able to write a formula, $Prv(x)$, that will be a relation that holds true if and only if $x$ is a code for a sentence that has a hyperarithmetical proof (or proof-tree). Now one can see the importance of the well-foundedness of the proof-trees, as I can now use recursion in the formula in the knowledge that every time I call on a subformula to be provable it will have a shorter proof. I will first create a formula $Prf(y, x)$ which says that $y$ is the code of a proof-tree for $x$. $Prv(x)$ will then be $\exists y Prf(y, x)$.

Let's look at some examples of the steps that need to be in the code of $Prf(y, x)$. Let $x$ be the code for any formula $\phi$, and assume that $\phi$ is proven by modus ponens, $\psi \rightarrow \phi$, $\psi \vdash \phi$. Then one can assume that we have proof-
trees for $\psi \to \phi$ and $\psi$, which I will write as $T_{\psi \to \phi}$ and $T_{\psi}$ respectively. And also that these trees have hyperarithmetical codes, say $z$ and $v$. Then one creates a proof-tree that gives a proof of $\phi$ by letting the two shorter proof-trees, $T_{\psi \to \phi}$ and $T_{\psi}$, be the two only immediate sub-trees of the proof-tree. So the tree $y$ will be of the form: $<< k, x >, \{< i, z >, < j, v >\} >$ where $i \neq j$, $i, j \in \{1, 2\}$ and $k$ is the index dedicated to modus ponens. What index $k$ is is irrelevant as long as it is chosen to uniquely determine the code for the tree, but since there are only finitely many clauses and each clause requires only countably many indexes everything can be coded by the natural numbers in some way or form. I will not go into too much technical detail about what index goes to what symbol or rule of inference or any other thing as it in my opinion only decreases readability and it is in not essential exactly how this is done, only that it can be done.

The part of $Prf(y, x)$ that deals with modus ponens will then look something like this pseudo code:

$$y = << k, x >, \{< i, z >, < j, v >\} > \land (i = 1 \lor i = 2) \land (j = 1 \lor j = 2) \land (i \neq j) \land Prf(v, (\text{TopNode}(v))_2) \land Prf(z, (\text{TopNode}(z))_2) \land (\text{TopNode}(z))_2 = 7, 6, (\text{TopNode}(v))_2, < 7, x >>$$

This formula says that $y$ is a three, the top node of $y$ is $< k, x >$, and the rest of the tree is built so that it represents the use of the rule of inference indexed by $k$. So this formula checks if $y$ is a tree that has $< k, x >$ as its top node, and that its two only direct subtrees are the proof-trees of $\psi$ and $\psi \to \phi$. The definition is obviously recursive because it is needed to call upon $Prf$ to show that the subtrees are proof-trees as well, but as discussed this is safe because the proof subtrees are strictly shorter than the original tree.

Another example. Assume $x$ is code for $\phi = \bigwedge_{i \in \mathbb{N}} \theta_i$ and it is proved by the infinite rule $\theta_1, \theta_2, \theta_3, \ldots \vdash \bigwedge_{i \in \mathbb{N}} \theta_i$. Then a potential proof-tree will have $< k, \phi >$ as the top node where $k$ is the index of this rule (again; it does not really matter exactly what it is. Say: $k = 777546$). Then we assume $T_{\theta_i}$ is the proof-tree of $\theta_i$ then each such tree will be an immediate subtree of the proof-tree $T_{\phi}$, that is all branches out from the top node in $T_{\phi}$ will be of the form $< j, \sigma >$ for some $\sigma$ and $i$ where $\sigma \in T_{\theta_i}$.

Then the tree $y$ will look like this: $<< k, x >, \{< i, z >\} >$, where $k$ is the index for the rule of inference and $z$ is the code for the subtree $T_i$. 
3.3. GÖDEL-NUMBERING

Remember x will be on this form: \(< 9, \{< 10, i, r\theta n >\} > So in pseudo code the part of Prf(y, x) that deals with this rule will look something like this:

\[ \text{Tree}(y) \land (\text{TopNode}(y))_2 = x \land (\text{TopNode}(y))_1 = k \land \forall z \in \text{Level}_1(y) \exists v \in (x)_1 [\text{Prf}(z, (v)_2) \land \forall v \in (x)_1 \exists z \in \text{Level}_1(y) [\text{Prf}(z, (v)_2) \land \forall v \in (x)_1 \exists z \in \text{Level}_1(y) \text{Prf}(z, (v)_2)] \]

\( \text{Level}_1(y) \) is the set of nodes at the first level under the top-node. An important thing to point out in this code; even though it is pseudo code one can see that all use of quantifiers are bounded in \( \text{HC(HYP)} \), it is easily seen that the same bounds will work in the full code. This makes the code, when written out completely in \( L \), into a \( \Delta_1 \) formula.

