LECTURE NOTES

"ON THE INTERPRETATION OF PROGRAM SCHEMES:
AN ALGEBRAIC APPROACH"
M. NIVAT
(IRIA & PARIS VII)

by
M.M. FOKKINGA
(T.H. DELFT)
28-3-1974

This manuscript is an elaboration of the notes made during the lectures of M.Nivat at the "Advanced Course on Semantics of Programming Languages" (Saarbrücken, Germany, Feb 18 - March 1, 1974)

We would like to thank H. Goeman (R.U. Leiden) for stimulating discussions and the Netherlands Organization for Advancement of Pure Research (ZWO) for financial support for attending the course.
(32.2) The definition
(32.3) Theorem on equivalence with magma semantics

3.4 The fixed point semantics
(34.1) Preparation
(34.2) The definition
(34.3) Theorem on inclusion in magma semantics

4 EQUIVALENCE OF RECURSIVE PROGRAM SCHEMES

4.1 Introduction and preparation
(41.1) Preparing definition and theorem
(41.2) The method of Hopcroft and Korenjak

4.2 Imitation of the method of Hopcroft and Korenjak
(42.1) Acceptability and standard form
(42.2) The crucial lemma
(42.3) The algorithm
(42.4) Example

4.3 Remarks to section 4.2
(43.1) The crucial lemma and algorithm fail
(43.2) Other attempts that fail

Remark to section 4.3 (May '74)
1 INTRODUCTION

We will define the semantics of a recursive program scheme in three ways and show their relations (equivalence, inclusion). Thereafter we will discuss the decidability of the equivalence for a restricted class of recursive program schemes.

First of all we derive, by pure syntactic manipulations, some properties which are meaningful under all the semantics of rec.pr. schemes. Thus these results need not be proved more than once and they are very strong because they are valid without any specification of the meaning of the symbols.

The three different definitions of a semantics of rec.pr. schemes are the so-called

- magma semantics, $\text{Val}_I \Sigma$: the common value of the interpretation $I$ of the set of sequences of symbols produced by the rec.pr. scheme $\Sigma$, considered as a rewriting system

- operational semantics, $\text{Comp}_I \Sigma$: input-output behaviour defined by means of stepwise evaluation in the domain $D_I$ of interpretation $I$

- fixed point semantics, $\text{Fix}_I \Sigma$: least fixed point of a mapping $\Sigma$ induced by the scheme $\Sigma$, in the space $D_I$ of functions on the domain $D_I$ of function interpretation $I$.

The sequences of symbols which will be considered frequently consist of well-known so-called well-formed terms, inductively built up from variable symbols and "known function" and "unknown function" symbols, with parentheses and commas. Such a collection of terms is called the free magma generated by $V$, where $F$ the set of all function symbols, $V$ the set of variable symbols, and is denoted by $M(F, V)$.
2. Basic Definitions, Terminology, Properties

2.1. Free F-magma generated by V, \( M(F,V) \)

First of all, a precise description of the sequences of symbols we deal with.

Let \( F \) be a set of function symbols.
Let \( V \) be a set of variable symbols.
Let \( p(f) \) for each \( f \in F \) denote the arity of \( f \) (the arity of \( f \) is to be considered as a primitive notion), \( p(f) \geq 1 \) for all \( f \in F \)!!

The free F-magma generated by \( V, M(F,V) \), is the least set of finite sequences of symbols which contains the sequence \( V \) alone, for each \( v \in V \)
contains the sequence \( f(M, \ldots, M, p(f)) \) whenever \( f \in F \) and \( M, \ldots, M, p(f) \) are in \( M(F,V) \).

Remarks.
1. Each elt of \( M(F,V) \) consist of symbols ( and , and ) and function symbols from \( F \) and variable symbols from \( V \).
2. \( M(F,V) \) can also be considered as the cf. language generated by the cf. grammar

\[
S = \sum_{f \in F} f(S, \ldots, S) + \sum_{v \in V} v
\]

or in another notation

\[
p(f)
\]

by the cf grammar

\[
S \rightarrow f(S, \ldots, S) | f^p(S, \ldots, S) | \ldots | f^{p(n)}(S, \ldots, S) | v^1 | v^2 | \ldots | v^n
\]

all \( f \in F \) \hspace{1cm} all \( v \in V \)

2.2. Intermesso.

We could have been somewhat more general by defining:

- A F-magma is an ordered pair \( \langle E, \sigma \rangle \)
- Where \( E \) is a set, the domain, and
- \( \sigma \) is a collection mappings, for each \( f \in F \) one such mapping

\[
\sigma(f) : E \times (E, \sigma) \rightarrow E
\]

Hence a F-magma yields an interpretation for the system of function symbols \( F \), viz. the domain of interpretation is \( E \) and \( \sigma \) is the association of function symbols in \( F \) with mappings in the domain.

A morphism \( \phi \) from F-magma \( \langle E, \sigma \rangle \) to F-magma \( \langle E', \sigma' \rangle \) is a
mapping \( \varphi : E \to E' \) with
\[
\varphi(f)(m_1, \ldots, m_{p(f)}) = f(\varphi(m_1), \ldots, \varphi(m_{p(f)})) \quad \text{for each } f.
\]
Now, we can state that
for each set \( V \) there exists a unique \( F \)-magma \( M \)
such that \( V \) is contained in the domain of \( M \) and
for each \( F \)-magma \( <E, \sigma> \) and each mapping \( \varphi : V \to E' \)
there is an extension of \( \varphi \) into a homomorphism \( \varphi : M \to <E, \sigma> \).
Because of this property, \( M \) is called free, as usual, and
this unique free \( F \)-magma generated by \( V \) is given by the pair \( <M(F,V), \sigma> \).
where \( M(F,V) \) is defined as above in (2.1.1) and for each \( f \in F \)
\[
\sigma(f) : M(F,V) \to M(F,V) \text{ is defined by: } \quad \text{for all } m_i \in M(F,V)
\]
\[
\sigma(f)(m_1, \ldots, m_{p(f)}) = f(m_1, \ldots, m_{p(f)})
\]
Note that in the r.h.s. of the last equation there is one element
of \( M(F,V) \) which is a sequence of symbols to which \( f \), the parentheses
( and ), the commas) belong and also the sequences of symbols
named by \( m_1, \ldots, m_{p(f)} \). But in the l.h.s. the parentheses and the
commas are part of a notation for function application of
\( \sigma(f) \) on its arguments \( m_1 \) and \( \ldots \) and \( m_{p(f)} \).
By identifying \( M(F,V) \) with \( <M(F,V), \sigma> \), definition (2.1) is justified.

2.2 Substitution

(2.2.1) Definition of factors
The elements of \( M(F,V) \) are sequences of symbols and we can look at
subsequences of them. In relation to substitution processes the
following notions are of interest.
A factor \( f \in M(F,V) \) is a triple \( (\alpha; u; \beta) \)
with \( m = \alpha \cdot u \cdot \beta \) and
\[
u \in M(F,V)
\]
Here \( \alpha, u, \beta \) stands for the concatenation of sequences of symbols,
also called product of \( \alpha, u, \beta \).
Factors \( (\alpha; u; \beta) \) and \( (\alpha'; u'; \beta') \) are disjoint
if there exist \( \alpha'', \beta'' \) such that
\[
either \alpha = \alpha'' \cdot u \cdot \alpha'' \quad \text{or} \quad \alpha' = \alpha'' \cdot u \cdot \alpha'' \quad \text{and} \quad \beta = \beta'' \cdot \beta''
\]
informally, if $u$ and $u'$ are disjoint substrings of $m$:

\[ m \in M(F, U) \]

"$(u, u'; \beta)$ and $(u', u'; \beta')$ are disjoint."

Factor $(u, u'; \beta)$ of $m$ is contained in $(u', u'; \beta')$, $(u, u'; \beta) \subseteq (u', u'; \beta')$ iff there exist $\alpha''$, $\beta''$ such that

$\alpha = \alpha'' \cdot u \cdot \beta''$; $\beta = \beta'' \cdot u' \cdot \beta''$.

Informally, if $u$ is a substring of $u'$:

\[ m \in M(F, U) \]

$(2.2)$ Property of factors. If $(u, u'; \beta)$ and $(u', u'; \beta')$ are factors of $m$, then one of (i), (ii), (iii) holds:

(i) they are disjoint

(ii) $(u, u'; \beta) \subseteq (u', u'; \beta')$

(iii) $(u, u'; \beta) \supseteq (u', u'; \beta')$

Proof: By induction on the complexity; this is a well-known property.

We denote substitution of $M_i$ for $u_i$ for $i = 1, \ldots, n$ and $M_i, u_i \in M(F, U)$ in $f$ by $f(M_1, u_1, \ldots, M_n, u_n)$. It is strongly emphasized that the parentheses and commas displayed in $f(M_1, u_1, \ldots, M_n, u_n)$ do not belong to the resulting expression in $M(F, U)$.

2.3 Rewriting systems

In the sequel we let $\Phi = \{ \phi, \ldots, \phi_k \}$ be a finite set of "unknown function" symbols, and $F$ be a possibly infinite set of "known function" symbols. Moreover, we assume $V$ to be countable, and fix the enumeration $v_1, v_2, \ldots$ throughout the paper. Please note the difference between $M(F, U)$ and $M(F \cup \Phi, V)$. 
Definitions.

A rewriting system \( \Sigma \) over \( M(F, V) \) is a collection equations
\[
\{ q_i(x_1, \ldots, x_{\rho(q_i)}) = r_i \quad \text{in which} \quad \tau_i \in M(F \cup \Phi, \{ x_1, \ldots, x_{\rho(r_i)} \}) \}
\]
\( i = 1, \ldots, N \).

Example,

Let the rewriting system be given by
\[
\Sigma = \{ q_i(x, y) = f_i(x, y, \phi_i(y, x, \phi_i(y, x, \phi_i(x, y))) \}
\]
\( q_i(x, y) = g_i(x, y, f_i(x, y), x, x) \quad *) \)

Then
\( \Phi = \{ q_1, q_2 \} \) and \( N = 2 \), \( P(q_i) = P(q_j) = 2 \) and \( V = \{ x, y, \ldots, f \} \). Furthermore \( f, g \) and \( f \) belong to \( F \) and \( P(f) = 2, P(g) = 4, P(f) = 3 \).

For \( f, f' \in M(F \cup \Phi, V) \) we say \( f \) derives immediately \( f' \) in \( \Sigma \), \( f \Rightarrow f' \),
iff there is a factor \( (x_i, u, \beta) \in F \) such that
\( u = q_i(m, \ldots, m, \rho(q_i)) \) and
\( f' = \alpha \tau_i(m, u, \ldots, m, \rho(q_i), \rho(q_i)) \cdot \beta \).

Informally, \( f \) can be obtained from \( f \) by a rewriting of a subterm according to the rewriting equation in which the variables \( \phi_i \), \( \phi_j \) are replaced by the current arguments \( m, \ldots, m, \rho(q_i) \).

Similarly we say \( f \) derives \( f' \) in \( \Sigma \), \( f \Rightarrow f' \),
iff there are \( f_1, \ldots, f_{k+1} \) such that
\( f = f_1 \) and for \( h = 1, \ldots, k \) \( f_h \Rightarrow f_{h+1} \), and finally \( f_{k+1} = f' \),
\( \Rightarrow \) is the reflexive transitive closure of \( \Rightarrow \).

