Proving WAM Compiler Correctness

P.H.Schmitt

January 9, 1995

Abstract

In this note we analyse the proof of compiler correctness of the WAM given in the paper [Börger and Rosenzweig 92] with regard to the question how it could be assisted by an automated theorem prover. We will give further details of the proof methodology and present the proof obligations in a form that is amenable to automated deduction systems.

1 Introduction

The investigations reported in this note were triggered by the discussions within the nationwide project *Deduktion* of the DFG on challenge problems that could be used to evaluate the various theorem provers, that have been used and developped within the project. One suggestion that received some attention was the formal verification of the correctness of a Prolog compiler described in [Börger and Rosenzweig 92]. In this paper the authors start from a formal specification of Prolog using the familiar computation tree model and arrive after successive refinement steps at a formal description of Prolog at the Warren abstract machine (WAM) level. All specifications are formalized using evolving algebras and proved correct with respect to the previous level.
As was to be expected additional effort had to be invested to transform the mathematical correctness proofs given in [Börger and Rosenzweig 92] into something that could be handled by an automated or semi-automated theorem prover. We only consider the first, and comparatively simple, step in the series of succesive refinements of the Prolog tree model hoping that the following steps will be easier once the pattern to follow is understood. We will give further details of the proof methodology, taking over where the discussion in section 5 of [Börger and Rosenzweig 92] ends, and present the proof obligations in a form that is amenable to automated deduction systems.

We assume familiarity with the method of evolving algebras, see e.g. [Gurevich 93] and with the paper [Börger and Rosenzweig 92], though we will repeat most of the relevant data. This paper may be retrieved via ftp from apollo.di.unipi.it:pub/Papers/boerger
where also other papers related to evolving algebras may be found.
In section 2 and 3 we review the evolving algebras for the top level specification and for the first refinement. In section 4 the goals to be proved and our way to atack this task are precisely stated and complete proofs for all proof obligations are given using usual mathematical reasoning, maybe we are a little more fine grained than usual. In section 5 we analyse the proofs of the preceding section and point out the potentials of automated or semi-automated reasoning and possible difficulties. In section 6 we sum up our assessment of the feasability of the challenge problem. The investigations of this note have been continued in the study project [Oel 94] where the theorem prover ${}_3T^AP$ has been used to establish most of the proofs done here with paper and pencil automatically. Work continues. The relationship between the notion of correctness used in [Börger and Rosenzweig 92] and the tradional concept of a correct compiler is delineated in [Beckert, Hähnle 94]. Efforts to use the interactive theorem prover KIV to find the above mentioned proofs are underway.

2 Evolving Algebra P_1

2.1 Signature

Principal Universes		
name	definition	meaning
$NODE$	basic	node of computation tree
LIT	basic	literals
$TERM$	basic	Prolog terms
$CLAUSE$	basic	Prolog clauses
$SUBST$	basic	substitutions
$CODE$	basic	program lines
$PROGRAM$	basic	program
$GOAL$	$TERM^*$	Prolog goals
$DECGOAL$	$GOAL \times NODE$	decorated goals, i.e. goal plus cutpoint information
$MODE$	$\{Call, Select\}$	modes

The only universe that gets updated during evaluation of the rules is $NODE$. We say that $NODE$ is a **dynamic** universe, while all others are called **static**.

Auxiliary Universes		
name	definition	meaning
$I\!N$	basic	natural numbers
$BOOL$	basic	Boolean thruth values
$SPECIAL$	basic	universe for various special constants
$STOPMODE$	$\{0, 1, -1\}$	stop modes

All auxiliary universes are static.

We distinguish between functions and constants that are essential for understanding the present rule system, these we call **principal**, and those that are only auxiliary. Among the first group we distinguish those that get updated during the evaluation of the rule system and those that don`t. The former we call **dynamic** the latter **static** for the rule system P_1.

Dynamic principal functions		
name	signatur	meaning
$currnode$	$NODE$	current node
$father$	$NODE \rightarrow NODE$	yields father of a node. Not defined on argument *root*
$decglseq$	$NODE \rightarrow DECGOAL^*$	associates with a node in the computation tree the list of decorated goals still to be solved
s	$NODE \rightarrow SUBST$	substitution accumulated upto a given node
$cands$	$NODE \rightarrow NODE^*$	list of sons of a node that still have to be considered
cll	$NODE \rightarrow CODE$	clause line,
$mode$	$MODE$	active mode of computation
$stop$	$STOPMODE$	$stop = 1$ signifies successful termination, $stop = -1$ signifies termination with failure, $stop = 0$ still working
vi	$I\!N$	renaming level

Static principal functions		
name	signatur	meaning
$root$	$NODE$	root of computation tree
$procdef$	$LIT \times PROGRAM \to CODE^*$	yields the procedure definition of a literal in a given program
$clause$	$CODE \to CLAUSE$	$clause(m)$ is the clause at program line m
$unify$	$TERM \times TERM \to SUBST \cup \{nil\}$	unifier of two terms
nil	$SPECIAL$	special constant, used to signify failure of unification
$subres$	$DECGOAL^* \times SUBST \to DECGOAL^*$	$subres(G, s)$ is obtained by applying substitution s to G
db	$PROGRAM$	database of the Prolog program

The meaning the following rule system does of course strongly depend on the properties of the uninterpreted function $procdef$; we refer to [Börger and Rosenzweig 92] for further details.

Auxiliary functions		
name	signatur	meaning
[]	X	empty list for any sort X
$rest$	$X^* \to X^*$	tail of a list for any sort X. Not defined on argument []
fst	$X^* \to X$	first element of a list. Not defined on argument []
$[\,\vert\,]$	$X \times X^* \to X^*$	$[a \mid L]$ is the list obtained from L by adding a as the new top element
$length$	$X^* \to I\!N$	length of a list for arbitrary sort X
$proj$	$X^* \times I\!N \to X$	$proj(L,i) = i$-th element of list L
fst	$X^2 \to X$	projection on the first element of a pair
snd	$X^2 \to X$	projection on the second element of a pair
$<,>$	$X \times X \to X^2$	forms a pair out of two elements
hd	$CLAUSE \to LIT$	head of a clause
bdy	$CLAUSE \to LIT^*$	body of a clause
$rename$	$CLAUSE \times I\!N \to CLAUSE$	renaming of a Prolog term at a given renaming level
$+$	$I\!N \times I\!N \to I\!N$	addition of natural numbers
$is\_user\_defined$	$TERM \to BOOL$	yields "true" for user defined predicate

2.2 Rules

In the formulation of the following rules we will use the following **abbreviations** (see figure 1):

$$
\begin{aligned}
father &\equiv father(currnode)\\
cands &\equiv cands(currnode)\\
s &\equiv s(currnode)\\
decglseq &\equiv decglseq(currnode)\\
goal &\equiv fst(fst(decglseq))\\
cutpt &\equiv snd(fst(decglseq))\\
act &\equiv fst(goal)\\
cont &\equiv [< rest(goal), cutpt > \mid rest(decglseq)]
\end{aligned}
$$