I have not yet explained the axiom system used to create these proofs, this is done deliberately. In fact the axiom system has only one requirement, the set of axioms need to be \( \Sigma_1 \)-definable and has \( \text{HC(HYP)} \) as a model. This is a consequence of the fact that the incompleteness result will be proven using formulas from set theory, defining relations that hold for \( L_{\omega_1}^{\text{HYP}} \)-sentences. So the only requirement on the set of axioms is that they are easy enough to be meta-recursively enumerable, in other words \( \Sigma_1 \) definable over \( \text{HC(HYP)} \). This is analogous to the normal case, but PA is often used as the base case in normal recursion. For \( \text{HC(HYP)} \) there is no preferred axiom set in the same way.

Now to make a full formula of a proof-tree I would need to make clear substitution of free variables for terms. But this will all be done in exactly the same way as in normal Gödel-numbering. The only new form of formula in this extended language is the one which uses infinite disjunction, and checking for free variables or substitution in such a formula amounts to checking it for each of the disjunctions, so this will not add any problem (only an infinite search through the natural numbers, but in \( \text{HC(HYP)} \) this is \( \Delta_1 \)).

With similar explanations for the rest of the clauses in the logic I have proved the following:

\textbf{Claim 2} \( \text{Prf}(y, x) \) is \( \Sigma_1 \)

Again, \( \text{Prf}(y, x) \) is a relation written in the language of set theory that says, "\( y \) is a code from \( \text{HC(HYP)} \) for a proof-tree for the formula coded by \( x \)". The intuitive explanation for this relation to be \( \Sigma_1 \) is that no search through all the elements of \( \text{HC(HYP)} \) is needed. The only thing that causes the relation to be \( \Sigma_1 \) and not \( \Delta_1 \) is the confirmation that something is an
axiom which I allowed to be $\Sigma_1$, the rest of the searches will be bounded by
the input and their transitive closures.

$Pr(x)$ (that says: "x is provable") is then easily seen to be $\Sigma_1$-definable
over HC(HYP) as well as it is equal to $\exists y[Prf(y, x)]$. So the set of all
codes of provable formulas are $\Sigma_1$-definable over HC(HYP) and is therefore meta-
recursively enumerable. This is exactly analogous to the case of regular logic
being recursively enumerable. This allows us to conceive of a machine using
transfinite computations that could be the sort of proof machine Hilbert
hoped for in formal arithmetic, only in the hyper arithmetic case. And in
the same way this leads to an incompleteness result similar to the normal
case. In other words I will be able to find a $\Pi_1$ sentence that will be true
for HC(HYP), yet unprovable from the given axioms. The argument, which
I will elaborate below, uses the same technique as Gödel originally came up
with, namely a sort of diagonalization over all provable sentences.

3.3.3 Incompleteness

The formula $Pr(x)$ is written in the language of set theory, but it can easily
be seen as a sentence of $L_{\text{HYP}}^{\omega \omega}$. This is important as the proof of the incom-
pleteness theorem requires use of the formula and its Gödel-numbering. The
first thing I need is to define a relation:

$$A(x) \equiv Form(x) \land \neg Pr(x)$$

$A(x)$ is all the codes of formulas that are not provable in the given axiom
system in $L_{\text{HYP}}^{\omega \omega}$. It is clear that $A(x)$ is a $\Pi_1$-formula definable in $L_{\text{HYP}}^{\omega \omega}$.
Now it makes sense that all the sentences that satisfies the relation $A$ will
not be provable in the given theory, by definition of $A$. All I need is a sen-
tence that will also be a sentence that is true for HC(HYP). I do this in the
normal way for incompleteness results; I find a sentence, $\theta$, that states its
own unprovability:

$$\theta \iff \neg Pr(\bar{\theta}) \iff A(\bar{\theta})$$

So I need a code, $a$, which is the code for the formula $A(a)$, this sort of
self-reference is the key to the incompleteness result. Let $\text{Sub}(\bar{A}, n, x, y)$ be
the relation were $y$ is the code for the formula $A(x)$ with $x$ replaced with the
term that $n$ encodes, so $y = \bar{A}(n)$. What I need is an element of HC(HYP),
3.3. GÖDEL-NUMBERING

say a, such that $\text{Sub}(\overline{A}, a, x, a)$ holds, because then $a = \overline{A(a)}$. Then the formula represented by $a$ will have the desired qualities.