A derivation of \( f \) into \( f' \) in \( \Sigma \) is a \( k+1 \)-tuple \( \langle f_1, \ldots, f_{k+1} \rangle \)
such that \( f = f_1 \) and for \( h = 1, \ldots, k \) \( f_h \Rightarrow f_{h+1} \), and finally \( f_{k+1} = f' \).

By convention we denote by \( (x_i, u, \beta) \) the factor to be replaced in \( f \) in order to obtain \( f' \).

A derivation is called leftmost.
iff for all \( h \) \( |x_i| \leq |x_{k+1}| \), where \( |x_i| \) is number of symbols in \( x_i \).

(23.2) Theorem on leftmost derivations.

\( f \Rightarrow f' \) if and only if there is a leftmost derivation \( g \) of \( f \) into \( f' \) in \( \Sigma \).

Proof:

(essentially the proof of H. Fischer given for macro-grammar.

*) which can be interpreted as computing the greatest common divisor, see ex (31.1).
let $d = \langle f_1, ..., f_{kh} \rangle$ be a derivation of $f$ into $f'$ in $\Sigma$.

Let $\pi(d) = \text{card} \{ h \in \{1, ..., kh\} : |x_h| > |x_h'| \} =$ number of steps

We will give an algorithm such that a derivation $d'$ of $f$ into $f'$ with $\pi(d') > 0$ is transformed into a derivation $d''$ of $f$ into $f'$ with $\pi(d'') < \pi(d')$. Thus by induction $d$ can be transformed into a derivation $d''$ of $f$ into $f'$ with $\pi(d'') = 0$, i.e. $d''$ is leftmost.

Now, let $h$ be the smallest number s.t. $|x_h| > |x_{h+1}|$, then

\[ f_{h+1} = \alpha_h \cdot u_h \cdot \beta_h \]

\[ f_{h+1} = \alpha_h \cdot u_h \cdot \beta_h \quad \text{where} \quad u_h = q_j(m_1, ..., m_r) \quad \text{and} \quad u_h = \tau_j(m_1, l_1, ..., l_s) \]

and looking at the next derivation step we can set

\[ f_{h+1} = \alpha_h \cdot u_h \cdot \beta_h \]

According to property (22.2) the factors $(\alpha_h \cdot u_h ; \beta_h)$ and $(\alpha_h \cdot u_h ; \beta_h)$ either are contained or are disjoint.

If disjoint then due to $|x_h| > |x_{h+1}|$ we have

\[ f_{h+1} = \alpha_h \cdot u_h \cdot \beta_h \quad \text{and we define} \]

\[ f_{h+1} = \alpha_h \cdot u_h \cdot \beta_h \quad \text{where} \quad f_{h+1} = \alpha_h \cdot u_h \cdot \beta_h \leq \pi(d) \]

Now, it is easy to see

\[ f_{h+1} = \alpha_{h+1} \cdot f_{h+2} \cdot f_{h+2} \cdot f_{h+2} \cdot f_{h+2} \quad \text{and moreover} \]

\[ \pi(f_{h+1} \cdot f_{h+2} \cdot f_{h+2} \cdot f_{h+2} \cdot f_{h+2}) < \pi(d) \]

If contained then due to $|x_h| > |x_{h+1}|$ we have $(\alpha_h \cdot u_h ; \beta_h) \subset (\alpha_h \cdot u_h \cdot \beta_h)$ and because $u_h \neq u_{h+1}$ we have (loosely formulated) that $u_h$ is contained in one of the factors of $u_{h+1} = q_j(m_1, ..., m_r \cdot \phi_j)$, say in $m_k$, so $m_k = \tau_j u_h \cdot \beta_h$.

Denote by $m_k = \tau_j u_h \cdot \beta_h$, then we have

\[ f_{h+1} = \alpha_h \cdot u_h \cdot \beta_h \]

\[ f_{h+1} = \alpha_h \cdot \tau_j \cdot (m_1, ..., m_r \cdot \phi_j) \cdot \beta_h \]

\[ f_{h+2} = \alpha_h \cdot \tau_j \cdot (m_1, ..., m_r \cdot \phi_j) \cdot \beta_h \]

\[ f_{h+2} = \alpha_h \cdot \tau_j \cdot (m_1, ..., m_r \cdot \phi_j) \cdot \beta_h \]

Define

\[ \beta_{h+1} = \alpha_h \cdot \tau_j \cdot (m_1, ..., m_r \cdot \phi_j) \cdot \beta_h \]

Then clearly
In one leftmost step, and to do

in a leftmost way we have to substitute

for \( m^n \) in each occurrence of \( m^n \) coming

from an occurrence of \( v_\ell \) in \( \tau_i \). This indeed

can be done in a leftmost way. Hence

\[
\tau_i \left( \begin{array}{c}
\frac{f}{m^n} \\
\frac{f}{m^n} \\
\end{array} \right) \leq \tau_i \left( \begin{array}{c}
\frac{f}{m^n} \\
\frac{f}{m^n} \\
\end{array} \right).
\]

end of the proof.

(23.3) Remark.

Let \( d( f, \ldots, f_{n+1} ) \) be a leftmost derivation of \( f \) into \( f' \in M(F, V) \),

then it is easy to check that each \( d_\ell \) does not contain any \( q \in \Phi \).

Call a factor \( \alpha_i ; u_j \beta \) replaceable iff \( u = \varphi_i (\ldots) \) with \( \varphi_i \in \Phi \).

Call a replaceable factor maximal iff it is not contained in

an other one. Then in every leftmost derivation \( \Sigma \) \( f \to f' \in M(F, V) \),
each factor \( \langle d_\ell ; u_j ; \beta \rangle \) is a maximal replaceable factor.

Hence leftmost often is called leftmost outermost.

2.4 The symbol \( \Sigma \), schematic variants of rewriting systems

(24.1) Definitions.

In the sequel we let \( V \) be \( \{ v_1, v_2, \ldots \} \).

Thus \( \Sigma \) being a distinguished set of \( V \), not appearing in the

elementation \( v_1, v_2, \ldots \).!!

The schematic variant \( \Sigma \) of a rewriting system \( \Sigma \) is defined

by \( \Sigma \): \( \{ q_\ell (v_1, \ldots, v_0) \} = \tau_i \in \Sigma \)

\[ i = 1, \ldots, N \]

where for \( i = 1, \ldots, N \), \( q_\ell (v_1, \ldots, v_0, v_j) = \tau_i \in \Sigma \).

We adjust the definition \( \Sigma \) of derives (immediately) \( f' \in \Sigma \) as

follows: if \( \alpha \Sigma \beta \) is a factor \( \Sigma \), and \( u = q_\ell (m_1, \ldots, m_0, v_j) \)

then both \( f' = \alpha \Sigma \beta \) \( m, v_1, \ldots, m_0, v_j \Sigma v_j \beta \).

and \( f'' = \alpha \Sigma \beta \) \( v_j \Sigma v_j \beta \),

derive immediately from \( f \in \Sigma \).!!
The ordering relation $\leq$.

Due to the introduction of the symbol $\leq$ with intended interpretation "the unspecified sequence of symbols", it is natural to express the interpretation of "being less specified than" by an partial ordering relation $\leq$.

We will define it and investigate its properties.

The magma ordering $\leq$, on $M(FuS, V)$ is defined as the coarsest relation compatible with the magma structure and such that $\sigma \leq \tau$ for all $\sigma, \tau \in V$.

i.e. for $m, m' \in M(FuS, V)$, we have $m \leq m'$ if

- $\exists \alpha, \beta, \ldots, \alpha_{p+1}$ and $\omega, \omega_1, \ldots, \omega_p$ all $\omega_i \in M(FuS, V)$ s.t.
  
  $m = \alpha \cdot \omega \cdot \beta \cdot \omega_1 \cdot \ldots \cdot \omega_p \cdot \alpha_{p+1}$
  
  $m' = \alpha \cdot \omega \cdot \beta \cdot \omega_1 \cdot \ldots \cdot \omega_p \cdot \alpha_{p+1}$

Note that this defines $\iota' \leq \iota''$ for $\iota', \iota'' \in V$ iff $\iota' = \iota''$ or $\iota' \neq \iota''$.

(24.3) Theorem on the lattice structure of $\leq$.

The magma-language of $\leq$ system $\Sigma$ is defined as

$L(\Sigma, \leq) = \{ f' \in M(F, V) : f \leq f' \}$

for $f \in M(FuS, V)$ and $\Sigma$, a retor system or schematic variant.

Theorem.

The restriction $f \leq$ to $L(\Sigma, \leq)$ is a lattice order, i.e.

- if $f \leq f_1$ and $f \leq f_2$ then there exist $f_3, f_4 \in L(\Sigma, \leq)$ such that both $f_3 < f_1$ and $f_3 < f_2$
  
  $\cup$ both $f_3 < f_1$ and $f_3 < f_2$

- and $\cup$ if for some $f_3 \in L(\Sigma, \leq$) both $f_3 < f_1$ and $f_3 < f_2$ then $f_3 < f_3$

- if for some $f_4 \in L(\Sigma, \leq)$ both $f_4 < f_1$ and $f_4 < f_2$ then $f_4 < f_4$.

Proof:

(Only part (b) is needed in the sequel). We only prove part (a) and (b) and leave (c) and (d) as an exercise.

Informally, look at leftmost derivations $d_1, d_2, d_3, f$ into $f_1, f_2, f$ and take for the derivations $d_3, d_4$ of $f$ into $f_3, f_4$ the corresponding rewriting steps, if they are equal, and otherwise in $d_3$ the
rewriting step into $\Sigma$ and in $a_1$, the rewriting into $\text{not-}\Sigma$.

Formally, by induction to the sum of lengths of leftmost
derivations $d_1 = \langle q_1, \ldots, q_{k + 1} \rangle$ and $d_2 = \langle \ell_1, \ldots, \ell_{k + 1} \rangle$ for
$f \mapsto f_1$ and $f \mapsto f_2$ respectively, in $\Sigma$.

If $k + 1 = 0$,
then $f = f_1 = f_2 \in M(\Sigma)$, so we can take $f_3 = f_1$ to be $f_1 = f_2 = f$.

If $k + 1 > 0$,
then $f = g_1 = \alpha_1 \cdot u_1 \cdot \beta$, with $g_2 = \alpha_2 \cdot u_2 \cdot \beta_2$ and $g_{k + 2} = f_1 \in M(\Sigma)$
and $f_2 = f_3 = \alpha_3 \cdot u_1 \cdot \beta$, with $h_2 = \alpha_4 \cdot u_2 \cdot \beta_2$ and $h_{k + 1} = f_2 \in M(\Sigma)$.

Now,
if $(\alpha, u, \beta)$ and $(\alpha', u', \beta')$ are disjoint, say $\alpha_1 = \alpha' \cdot u' \cdot \alpha''$
then $\alpha_1$ contains a $q \in \Phi$ and due to the leftmost
property $f_1$ would contain that $q \in \Phi :$ contradiction $f_2$.

And
if $(\alpha; u; \beta')$ is proper contained in $(\alpha'; u'; \beta')$, say $\alpha_1 = \alpha' \cdot u''$
then similarly $\alpha''$ hence $\alpha_1$ contains a $q \in \Phi$. $f_2$.