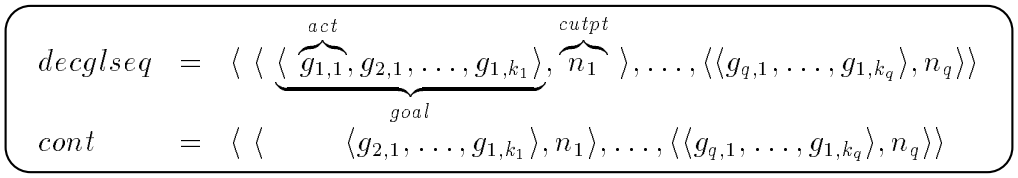

Figure 1: Visualizing the abbreviations

P_1-Rule 1 (final-success-rule)

IF $stop := 0$ *&* $decglseq(currnode) = []$
THEN $stop = 1$

P_1-Rule 2 (success-rule)

IF $stop = 0$ *&* $goal = []$
THEN $decglseq := rest(decglseq)$

P_1-Rule 3 (call-rule)

IF $stop = 0$ *&* $is\_user\_defined(act)$ *&* $mode = Call$
THEN
$LET\ n = length(procdef(act, db))$
$EXTEND\ NODE\ by\ temp_1, \ldots, temp_n$
WITH
$father(temp_i) := currnode$
$cll(temp_i) := proj(procdef(act, db), i)$
$cands := [temp_1, \ldots, temp_n]$
ENDEXTEND
$mode := Select$

P_1-Rule 4 (select-rule)

IF $stop = 0$ *&* $is\_user\_defined(act)$ *&* $mode = Select$
THENIF $cands = []$
THEN $backtrack$
ELSE $LET\ clause = rename(clause(cll(fst(cands))), vi)$
$LET\ unify = unify(act, hd(clause))$
IF $unify = nil$
THEN $cands := rest(cands)$
ELSE $currnode := fst(cands)$

$$
\begin{aligned}
&decglseq(fst(cands)) := \\
&subres([< bdy(clause), father >| cont], unify) \\
&s(fst(cands)) := s \circ unify \\
&cands := rest(cands) \\
&mode := Call \\
&vi := vi + 1
\end{aligned}
$$

where

$$
\begin{aligned}
backtrack \quad \equiv \quad & IF\ father = root \\
& THEN stop := -1 \\
& ELSE\ currnode := father \\
& \qquad mode := Select
\end{aligned}
$$

P_1-Rule 5 (cut-rule)

$$
\begin{aligned}
IF \quad & stop = 0\ \&\ act =! \\
THEN \quad & father := cutpt \\
& decglseq := cont
\end{aligned}
$$

2.3 Initial P_1-algebra

We start with a given prolog program *program* and a goal *goal*. We will denote the initial P_1-algebra by $\mathcal{A}_1^0$. It suffices to describe the dynamic universes and functions of $\mathcal{A}_1^0$.

The universe $NODE$ of $\mathcal{A}_1^0$ consists of two elements:

$$NODE = \{n_0, n_1\}$$

function	value
$root$	n_0
$currnode$	n_1
$father(n_1)$	n_0
$decglseq(n_1)$	$[< goal, n_0 >]$
$s(n_1)$	[]
vi	0
$stop$	0
$mode$	$Call$
db	$program$

For arguments not mentioned in this table functions are undefined.

3 Evolving Algebra P_2

The representation of the Prolog interpreter by the rule system P_2 deviates from P_1-System essentially in two points.
First the new system does no longer contain the constant *currnode*. The former functions values $f(currnode)$ are now stored so to speak in a separate register and named by the corresponding constants f.
More substantial are the changes inthe division of labour between rule 3 and rule 4 in both systems. A synopsis is depicted in figure 2.

3.1 Signature

Principal Universes		
name	definition	meaning
$STATE$	basic	node of computation tree
LIT	basic	literals
$TERM$	basic	Prolog terms
$CLAUSE$	basic	Prolog clauses
$SUBST$	basic	substitutions
$CODEAREA$	basic	program lines
$PROGRAM$	basic	program
$GOAL$	$TERM^*$	Prolog goals
$DECGOAL$	$GOAL \times STATE$	decorated goals, i.e. goal plus cutpoint information
$MODE$	$\{Call, Select\}$	modes
vi	$I\!N$	renaming level

$STATE$ is the only dynamic universe.

All auxiliary universes remain unchanged.