The argument I use here will be analogous to the proof of Gödel’s self reference lemma in Leary [4] for normal first order logic. Let $\psi(v_1)$ be the formula:

$$\forall y[\text{Sub}(v_1, v_1, < 2, 1 >, y) \rightarrow A(y)]$$

First notice that $\psi(v_1)$ is a $\Pi_1$-formula. A thorough explanation of the formula follows since it is not only a very important formula, it is also confusing. $A(y)$ is really not important in this formula, it is there so the wanted equivalence will go through in the end, but the same argument would work in the case of any formula replacing $A(x)$. It is a fact that for any sentence one can find a self-referencing number like this, and when the formula is $\neg Pr(x)$ this leads to the incompleteness theorem.

Now what the formula says is that $y$ is the code of the input, $v_1$, with the first variable in the formula coded by $v_1$ substituted with the code for the formula itself. The intention is that the input will be the code for a formula with one free variable, say $\phi(v_1)$. Then $y = \overline{\phi(\phi(v_1))}$. The double use of $v_1$ both as the input in the formula $\psi(v_1)$ and as a theoretical variable in the input, which again is $v_1$, is confusing at first, but it is essential. For now; let

$$\theta \leftrightarrow \psi(\psi(v_1)) \leftrightarrow \forall y[\text{Sub}(\psi(v_1), \psi(v_1), < 2, 1 >, y) \rightarrow A(y)]$$

Then $y$ becomes the code for the formula $\theta$ and $\theta$ then does what it should, namely it says that if $y$ is the code for itself then $A(y)$ holds, this makes sure that the wanted equality holds, namely:

$$\theta \leftrightarrow A(\overline{\theta}) \leftrightarrow \neg Pr(\theta)$$

If $\theta$ holds, then by construction of the formula $A(y)$ holds for $y = \overline{\theta}$, so one way is shown. Now if $A(\overline{\theta})$ holds, then $\text{Sub}(\overline{\theta}, \overline{\theta}, < 2, 1 >, y) \rightarrow A(y)$ holds, so the converse is shown and the equivalence holds.

Thus $\theta$ is a formula which is equivalent to the fact that $\theta$ itself is unprovable in the given theory. If it is provable then it is false, but this does not make sense as all provable sentences are true if the set of axioms are consistent and are satisfied by the model. Therefore the sentence must be true and also unprovable. So we have a form of incompleteness result in the language.
with any consistent $\Sigma_1$-set of axioms. The way the incompleteness is shown uses properties that can be found in more general settings, as I will briefly discuss in Section 4.

As earlier mentioned, a subset of HC(HYP) is meta-recursively enumerable if and only if it is $\Sigma_1$-definable over HC(HYP). I have already shown that one can given a $\Sigma_1$ definable set of axioms, one can find a $\Sigma_1$ formula that represents all provable formulas for the given set of axioms. From the provable formulas, or a representation of them in code, I can then find a the code for a Gödel-sentence for the given axiom-set in the way shown above.

Let $\mathcal{A}$ be the representation of any consistent $\Sigma_1$-definable axiom-set which are true for the model HC(HYP), what I have just shown is the existence of a meta-computable function, $f$, such that $f(\mathcal{A}) = \theta_\mathcal{A}$, where $\theta_\mathcal{A}$ is a $\Pi_1$ Gödel-sentence for $\mathcal{A}$.

### 3.3.4 Iterations

Now I am in a situation similar to the one I was in in Section 1. I have a function creating true, unprovable sentences for any consistent $\Sigma_1$ set of axioms with HC(HYP) as a model. The two changes are the complexity of the function and the model. It seems likely that I will be able to proceed in the same fashion as in Section 1 and create an iterative process of making stronger and stronger theories for HC(HYP), and this is exactly what I intend to do!

**Theorem 6** 1.8 Proposition Transfinite Recursion p.120 Sacks/ If $I$ is metar-erecursive, then the solution of

$$f(\delta) = I(f \upharpoonright \delta)(\delta < \omega_1^{CK})$$

is metar-erecursive.

With the Recursion Theorem for meta-computable functions this is easy. In the construction from Section 1 it is easily seen that $Add$ is meta-computable as well as computable. Then the Recursion Theorem makes sure the iteration works as it should. An important thing to note is that at any stage of the
3.4. Iteration for $O^{HYP}$

iteration the set of axioms plus the Gödel-sentences already iterated will form a $\Sigma_1$-definable set, thus making it possible to construct a new Gödel-sentence in the way already described. First I will use it only to iterate through the constructive ordinals, but the restricting quality will be that any initial segment of the relation used in the iteration must be $\Sigma_1$, i.e. meta-computable.