Hence
$(\alpha, u, \beta)$ equals $(\alpha', u', \beta')$, so that we can say
$\alpha_1 = \alpha' = \alpha''$, $\beta_1 = \beta' = \beta'$ and $u_1 = u' = \Phi (M_1, \ldots, M_{\nu(F)}).$

Define $w = F \cdot (M_1, u_1, \ldots, M_{\nu(F)} / u_1(F))$, and now
there are four cases for $g_2$ and $h_2$:

(i) $g_2 = \alpha_2 \cdot \omega_2 \cdot \beta$, (ii) $g_2 = \alpha_1 \cdot \omega_2 \cdot \beta$, (iii) $g_2 = \alpha_1 \cdot \omega_2 \cdot \beta$,
and $h_2 = \alpha_2 \cdot \omega_2 \cdot \beta$. $h_2 = \alpha_2 \cdot \omega_2 \cdot \beta$.

So that, due to the leftmost property, in each $\Sigma$ the cases
$\forall j = \alpha_2 \cdot \omega_2 \cdot \beta$, with either $\omega_2 = \Sigma^* \cdot \omega_2$ and $\beta \not\in \beta_1 \in M(\Sigma)$
$\forall k = \alpha_2 \cdot \omega_2 \cdot \beta$, with either $\omega_2 = \Sigma^* \cdot \omega_2$ and $\beta = \beta_2 \in M(\Sigma)$.

By induction hypothesis, there exist $\omega_3$, $\omega_4$ and $\beta_3$, $\beta_4$:

(i) $\omega_3 = \omega_2$, (ii) $\omega_4 = \omega_2$, (iii) $\omega_4 = \omega_2$, (iv) $\omega_4 = \omega_2$ (by induction)

$\omega_3 < \omega_4 < \omega_4 < \omega_4$, $\omega_4 < \omega_4$, $\omega_4 < \omega_4$, $\omega_4 < \omega_4$, $\omega_4 < \omega_4$, $\omega_4 < \omega_4$, $\omega_4 < \omega_4$, $\omega_4 < \omega_4$.

$\beta_3 < \beta_1 \cdot \beta_2 < \beta_1 \cdot \beta_2 < \beta_1 \cdot \beta_2$, $\beta_3 < \beta_1 \cdot \beta_2 < \beta_1 \cdot \beta_2$, $\beta_3 < \beta_1 \cdot \beta_2 < \beta_1 \cdot \beta_2$.

Therefore, we can take
$\beta_3 = \alpha_2 \cdot \omega_2 \cdot \beta_2$ and $\beta_4 = \alpha_2 \cdot \omega_2 \cdot \beta_4$,
and clearly $\beta \mapsto f_3$, $f_4$ in $\Sigma$, and $\beta_3 < f_3$, $f_4 < f_4$.

end of proof.
Kleene's sequences.

We now give the interesting notion of a Kleene's sequence $\varepsilon_f$ with $\varepsilon_f$, being defined as an increasing infinite sequence $\langle f^{(k)} \rangle_{k=0}^{\infty}$, so $f^{(k)} \in L(\varepsilon_f)$, which majorizes all elements in $L(\varepsilon_f)$, i.e., $f^{(0)} < f^{(1)} < \ldots < f^{(k)} < \ldots$ where $f^{(k)} \in L(\varepsilon_f)$, and for each $f' \in L(\varepsilon_f)$ there is an $k$ such that $f' \prec f^{(k)}$.

Strong and weak derivations.

For the construction of Kleene's sequences and for other purposes, we need the following definitions.

$\varepsilon_f \rightarrow f'$ is a strong derivation step, $f \Rightarrow f'$, if $f = \alpha \cdot (m_1, \ldots, m_{\rho(f)}) \cdot \beta$ and $f' = \alpha \cdot \gamma (m_1, v_1, \ldots, m_{\rho(f)} | v_{\rho(f)}) \cdot \beta$.

$\varepsilon_f \rightarrow f'$ is a weak derivation step, $f \rightarrow f'$, if $f = \alpha \cdot (m_1, \ldots, m_{\rho(f)}) \cdot \beta$ and $f' = \alpha \cdot \gamma \cdot \beta$.

With the obvious extensions for $\Rightarrow$ and $\rightarrow$.

We will give the construction and the proof of the existence of Kleene's sequences after six lemmas, establishing among others that in derivations $\varepsilon_f \rightarrow f'$, the strong derivation steps can be done before the weak ones (lemma 6).

Let $S : M(F, E, V) \rightarrow M(F, E, V)$ be a mapping, inductively defined as:

$S(\emptyset) = \emptyset$

$S(f(m_1, \ldots, m_{\rho(f)}) = f(S(m_1), \ldots, S(m_{\rho(f)}))$

$S(\phi(m_1, \ldots, m_{\rho(f)}) = \overline{\gamma}(S(m_1) | v_1, \ldots, S(m_{\rho(f)}) | v_{\rho(f)})$

Thus performing at once all possible strong derivation steps simultaneously.

Let $W : M(F, E, V) \rightarrow M(F, V)$ be a mapping, inductively defined by:

$W(\emptyset) = \emptyset$

$W(f(m_1, \ldots, m_{\rho(f)}) = f(W(m_1), \ldots, W(m_{\rho(f)}))$

$W(\phi(m_1, \ldots, m_{\rho(f)}) = \emptyset$

Thus performing at once all possible weak derivation steps simultaneously, by simply replacing all replaceable factors by $\emptyset$. 


Lemma 1. \( f \Rightarrow f' \) implies \( f' \Rightarrow S(f) \)

Proof by induction on the number \( \sum f_i \) of symbols in \( f \).

Lemma 2. \( f \Rightarrow f' \) implies \( S(f) \Rightarrow S(f') \)

Proof by induction on the number \( \sum f_i \) of symbols in \( f \).

Lemma 3. \( f \Rightarrow f' \) implies \( S(f) \Rightarrow S(f') \)

Proof by induction on the length \( |f| \) of derivation.

Lemma 4. \( f \Rightarrow f' \) implies \( f' \Rightarrow S^k(f) \)

Proof by induction on the length \( |f| \) of derivation.

Lemma 5. \( f \Rightarrow f'_2 \Rightarrow f'_3 \) implies \( f 
\Rightarrow f'_y \Rightarrow f'_3 \) for some \( f'_y \)

Proof: by induction on the length \( |f| \) of the derivation \( f \Rightarrow f'_2 \)

Length = 0: Trivial

Length = 1: We have to make the diagram

\[
\begin{array}{c}
\downarrow \quad \downarrow \\
f \Rightarrow f'_1 \\
\downarrow \quad \downarrow \\
 f_2 \Rightarrow f'_3
\end{array}
\]

Commutative, which can easily be done.

Length > 1: We have the situation (i), which by the preceding case reduces to (ii), which by ind. hyp. reduces to (iii).

\[
\begin{array}{c}
\downarrow \quad \downarrow \\
f \Rightarrow f'_1 \\
\downarrow \quad \downarrow \\
 f_2 \Rightarrow f'_3
\end{array}
\]

\[
\begin{array}{c}
\downarrow \quad \downarrow \\
f \Rightarrow f'_1 \\
\downarrow \quad \downarrow \\
 f_2 \Rightarrow f'_3
\end{array}
\]

\[
\begin{array}{c}
\downarrow \quad \downarrow \\
f \Rightarrow f'_1 \\
\downarrow \quad \downarrow \\
 f_2 \Rightarrow f'_3
\end{array}
\]

Lemma 6: \( f \Rightarrow f' \Rightarrow f'' \in M(F,V) \) and in the derivation there are exactly \( k \) strong derivation steps,

then \( f \Rightarrow f'' \) and \( W(f'') = f' \) for some \( f'' \).

Proof: by induction on \( k \) we show \( f \Rightarrow f'' \Rightarrow f' \). Then, because \( f' \in M(F,V) \), clearly \( W(f'') = f' \) (no q's are left!).

\( k = 0 \): Trivial.

\( k = 1 \): \( f \Rightarrow f_1 \Rightarrow f_2 \Rightarrow f' \) reduces by ind. hyp. to \( f \Rightarrow f_1 \Rightarrow f_2 \Rightarrow f_3 \Rightarrow f' \)

and by Lemma 5 this reduces to \( f \Rightarrow f'' \Rightarrow f'_y \Rightarrow f'_3 \Rightarrow f' \).

Compare the following process in making the diagram commutative: each step is done by applying Lemma 5:
Theorem (24.6)
The sequence \( f(i) \) where \( f(k) = W(S^k(f)) \) is a Kleene sequence w.r.t. \( \Sigma \).

Proof: (a) it is an increasing seq. w.r.t. \( \leq \), (b) it majorizes \( L(\Sigma, f) \).

(a) By replacing in \( S^k(f) \) all replaceable factors by \( \Sigma \) we obtain \( f(k) \),

whereas \( f(k+1) \) is obtained from \( S^k(f) \) by replacing all replaceable
factors, say \( q_i(m, \ldots) \), by \( W(q_j(m, \ldots)) \); hence \( f(k) \leq f(k+1) \).

(b) Let \( f' \in L(\Sigma, f) \), i.e., \( f(k) \rightarrow f' \) say with exactly \( k \) strong steps, \( f' \in M(f) \).

By lemma 6 \( f(k) \Rightarrow f' \) with \( W(f') = f'' \) and

by lemma 4 \( f'' \rightarrow S^k(f) \), whereas \( W(S^k(f)) = f(k+1) \).

Hence \( f(k+1) \) and \( f \) are the same except for some occurrences of \( \Sigma \) in \( f' \) which are replaced by something else in \( f(k) \): \( f' \preceq f(k) \).

Example

Let \( \Sigma = \{ q_1(xy) = a(x, q_0(x, y)) \}

\( q_2(x, y) = g(x, y, q_1(x, y)) \}

Let \( f = q_1(x, y) \).

Then we construct the Kleene sequence of \( f \) w.r.t. \( \Sigma \) as follows:

\( S^0(f) = q_1(x, y) \)

\( S^1(f) = a(x, q_0(x, y)) \)

\( S^2(f) = a(x, g(x, y, q_1(x, y))) \)

\( S^3(f) = a(x, g(x, y, a(x, q_0(x, y)))) \)

\( f(0) = W(S^0(f)) = a(x, q_0(x, y)) \)

\( f(1) = W(S^1(f)) = a(x, q_0(x, y)) \)

\( f(2) = W(S^2(f)) = a(x, g(x, y, q_1(x, y))) \)

\( f(3) = W(S^3(f)) = a(x, g(x, y, a(x, q_0(x, y)))) \)

Characterization of the language \( \Sigma \) a rewriting system

Informally.

In considering a recursive program scheme as a rewriting system \( \Sigma \),
the language of the system could be considered as yielding the
meaning of the program. Therefore we look for a convenient
characterization of that language. The definition will be
given by means of the notion derivation, which corresponds in an obvious way to an operational approach for the semantics of the program. We prefer "let us say" a fixed point approach.

This means that we have to associate with $\Xi$ a mapping $\hat{\Xi}$ such that the language can be a fixed point and is indeed the least one under an appropriate ordering. We take the domain to be the set $t$ of terms in $M(Fu^{P}V)$ and we take $\hat{\Xi}$ to map each set $t$ of terms in $M(Fu^{P}V)$ into the set $t^t$ of terms which are derivable from the r.h.s of the $i$-th rewriting equation, according to rewritings induced by the given terms $t_i$.