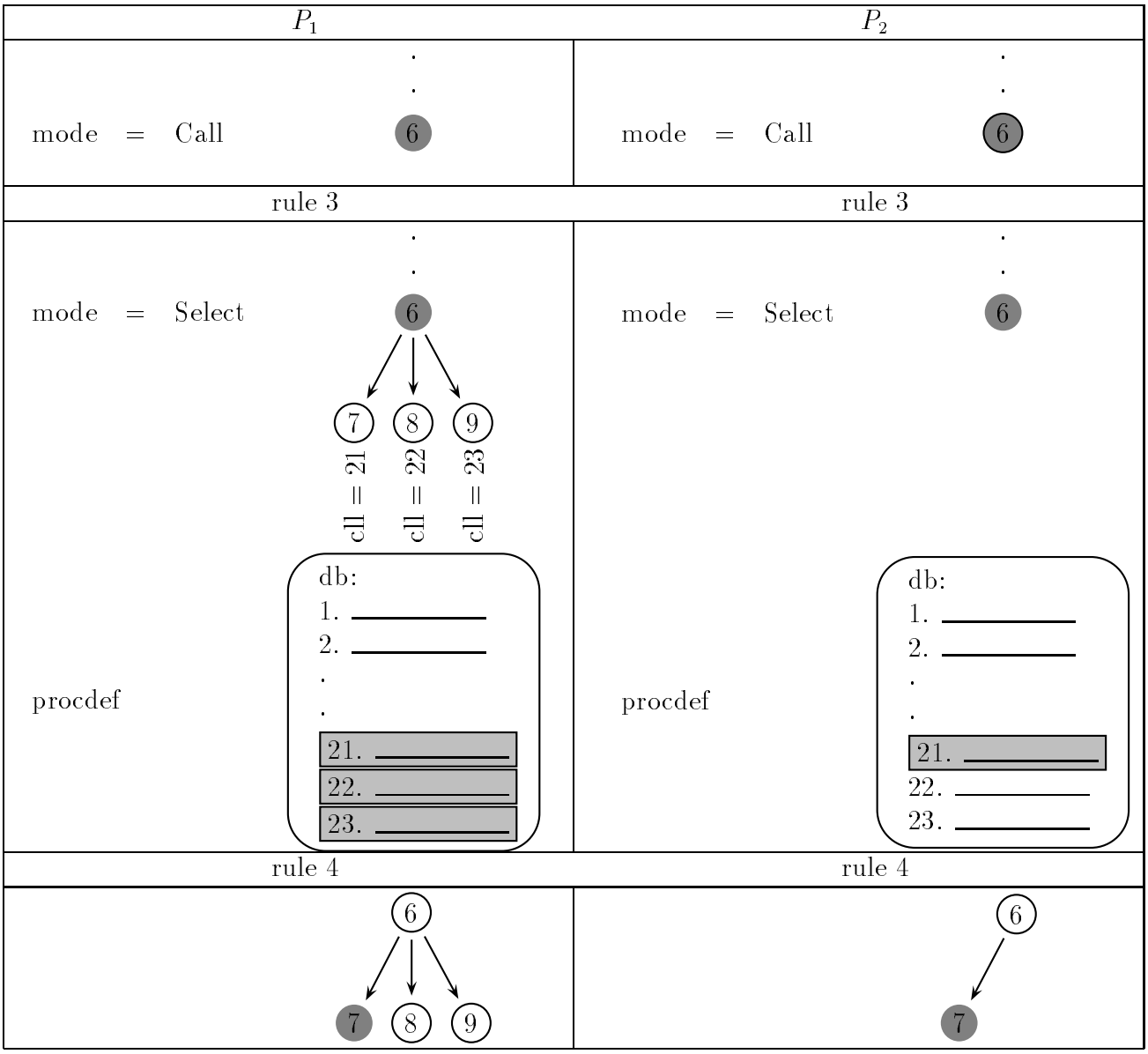

Figure 2: Comparing P_1 and P_2

Dynamic principal functions		
name	signatur	meaning
b	$STATE$	father of current state
b	$STATE \rightarrow STATE$	yields father of a state. Not defined on argument *bottom*
$decglseq$	$STATE \rightarrow DECGOAL^*$	associatesd with a state in the computation tree the list of decorated goals still to be solved
$decglseq$	$DECGOAL^*$	decorated goal of current state
s	$STATE \rightarrow SUBST$	substitution accumulated upto a given state
s	$SUBST$	substitution of current state
cll	$CODEAREA$	clause line of current state
cll	$STATE \rightarrow CODEAREA$	$cll(n)$ is clause line for node n
$mode$	$MODE$	active mode of computation
$stop$	$STOPMODE$	$stop = 1$ signifies successful termination, $stop = -1$ signifies termination with failure, $stop = 0$ still working
vi	$I\!N$	renaming level

Static principal functions		
name	signatur	meaning
$bottom$	$STATE$	bottom of computation tree
$procdef$	$LIT \times PROGRAM \rightarrow CODEAREA$	yields the procedure definition of a literal in a given program
$clause$	$CODEAREA \rightarrow CLAUSE \cup \{nil\}$	$clause(m)$ is the clause at program line m
$unify$	$TERM \times TERM \rightarrow SUBST \cup \{nil\}$	unifier of two terms
nil	$SPECIAL$	special constant, used to signify failure of unification or end of list
$subres$	$DECGOAL^* \times SUBST$ $\rightarrow DECGOAL^*$	$subres(G, s)$ is obtained by applying substitution s to G
$+$	$CODEAREA \rightarrow CODEAREA$	successor function on the universe $CODEAREA$
db	$PROGRAM$	database of the Prolog program

Though we did use the same name for it the meaning of the function $procdef$ in P_1-algebras is different from its meaning in P_2-algebras, as can already be seen from the different type declarations. Since $procdef$ is in both cases an uninterpreted function we have to explicitely state a connection between both meanings if we want to get any reasonable correspondance between P_1- and P_2-algebras at all. For the convenience of the reader we recall from [Börger and Rosenzweig 92]:

$$\begin{aligned} clls(Ptr) \quad = \quad & \text{IF } clause(Ptr) = nil \\ & \text{THEN } [] \\ & \text{ELSE } [Ptr \mid clls(Ptr+)] \end{aligned}$$

For any P_1-algebra $\mathcal{A}$ and any P_2-algebra $\mathcal{B}$ we stipulate

$$\mathcal{A}(procdef(L, db)) = \mathcal{B}(clls(procdef(L, db))$$

3.2 Rules

This time the following **abbreviations** will suffice:

$$
\begin{aligned}
goal &\equiv fst(fst(decglseq)) \\
cutpt &\equiv snd(fst(decglseq)) \\
act &\equiv fst(goal) \\
cont &\equiv [< rest(goal), cutpt >| rest(decglseq)]
\end{aligned}
$$

P_2-Rule 1 (final-success-rule)

$IF \quad stop = 0 \ \& \ decglseq = []$
$THEN \quad stop := 1$

P_2-Rule 2 (success-rule)

$IF \quad stop = 0 \ \& \ goal = []$
$THEN \quad decglseq := rest(decglseq)$

P_2-Rule 3 (call-rule)

$IF \quad stop = 0 \ \& \ is\_user\_defined(act) \ \& \ mode = Call$
$THEN \quad cll := procdef(act, db)$
$mode := Select$

P_2-Rule 4 (select-rule)

$IF \quad stop = 0 \ \& \ is\_user\_defined(act) \ \& \ mode = Select$
$THENIF \quad clause(cll) = nil$
$THEN \quad backtrack$
$ELSE \quad LET \ clause = rename(clause(cll), vi)$
$LET \ unify = unify(act, hd(clause))$
$IF \quad unify = nil$
$THEN cll := cll+$
$ELSE \ EXTEND \ STATE \ BY \ temp \ with$
$b := temp$
$b(temp) := b$
$decglseq(temp) := decglseq$
$s(temp) := s$
$cll(temp) := cll+$
$ENDEXTEND$
$decglseq := subres([< bdy(clause), b >| cont], unify)$
$s := s \circ unify$
$mode := Call$
$vi := vi + 1$

where

$backtrack \equiv$ $IF\ b = bottom$
$THEN stop := -1$
$ELSE\ decglseq := decglseq(b)$
$s := s(b)$
$b := b(b)$
$cll := cll(b)$
$mode := Select$

P_2-Rule 5 (cut-rule)
IF $stop = 0\ \&\ act =!$
THEN $b := cutpt$
$decglseq := cont$

3.3 Initial P_2-algebra

The initial P_2-algebra will be denoted by $\mathcal{A}_2^0$.

The universe $STATE$ of $\mathcal{A}_2^0$ consists of one element:

$$STATE = \{n_0\}$$

Furthermore $CODEAREA = CODE \cup \{l_e\}$ for a new line l_e with $clause(l_e) = nil$.

function	value
$bottom$	n_0
b	n_0
$decglseq$	$[< goal, n_0 >]$
s	[]
vi	0
$stop$	0
$mode$	$Call$
db	$program$

4 The Proof Task

4.1 General Set-up

We assume that parameters $program$ and $goal$ are given.
A sequence of P_1-algebras $\mathcal{A}_1^0, \mathcal{A}_1^1, \ldots, \mathcal{A}_1^n$ is called a P_1-sequence if for every

$0 < i \leq n$ there is a P_1-rule R, such that $\mathcal{A}_1^i$ results from $\mathcal{A}_1^{i-1}$ by application of rule R.

A P_1-algebra $\mathcal{A}$ is called **reachable** , or more precisely P_1-reachable, if there is a P_1-sequence $\mathcal{A}_1^0, \mathcal{A}_1^1, \ldots, \mathcal{A}_1^n$ with $\mathcal{A} = \mathcal{A}_1^n$.