**Proposition 5** For any consistent, $\Sigma_1$-definable theory, $A$, with $HC(HYP)$ as a model there exists a meta-computable function $\text{Göd}_{HYP} : \omega_1^{CK} \rightarrow HC(HYP)$ such that $\text{Göd}_{HYP}(\delta)$ is the code for a Gödel-sentence for $A \cup \{\text{Göd}_{HYP}(\sigma) | \sigma < \delta\}$.

Just as iteration through the natural numbers only was the starting point for regular recursion, so is iteration up to $\omega_1^{CK}$ only the beginning for meta-recursion. As I have mentioned in the previous Chapter the only restriction is that initial segments of the iteration ordering must be meta-recursive, and as I will show this takes us pretty far. $\Sigma_1$-definable over $HC(HYP)$ is by definition the same as meta-computable. This is why this restriction on the ordering is important in using the transfinite recursion. Since $\omega_1^{CK}$ itself is meta-computable, there will be a $\Delta_1$ relation of length $\omega_1^{CK}$, by the use of a uniformization principle used in the Section on $O^\ast$. Let $R(x, y)$ be such a relation. Now consider the lexicographical relation on ordered pairs of elements of $R$:

$$R^\ast(<x, y>, <z, v>) \leftrightarrow R(x, z) \lor [x = z \land R(y, z)]$$

This ordering will have length $\omega_1^{CK} \cdot \omega_1^{CK}$, and it will be $\Delta_1$ as long as $R$ is $\Delta_1$. Thus I can use the same sort of transfinite iteration, only this time the iteration will go on until the ordinal $(\omega_1^{CK})^2$.

This can be expanded even further as a lexicographical ordering can be defined in mostly the same way for any $n$-tuple of elements. Thus I easily get $\Delta_1$-orderings of length $(\omega_1^{CK})^n$ for any finite $n$.

**Proposition 6** For any natural number, $n$, there exists a meta-computable function, $f : L(\omega_1^{CK}) \rightarrow \omega_1^{CK}$, that iterates Gödel-sentences through the ordinal $(\omega_1^{CK})^n$.

3.4 Iteration for $O^{HYP}$

3.4.1 Definition and Iteration

In Section two I used Kleene's $O$ as a way of expanding the iteration to the length of the constructive ordinals. It was also an extension in width, as $O$
was not a well-ordering, but a partial ordering that can be viewed more as a
tree with a least element with branches of order-length up to $\omega_1^{CK}$. The con-
struction of $O$ used the enumeration of the computable functions to create
notations for ordinals in the natural numbers. It is easy to imagine something
similar being done in the case of meta-recursion; creating notations for the
meta-recursive ordinals using $L(\omega_1^{CK})$, and then using a fixed-point theorem
for meta-recursion and argue for a meta-computable iteration through this
ordering as well. The important properties I need to preserve when general-
izing the construction of $O$ is that for any element in $O$ the initial segment
below this element will be a meta-computable well-ordering. The fact that
it is well-ordered will follow directly from its definition.

I will call this ordering $O^{hyp}$. When trying to apply the same construc-
tion as in $O$, some problems arise. A meta-recursive function is often said to
be from $\omega_1^{CK}$ to $\omega_1^{CK}$, but here it is necessary to have functions with domain
and range in $L(\omega_1^{CK})$. Fortunately the elements of $L(\omega_1^{CK})$ can be put in a
one-to-one correspondence with $\omega_1^{CK}$, so any element of $L(\omega_1^{CK})$ can be re-
presented by its unique ordinal. Therefore the meta-recursive functions can
be seen as functions from $L(\omega_1^{CK})$ into $L(\omega_1^{CK})$.