But as we have $i = 1,\ldots, N$, we have to do all in $N$-tuples and coordinate wise.

(25.2)

Definitions.

The language of the rewrite system $\Xi$ is $\mathcal{L} = \langle L_1, \ldots, L_N \rangle$, where $L_i = \mathcal{L}(\Sigma, q_i, \varphi_i(t, \varphi_i')) = \{ f \in M(F^P) : q_i(t, \varphi_i') \overset{\Xi}{\Rightarrow} f \}$ for $i = 1,\ldots, N$.

Let $T$ be the set of all $N$-tuples $t = (t_1, \ldots, t_N)$ with $t_i \in M(F^P \{ q_i \varphi_i' \})$.

Let $T$ be ordered by coordinatewise set-theoretic inclusion, i.e., $t \subseteq t'$ iff $t_i \subseteq t'_i$ for $i = 1,\ldots, N$. Then $T$ forms a complete lattice.

Let for each $t = (t_1, \ldots, t_N) \in T$ a mapping $\lambda_t : \text{sets of subsets of } M(F^P, V) \rightarrow \text{sets of subsets of } M(F^P, V)$, by defining $\lambda_t$ for singleton subsets inductively by

$\lambda_t(\{ v \}) = \{ v \}$

$\lambda_t(\{ f(m_1, \ldots, m_p) \}) = \{ f(m'_1, \ldots, m'_p) : m'_i \in \lambda_t(m_i) \text{ for } i = 1,\ldots, N \}$

$\lambda_t(\{ g(m_1, \ldots, m_p) \}) = \{ g(m'_1, \ldots, m'_p) : g \in t_i \text{ and } m'_i \in \lambda_t(m_i) \}$

and by defining

$\lambda_t(\Sigma) = \bigcup_{s \subseteq \Sigma} \lambda_t(s)$ for $\lfloor t \rfloor$-singleton $\Sigma \subseteq M(F^P, V)$ and by coordinatewise application $\lambda_t$ can deal with $N$-tuples.

Now we associate with $\Xi$ a mapping $\hat{\Xi}$,

$\hat{\Xi} : T \rightarrow T$ defined by

$\hat{\Xi}(t) = \lambda_t(t)$ where $t = (t_1, \ldots, t_N)$

Then $\hat{\Xi}$ is increasing, i.e., for $t \subseteq t'$ we have $\hat{\Xi}(t) \subseteq \hat{\Xi}(t')$, and even $\hat{\Xi}$ is continuous, i.e., for $t_1 \subseteq t_2 \subseteq \ldots$ we have $\hat{\Xi}(\bigcup t_i) \subseteq \bigcup \hat{\Xi}(t_i)$.

Hence by the Knaster-Tarski theorem,
There is a least fixed point \( s \) of \( \hat{\Sigma} \) in \((T, \subseteq)\), viz.
\[
s = \bigcup_{k \geq 0} \hat{\Sigma}^k(\emptyset)
\]
where \( \emptyset = \langle \emptyset, \ldots, \emptyset \rangle \in T \).

And intuitively, this can be read as follows:
the least fixed point of \( \hat{\Sigma} \) consists of the (\( T \)-tuples of
sets \( \emptyset \)) terms obtained from the r.h.s. of the new eqns
by replacing the \( \emptyset \) by "something from \( \emptyset \)" or by previously obtained
terms for the \( j \)-th r.h.s.

(25.3) **Theorem**

\( L \) is the least fixed point of \( \hat{\Sigma} \)

(This is an extension of Schützenberger's Theorem for c.f. languages)

We are going to prove \( L = s \) (see previous section) by use of

**Lemma 1:** if \( g' \in \Lambda_t(g) \) for \( g \in M(F, \Phi, V) \) then \( g \stackrel{\Lambda_t}{\rightarrow} g' \) (Exercise),

**Lemma 2:** if \( g \not\rightarrow g' \) then \( \Lambda_t(g) \subseteq \hat{\Lambda}_{\hat{\Sigma}^*}(t)(g) \) (Proof???)

where
\[
\hat{\Lambda}_t(t) = \hat{\Sigma}(t) \cup t \quad \text{and} \quad \hat{\Lambda}_{\hat{\Sigma}^*}(t) = \bigcup_{k \geq 0} \hat{\Lambda}_{\hat{\Sigma}^k}(t) \quad (\text{a clever trick, isn't it?})
\]

Now we prove both \( L \leq s \) and \( L \geq s \).

\( \leq \) : take \( g = g_i(v_i, \ldots, v_{\phi(p_i)}) \) and \( t = \emptyset \in T \),
then for all \( g' \in M(F, V) \) with \( g \stackrel{\Lambda_t}{\rightarrow} g' \), i.e. for all \( g' \in L \),
by applying lemma 2,
\[
\Lambda \emptyset (g') = \{ g' \} \subseteq \hat{\Lambda}_{\hat{\Sigma}^*}(\emptyset) \quad g_i(v_i, \ldots, v_{\phi(p_i)}) = \left[ \hat{\Sigma}^*(\emptyset) \right]_i \text{-th comp.}
\]
hence
\[
L_i \subseteq \left[ \hat{\Sigma}^*(\emptyset) \right]_i \text{-th component for } i = 1, \ldots, N \text{, so}
L \subseteq \hat{\Sigma}^*(\emptyset) = \hat{\Sigma}^*(\emptyset) = s
\]

\( \geq \) : we will show that \( L \) is a fixed point of \( \hat{\Sigma} \), then
\( L \geq s \). holds too, because \( s \) is the least fixed point.

Take \( g = t_i \) and \( t = L \),
then for all \( g' \in \hat{\Sigma}(L) \), i.e. for all \( g' \in \lambda_L(t_i) \),
by applying lemma 1,
\[
g \not\rightarrow g' \text{ i.e. } g' \in L_i \text{; hence } \hat{\Sigma}(L) \subseteq L
\]
and now using the additional property \( \hat{\Sigma}^*(L) \subseteq L \) and
by applying lemma 2:
\[
\lambda_L(g') = \{ g' \} \subseteq \hat{\Lambda}_{\hat{\Sigma}^*(L)}(t_i) \subseteq \lambda_L(t_i) = \left[ \hat{\Sigma}(L) \right]_i \text{-th comp.}
\]
Hence

\[ (*) \] so that in this case the whole expression disappears, cfr. \( \lambda_L(t_m) \).
\[ L_i \leq [\hat{E}(L)] \text{th-comp}, \text{ so } L = \hat{E}(L) \]
Together (i), (ii) state \( L = \hat{E}(L) \)

end of sketch
3 REC. PR. SCHEMES, INTERPRETATION AND SEMANTICS.

We now try to use the previous results of the theory in the study of semantics of recursive program schemes.

3.1 Preliminary

(3.1.1) Informal introduction.

Suppose the following is an example of a recursive program:

\[ \text{gcd} : \begin{cases} g(x, y) = \text{if } x = 0 \text{ then } y \text{ else } g(y, g(x, y)) \\ g_2(x, y) = \text{if } x < y \text{ then } x \text{ else } g_2(x - y, y) \end{cases} \]

We will consider \( \text{gcd} \) as an instance of the following axiom system \( \Sigma \):

\[ \begin{cases} f(x, y) = \text{if } x = y \text{ then } x \text{ else } \text{gcd}(y, f(x, y)) \\ f_2(x, y) = \text{if } x < y \text{ then } x \text{ else } f_2(f(x, y), y) \end{cases} \]

by the interpretation \( I \), given by

\[ I : \begin{cases} \text{Domain } D_I = \text{ the set of natural numbers} \\ h_I(m, n, p) = \text{if } m = 0 \text{ then } n \text{ else } p \\ q_I(m, n, p) = \text{if } m < n \text{ then } m \text{ else } p \\ f_I(m, n) = m - n \end{cases} \]

Now, we want to define a semantics, assigning to each program \( \langle \Sigma, I \rangle \) a function in the domain \( D_I \) which can be considered as the meaning for the program \( \langle \Sigma, I \rangle \).

We will do this in three ways: a magma semantics, an operational semantics, a fixed point semantics.

(3.1.2) Interpretation and valuation.

An interpretation \( I \) is given by

- a nonempty domain \( D_I \) which is a natural number domain,
- for all \( f \in F \) a partial mapping \( f_I : D_I^{|f|} \rightarrow D_I \)

but as we will deal with total functions we extend the domain

- with a new element \( * \) ( the "undefined value"

and we extend the \( f_I \) into

\[ f_I : (D_I \cup \{*\})^{|f|} \rightarrow D_I \cup \{*\} \]

by defining

- for all \( d_1, \ldots, d_{|f|}, p \in D_I \)

\[ f_I(d_1, \ldots, d_{|f|}, p) = \begin{cases} f_I(d_1, \ldots, d_{|f|}) & \text{if this value is defined} \\ * & \text{otherwise} \end{cases} \]

\(*\): so that \( \text{gcd} \) computes the greatest common divisor.
for all $d_1, \ldots, d_k, \phi \in D_\Sigma \cup \{w\}$, where exactly $d_1 = w, \ldots, d_k = w$,
$$f_\Sigma(d_1, \ldots, d_k, \phi) = \begin{cases} f_\Sigma((d_1, \ldots, d_k, \phi), (d_1/d_1, \ldots, d_k/d_k)) & \text{if this value does not vary with } d_1, \ldots, d_k \in D_\Sigma \\ w & \text{otherwise} \end{cases}$$

That is, we use the call-by-name evaluation for the function $f_\Sigma$.

By convention, we assume from here onwards:

- $D_\Sigma$ contains $w$
- for each $f \in F$, $f_\Sigma$ is a total mapping $f_\Sigma : D_\Sigma^\phi \to D_\Sigma$ with $f_\Sigma((d_1, \ldots, d_k, \phi), (d_1/d_1, \ldots, d_k/d_k)) = d + w$ whenever $f_\Sigma(d_1, \ldots, d_k, \phi) = d$.

The discrete ordering $\leq$ on $D_\Sigma$ is given by $d_1 \leq d_2$ if $d_1 = w$ or $d_1 = d_2$,

and clearly the $f_\Sigma$ are increasing wrt $\leq$, i.e. $d_1 \leq d_2$, $d_k, \phi \in D_\Sigma^\phi$ implies $f_\Sigma(d_1, \ldots, d_k, \phi) \leq f_\Sigma(d_1, \ldots, d_k, \phi)$.

A valuation is a mapping $\nu : V \to D_\Sigma$

The mapping $(I, \nu) : M(F, V) \to D_\Sigma$ is inductively defined by

$(I, \nu)(\nu) = \nu(\nu)$ for all $\nu \in V$

$(I, \nu)(w) = w$

$(I, \nu)(f(M_1, \ldots, M_k, \phi)) = f_\Sigma((I, \nu)(M_1), \ldots, (I, \nu)(M_k, \phi))$

3.2 The magma semantics

Preparation.

Lemma. $(I, \nu)$ is order preserving, i.e. $\forall m, m' \in M(F, V); m < m' \Rightarrow (I, \nu)m < (I, \nu)m'$

proof: by induction on the number $\| m \|$ of function symbols in $m$.