Definition 1 (Correct refinement)
We call P_2 a **correct refinement** *of P_1 if*

1. *whenever there is a terminating P_1-sequence*
   $\mathcal{A}_1^0, \ldots, \mathcal{A}_1^n$
   there is also a terminating P_2-sequence
   $\mathcal{A}_2^0, \ldots, \mathcal{A}_2^m$,
   not necessarily of the same length, such that

   $$\begin{array}{lcl} \mathcal{A}_2^n(stop) & = & \mathcal{A}_1^m(stop) \\ \mathcal{A}_2^n(s) & = & \mathcal{A}_1^m(s(currstate)) \end{array}$$

2. *if there is no terminating P_1-sequence starting in $\mathcal{A}_1^0$, then there is also no terminating P_2-sequence starting in $\mathcal{A}_2^0$*

Since the evolving algebras considered here are deterministic this definition reduces to:

Definition 2 (Correct refinement, deterministic case)
We call P_2 a **correct refinement** *of P_1 if*

1. *whenever the P_1-computation*
   $\mathcal{A}_1^0, \ldots, \mathcal{A}_1^n$
   terminates, then also the P_2-computation
   $\mathcal{A}_2^0, \ldots, \mathcal{A}_2^m$,
   terminates, not necessarily after the same number of steps, such that

   $$\begin{array}{lcl} \mathcal{A}_2^n(stop) & = & \mathcal{A}_1^m(stop) \\ \mathcal{A}_2^n(s) & = & \mathcal{A}_1^m(s(currstate)) \end{array}$$

2. *if the P_1-computation starting in $\mathcal{A}_1^0$ does not terminate then also the P_2-computation starting in $\mathcal{A}_2^0$ does not terminate.*

The relation between two successive levels of refinement may be as complicated as possible. Here, passing from P_1 to P_2, we face a relatively simple case. Ler us give at this point a few hints on the general situation.
The signatures of the two levels to be compared need not, and will in general not be, the same. A constant that is named *nil* in the fist algebra might be named *bottom* in the second and only the person who devised the algebras

know this. As an example in our case the constant *stop* in the signature of P_2 plays the same role as the function value $stop(currnode)$ in P_1. There may also be function names that occur in both signatures but have different meaning, this is the case with the function $procdef$ in our example. A function f at the first level might correspond to a complicated algorithm using the functions of the second level. Evolving algebra specifications will usually also contain uninterpreted functions, like $procdef$ in our example. It may also happen that an uninterpreted function symbol at the first level is replaced by a combination of other uninterpreted functions on the next level. In general we need some way to associate with every epression of the first level a syntactic entity of the next refinement level. We do not make this correspondance explicit in the treatment of P_1 and P_2, since it is of only moderate complexity. It is nevertheless implicitely present when we speak e.g. of s in P_2-algebras and of $s(currnode)$ in P_1-algebras.
The correspondance between the start algebras of both levels is again very simple in our case, since both depend only on the parameters *db* and *goal* and both are not affected by the refinement step. This may be different in general. On the refined level there may occur start algebras, that cannot be obtained as refinements of a start algebra on the abstract level. This will then lead to a more careful definition of the concept of a correct refinement. The ideas that lead to a particular refinement will most precisely be decsribed by mapping $\mathcal{F}$ from algebras from the second level to algebras on the first level. In case P_2 is a correct refinement ofP_1 if and only if P_1 is a correct refinement of P_2. this is a consequence of the deterministic nature of the aevolving lgebras involved and the simple correspondance between start algebras. In such cases it would also be possible to consider a mapping $\mathcal{G}$ from P_1-algebras to P_2-algebras and it is a matter of taste which one to prefer.

4.2 Definition of $\mathcal{F}$

In the case at hand we define a mapping $\mathcal{F}$, that maps every reachable P_2-algebra into a P_1-algebra such that

1. $\mathcal{F}$ is defined on the initial algebra $\mathcal{A}_2^0$ and $\mathcal{F}(\mathcal{A}_2^0) = \mathcal{A}_1^0$
2. for every pair $\mathcal{A}$,$\mathcal{B}$ of reachable P_2-algebras, such that $\mathcal{B}$ results from $\mathcal{A}$ by an application of P_2-Rule k also $\mathcal{F}(\mathcal{B})$ results from $\mathcal{F}(\mathcal{A})$ by an application of P_1-Rule k. See figure 3.
3. If $\mathcal{F}$ is defined on the P_2-algebra $\mathcal{A}$ then

$$\mathcal{A}(stop) = (\mathcal{F}(\mathcal{A}))(stop)$$

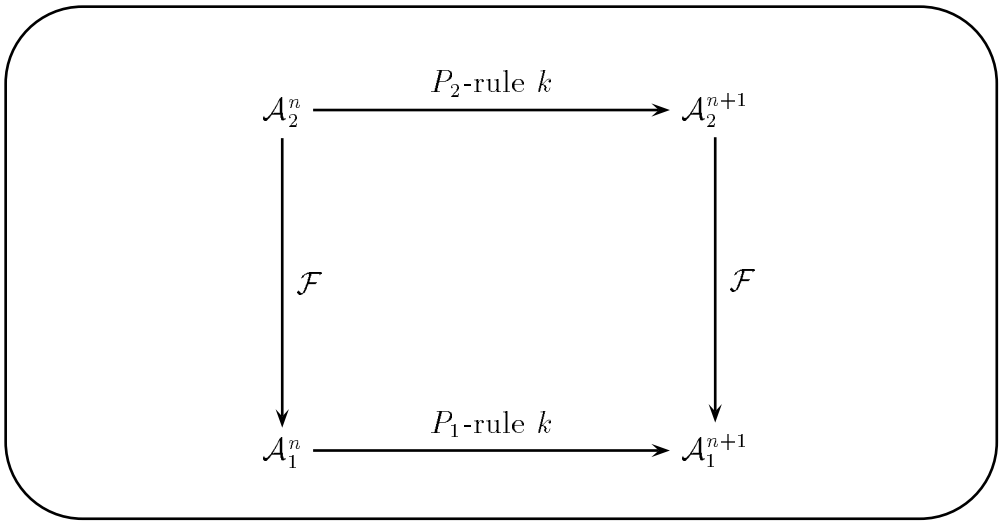

Figure 3: The mapping $\mathcal{F}$

and

$$\mathcal{A}(s) = (\mathcal{F}(\mathcal{A}))(s(currnode))$$

Sometimes there may be an easy definition for $\mathcal{F}$, but this time there is not. The reference [Börger and Rosenzweig 92] gives all the necessary hints but no explicit definition of $\mathcal{F}$. If one tries to do this one inevitably ends up with an inductive definition of $\mathcal{F}(\mathcal{A}$. We will define $\mathcal{F}$ by induction on the number of rule applications, i.e. part 2 of the above requirements is changed to 2^*:

for every pair $\mathcal{A}$,$\mathcal{B}$ of reachable P_2-algebras, such that $\mathcal{B}$ results from $\mathcal{A}$ by an application of P_2-Rule k and $\mathcal{F}(\mathcal{A})$ is already defined, there is also a P_1-algebra $\mathcal{F}(\mathcal{B})$ such that $\mathcal{F}(\mathcal{B})$ results from $\mathcal{F}(\mathcal{A})$ by an application of P_2-Rule k.

The static universes and functions will be identical with the exceptions already mentioned above

$$\mathcal{A}(CODEAREA) = (\mathcal{F}(\mathcal{A}))(CODE) \cup \{l_e\}$$

for one new line l_e with $clause(l_e) = nil$ and

$$(\mathcal{F}(\mathcal{A}))(procdef(L, db)) = \mathcal{A}(clls(procdef(L, db))$$

There will furthermore be an auxiliary function

$$F : STATE \rightarrow NODE$$

as part of the definition for $\mathcal{F}$.