I will define $O^{hyp}$ inductively, with three clauses of inclusion and simul-
taneously define the ordering on $O^{hyp}$. I will use the enumeration of meta-
computable functions using the ordinals less than $\omega_1^{CK}$.

\begin{align*}
x \in O^{hyp} & \rightarrow < 1, x \in O^{hyp} \\
x \in O^{hyp} & \land x \text{ is a total order} \land X \in HY P \land \sup\{X\} \notin X
& \rightarrow < 2, X \in O^{hyp} \\
\delta < \omega_1^{CK} & \land \{\delta\} \text{ is total} \land [\sigma < \lambda \rightarrow \{\delta\}(\sigma) < \{\delta\}(\lambda)]
& \rightarrow < 3, \delta \in O^{hyp}
\end{align*}

The ordering will be defined using clauses similar to $O$. Notice that clause 2
and 3 from the definition of $O^{hyp}$ needs an ordering already present. There-
fore the definition can be seen as an inductive process where the definition
of $O^{hyp}$ and the ordering $<_{O^{hyp}}$ are defined simultaneously, and in stages.
This is exactly analogous to the definition of $O$. The ordering is defined by
these clauses:

\[ x <_{O^{HYP}} 1, x > \]

\[ x \in X \rightarrow x <_{O^{HYP}} 2, X > \]

\[ \{ \delta \}(\sigma) <_{O^{HYP}} 3, \delta > \]

\[ x <_{O^{HYP}} y \land y <_{O^{HYP}} z \rightarrow x <_{O^{HYP}} z \]

< 1, x > is meant to be the successor, \( S(x) \). The limit clause is split in two as there will be two different kind of limits, \( \omega \)-limits and \( \omega_1^{CK} \)-limits. < 2, X > is meant to deal with \( \omega \)-limits, the point is that < 2, X > will be representing \( sup\{X\} \). Therefore any hyperarithmetic limit, be it \( \omega \) or longer, will have a representation. The requirement that \( sup\{X\} \) is not an element of X is to make sure that only infinite limits will be added. Without this requirement the successor would not be the unique next element in the ordering, and this is a property I would like to keep. Notice that this clause also secures that the ordering has a least element as \( X = \emptyset \) satisfies the requirements, and < 2, \emptyset > will be a unique least element. < 3, \delta > is meant to deal with \( \omega_1^{CK} \)-limits, this is the clause that takes branches of the ordering past \( \omega_1^{CK} \). The ordering on \( O^{HYP} \) is the natural one considering the way \( O^{HYP} \) is built.

\( O^{HYP} \) has the following properties:

\[ \forall x, y \in O^{HYP}[x <_{O^{HYP}} y \rightarrow S(x) \leq_{O^{HYP}} y] \]

\[ \forall x, y \in O^{HYP}[y =< 2, z > \land x <_{O^{HYP}} y \rightarrow \exists v \in z(x <_{O^{HYP}} v)] \]

\[ \forall x, y \in O^{HYP}[y =< 3, \delta > \land x <_{O^{HYP}} y \rightarrow \exists \sigma \in \omega_1^{CK}(x <_{O^{HYP}} \{ \delta \}(\sigma))] \]

These properties makes any initial segment of \( O^{HYP} \) well-ordered as limits and successors will be uniquely defined. Any element of \( O^{HYP} \) will be reached in a meta-computable way, therefore the initial segment under any element will be meta-computable and \( \Sigma_1 \) as well. The complete proof of this is very similar to the one for Kleene's \( O \), so I will not show them explicitly for \( O^* \).
An iteration of Gödel-sentences will now also be possible through $O^{HYP}$. The proof follows the same lines as for Kleene's $O$ using ETR. A fix-point for any meta-computable iterator, $I$, is already assured. The only thing needed is to show that the iteration will be defined for all of $O^{HYP}$. This is proved by an inductive argument; the zero-instance is obvious since it uses no information other than the already fixed set of starting axioms. Assume $a$ is an element for which the construction breaks down, then there has to be a $b <_{O^{HYP}} a$ so that the construction breaks down. This will lead to an infinite downward sequence in the initial segment of $O^{HYP}$ below $a$, which is a contradiction. Again the argument is the same as for Kleene's $O$, since the construction of $O^{HYP}$ is basically the same.

**Proposition 7** Let $O^{HYP}$ be defined as above. Then for any consistent set of axioms, $A$, $\Sigma_1$-definable over $HC(HYP)$ with $HC(HYP)$ as a model there exists a function $Göd_{O^{HYP}}$ that is definable on all of $O^{HYP}$ such that for any element, $a$, of $O^{HYP}$ $Göd_{O^{HYP}}(a)$ is a $\Pi_1$ Gödel-sentence for the set of sentences $A \cup \{Göd_{O^{HYP}}(b) | b <_{O^{HYP}} a \}$.

In the same way as the branches of Kleene's can have orderings of up to the first non-recursive ordinal, the branches of $O^{HYP}$ will be of length up to the first non-meta-recursive ordinal, usually denoted by $\omega^2_{CK}$.

