

A VM program starts with a byte indicating the number of predicates P in the program. Next, there are several components:

- An unsigned integer indicating the number N of nodes to instantiate, followed by $2N$ unsigned integers corresponding to one pair of unsigned integers one for node. The first value is the node ID to use during execution and the second one the ID given by the user.
- An unsigned integer indicating the number of arguments needed to run the program.
- An unsigned integer describing the number of rules R in the program. Followed by R byte regions. Each region contains an unsigned integer, N , indicating the size of the rule and then N bytes with the string for this rule.
- An unsigned integer indicating the number S of constant strings in the program followed by S pairs containing the length of the string and the string itself.
- A byte indicating the number of code constants C and then C bytes for the types of such constants. Finally, there's an unsigned integer describing the code size for computing the constants and the code itself.
- A set of P *predicate descriptors*, with 69 bytes each.
- A set of P byte-code instructions, one for each predicate.

A predicate descriptor consists of the following fields:

- A short integer indicating the size, in bytes, of the corresponding byte-code instructions.
- 1 byte describing the predicate's properties.
- 1 byte indicating the aggregate's type, if any. The high nibble if the aggregate type and the low nibble the aggregate field.
- A byte indicating the predicate's number of fields F .
- 32 bytes with information about the fields' types. Actually, only F bytes are used, and the remaining bytes are zeroes.
- 32 bytes containing the predicate's name representing as a string. As before, unused bytes are left as zeroes.

INSTRUCTION	BYTE FORMAT	ARGS																																																																
IF	<table><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr></table> <p>if $reg \neq 0$ then process until ELSE and then jump. if $reg = 0$ then jump to ELSE (note: IFs may be nested)</p>	0	1	1	0	0	0	0	0	0	0	0	r	r	r	r	r	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	$reg, jump_offset$																
0	1	1	0	0	0	0	0																																																											
0	0	0	r	r	r	r	r																																																											
j	j	j	j	j	j	j	j																																																											
j	j	j	j	j	j	j	j																																																											
j	j	j	j	j	j	j	j																																																											
j	j	j	j	j	j	j	j																																																											
ELSE	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table> <p>a marker for if blocks</p>	0	0	0	0	0	0	1	0	—																																																								
0	0	0	0	0	0	1	0																																																											
ITER	<table><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td></tr><tr><td>o</td><td>o</td><td>o</td><td>o</td><td>o</td><td>o</td><td>o</td><td>o</td></tr><tr><td>a</td><td>a</td><td>a</td><td>a</td><td>a</td><td>a</td><td>a</td><td>a</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr></table> <p>iterates over all the tuples of type id that match according to the following $matchlist$. after all matching facts have been processed, use $jump_offset$ to jump to the next instruction</p>	1	0	1	0	0	0	0	0	0	i	i	i	i	i	i	i	o	o	o	o	o	o	o	o	a	a	a	a	a	a	a	a	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	$id, options, options\ arg, jump_offset, matchlist$
1	0	1	0	0	0	0	0																																																											
0	i	i	i	i	i	i	i																																																											
o	o	o	o	o	o	o	o																																																											
a	a	a	a	a	a	a	a																																																											
j	j	j	j	j	j	j	j																																																											
j	j	j	j	j	j	j	j																																																											
j	j	j	j	j	j	j	j																																																											
j	j	j	j	j	j	j	j																																																											
NEXT	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <p>return to iter and process next matching fact</p>	0	0	0	0	0	0	0	1	—																																																								
0	0	0	0	0	0	0	1																																																											
SEND	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r_1</td><td>r_1</td><td>r_1</td><td>r_1</td><td>r_1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r_2</td><td>r_2</td><td>r_2</td><td>r_2</td><td>r_2</td></tr></table> <p>sends the tuple in reg_1 along the path in reg_2 if $reg_1 = reg_2$ then the tuple is stored locally</p>	0	0	0	0	1	0	0	0	0	0	0	r_1	r_1	r_1	r_1	r_1	0	0	0	r_2	r_2	r_2	r_2	r_2	reg_1, reg_2																																								
0	0	0	0	1	0	0	0																																																											
0	0	0	r_1	r_1	r_1	r_1	r_1																																																											
0	0	0	r_2	r_2	r_2	r_2	r_2																																																											
REMOVE	<table><tr><td>1</td><td>0</td><td>0</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table> <p>delete tuple stored in reg from database</p>	1	0	0	r	r	r	r	r	reg																																																								
1	0	0	r	r	r	r	r																																																											
OP	<table><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>v_1</td><td>v_1</td><td>v_1</td><td>v_1</td><td>v_1</td><td>v_1</td></tr><tr><td>0</td><td>0</td><td>v_2</td><td>v_2</td><td>v_2</td><td>v_2</td><td>v_2</td><td>v_2</td></tr><tr><td>0</td><td>0</td><td>v_3</td><td>v_3</td><td>v_3</td><td>v_3</td><td>v_3</td><td>v_3</td></tr><tr><td>0</td><td>0</td><td>0</td><td>o</td><td>o</td><td>o</td><td>o</td><td>o</td></tr></table> <p>sets $val_3 = val_1\ op\ val_2$</p>	1	1	0	0	0	0	0	0	0	0	v_1	v_1	v_1	v_1	v_1	v_1	0	0	v_2	v_2	v_2	v_2	v_2	v_2	0	0	v_3	v_3	v_3	v_3	v_3	v_3	0	0	0	o	o	o	o	o	val_1, val_2, val_3, op																								
1	1	0	0	0	0	0	0																																																											
0	0	v_1	v_1	v_1	v_1	v_1	v_1																																																											
0	0	v_2	v_2	v_2	v_2	v_2	v_2																																																											
0	0	v_3	v_3	v_3	v_3	v_3	v_3																																																											
0	0	0	o	o	o	o	o																																																											
NOT	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>v_1</td><td>v_1</td><td>v_1</td><td>v_1</td><td>v_1</td><td>v_1</td></tr><tr><td>0</td><td>0</td><td>v_2</td><td>v_2</td><td>v_2</td><td>v_2</td><td>v_2</td><td>v_2</td></tr></table> <p>sets $val_2 = not\ val_1$</p>	0	0	0	0	0	1	1	1	0	0	v_1	v_1	v_1	v_1	v_1	v_1	0	0	v_2	v_2	v_2	v_2	v_2	v_2	val_1, val_2																																								
0	0	0	0	0	1	1	1																																																											
0	0	v_1	v_1	v_1	v_1	v_1	v_1																																																											
0	0	v_2	v_2	v_2	v_2	v_2	v_2																																																											