$\| m \| = 0$: then $m = w$ and $(I, \nu)m = w$ so for all $m'$, $(I, \nu)m \leq (I, \nu)m'$.

or $m = \nu$ and consequently $m' = \nu$ hence $(I, \nu)m = (I, \nu)m'$.

$\| m \| > 0$: $m = f(M_1, \ldots, M_k, \phi)$ and consequently for $m < m'$

$m' = f'(M'_1, \ldots, M'_k, \phi)$ with $f = f'$ and $M'_e < M_e$ for $e = 1 \ldots p(f)$.

Hence by induction $(I, \nu)M'_e \leq (I, \nu)M_e$ for $e = 1 \ldots p(f)$ and because $f_\Sigma$ is increasing $f_\Sigma((I, \nu)M_1, \ldots, (I, \nu)M_k, \phi) \leq f_\Sigma((I, \nu)M'_1, \ldots, (I, \nu)M'_k, \phi)$

i.e. $(I, \nu)m \leq (I, \nu)m'$. 
Lemma. For $m, m' \in L(\Sigma, f)$: $(I, y) m \neq \omega \land (I, y) m' \neq \omega$ implies $(I, y) m = (I, y) m'$.

Proof. By Theorem (24.3) in $L(\Sigma, f)$ the join w.r.t. $L$ exists, i.e., for $m, m' \in L(\Sigma, f)$ there is an $m'' \in L(\Sigma, f)$ with $m \sqsubseteq m'', m' \sqsubseteq m''$.

Now, $\omega \neq (I, y) m \in (I, y) m''$ implies $(I, y) m = (I, y) m''$ and also $\omega \neq (I, y) m' \in (I, y) m''$ implies $(I, y) m' = (I, y) m''$.

Hence $(I, y) m = (I, y) m'$.

3.3. The Operational Semantics

(33.1) Preparation.
a computation $\delta_{\Sigma}$ under $I$ at point $y$ is a possibly infinite sequence $e_0, e_1, \ldots$ of elements from $M(Fu, \Sigma, D_\Sigma)$ such that

(i) $e_0 = \delta_{\Theta} (y, \ldots, \delta_{\Theta} (y))$
and by $\nu$ we denote $\nu (\nu)$

(ii) Either

\[ e_{n+1} \text{ is obtained from } e_n \text{ by a reduction step, that is to say: } \]

\[ e_{n+1} = \alpha \cdot \gamma \cdot \beta \text{ and } e_n = \alpha \cdot \delta (m_1, \ldots, m_{\rho (\gamma)}) \cdot \beta \text{ with } \]

\[ m \in M(Fu, \Sigma, D_\Sigma) \]

\[ \text{and for all } d_1, \ldots, d_{\rho (\beta)} \in D_\Sigma \text{ s.t. } d_i = m_i \text{ whenever } m_i \in D_\Sigma \]

\[ e_{n+1} (d_1, \ldots, d_{\rho (\beta)}) = \alpha \neq \omega \]

or

\[ e_{n+1} \text{ is obtained from } e_n \text{ by a rewriting step, that is to say: } \]

\[ e_{n+1} = \alpha \cdot \delta (m, \nu_1, \ldots, m_{\rho (\gamma)}) \cdot \beta \text{ and } e_n = \alpha \cdot \Theta (m, \ldots, m_{\rho (\gamma)}) \cdot \beta \]

(iii) If there is a last element in the sequence, then no rewriting or reduction steps are applicable and consequently the last element is in $D_\Sigma$.

And then we say the computation terminates.

Lemma. The results of two terminating computations of $\Sigma$ under $I$ at $y$ if both defined (i.e. $\neq \omega$), are the same.
proof: By the lemma in the next subsection, for each terminating computation \( \varphi(Y, \ldots, \psi(Y)) = e_0, e_1, e_2, \ldots, e_n \) there exists a strong derivation in \( \Xi \): \( \varphi(Y, \ldots, \psi(Y)) = e_0 \Rightarrow e_1 \Rightarrow \cdots \Rightarrow e_n \) with \((IY) W(e_n) = e_n \) (and of course \( e_n \in L(\Xi, \psi) \)).

and by Lemma 2 of (21.1), \((IY)\) does not vary on \( L(\Xi, \psi) \) whenever the value \( \neq \omega \).

The definition
The operational semantics \( \text{Comp}_\Xi \psi \) of a rec. program \( \langle \Xi, \psi \rangle \) is given by:

\[
\text{Comp}_\Xi \psi (Y) = \begin{cases} 
\text{the common value of all terminating comp's of } \Xi \text{ under } \text{Eval} & \text{if such exist} \\
\omega & \text{(otherwise)}
\end{cases}
\]

Theorem on equivalence with magma semantics.

Lemma: If \( e_0, e_1, \ldots, e_n \) is a terminating comp. seq. of \( \Xi \) under \( I \) at \( Y \) with exactly \( k \) rewriting steps, then there exists a strong derivation of length \( k \) in \( \Xi \): \( e_0 = \varphi(Y, \ldots, \psi(Y)) \) into some \( e_n \) s.t. \((IY) W(e_n) = e_n \).

Sketch of proof:
(Due to B. Rosen's paper "Subtree manipulations and Church-Rosser properties") (Quite similar to the proof of Thm (24.6) and lemma 6 in (24.5)).

Denote a comp. step by \( \Rightarrow \), a rewriting step by \( \Rightarrow \), a reduction step by \( \Rightarrow \), then by B. Rosen's results we can make the following diagram commutative:

\[
\begin{array}{c}
\varphi(Y, \ldots, \psi(Y)) = e_0 \Rightarrow \Rightarrow \cdots \Rightarrow e_k \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\psi(Y) \Rightarrow e_1 \Rightarrow \cdots \Rightarrow e_n \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\cdots \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
e_k \Rightarrow e_n \Rightarrow \cdots \Rightarrow e_n
\end{array}
\]

and corresponding to the top line we can make a strong derivation in \( \Xi \): \( \varphi(Y, \ldots, \psi(Y)) = e_0 \Rightarrow \Rightarrow \cdots \Rightarrow e_k \) with \( e_k \not\rightarrow \omega \in L(\Xi, \psi) \), i.e. \( W(e_k) = \varepsilon \) such that \((IY) \varepsilon = e_n \).

Theorem
The operational semantics and the magma semantics are equivalent.

proof:
We have to prove: for all \( \Sigma, \tau \) and for all \( \nu \) \( \text{Val}_\tau \Sigma (\nu) = \text{Comp}_\tau \Sigma (\nu) \).

By the lemma

\[
\text{Comp}_\tau \Sigma (\nu) = \begin{cases} (I, y) W(y, \epsilon) & \text{if some term comp. } e_0 \stackrel{\tau}{\Rightarrow} e_n \neq \omega \text{ exists and } e_0 \stackrel{\tau}{\Rightarrow} e_k \text{ is constructed according the lemma} \\ \omega & \text{otherwise} \end{cases}
\]

hence by lemma 2 of (32.1)

\[
\text{Comp}_\tau \Sigma (\nu) = \begin{cases} (I, y) (\epsilon) & \text{if some } \epsilon \in L(\Sigma, \chi) \text{ exists with } (I, y) \epsilon \neq \omega \\ \omega & \text{otherwise} \end{cases}
\]

so that we conclude on basis of the definition of \( \text{Val}_\tau \Sigma \)

\[
\text{Comp}_\tau \Sigma (\nu) = \text{Val}_\tau \Sigma (\nu).
\]

### 3.4 The fixed point semantics

#### 3.4.1 Preparation.

We have to associate with a rec. program \( \langle \Sigma, \tau \rangle \) a mapping \( \Sigma_\tau \), such that a fixed point of \( \Sigma_\tau \) can be considered as the meaning of the program. Hence the \( \Sigma_\tau \) must work on a space \( D_{\tau}^{(N)} \) of \((N\text{-tuples of}) \) functions and its arguments are to be considered as giving a value for the unknown function symbols in the \( \Sigma_\tau \).

Formally, we define the domain as follows:

\[
D_{\tau} = \bigcup_{n \geq 1} (D_{\tau}^n \rightarrow D_{\tau}),
\]

\( E \) on \( D_{\tau} \) a partial ordering induced by the \( E \supseteq \) \( D_{\tau} \):

\[
\forall \psi, \psi' \in D_{\tau}^n \iff \exists \psi, \psi' \in (D_{\tau}^n \rightarrow D_{\tau}) \text{ for some } n, \text{ and } \forall (d_1, \ldots, d_n) \in \psi' (d_1, \ldots, d_n) \text{ for all } (d_1, \ldots, d_n) \in D_{\tau}^n.
\]

Then, due to the discreteness of \( E \) on \( D_{\tau} \) we can define

\[
\bigwedge \psi^{(i)} \text{ for chains } \psi^{(i)} \in \psi^{(i)} \in \ldots \in \psi^{(i)} \in \ldots \text{ as } \bigwedge \psi^{(i)} (d_1, \ldots, d_n) = \begin{cases} \text{the common value of } \psi^{(i)} (d_1, \ldots, d_n) \text{ for all } k \text{ with } \psi^{(i)} (d_1, \ldots, d_n) \neq \omega \text{ (if such exist)} \\ \omega \text{ (otherwise)} \end{cases}
\]

\( D_{\tau}^N \) is just the set of \( N\text{-tuples of functions} \) and is a chain-closed set w.r.t. the components-wise ordering with \( E \) on \( D_{\tau} \).

Secondly we define a mapping "subscript \( \tau \)" \( : M(F, V)^n \rightarrow D_{\tau} \) by

\[
\text{for each } n \text{ (but we will omit the indication } n \text{ when no confusion results).}
\]
$$\begin{align*}
\Omega_\Sigma &= \Lambda d_1 \ldots d_n \cdot \omega \\
\Sigma_1 &= \Lambda d_1 \ldots d_n \cdot \delta_i \quad \text{where} \quad \nu = \nu_1 \in V = \{ \nu_1, \nu_2, \nu_3, \ldots \} \setminus \Delta \\
\phi^i(m_1, \ldots, m_{\phi(p)}) I &= \phi^i \left( m_1, \ldots, m_{\phi(p)} I \right) \quad \text{(function composition)}.
\end{align*}$$

Now, \((\Sigma_1) m = m_1 \cdot (\nu_1 \delta_i), \ldots, \nu_n \delta_i))\), where \(\{\nu_1, \ldots, \nu_n\}\) contains all variable symbols occurring in \(m\).

Now, we associate with a rec. program \(\langle \Sigma, I \rangle\) the mapping \(\Sigma_1^I: \bar{D}^I \rightarrow \bar{D}^I\) defined as

\[\Sigma_1^I(\nu, \eta) = \langle \zeta_1, \ldots, \zeta_{\phi(p)} \rangle,\]

where \(\zeta_i\) is obtained from \(\tau_i\) by replacing the unknown function symbols \(\psi_i\) by the functions \(\eta_i\) and considering the expression thus obtained as a face: \(\bar{D}_1^\phi(\psi_i^I) \rightarrow \bar{D}_1\), i.e.

\[\zeta_i = (\tau_i / \psi_i^1, \ldots, \psi_i^p) \eta_i\]

where the \(\psi_i^k\) are function symbols with \(\psi_i^1 = \psi_i^1\).