### 3.4.2 A completeness result

In the case of normal recursion through $O$, Turing showed that any $\Pi_1$ sentence could be proved in a branch of $O$ and already at the level $\omega + 1$. I will in this section show that the same result can be found in the case of meta-recursion through $O^{HYP}$. The proof I will give is taken from Feferman [11] and adapted to the meta-recursive case. For this part I will have to add the extra requirement on the initial set of axioms that any true $\Sigma_1$ sentence can be proved. The regular extension of PA to HC(HYP), $PA_{HYP}$, has this property. $PA_{HYP}$ is the sentences of PA only with the induction axiom adopted to $\omega^1_{CK}$.

**Proposition 8** Let $\psi$ be a $\Pi_1$ sentence that is true in $HC(HYP)$ and let $A_a$ be the axiom set gained by an iteration process up to $a \in O^{HYP}$ starting with $PA_{HYP}$ then there exists an element, $d$, in $O^{HYP}$ such that $\vdash_{A_d} \psi$ and $|d| = \omega^1_{CK} + 1$.

**Proof:** Let $\phi(x)$ be a $\Delta_1$ formula such that $\psi \equiv \forall x \phi(x)$. Since $\psi$ is a true sentence for HC(HYP) $\phi(\sigma)$ will be true for any $\sigma \in HC(HYP)$. Let $\delta$ be an
index for the meta-recursive function:

\[ \{ \delta \}(\sigma) = \begin{cases} < 1, \beta > & \text{if } \forall \lambda \leq \text{O}HY P \ \sigma \phi(\lambda) \land \sigma \text{ is the successor of } \beta \\ < 2, \{\delta\}(\lambda) |\lambda < \text{O}HY P \ \sigma > > & \text{if } \forall \lambda \leq \text{O}HY P \ \sigma \phi(\lambda) \land \sigma \text{ is a limit ordinal} \\ < 1, < 3, \delta > > & \text{if } \exists \lambda \leq \text{O}HY P \neg \sigma \phi(\lambda) \end{cases} \]

So as long as \( \phi(x) \) holds, \( \{ \delta \} \) defines a branch in \( \text{O}HY P \), where each ordinal is mapped to a notation of itself. If \( \phi(x) \) does not hold at some point the function then jumps to the successor of the ordinal representing the function.

Let \( b = < 3, \delta > > \) and \( d = < 1, < 3, \delta > = < 1, b > \), and let \( \theta_b \) be the Gödel-sentence constructed for \( A_b \). It is clear by definition that \( \vdash_{A_b} \theta_b \) as this is the new Gödel-sentence. Equally obvious is the fact that \( \vdash_{A_{<1,b>} \theta_b} \) as \( \theta_b \) is the one axiom added to \( A_b \) to make \( A_{<1,b>} \).

Now assume for one moment that \( \exists x \neg \phi(x) \), let \( \lambda \) be the least element such that \( \neg \phi(\lambda) \). Then \( \{ \delta \}(\lambda) = < 1, < 3, \delta > = d \) and as \( A_b = \bigcup_{a \leq \text{O}HY P} A_a \) and \( \{ \delta \}(\lambda) < \text{O}HY P < 3, \delta > = \theta_b \) it would follow that \( A_d \subseteq A_b \). Furthermore, this is provable in \( PA_{HY P} \) as the function is meta-recursive and therefore \( \Sigma_1 \) definable. Now I get.

\[ A_b \vdash \theta_b \leftrightarrow \neg Pr_{A_{b}}(\overline{\theta_b}) \text{ (by the definition of } \theta_b) \]

\[ A_d \vdash \theta_b \text{ (since } d \text{ is the successor of } b) \]

\[ PA_{HY P} \vdash \exists x \neg \phi(x) \rightarrow A_d \subseteq A_b \text{ (since } \{ \delta \}(\lambda) = d \text{ for some } \lambda) \]

\[ PA_{HY P} \vdash \exists x \neg \phi(x) \rightarrow Pr_{A_{b}}(\overline{\theta_b}) \text{ (since a proof in } A_d \text{ leads to a proof in } A_b) \]

\[ PA_{HY P} \vdash \exists x \neg \phi(x) \rightarrow \neg \theta_b \text{ (by the first and previous line)} \]

\[ PA_{HY P} \vdash \theta_b \rightarrow \forall x \phi(x) \text{ (by the previous line)} \]

\[ A_d \vdash \theta_b \rightarrow \forall x \phi(x) \text{ (since } A_d \text{ is stronger than } PA_{HY P}) \]

\[ A_d \vdash \forall x \phi(x) \text{ (by the second and previous line)} \]

Therefore iteration up to \( d \) proves the sentence, and \( |d| = |b| + 1 = \omega_1^{CK} + 1 \). □
A slight modification of the proof shows that for any element in $O^{HYP}$ and $\Pi_1$ sentence one can find an extension in $O^{HYP}$ that proves the sentence. This is analogous to the case in $O$. This also shows that all $\Pi_1$ sentences can be proved in $O^{HYP}$.