We repeat for the convenience of the reader the definition of $clls$:

$$\begin{array}{lcl} clls(Ptr) & = & \text{IF } clause(Ptr) = nil \\ & & \text{THEN } [] \\ & & \text{ELSE } [Ptr \mid clls(Ptr+)] \end{array}$$

We furthermore need as an auxiliary function $mapcll$ informally definied by:

$$mapcll([n_1, \ldots, n_k]) = [cll(n_1), \ldots, cll(n_k)]$$

For the induction step to work we have to replace the requirement 3 by the following stronger version 3*:

1. $\mathcal{A}(stop) = (\mathcal{F}(\mathcal{A}))(stop)$
2. $\mathcal{A}(mode) = (\mathcal{F}(\mathcal{A}))(mode)$
3. $\mathcal{A}(s) = (\mathcal{F}(\mathcal{A}))(s(currnode))$
4. $F(\mathcal{A}(bottom)) = (\mathcal{F}(\mathcal{A}))(root)$
5. $F(\mathcal{A}(b)) = (\mathcal{F}(\mathcal{A}))(father(currnode))$
6. $F(\mathcal{A}(decglseq)) = (\mathcal{F}(\mathcal{A}))(decglseq(currnode))$
7. $\mathcal{A}(vi) = (\mathcal{F}(\mathcal{A}))(vi)$
8. $clls(\mathcal{A}(cll)) = mapcll((\mathcal{F}(\mathcal{A}))(cands(currnode)))$
9. $(\mathcal{A}(s))(n) = (\mathcal{F}(\mathcal{A})(s))(F(n))$
10. $F((\mathcal{A}(b))(n)) = (\mathcal{F}(\mathcal{A})(father))(F(n))$
11. $F((\mathcal{A}(decglseq))(n)) = (\mathcal{F}(\mathcal{A})(decglseq))(F(n))$
12. $clls((\mathcal{A}(cll))(n)) = mapcll((\mathcal{F}(\mathcal{A})(cands))(F(n)))$

The parameters n and m occuring in this list range over all elements in the universe $\mathcal{A}(STATE)$. Since decorated goal sequences contain references to nodes , respectively states, the function F must be applied to them. $F(G)$ for $G \in DECGOAL$ is defined as one would expect:

$$\begin{array}{lcl} F([]) & = & [] \\ F([< L, n > \mid T]) & = & [< L, F(n) > \mid F(T)] \end{array}$$

4.3 The Proofs

We start the inductive proof by observing that $\mathcal{F}(\mathcal{A}_2^0) = \mathcal{A}_1^0$ with $F(n_0) = n_0$ satisfies all requirements.

As the first induction step consider two P_2-algebras $\mathcal{A}$ and $\mathcal{B}$, such that $\mathcal{B}$ results from $\mathcal{A}$ by an application of P_2-rule 1. Thus

$$\mathcal{A}(decglseq) = []$$

Also assume that $\mathcal{F}(\mathcal{A})$ is already defined and satisfies all the above requirements, in particular we by $3^*(6)$

$$F(\mathcal{A}(decglseq)) = (\mathcal{F}(\mathcal{A}))(decglseq(currnode))$$

This shows that P_1-rule 1 is applicable to $\mathcal{F}(\mathcal{A})$. The resulting P_1-algebra will be used as $\mathcal{F}(\mathcal{B})$. It is almost obvious that $\mathcal{B}$ and $\mathcal{F}(\mathcal{B})$ satisfy all requirements.

The case that $\mathcal{B}$ results from $\mathcal{A}$ by P_2-rule 2 or 5 is treated in the same way. In the latter case we use the equality $(\mathcal{F}(\mathcal{A}))(father(currnode)) = \mathcal{A}(b)$.

As the next case assume that $\mathcal{B}$ results from $\mathcal{A}$ by P_2-rule 3. Since the static function $is\_user\_defined$ is interpreted in all P_1- and all P_2-algebras in the same way and since $\mathcal{A}(stop) = (\mathcal{F}(\mathcal{A}))(stop)$ and $\mathcal{A}(mode) = (\mathcal{F}(\mathcal{A}))(mode)$ we see that P_1-rule 3 is also applicable to $\mathcal{F}(\mathcal{A})$. As agreed above we take $\mathcal{F}(\mathcal{B})$ to be the algebra that arises from applying P_1-rule 3 to $\mathcal{F}(\mathcal{A})$. It remains to show that the pair $\mathcal{B}$ and $\mathcal{F}(\mathcal{B})$ satisfies all the requirements of 3^*. This is trivial for $3^*(2)$. Since $\mathcal{F}(\mathcal{B})(cands(currnode))$ has received a new value also $3^*(8)$ needs checking.

	$mapcll((\mathcal{F}(\mathcal{B}))(cands(currnode)))$	
$=$	$(\mathcal{F}(\mathcal{A})(procdef(act, db))$	P_1-rule 3
$=$	$clls(\mathcal{A}(procdef(act, db)))$	global
$=$	$clls(\mathcal{B}(cll))$	P_2-rule 3

All remaining parts of 3^* are satisfied since no other function has been changed.

As the next case assume that $\mathcal{B}$ results from $\mathcal{A}$ by P_2-rule 4. It is easy to see that P_1-rule 4 is also applicable to $\mathcal{F}(\mathcal{A})$. Let $\mathcal{F}(\mathcal{B})$ be the algebra that arises from $\mathcal{F}(\mathcal{A})$ by application of P_1-rule 4.

As **subcase 4.1** let us assume that $\mathcal{A}(cll) = nil$ and therefore $\mathcal{B}$ is obtained by performing the $P_2 - backtrack$ transition. Considering the equivalences

	$\mathcal{A}(cll) = nil$	
iff	$clls(\mathcal{A}(cll)) = []$	def. of $clls$
iff	$mapcll(\mathcal{F}(\mathcal{A}))(cands(currnode))) = []$	$3^*(8)$
iff	$(\mathcal{F}(\mathcal{A}))(cands(currnode)) = []$	def. of $mapcll$

we see that also $\mathcal{F}(\mathcal{B})$ arises from $\mathcal{F}(\mathcal{A})$ by performing the corresponding $P_1 - backtrack$ transition. Inspection of the rules shows that the following functions (may) have changed:

for P_2 $decglseq$,s,b,cll,$mode$,$stop$

for P_1 $stop$,$mode$,$currnode$

All conditions from 3^* that contain one of these functions have to be checked. These are 3^* (1), (2), (3), (5), (6).

Verification of $3^*(1)$, (2) is simple.