**Example**

Let \(G(x, y) = \begin{cases} 1 & \text{if } x < y \text{ then } x \text{ else } \eta_1(x, y), \\ 0 & \text{if } y = 0 \text{ then } x \text{ else } \eta_2(x, y). \end{cases}\)

Then

\[\Sigma_1^I(\eta_1, \eta_2) = \begin{cases} \eta_1(x, y) & \text{if } x = 0 \text{ then } y \text{ else } \eta_2(x, y), \\ \eta_2(x, y) & \text{if } x < y \text{ then } x \text{ else } \eta_2(x, y), \end{cases}\]

Now, we note that \(\Sigma_1^I\) is continuous, i.e., for each chain \(\psi_i^1 \sim \psi_i^2 \sim \cdots \)

\[\Sigma_1^I(\psi_i, \psi_i^1, \psi_i^2, \ldots, \psi_i^n) = \lim_{k \to \infty} \Sigma_1^I(\psi_i^1, \psi_i^2, \ldots, \psi_i^n),\]

Hence

\[\Sigma_1^I\] has a least fixed point \(\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in \bar{D}^I\), \(\forall \zeta_i\).

\[\langle \sigma_1, \ldots, \sigma_n \rangle = \lim_{k \to \infty} \Sigma_1^I(\omega_1, \ldots, \omega_n) \quad \text{where} \]

\(\omega_i\) is the "undefined function on \(\bar{D}_1^\phi(\psi_i^I)\)" i.e.

\(\omega_i\left(d_1, \ldots, d^\phi(\psi_i^I)\right) = \omega\quad \text{for all } d_1, \ldots, d^\phi(\psi_i^I) \in \bar{D}_1\).

(34.2) The Fixed Point Semantics \(\text{Fix}_I\Sigma\) of a rec. program \(\langle \Sigma, I \rangle\) is given by

\[\text{Fix}_I\Sigma = \text{the first component of the least fixed point of } \Sigma_1^I: \bar{D}^I \rightarrow \bar{D}^I.\]

(34.3) Theorem

The Fixed Point Semantics is included in the Standard Semantics.

Proof:

We have to prove \(\text{Fix}_I\Sigma \subseteq \text{Val}_I\Sigma\) in \(\bar{D}_1\) for all rec. programs \(\langle \Sigma, I \rangle\).


For $t \in \mathcal{T}$ (see 25.2), if $(I,y) t \neq (w)$ then $(I,y) \hat{\Xi}(t) = \overline{\Xi}_I(t) = \overline{\Xi}_I(t_I)$.

This should be clear from the definitions of $\hat{\Xi}$ and $\overline{\Xi}_I$, but it is rather complicated to write down in a precise formulation.

Hence, if a fixed point in $(\hat{\Xi}^\mathcal{T}, \leq)$ of $\hat{\Xi}$ is of the form $t_I$ for some $t \in \mathcal{T}$, then $t$ is also a fixed point of $\hat{\Xi}$ in $(\mathcal{T}, \leq)$, because $(I,y) t = (t_I) = \hat{\Xi}_I(t_I) = (I,y) \hat{\Xi}(t) \Rightarrow (I,y) t = (I,y) \hat{\Xi}(t)$.

And conversely, a fixed point $t \in \hat{\Xi}$ in $(\mathcal{T}, \leq)$ induces the fixed point $t_I \in \hat{\Xi}_I$ in $(\hat{\Xi}^\mathcal{T}, \leq)$.

Now,

$\text{Val}_I \hat{\Xi} = L_I(\hat{\Xi}, \Phi_I) = L_I[I_{\text{first comp}}]$, whereas $L_I \in \mathcal{T}$ is a fixed point $L_I$ of $\hat{\Xi}$ in $(\mathcal{T}, \leq)$, so that $L_I$ is a fixed point of $\hat{\Xi}_I$ in $(\hat{\Xi}^\mathcal{T}, \leq)$, and consequently includes the least one.

Hence $\text{Fix}_{I_{\text{first comp}}} = L_I[I_{\text{first comp}}] = \text{Val}_I \hat{\Xi}$.

Remark

We can strengthen the above result.

In spite of the fact that we did not prove $\text{Val}_I \hat{\Xi}$ to be the least fixed point of $\hat{\Xi}_I$, we can prove that it is the least one of $\hat{\Xi}_I$ with respect to all fixed points of the form $L_I[I_{\text{first comp}}]$, for $L_I \in \mathcal{T}$.

Let $L_I$ be a fixed point of $\hat{\Xi}_I$, then $L_I$ is a fixed point of $\hat{\Xi}$ and because $L_I$ is the least one (Theorem 25.2), $L_I \leq L_I$ (the ordering in $\mathcal{T}$) and because $L_I$ is ordering preserving with $\Xi$ and $\overline{\Xi}$ (clear from def's) $(I,y) L_I \leq (I,y) L_I$, i.e. $L_I \in L_I[I_{\text{first comp}}]$.

Hence

If we restrict $\hat{\Xi}_I$ to be the union of the magma-definable mappings $D_\Xi \rightarrow D_\Xi$, then $\text{Val}_I \hat{\Xi} = \text{Fix}_{I_{\text{first comp}}} \hat{\Xi}$. 


4 EQUIVALENCE OF RECURSIVE PROGRAM SCHEMES

B. Courtois and T. Vuillemin have proved the decidability of the equivalence of rec. pr. schemes with one variable only and an additional restriction on the rec. pr. schemes, viz. the property of being "acceptable". We will try to formulate their proof in our terminology. *)

4.1 Preparation

4.1.1 Definitions and Theorem.
\( Z \equiv Z' \), \( Z \) is equivalent with \( Z' \), iff for all \( I \) \( \text{Val}_I Z = \text{Val}_I Z' \).
We prefer to reason as syntactically as possible, therefore
\( L \sim L' \), \( L \) is equivalent with \( L' \), iff \( \text{both } L \leq L' \text{ and } L' \leq L \), where
\( L \leq L' \), \( L \) majorizes \( L' \), iff \( \forall m \in L \exists m' \in L': m \leq m' \).

Theorem
\( Z \equiv Z' \) if and only if
\[ L(\overline{Z}, q) \sim L(\overline{Z}', q) \]

Proof:
\[ (\leq) : \text{Immediately from def's } L \sim L', \text{ def Val}_I Z, \text{ lemma } 2 (32.1). \]
\[ (\geq) : \text{Take Herbrand interpretation } (I', \Gamma) \text{ given by } \\
\quad \quad \quad \quad D = M(F, V) \text{ and } \Gamma(v) = \{ v \} \text{ for all } v \in V, \quad \overline{f}_I = \{ f \} \text{ for all } f \in F. \]

4.1.2 The method of Hopcroft and Karpunjah.
We now sketch the way Hopcroft and Karpunjah prove the decidability of the equivalence of simple deterministic grammars (s.d.g.). Our attempt will be an imitation of this one.
Sketch
s. d. g. have production rules of the form \( \xi_i \rightarrow a \xi_i \ldots \xi_i \), where
\( a \in \text{ terminal alphabet and all } \xi \in \text{ nonterminal alphabet, and } \\
\text{for all } \xi_i, a \text{ there exist at most one prod rule of the form } \xi_i \rightarrow a \ldots. \)
Equivalence is defined as \( G \equiv G' \iff L(G, \xi_i) = L(G', \xi_i) \).
The deciding algorithm is sketched below.
First define for words \( w, w' \vdash w \equiv w' \iff L(G, w) = L(G', w') \). Then
\( G \equiv G' \) is equivalent with \( \xi_i \equiv \xi_i \). Secondly, bring \( G \) and \( G' \) into a normal form, viz. the reduced form defined by: \( G \) is reduced
iff \( L(G, \xi) \equiv \emptyset \) for all \( \xi \) in \( G \). Thirdly let \( d(\xi) = \min \{ 1/|W| : W \in L(G, \xi) \} \).

*) (added in May '74; see page 34 for further remarks)
Now, for deciding $\xi_1, \ldots, \xi_n \equiv \xi_{i_1}, \ldots, \xi_{i_m}$ proceed as follows:

while no negative tests are encountered and some equivalence relations still have to be tested

do

either for $\xi_{i_1}, \ldots, \xi_{i_K} \equiv \xi_{j_1}$ and $\xi_{i_{K+1}}, \ldots, \xi_{i_n} \equiv \xi_{j_1}, \ldots, \xi_{j_m}$
or for $\xi_{i_1} \equiv \xi_{j_1}, \ldots, \xi_{j_K}$ and $\xi_{i_{K+1}}, \ldots, \xi_{i_n} \equiv \xi_{j_{K+1}}, \ldots, \xi_{j_{m}}$

(where both the $k$ and the case are determined by means of $\delta$; and this is based on a crucial lemma)

test $a = a'$, where $\xi_{i_j} \rightarrow a_{i_j}, \ldots, \xi_{i_m}$ and $\delta \overleftarrow{\rightarrow} a', \xi_{i_1}, \ldots, \xi_{i_m}$

and

go on with the relations

either $\xi_{i_1}, \ldots, \xi_{i_m} \equiv \xi_{j_1}, \ldots, \xi_{j_m}$ and $\xi_{i_{K+1}}, \ldots, \xi_{i_n} \equiv \xi_{j_{K+1}}, \ldots, \xi_{j_{m}}$
or $\xi_{i_1}, \ldots, \xi_{i_m} \equiv \xi_{j_1}, \ldots, \xi_{j_m}, \xi_{j_{K+1}}, \ldots, \xi_{j_{m}}$

od

and this process terminates by virtue of the properties of $s$ and $\delta$'s.

4.2 Imitation of the method of Hopcroft and Karp Johan.

4.2.1 Acceptability and Standard Form.

Let $\delta : M(F, \Phi, V) \rightarrow \mathbb{N} \cup \{ \infty \}$ be defined w.r.t. $\Sigma$ by

$\delta(V) = 0$

$\delta(P(M_{m_{1}}, \ldots, M_{m_{p}}(g))) = 1 + \min \{ \delta(M_{m_{1}}, \ldots, \delta(M_{m_{p}}(g)) \}$

$\delta(g(M_{m_{1}}, \ldots, M_{m_{p}}(g))) = \delta(\delta(M_{m_{1}/V}, \ldots, M_{m_{p}}(g)) / \delta(g))$

Define

$\Sigma$ is acceptable iff $\delta(g) < \infty$ for $i = 1, \ldots, n$.

Intuitively, $\delta(M)$ is finite iff there exists a derivation $m \Rightarrow m'$ such that $m \in M$ there is an occurrence of a variable $y$ not lying in the scope of any $g \in \Phi$. This property is decidable.

$\Sigma$ is in Standard Form iff all $\xi_i$ in $\Sigma$ are of the form $f(M_{m_{1}}, \ldots, M_{m_{p}}(g))$ with $M_{m_{p}} \in M(F, \Phi, V)$ for $i = 1, \ldots, p(g)$.

For each acceptable rewrite system $\Sigma$, we effectively can construct another one which is in Standard Form, is acceptable and equivalent to the original one, by the following algorithm:

while there are some $\xi_i$ of the form $g(M_{m_{1}}, \ldots, M_{m_{p}}(g))$

do replace such a $\xi_i$ by $\xi_{j}(M_{m_{1}/V}, \ldots, M_{m_{p}}(g)) / \delta(g)$

od
and this process terminates by virtue of acceptability.

while there are proper subexpressions \( f(\ldots) \) of the \( \varphi \)'s

    do add a new \( \varphi \) to the current \( \Phi \);
        replace such an innermost \( f(\ldots) \) by \( \varphi(\varphi, \ldots \varphi(\varphi)) \);
        and add a new equation \( \varphi(\varphi, \ldots \varphi(\varphi)) = f(\ldots) \) to the
        current system

end

and now the resulting system is in standard form and is

equivalent to the original one.