**Proposition 9** For any $\Pi_1$ sentence, $\phi$, of $\mathcal{L}_{\omega_1, \omega}^{HYP}$ that is true for $HC(HYP)$, $\bigcup_{a \in O^{HYP}} A_a \vdash \phi$. 
Chapter 4

α-recursion on admissible ordinals

HC(HYP) was used in the previous sections as a natural environment for a meta-recursive iteration of theories because of its strong connection with the meta-recursively enumerable sets. A subset of $\omega_1^{CK}$ is meta-r.e if and only if it is $\Sigma_1$ definable over HC(HYP). The same relation holds between $P(\mathbb{N})$ and the recursively enumerable sets. This seems to indicate that a generalisation can be made to other sets with similar properties to the ones that HC(HYP) ($= L(\omega_1^{CK})$) and $P(\omega)$ ($= L(\omega)$) has. This leads to $\alpha$-recursion and theory of admissible sets.

HC(HYP) was useful because of its equivalent definition as $L(\omega_1^{CK})$, and the fact that meta-recursive subsets of $\omega_1^{CK}$ then was equivalent to $\Sigma_1$-definability over HC(HYP). The choice of language was after that a forced choice in that it had to be complex enough to be able to define all sets in HC(HYP) in an easy way, but also easy enough to be coded into HC(HYP). Likewise the choice of logic was the natural one to this language. It is not hard to imagine that one can get a similar result with a different definition of recursion. This will possibly lead to a new choice of language, logic, and model, but many of the results from previous chapters can be lifted directly.

I have already given an introduction to admissible sets. Now I will say that an ordinal, $\alpha$, is admissible if $L(\alpha)$ is an $\Sigma_1$ admissible set. It is these sets that will be the new model, replacing $L(\omega_1^{CK})$ in a natural way. The $\Sigma_1$-admissible ordinals were first studied by Saul Kripke and Richard Platek as models for Kripke-Platek set theory, a weakening of the axioms of set theory. Using results and terminology from Sacks [I] I call a subset of $\alpha$ $\alpha$-recursively enumerable if it is $\Sigma_1$-definable over $L(\alpha)$, and it is $\alpha$-finite if it
is an element of $L(\alpha)$. Notice here that $\alpha$-recursion is defined to behave similarly to meta-recursively enumerable subsets of $\omega_1^{CK}$. $\alpha$-recursion is in this way a straight generalisation of recursion using $\Sigma_1$ as the defining quality of recursion. Both recursion, and to some degree meta-recursion, has a natural interpretation. Recursion theory is by Church’s thesis closely connected to what actual computers are capable of. Meta-recursion, while allowing infinite computations, only allow recursively infinite computations, so there is still some real-world applicability. Not so when general $\alpha$-admissible ordinals are considered, but the result might be interesting from a theoretic point of view.

As already explained $L(\alpha)$ will be the model for truth when using $\alpha$-recursion for iteration. In the same way the choice of meta-recursion forced my hand in choosing the language $L^{HYP}_{\omega_1, \omega}$. I am now forced to a similar choice. The choice is natural; I let $L^{L(\alpha)}_{|\alpha|', |\alpha|}$ be the language that can have $< |\alpha|'$ many connectives, where $|\alpha|'$ is the successor cardinal of $|\alpha|$, in a formula and all formulas of the language has a code in $L(\alpha)$.

The change of logic will also be minor as the only potentially new thing will be if $|\alpha|' > \omega_1$ so that uncountably infinite connectives are allowed. And then only a change in the one logical rule dealing with infinite connectives will have to be changed. The generalisation is in this way quite straightforward and will not need much explanation.