Verification of $3^*(3)$:

	$\mathcal{B}(s)$	
$=$	$\mathcal{A}(s(b))$	$P_2 - backtrack$
$=$	$(\mathcal{F}(\mathcal{A}))(s(F(b))$	$3^*(9)$
$=$	$(\mathcal{F}(\mathcal{A}))(s(father(currnode))$	$3^*(5)$
$=$	$(\mathcal{F}(\mathcal{B}))(s(currnode))$	$P_1 - backtrack$

Verification of $3^*(5)$:

	$F(\mathcal{B}(b))$	
$=$	$F(\mathcal{A}(b(b)))$	$P_2 - backtrack$
$=$	$(\mathcal{F}(\mathcal{A}))(father(F(b))$	$3^*(10)$
$=$	$(\mathcal{F}(\mathcal{A}))(father(father(currnode))$	$3^*(5)$
$=$	$(\mathcal{F}(\mathcal{B}))(father(currnode)$	$P_1 - backtrack$

Verification of $3^*(6)$:

	$F(\mathcal{B}(decglseq))$	
$=$	$F(\mathcal{A}(decglseq(b)))$	$P_2 - backtrack$
$=$	$(\mathcal{F}(\mathcal{A}))(decglseq(F(b)))$	$3^*(11)$
$=$	$(\mathcal{F}(\mathcal{A}))(decglseq(father(currnode)))$	$3^*(5)$
$=$	$(\mathcal{F}(\mathcal{B}))(decglseq(currnode))$	$P_1 - backtrack$

Now consider **subcase 4.2** in the application of the P_2-rule 4 to $\mathcal{A}$. Our first objective is to show that the local designator "clause" used in both the P_2- and the P_1-rule satisfies $\mathcal{A}(clause) = \mathcal{F}(\mathcal{A})(clause)$. For this it suffices to show

$$\mathcal{A}(cll) = \mathcal{F}(\mathcal{A})(cll(fst(cands))).$$

This is proved as follows:

	$\mathcal{F}(\mathcal{A})(cll(fst(cands(currnode))))$	
$=$	$fst(mapcll(\mathcal{F}(\mathcal{A})(cands(currnode))))$	fact
$=$	$fst(clls(\mathcal{A}(cll)))$	$3^*(8)$
$=$	$\mathcal{A}(cll)$	fact

For greater clarity we state the used facts once again seperately. They involve only static functions:

$$cll(fst(X)) = fst(mapcll(X))$$
$$fst(clls(P)) = P$$

Subcase 4.2 splits again into two subcases depending on wether $\mathcal{A}(unify) = nil$ or not. In the first case, **4.2.1**, the functions cll (in P_2) and $cands(currnode)$ (in P_2) may have changed. We need thus to check $3^*(8)$ for $\mathcal{F}(\mathcal{B})$ and $\mathcal{B}$.

	$clls(\mathcal{B}(cll))$	
$=$	$clls(\mathcal{A}(cll)+)$	P_2-rule 4
$=$	$rest(clls(\mathcal{A}(cll)))$	fact
$=$	$rest(mapcll(\mathcal{F}(\mathcal{A})(cands(currnode)))$	$3^*(8)$
$=$	$mapcll(\mathcal{F}(\mathcal{A})(rest(cands(currnode))))$	fact P_2-rule 3
$=$	$mapcll(\mathcal{F}(\mathcal{B})(cands(currnode)))$	P_1-rule 4

In the second subcase, **4.2.2**, the following functions may have changed:
for P_2 $decglseq$,s,b,$mode$,vi,$decglseq(temp)$,$s(temp)$,$b(temp)$, $cll(temp)$
for P_1 $mode$,$currnode$,$decglseq$,$cands$,vi
which requires verification of the following parts of 3^*:
(2),(3),(5),(6),(7),(9).

Since $\mathcal{B}(STATE) = \mathcal{A}(STATE) \cup \{temp\}$ we also have to extend the function F. This we do by

$$F(temp) = father(fst(\mathcal{F}(\mathcal{A})(cands(currnode))))$$

We skip the easy verification of $3^*(2)$
Verification of $3^*(3)$

	$\mathcal{B}(s)$	
=	$\mathcal{A}(s) \circ unify$	P_2-rule 4
=	$\mathcal{F}(\mathcal{A})(s) \circ unify$	$3^*(3)$
=	$(\mathcal{F}(\mathcal{B})(s))(\mathcal{F}(\mathcal{A})(fst(cands(currnode))))$	P_1-rule 4
=	$(\mathcal{F}(\mathcal{B})(s))(\mathcal{F}(\mathcal{B})(currnode))$	P_1-rule 4
=	$\mathcal{F}(\mathcal{B})(currnode)$	algebra

Verification of $3^*(5)$

	$F(\mathcal{B}(b))$	
=	$F(temp)$	P_2-rule 4
=	$\mathcal{F}(\mathcal{A})(father(fst(\mathcal{F}(\mathcal{A})(cands(currnode)))))$	def. of F
=	$\mathcal{F}(\mathcal{B})(father(currnode))$	P_1-rule 4

Verification of $3^*(6)$

	$F(\mathcal{B}(decglseq))$	
=	$F(subres([< bdy(clause), \mathcal{A}(b) >\mid \mathcal{A}(cont)], unify))$	P_2-rule 4
=	$subres([< bdy(clause), F(\mathcal{A}(b)) >\mid F(\mathcal{A}(cont))], unify)$	def. of F
=	$subres([< bdy(clause), \mathcal{F}(\mathcal{A})(father(currnode)) >\mid F(\mathcal{A}(cont))], unify)$	$3^*(5)$
=	$subres([< bdy(clause), \mathcal{F}(\mathcal{A})(father(currnode)) >\mid \mathcal{F}(\mathcal{A})(cont)], unify)$	auxiliary1
=	$(\mathcal{F}(\mathcal{B})(\mathcal{F}(decglseq))(\mathcal{F}(\mathcal{A})(fst(cands(currnode)))))$	P_1-rule 4
=	$(\mathcal{F}(\mathcal{B})(decglseq))(\mathcal{F}(\mathcal{B})(currnode))))$	P_1-rule 4
=	$\mathcal{F}(\mathcal{B})(decglseq(currnode))$	algebra

We did use as an auxiliary equation, auxiliary1:

$$F(\mathcal{A}(cont)) = \mathcal{F}(\mathcal{A})(cont)$$

Here is the verification for it

$$\begin{array}{lll}
 & F(\mathcal{A}(cont)) & \\
= & F(\mathcal{A}([< rest(goal), cutpt > | \; rest(decglseq(currnode))]) & \\
 & & \text{def. of } cont \\
= & F([< rest(\mathcal{A}(goal)), \mathcal{A}(cutpt) > | \; rest(decglseq(currnode))]) & \\
 & & \text{algebra} \\
= & [< rest(\mathcal{A}(goal)), F(\mathcal{A}(cutpt)) > | \; F(\mathcal{A}(rest(decglseq(currnode))))] & \\
 & & \text{def. of } F(list) \\
 & [< rest(\mathcal{A}(goal)), \mathcal{F}(\mathcal{A})(cutpt)) > | \; F(\mathcal{A}(rest(decglseq(currnode))))] & \\
 & & \text{auxiliary2} \\
 & [< rest(\mathcal{A}(goal)), \mathcal{F}(\mathcal{A})(cutpt)) > | \; \mathcal{F}(\mathcal{A})(rest(decglseq(currnode))))] & \\
 & & \text{local ind.hyp.} \\
= & \mathcal{F}(\mathcal{A})([< rest(\mathcal{A}(goal)), cutpt > | \; rest(decglseq(currnode))]) & \\
 & & \text{algebra} \\
= & \mathcal{F}(\mathcal{A})(cont) & \\
 & & \text{def. of } cont
\end{array}$$

We have used auxiliary2:

$$F(\mathcal{A}(cutpt)) = \mathcal{F}(\mathcal{A})(cutpt)$$

which may be verified as follows:

$$\begin{array}{lll}
 & F(\mathcal{A}(cutpt)) & \\
= & F(\mathcal{A}(snd(fst(decglseq)))) & \text{def. of } cutpt \\
= & snd(fst(F(\mathcal{A}(decglseq)))) & \text{def.of } F(list) \\
= & \mathcal{F}(\mathcal{A})(snd(fst(decglseq))) & 3^*(6) \\
= & \mathcal{F}(\mathcal{A})(cutpt) & \text{def. of } cutpt
\end{array}$$

We again skip the easy verification of $3^*(7)$.
Verification of $3^*(9)$

$$\begin{array}{lll}
 & (\mathcal{B}(s))(temp) & \\
= & \mathcal{A}(s) & P_2\text{-rule } 4
\end{array}$$

=	$\mathcal{F}(\mathcal{A})(s(currnode))$	$3^*(3)$
=	$\mathcal{F}(\mathcal{A})(s(father(fst(cands(currnode)))))$	fact
=	$(\mathcal{F}(\mathcal{B})(s))(\mathcal{F}(\mathcal{A})(father(fst(cands(currnode)))))$	P_1-rule 4
=	$(\mathcal{F}(\mathcal{B})(s))(F(temp)$	def. of $F(temp)$

Two remarks are necessary on this derivation.