(48.2)

The crucial lemma.

From here onwards we let \( \Sigma \) and \( \Sigma' \) be in standard form

and one variable systems, i.e. \( V = \{ x; \alpha \} \) and for

all \( \varphi \in \Phi \), \( \rho(\varphi) = 1 \) and for \( f \in F \) \( \rho(f) \geq 1 \).

example

\[
\begin{align*}
    \varphi_1(x) &= g_1(x, \varphi_1(\phi_1(x))) \\
    \varphi_2(x) &= g_2(\varphi_2(x), \varphi_2(\phi_2(x))) \\
    \varphi_3(x) &= g_3(\varphi_3(x), \phi_3(x))
\end{align*}
\]

Define for \( m = \Pi(F, \Phi, V) \) and \( m' = \Pi(F, \Phi', V) \)

\( m \equiv m' \), \( m \) is equivalent with \( m' \) with respect to \( \Sigma \) and \( \Sigma' \),

iff \( L(\Sigma, m) \sim L(\Sigma', m') \) (Note that we omit \( \Sigma \) and \( \Sigma' \) !).

\( \Sigma \equiv \Sigma' \) iff \( \varphi_i(-) \equiv \varphi_i'(-) \) is now immediately clear from def.'s.

lemma 1: \( f(m_1, \ldots, m_k, \varphi_1) \equiv f'(m_1', \ldots, m_k, \varphi_1) \) iff \( f = f' \) and \( m_i \equiv m_i' \) \( i = 1 \ldots k \).

proof: (\( \Leftarrow \)) clear and for (\( \Rightarrow \)) we reason as follows:

by def, for all \( m_i, \varphi_1 \) there are \( m_i^* \in L(\Sigma, m_i) \)

such that \( f(m_i^*, \ldots, m_k, \varphi) \equiv f'(m_i^*, \ldots, m_k, \varphi_1) \). Hence

by def, \( \equiv \) , \( f = f' \) and \( m_i^* \equiv m_i^* \).

But similarly for the reverse order \( \Rightarrow \).

Hence \( f = f' \) and \( m_i \equiv m_i' \) for \( i = 1 \ldots k \).

lemma 2: for all \( m, n \in \Pi(F, \Phi, V) \) and \( m', n' \in \Pi(F, \Phi', V) \)

\( a) \) when \( m = m' \) \( \Rightarrow \) \( m(n/x) \equiv m'(n/x) \) iff \( n \equiv n' \)

\( b) \) when \( n = n' \) \( \Rightarrow \) \( m(n/x) \equiv m'(n/x) \) iff \( m \equiv m' \)

proof: \( a) \Rightarrow \) by induction on the number \( ||m|| \) of symbol in \( m \):
\[ \|m\| = 0\] then \(x = m = m'\), so \(m' = x\) and \(n = m(n/x) \equiv m'(n'/x) = n'.\)

\[\|m\| > 0\]

if \(m = \phi_k(m^*)\) then \(m \equiv \tau_i(m^*/x) = \phi(m_i, ..., m_{\phi(k)})\), and otherwise \(m = \phi(m_i, ..., m_{\phi(k)})\) so we have

\[m \equiv \phi(m_i, ..., m_{\phi(k)})\].

Because \(m'\) cannot be \(x\), we similarly get \(m' \equiv \phi(m'_i, ..., m'_{\phi(k)})\).

By Lemma 1, from \(m \equiv m'\) we get \(f = f'\) and \(m_i \equiv m_i'\).

Now \(f(m_i(n/x), ..., m_{\phi(k)}(n/x)) = f(m_i, ..., m_{\phi(k)})(n/x) \equiv m'(n/x) \equiv m'_i(n'/x) \equiv f'(m'_i(n'/x), ..., m'_{\phi(k)}(n'/x)) \equiv m'(n'/x)\) by Lemma 1, \(f = f'\) (again) and \(m_i \equiv m'_i\).

Hence, by induction hypothesis \(n \equiv n'\).

(a) \(\Rightarrow\) Obvious, also provable by induction.

(a) \(\Rightarrow\) By induction on \(\|m\| + \|m'\|\).

\[\|m\| + \|m'\| = 0\] then \(m = x = m'\) hence they are equivalent.

\[\|m\| + \|m'\| > 0\] then as in case (a), \(m \equiv f(m_i, ..., m_{\phi(k)})\), \(m' \equiv f(m'_i, ..., m'_{\phi(k)})\)

Now as in case (a) from \(m_i \equiv m_i'\) we derive \(f = f'\) and \(m_i \equiv m_i'\) for \(i = 1, ..., \phi(k)\).

By induction hypothesis \(m_i \equiv m_i'\) hence by Lemma 1

\(m \equiv f(m_i, ..., m_{\phi(k)}) \equiv f(m'_i, ..., m'_{\phi(k)})) \equiv m'\).

(b) \(\Rightarrow\) Obvious.

**Lemma 3:** "**CRUCIAL LEMMA**"

"**THE LEMMA PRESENTED AT THE LECTURES WAS ILL-FORMULATED AND EVEN NOT TRUE IN ITS PRESUMABLY INTENDED MEANING. MOREOVER, DUE TO THIS FAILURE THE COROLLARY AND THE ALGORITHM WILL FAIL TOO."

**IT IS ONLY FOR CURiosity THAT I PRESENT THE COROLLARY AND THE ALGORITHM. PLEASE SEE THE NEXT SECTION FOR FURTHER REMARKS.**

**Corollary to Lemma 3:** (omitting parentheses when no confusion results)

\[\phi_1, \phi_2, ..., \phi_m x = \phi_{i_1} \phi_{i_2} \phi_{i_m} x\]

with \(\delta(\phi_1) \leq \delta'\phi_{i_1}\)

\[\phi_1 \phi_2 \phi_m x = \phi_{i_1} \phi_{i_2} \phi_{i_m} x\]

\[\phi_{i_1} \phi_{i_2} \phi_{i_m} x \equiv \phi'_{i_1} \phi'_{i_2} \phi'_{i_m} x\]

for \(i = 1, ..., \phi(k)\) \(\phi_1, ..., \phi_m x \equiv \phi_{i_1}, ..., \phi_{i_m} x \wedge f = f'\) where

\[\phi_1 x = f(\phi_1 x, ..., \phi_m x)\] in \(\Sigma\)

and \(\phi_{i_1} x = f'(\phi'_{i_1} x, ..., \phi'_{i_m} x)\) in \(\Sigma'\).
proof: by the lemma \( q_i \ldots q_m x = q_i' \ldots q_m' x \) if and only for some \( k \leq m \)
\( q_i \ldots q_k x = q_i' \) and \( q_{k+1} \ldots q_m x = q'_{k+1} \ldots q_m' x \) (provided \( \delta(q_i) \leq \delta(q_i') \))
The conclusion follows by lemma 1.

(4.2.3) The algorithm.

Now we present the algorithm for deciding the equivalence of acceptable rewriting systems \( \Sigma \) and \( \Sigma' \).
First bring \( \Sigma \) and \( \Sigma' \) into standard form, then perform

1. algorithm Decide:
2. begin
3. list \( \mathbf{EAT} \) \& expressions \# already tested \#;
4. proc TEST = ( expression \# to be tested \# \( q_i \ldots q_m x \equiv q_i' \ldots q_m' x \) );
5. if expression does not appear in list \( \mathbf{EAT} \)
6. then write down \( q_i \ldots q_m x \equiv q_i' \ldots q_m' x \) in list \( \mathbf{EAT} \);
7. if \( \delta(q_i,x) \leq \delta(q'_i,x) \)
8. then search for \( k \) such that \( \delta(q_i \ldots q_k x) \equiv \delta(q'_i x) \)
9. with escape ( print(“NO”) ) ; leave algorithm Decide;
10. let \( q_i x = f(q_i x, \ldots, q_k x) \) be \# \( \Sigma \),
11. \( q_i x = f(q_i x, \ldots, q_k x) \) be \# \( \Sigma' \);
12. if \( f = f' \)
13. with escape ( print(“NO”) ) ; leave algorithm Decide;
14. \( \text{(call TEST( } q_i \ldots q_k x \equiv q_i' \ldots q_k' x \text{ ) for } \ell \in \{1 \ldots p(f)\} \text{)}
15. \( \text{(call TEST( } q_i \ldots q_m x \equiv q_i' \ldots q_m' x \text{ ) )}
16. \text{else \{ } \delta(q_i x) \geq \delta(q'_i x) \text{ and }
17. \text{do something similar with } \mathbf{EAT} \rightarrow \mathbf{EAT} \text{ };
18. \text{fi}
19. \text{fi proc TEST;}
20. \text{EAT := empty ;}
21. \text{call TEST( } q_i x \equiv q_i' x \text{ ) ; print(“YES” ) ;
22. end algorithm Decide .

Clearly the corollary implies the partial correctness of the algorithm and moreover the termination follows from the following argument:
Let \( M = \max \{ ||\tilde{q}_\ell x|| : f(\tilde{q}_\ell x, \ldots, \tilde{q}_m x) \text{ is } \mathbf{RHS} \in \Sigma \lor \Sigma' \text{, } 1 \leq \ell \leq p(f) \} \)
Let \( \Delta = \max \{ \delta(q,x) : q \in \mathbf{EAT} \text{ or } q' \in \mathbf{EAT}' \} \leq \infty \text{ (acceptability !) } \)
Then initially the length of the expression to be tested is bound by $2M + \Delta$, and inductively a call $\delta$ TEST with an expression of length less than $2M + \Delta$ induces calls $\delta$ TEST with expressions of length less than $2M + \Delta$, because $\delta(q_i \ldots q_k x) = \delta(q_i x)$ implies $\|q_i \ldots q_k x\| \leq \Delta$.

(42.4) **Example.**

We now present an example of execution of the algorithm "decide" for the following acceptable rew systems in standard form.

$\Sigma_i = \left\{ \begin{array}{l} g_1 x = g_1(x, q, q, q, x) \\
g_2 x = g_2(q, x, q, q, x) \\
g_3 x = g_3(q, x, q, x) \end{array} \right\}$

$\Sigma'_i = \left\{ \begin{array}{l} g_1 x = g_1(x, q, x) \\
g_2 x = g_2(q, x, q, x) \\
g_3 x = g_3(q, x, q, x) \end{array} \right\}$

Hence

<table>
<thead>
<tr>
<th>$\delta_i$</th>
<th>$\delta_i'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>$2$</td>
<td>$2$</td>
</tr>
<tr>
<td>$2$</td>
<td>$2$</td>
</tr>
<tr>
<td>$5$</td>
<td>$5$</td>
</tr>
<tr>
<td>$4$</td>
<td>$4$</td>
</tr>
</tbody>
</table>

**Notation:** For each actual parameter, we denote beneath it the tests performed and the new actual parameters for the recursive calls $\delta$ TEST. For each depth in the tree, the tree expr. above it belong to the list EAT, [eq. expr.] denotes the generation of an expr. already belonging to EAT.