There really is not that much new to this, any formula of $L^{L(\alpha)}_{|\alpha|', |\alpha|}$ can be inductively mapped into $L(\alpha)$. Relations are then defined for an element of $L(\alpha)$ to be a constant, a term, an atomic formula, a formula and finally a proof from $L^{L(\alpha)}_{|\alpha|', |\alpha|}$. All this will be $\Sigma_1$-definable, thus $\alpha$-r.e. Then a $\Pi_1$-formula expressing its own unprovability can be found and the completeness result follows for any $\Sigma_1$-set of sentences with $L(\alpha)$ as its model. It is easily shown, in Sacks for instance, that for any admissible $\alpha$ the $\alpha$-recursive functions are closed under transfinite $\Sigma_1$-on-$L(\alpha)$ recursion. This makes it possible to once again consider iteration of Gödel-sentences for any $\Delta_1$-on-$L(\alpha)$ ordering.

As theory gets generalized more and more, the application of its results gets less and less. And this is the case here; even if the iteration allowed in some huge $\Sigma_1$-admissible ordinal will be very complex and allows orderings of theories of great complexity.
Proposition 10 For any $\Sigma_1$-admissible ordinal, $\alpha$, let $A_\alpha$ be a set of axioms of $L_{[\alpha], [\alpha]}$ that is $\Sigma_1$ definable over $L(\alpha)$ and has $L(\alpha)$ as a model. Then there exists a $\alpha$-recursive function, $\text{Göd}_\alpha$, such that for any $\sigma < \alpha \text{Göd}_\alpha(\sigma)$ is a code for a Gödel-sentence for $A_\alpha \cup \{\text{Göd}_\alpha(\theta) | \theta < \sigma\}$. 
CHAPTER 4. α-RECURSION ON ADMISSIBLE ORDINALS
Chapter 5

Conclusion

Computability is an important part of proof theory because it is equivalent to formal finite proofs. A computable function is a function where; if $\phi(x)$ terminates then it can be proved in finite time. Conversely a computable proof can be seen as a $\Sigma_1$ relation which again is computable. The equivalence of computation and $\Sigma_1$-relation is similarly connected with finiteness, as a search for a witness to a $\Sigma_1$-relation ends in finite time when such a witness exists. This text has explored the connection between computable and provable, and generalised to different definitions of computability.

When provable is viewed as finite, Gödel's incompleteness result is intuitively clear, as a finite procedure cannot deal with a truly infinite search like checking every instance of a $\Pi_1$ sentence. Of course this is an oversimplification as there certainly are provable $\Pi_1$ sentences, but the intuition is sound nonetheless. Some of the results in Chapter 2 are surprising when viewed this way, Proposition 3 in particular. These chains of axiom sets go way past the finite and even past well-foundedness. That these iterations remain computable shows that the view of computable as finite is flawed. The fact remains though that for every recursively enumerable set of axioms for the natural numbers there is a $\Pi_1$ Gödel-sentence. Even limits of such r.e sets, which can conceivably be used for formal proofs, as only a finite part will be used for each proof, are not complete. An iteration process through $O^*$ creates a countable dense set of axiom sets of varying strength. Potentially this means that for any two axiom sets where one is stronger than the other, one can find an axiom set strictly between these two sets.

Going over to meta-computations the formalisation of proofs is very similar to the computable case and the results reflect this. Meta-computation
does not have the same connection to the real world as computer computations. Meta-computation uses infinite rules and will therefore not occur in the real world, but the infinite rules are chosen to be computable so the extension is in a way the smallest one can do. Meta-computation is therefore seen equivalent to provability using a computably infinite rule of inference, i.e. the logic I use in Chapter 3. This logic system is also seen to be incomplete for meta-recursively enumerable sets of axioms with $L(\omega_1^{CK})$ as a model. The incompleteness is again a consequence of the formalisation of provability. A meta-computable iteration process is then definable. The meta-computable iteration of Gödel-sentences will naturally be longer than a computable one, but as the link with the real world weakens the practical implications lessens. The same applies to an even greater extent to the results of Chapter 4, which is why I only have given a brief overview of this part. The generalisation of the computations and formalisation of proofs to $\Sigma_1$-admissible ordinals is straightforward, the arguments are straightforwardly lifted from meta-recursion. The essential point of using recursion theory and proof theory in tandem is in danger of getting lost in the generalisation though. After all proofs are supposed to be discovered not only studied.
Bibliography


[2] Devlin, K.J.; Perspectives in Mathematical Logic; Constructibility. Publisher: Springer-Verlag, Germany, 1984

[3] Normann, D; Mathematical Logic II. The University of Oslo, Norway, 2005


[7] Turing, A.M.; Systems of Logic Based on Ordinals. Publisher; Princeton University, 1938


59

[10] Schütte, K.; Proof Theory. Publisher; Springer-Verlag, Great Britain, 1977