(1) The last but one step, justified by an appeal to P_1-rule 4, is a type of frame axiom. Since we know that in all P_1-algebras for all nodes n $father(n) \neq n$, we get in particular in $\mathcal{F}(\mathcal{A})$ that $father(fst(cands(currnode))) \neq fst(cands(currnode))$. The value of $\mathcal{F}(\mathcal{B})(s)$ is only changed at the argument position $\mathcal{F}(\mathcal{A})(fst(cands(currnode)))$. Thus $(\mathcal{F}(\mathcal{B})(s))(\mathcal{F}(\mathcal{A})(fst(cands(currnode))))$ remains unaltered, i.e. is equal to $(\mathcal{F}(\mathcal{A})(s))(\mathcal{F}(\mathcal{A})(fst(cands(currnode))))$.

(2) The fact that is used in the above derivation states that for every reachable P_1-algebra $\mathcal{C}$ and all nodes n the following is true:

$$mode(n) = Select \rightarrow father(fst(cands(n))) = n$$

Looking at the rules it is not to difficult for the human mind to see that it is true, but it poses additional difficulties for mechanical verification:

- it is for the first time a conditional equation instead of simply an equation.
- it needs proof by induction on the number of transitions neccessary to reach $\mathcal{C}$ from the initial P_1-algebra.
- this induction will not be straightforward since in one-step transitions the premisse of the implication we want to prove, i.e. $mode(n) = Select$, is not true in the induction hypothesis.

Verification of $3^*(10)$

	$F(\mathcal{B}(b)(temp))$	
=	$F(\mathcal{A}(b))$	P_2-rule 4
=	$\mathcal{F}(\mathcal{A})(father(currnode))$	$3^*(5)$
=	$\mathcal{F}(\mathcal{A})(father(father(fst(cands(currnode)))))$	fact
=	$(\mathcal{F}(\mathcal{B})(father))(\mathcal{F}(\mathcal{A})(father(fst(cands(currnode)))))$	P_1-rule 4
=	$(\mathcal{F}(\mathcal{B}))(father(F(temp)))$	def. of $F(temp)$

The fact used is the same as in the previous derivation. The effect of P_1-rule 4 concerns again the fact that a function, $father$ in this case, remains unchanged. We have classified $father$ among the dynamic functions. This makes sense since $father(temp)$ has to updated for new elements $temp \in NODE$. One the other hand this is the only dynamic behaviour for $father$; a function value once assigned is never changed again. One could think of creating a new type of semi-dynamic or extensible functions.

Verification of $3^*(11)$

	$F(\mathcal{B}(decglseq(temp)))$	
$=$	$F(\mathcal{A}(decglseq))$	P_2-rule 4
$=$	$\mathcal{F}(\mathcal{A})(decglseq(currnode))$	$3^*(6)$
$=$	$(\mathcal{F}(\mathcal{A})(decglseq))(\mathcal{F}(\mathcal{A})(father(fst(cands(currnode)))))$	fact
$=$	$(\mathcal{F}(\mathcal{B})(decglseq))(\mathcal{F}(\mathcal{A})(father(fst(cands(currnode)))))$	P_1-rule 4
$=$	$(\mathcal{F}(\mathcal{B})(decglseq))(F(temp))$	def. of $F(temp)$

Again we did use the same fact as in the previous two derivations.

Verification of $3^*(12)$

	$clls((\mathcal{B}(cll))(temp))$	
$=$	$clls(\mathcal{A}(cll+))$	P_2-rule 4
$=$	$rest(clls(\mathcal{A}(cll)))$	fact
$=$	$rest(mapcll(\mathcal{F}(\mathcal{A})(cands(currnode))))$	$3^*(8)$
$=$	$mapcll(\mathcal{F}(\mathcal{A})(rest(cands(currnode))))$	fact
$=$	$mapcll((\mathcal{F}(\mathcal{B})(cands))(\mathcal{F}(\mathcal{A})(currnode)))$	P_1-rule 4
$=$	$mapcll((\mathcal{F}(\mathcal{B})(cands))(F(temp)))$	fact and def. of $F(temp)$

The two facts we used here, besides the one that we did already employ in the previous derivations, are:

$rest(clls(P)) = clls(P+)$
$rest(mapcll(L)) = mapcll(rest(L))$

5 Analysis

The way to proof completeness and correctness of P_2 for P_1 presented in the previous section is not the only one, but it seem to be a very plausible one.

We list the various tasks that have to be solved in a proof of the completeness of P_2 for P_1 together with explanations of their complexity and pecularities.

1. **Induction hypothesis:**
   In the previous section these were listed in 3*. This is hard to automize and the correct version may only be obtained after some experiments. In the previous section I had in the beginning additional requirements

   - $\mathcal{A}(cll) = (\mathcal{F}(\mathcal{A}))(cll(currnode))$
   - $(\mathcal{A}(cll))(n) = (\mathcal{F}(\mathcal{A})(cll))(F(n))$
   - $\mathcal{A}(b)(n) = m$ iff $(\mathcal{F}(\mathcal{A}))(father(F(n)) = F(m)$

   which turned out later to be superfluous. Also the given from of 3*(8) and (12) were only obtained at second thought. So one should leave the task to find the appropriate induction hypothesis, at least in the first attempt, totally to the user.

2. **Definition of F:**
   More precisely, the update of F for fresh elements added to the universe $STATE$ have to be given. This should be completely done by the user since it is incorporated in the whole set-up of the refinement from one evolving algebra system to another. There is furthermore no unique choice. In the above case instead of the chosen definition

   $$F(temp) = father(fst(\mathcal{F}(\mathcal{A})(cands(currnode))))$$

   also

   $$F(temp) = \mathcal{F}(\mathcal{A})(currnode)$$

   is possible, with no improvement on the ensuing computations.

3. **Control:**
   The system has to know which induction steps (= transition rules) have already been treated. Within a transition rule record has to be kept of various subcases. The task to show equivalence of the test conditions in a transitions rule (IN-conditions) has to be distinguished from the task of verifying correspondence of the resulting algebras,(OUT-conditions). Also a possible preprocessing step could be included, that

sorts out those induction hypothesis that remain true since they are not affected by a transition step. This task can be completely automated. It is not clear how much support existing theorem provers and verification systems provide in this respect. In the worst case it has to be imnplemented from scratch.