**Execution:**

$\Sigma \not= \Sigma'$ because: "NO" is printed, $k$ cannot be found.
4.3 Remarks on section 4.2

The crucial lemma and the algorithm fail

The ill-formulated lemma presented at the lectures was

\[ m(n/x) \equiv m'(n'/x) \iff \exists m'': n = (o \equiv ?) m''(n'/x) \]
\[ \delta(m) \leq \delta(m') \]

Clearly the omission of the respective \( \Sigma, \Sigma' \) in "\( \equiv \)" here has lead to a confusion: both the possibilities of \( \equiv \) and \( \delta \equiv \) (w.r.t. \( \Sigma, \Sigma' \)) and \( \delta \equiv \) (w.r.t. \( \Sigma, \Sigma' \)) in the line \( n = (o \equiv ?) m''(n'/x) \) lead to contradictions. Presumably the lemma should read

\[ m(n/x) \equiv m'(n'/x) \iff \exists m'', m''': n = m''(m'''/x) \]
\[ \delta(m) \leq \delta(m') \]

Because now the corollary follows by taking

\[ \begin{cases} 
  m'' = \phi_{i_1}, & m''' = \phi_{i_2}, \\
  m'' = \phi_{i_2}, & m''' = \phi_{i_3}, \\
  n = \phi_{i_2}, & n = \phi_{i_3}.
\end{cases} \]

The proof of the lemma is given such that the failure becomes as clear as possible:

"proof": by induction on \( \delta(m) \):

\( \delta(m) > 0 \): \( m = \varphi(m) \) and \( m' = \varphi(m') \). Let \( \varphi(x) = f(...), \varphi'(x) = f'(...), \varphi''(x) = \Xi, \Xi' \).

Then \( f(m, m_0(x)(m'/x), ..., m_0^k(x)(m'/x)) \equiv m(n/x) \equiv m'(n'/x) \equiv f(m', m_0(x)(m'/x), ..., m_0^k(x)(m'/x)) \).

Hence \( f = f' \) and \( m_0(m_0(x)(m'/x)) \equiv m_0'(m_0(x)(m'/x)) \) for all \( \xi \in \{ \phi, \phi' \} \).

Now, for \( o \in : \)

\[ \delta(m_0(m_0(x))) = \min \{ \delta(m_0(m_0(x))) : x = 1, ..., \xi \} \phi \]

\[ \delta(m_0(m_0(x))) = \min \{ \delta(m_0(m_0(x))) : x = 1, ..., \xi \} \phi' \]

\[ \leq \delta'(m') - 1 = \delta'(\phi''(m''')) = \min \{ \delta'(m''')(x) : x = 1, ..., \xi \} \phi'. \]

Hence we have

\[ m_0(m_0(x)(m'/x)) \equiv m_0'(m_0(x)(m'/x)) \]
\[ \delta(m_0(m_0(x))) \leq \delta'(m_0'(m_0'(x))) \]

and moreover...
\[ \delta(m_0(m_1/x)) \leq \delta(m) . \]

So by ind-hypo, there exist \( m'' \) and \( m''' \) such that
\[ n = m''(m''''/x) \text{ and } m''' = n' \]
(1)

In addition, by associativity of substitution we get
\[ m(m''/x)(m''''/x) = m(m''(m''''/x)/x) = m(n'/x) \]
So by lemma 2 and \( m''' \equiv n' \) we get \( m(m''/x) \equiv m' \) .... (2)

By (1) and (2) the induction step is proved.

end of proof.

Due to the case \( m = x \) and \( n \neq x \) the lemma fails, but we can try to overcome this by formulating the premise of the lemma as:
\[ m(n/x) \equiv m'(n'/x) \text{ and } \delta(m + n) \leq \delta'(m') \]
where \( m + n \equiv m \) whenever \( m \neq x \) or \( n = x \), and \( eq \) otherwise, where \( eq \) is the first symbol of \( n \).

But again we cannot prove it, essentially by the same reason. Other formulations or inductions do not hold either. Indeed:

Counter-example to the lemma, corollary, algorithm

Let \( \Sigma \) and \( \Sigma' \) as follows:
\[ \begin{align*}
\sigma & : \sigma_1 = f \sigma_2 \\
\psi & : q_1 \sigma_1 = \sigma_2 \\
\phi & : q_2 \sigma_1 = \sigma_2 x
\end{align*} \]

Then clearly
\( \Sigma \) and \( \Sigma' \) in standard form, and \( \phi \sigma_1 \sigma_2 \equiv \phi_3 \sigma_3 \) (because
\[ \{ \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5 \} \text{ and } \{ \phi_1, \phi_2, \phi_3, \phi_4, \phi_5 \} \]
and
\[ \delta'(\phi) \leq \delta'(\phi_1) = 2 , \]
but it is neither the case that \( \phi \sigma_1 \sigma_2 \equiv \phi_3 \sigma_3 x \), nor
the case that \( \phi_1 \sigma_1 \equiv \phi_2 x \land \sigma_2 \equiv \psi x \).

(43:2) Other attempts that failed.

Once we have demonstrated the incorrectness of the algorithm, we can look for an improved version. Obviously the line
(7): search for \( k \) such that \( \delta(q_1 \ldots q_{1k} x) = \delta'(q_i) \)
has to be changed, and accordingly lines (13), (14). We could try
(7): search for minimal \( (k, l) \) such that
\[ \delta(q_1 \ldots q_{1k} x) = \delta'(q_i \ldots q_{ij} x) \]
justified by the following lemma:
\[ \phi_1 \cdots \phi_m \in \phi'_1 \cdots \phi'_n \quad \text{for some } (k, l) \text{ with } \delta(\phi_1 \cdots \phi_m, x) = \delta(\phi'_1 \cdots \phi'_n, x) \]

which should be obviously true,
and may be there exists an argument (acceptability of \( \Sigma, \Sigma' \))
by virtue of which the revisited algorithm should terminate.

However, for the revisited algorithm (line 7), and
similar with line (3), we have the following
counterexample to the existence of such a bound:

\[
\begin{align*}
\Sigma' & \models \phi \chi = \phi(\phi_1 \phi_2, \chi) \\
\Sigma & \models \phi_1 \chi = \phi(\phi_1, \phi_2 \chi)
\end{align*}
\]

Denote the states during the execution by the list \( \text{EAT} \),
denoted by \( \Sigma \cdots \Sigma \), and the expressions to be tested (thus
omitting the successful tests \( \chi = \chi \) and \( \chi = \chi \)).

Then there are infinitely many states of the form

\[ s(n) : \left[ \delta(n) \phi \chi = \phi_1 \cdots \phi_n \chi \right] \]

where \( \delta(n) \) denotes \( \delta_0 \cdots \delta_n \),

so that there is no bound on the length of \( \phi \chi = \phi_1 \cdots \phi_n \chi \).

For

\[ s(3) \text{ is } \phi_1 \chi = \phi_1 \phi_2 \chi \text{, the initial state in testing } \Sigma = \Sigma', \]

and from a state \( s(n) \) we get \( s(n+1) \) in some ways:

\[ s(n) : \left[ \delta(n) \phi \chi = \phi_1 \cdots \phi_n \chi \right] \]

\[ \Rightarrow s(n+1) : \left[ \delta(n+1) \phi_1 \chi = \phi_1 \cdots \phi_n \chi, \delta(n+1) \phi_2 \chi = \phi_1 \cdots \phi_n \chi \right] \]

\[ \Rightarrow \cdots \Rightarrow \left[ \delta(n+k) \phi_k \chi = \phi_1 \cdots \phi_n \chi \right] \]

where \( \delta(n) = 1, \delta(n) = 2, \delta(n) = 2, \delta(n) = 1 \). The minimal \( k, l \)
are \( n+1, n+1 \), so that we get the resulting line according to

\[ \phi_1 \chi = \phi(\phi_1 \phi_2 \chi, \chi) \in \Sigma \text{ and } \phi_2 \chi = \phi(\phi_1 \phi_2 \chi, \phi \chi) \in \Sigma'. \]

\[ \Rightarrow \] due to the \( \delta \)-values, again the minimal \( k, l \) yield the
whole expressions. The generated expressions are already
in the list EAT, so that they need not proceed further.

Finally, it is even not true that equivalence of \( \Sigma \) and \( \Sigma' \) or more generally, equivalence of \( \phi_i \ldots \phi_n \) with \( \phi'_i \ldots \phi'_n \) will guarantee that minimal \((k, l)\) are \( \neq (m, n) \), so that with an additional test

\[(b) \quad \text{test for } (k, l) \neq (m, n) \text{ with escape (print('no'), leave algorithm)}\]

a termination argument could hold.

This is shown by the first counterexample, where for the valid equivalence \( \phi_i \phi_j \equiv \phi' \phi_j \) neither \( \phi_i \phi_j \equiv \phi' \phi_j \) nor \( \phi_i \phi_j \equiv \phi' \phi_j \) nor \( \phi_i \phi_j \equiv \phi' \phi_j \) hold true.

Conclusion:
In this way we are not able to prove the decidability of the equivalence of acceptable rewriting systems.

*) The systems \( \Sigma, \Sigma' \) of the latter counterexample are not equivalent as can be proved by use of the Kleene sequences.

For \( \Sigma \) the Kleene's sequence for \( \phi_i \phi_j \) is \( \langle f_{(n)} \rangle \) where

\[
\begin{align*}
&f_{(0)} = \phi_i \phi_j \equiv \phi_i \\
&f_{(1)} = \phi_i \\
&f_{(m+1)} = f \cdot f_{(m)}(\phi_j/\phi_i) \cdot \phi_j \\
&f_{(n)} = \phi_j 
\end{align*}
\]

and for \( \Sigma' \) we get similarly

\[
\begin{align*}
&f_{(0)} = \phi_i \phi_j \equiv \phi_i \\
&f_{(1)} = \phi_i \\
&f_{(m+1)} = f \cdot f_{(m)}(\phi_j/\phi_i) \cdot \phi_j \\
&f_{(n)} = \phi_j
\end{align*}
\]

For instance, the first terms of the seq for \( \Sigma \) are

\[
\begin{align*}
f_{(0)} &= \phi_i \\
f_{(1)} &= \phi_i \\
f_{(2)} &= \phi_i \phi_j \\
f_{(3)} &= \phi_i \phi_j \phi_i \phi_j \\
f_{(4)} &= \phi_i \phi_j \phi_i \phi_j \phi_i \phi_j
\end{align*}
\]
Remark to chapter 4., p. to 4.3.

In a discussion with Bruno Courcelle after finishing this manuscript (i.e. the first 33 pages) it appeared that the reason of failure was the point that the two systems $\Sigma$ and $\Sigma'$ had to be merged into one system $\Sigma''$ over $\overline{\Sigma}'' = \overline{\Sigma} + \overline{\Sigma}'$. When this is done, the formulation of the crucial lemma should be like the first formulation in (4.3.1). The equivalence relation $\equiv$ should take both expressions of one system $\Sigma''$ in stead of an expression of $\Sigma$ at the left-hand side and an expression of $\Sigma'$ at the right-hand side.

References:
"Semantics and axiomatics of a simple recursive language"
by B. Courcelle and J. Ullamein
SIGACT 1974

(lemma? gives the crucial lemma).