4. **Axioms and Rules:**
The premisses for logical deduction come from various sources, which we list below. We have deliberatly mentioned rules in the caption, since it remains to be decided if a certain piece of information should be presented as an axiom or as a proof rule. The typical proof situation comprises four algebras $\mathcal{A}$, $\mathcal{B}$, $\mathcal{F}(\mathcal{A})$ and $\mathcal{F}(\mathcal{B})$ and statements about the relationship between interpretations of terms in these algebras.

   (a) AXIOMS FROM THE ACTION PART OF TRANSITION RULES
   Typical examples are:

$$\mathcal{B}(mode) = Select$$
$$\mathcal{F}(\mathcal{B})(currnode) = \mathcal{F}(\mathcal{A})(fst(cands(currnode)))$$

   Note that a transition rule like

$$cands(currnode) := rest(cands(currnode))$$

   leads to an axioms of the form:

$$\mathcal{F}(\mathcal{B})(\mathcal{F}(\mathcal{A})(currnode)) = \mathcal{F}(\mathcal{A})(rest(cands(currnode)))$$

   i.e. argument terms on the left hand side of an assignment are evaluated in the old algebra. These axioms can in most cases be easily generated from the transition rules with the possible exceptions of universe extension rules like the one encountered in P_1-rule 3 above.
   A much severe problem concerns the function values that do *not* change. It is prohibitive to include them all as axioms. A possible solution could be to keep a list of function values, that have changed, and if a term, say $(\mathcal{F}(\mathcal{B})(s))(\mathcal{F}(\mathcal{A})(fst(cands(currnode))))$, is encountered with $s(fst(cands(currnode)))$ not in the list of updated function values in the transition from $\mathcal{F}(\mathcal{A})$ to $\mathcal{F}(\mathcal{B})$, then rewriting into $(\mathcal{F}(\mathcal{A})(s))(\mathcal{F}(\mathcal{A})(fst(cands(currnode))))$ could be

triggered. Notice, that this neccesitates reasoning about inequality.

(b) AXIOMS ON UNINTERPRETED FUNCTIONS
These obviously have to be supplied by the user, sicne they exist only in his brain. In the previous section the crucial axiom of this type was:

$$\mathcal{C}(procdef(L, db)) = \mathcal{A}(clls(procdef(L, db))$$

for any P_1-algebra $\mathcal{C}$ and any P_2-algebra $\mathcal{A}$.

(c) LEMMATA ON DATA STRUCTURES
Typical examples are the facts used above concerning static functions:

$$cll(fst(X)) = fst(mapcll(X))$$
$$fst(clls(P)) = P$$
$$rest(clls(P)) = clls(P+)$$
$$rest(mapcll(L)) = mapcll(rest(L))$$

The task is twofold: first to find out what lemmata would be useful and second to prove that a hypothetical lemma is in fact true. Typically these equations involve inductively defined functions. Resolution provers like Otter are not very suited for this type of reasoning, while LIPS- based provers like the Boyer-Moore system are well adapted.
The first subtask will probably be handled best by user interaction. The second subtask could be automated, though it remains to be seen if this really pays off given the small number of lemmatas needed. In a first version one could just through this lemmatas in as verified truth.

(d) CRUCIAL LEMMATA
P_1-algebra $\mathcal{C}$ and all nodes n the following is true:

$$\forall \mathcal{C} \forall n \in \mathcal{C}(STATE)(mode(n) = Select \rightarrow$$
$$father(fst(cands(n))) = n)$$

These pose a real challenge, they are hard to find and hard to prove. Others like

$$F(\mathcal{A}(cont)) = \mathcal{F}(\mathcal{A})(cont)$$
$$F(\mathcal{A}(cutpt)) = \mathcal{F}(\mathcal{A})(cutpt)$$

are easy to deal with.

(e) ALGEBRA
As an example of what we have in mind consider:

$$\mathcal{F}(\mathcal{A})(decglseq(currnode)) = (\mathcal{F}(\mathcal{A})(decglseq))(\mathcal{F}(\mathcal{A})(currnode))$$

Rewritings of this kind are frequently necessary for subsequent replacment of inner subterms.
Under the same heading come equations like

$$\mathcal{B}(unify(L_1, L_2)) = \mathcal{A}(unify(L_1, L_2))$$

for static functions, that cannot be altered, or semi-static functions like $father$ that can be extended, but altered.
In the derivations of the previous section we have used rewritings of this kind without mentioning. Since there is, in particular for larger terms, an abundance of rewritings of this kind possible, they should be severely controlled. It is not clear at the moment how difficult this will be.

(f) EXCEPTIONS
This item is included here to bring to mind the negligence committed in the previous section: We did not check for exceptional cases, like undefined functions or the occurence of constants nil or l_e. Maybe this is trival, maybe one has to invent something new here.

5. **Derivations:**
The bulk of reasoning required is reasoning about equalities and, to a lesser extend, inequalities. This suggest rewriting as a basic technique. The challenge lies in the control of the derivations. Exhaustive search seems out of the question. On the other hand the derivations given in the previous section show a great degree of regularity, the overall proof pattern is very similar for all steps. So there is hope for a highly specialized proof tactic.

6. **new phenomena:**
The analysis given so far is based on the proof step detailed in the

previous section. This is only concerned with the first and simplest refinement step, from the tree model to the stack model. In later refinements more difficult situations occur. One transition rule in the P_{k+1}-algebras may then correspond to a sequence of more than one transition rule in the P_k-algebras. This will require a type of reasoning completely different from what we have encountered so far. We may need information on the possible sequences of rule applications, like

> after transition rule P_1-rule 3 has been applied only application of P_1-rule 4 is possible
> after finitely many successive applications of transition rule P_1-rule 4 $mode = Call$ will happen

6 Conclusion

It is possible to built a theorem proving system that supports the proof of the WAM compiler correctness given in [Börger and Rosenzweig 92], but the following cautions have to be clearly stated:

- it will be a huge enterprise, not an application that one does on the side.
- the theorem proving capabilites required are strongly biased on equational reasoning and rewriting.
- the theorem proving program proper will be small compared with other parts of the system.

7 References

[Beckert, Hähnle 94] Bernhard Beckert, Reiner Hähnle *Proving Compiler Correctness with Evolving Algebra Specifications* informal note Institut für Logik, Komplexität und Deduktionssysteme, Fakulktät für Informatik, Univeristät Karlsruhe, Dezember 1994

[Börger and Rosenzweig 92] Egon Börger and Dean Rosenzweig. *The WAM - Definition and Compiler Correctness* TR-14/92, Dipartimento di Informatica, Università di Pisa, 1992.

[Börger and Rosenzweig 93] Egon Börger and Dean Rosenzweig. *A Mathematical Definition of Full Prolog* to appear in: Science of Computer Programming

[Gurevich 93] Yuri Gurevich. *Evolving Algebras: An Attempt to Discover Semantics* In: *Current Trends in Theoretical Computer Science* eds. G. Rozenberg and A. Salomaa, World Scientific, 1993, 266–292. A previous version appeared in the Bulletin of the European Association for Theoretical Computer Science, no. 43, Feb. 1991, 264–284.

[Oel 94] Peter Oel *Machbarkeitsstudie: Einsatz taktischer und automatischer Theorembeweiser zur Verifikation eines Prolog/WAM Compilers* Studienarbeit am Institut für Logik, Komplexität und Deduktionssysteme, Fakulktät für Informatik, Universität Karlsruhe, Dezember 1994