INSTRUCTION	BYTE FORMAT	ARGS																																								
MOVE	<table><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr><tr><td>0</td><td>0</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td></tr></table> <p>copies <i>val</i>₁ to <i>val</i>₂</p>	0	0	1	1	0	0	0	0	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂	<i>val</i> ₁ , <i>val</i> ₂																
0	0	1	1	0	0	0	0																																			
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																																			
0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂																																			
MOVE-NIL	<table><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>v</td><td>v</td><td>v</td><td>v</td><td>v</td><td>v</td></tr></table> <p>sets <i>val</i> to the nil list</p>	0	1	1	1	0	0	0	0	0	0	v	v	v	v	v	v	<i>val</i>																								
0	1	1	1	0	0	0	0																																			
0	0	v	v	v	v	v	v																																			
TEST-NIL	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr><tr><td>0</td><td>0</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td></tr></table> <p>v₂ = 1 if v₁ is nil. v₂ = 0 if v₁ is not nil.</p>	0	0	0	0	0	0	1	1	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂	<i>val</i> ₁ , <i>val</i> ₂																
0	0	0	0	0	0	1	1																																			
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																																			
0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂																																			
ALLOC	<table><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td></tr><tr><td>0</td><td>0</td><td>v</td><td>v</td><td>v</td><td>v</td><td>v</td><td>v</td></tr></table> <p>allocates a tuple of type <i>id</i> and stores it in <i>val</i></p>	0	1	0	0	0	0	0	0	0	i	i	i	i	i	i	i	0	0	v	v	v	v	v	v	<i>id</i> , <i>val</i>																
0	1	0	0	0	0	0	0																																			
0	i	i	i	i	i	i	i																																			
0	0	v	v	v	v	v	v																																			
RETURN	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>finished processing the tuple - return</p>	0	0	0	0	0	0	0	0	—																																
0	0	0	0	0	0	0	0																																			
CALL	<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table> <p>call external function number <i>id</i> with <i>args</i> and store the result in <i>reg</i></p>	0	0	1	0	0	0	0	0	0	i	i	i	i	i	i	i	0	0	0	r	r	r	r	r	<i>id</i> , <i>reg</i> , <i>args</i>																
0	0	1	0	0	0	0	0																																			
0	i	i	i	i	i	i	i																																			
0	0	0	r	r	r	r	r																																			
CONS	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td><i>t</i></td><td><i>t</i></td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr><tr><td>0</td><td>0</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td></tr><tr><td>0</td><td>0</td><td>v₃</td><td>v₃</td><td>v₃</td><td>v₃</td><td>v₃</td><td>v₃</td></tr></table> <p>sets <i>val</i>₃ = <i>val</i>₁ :: <i>val</i>₂ <i>t</i> is the list type (00 = int, 01 = float, 02 = addr)</p>	0	0	0	0	0	1	0	0	0	0	0	0	0	0	<i>t</i>	<i>t</i>	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂	0	0	v ₃	v ₃	v ₃	v ₃	v ₃	v ₃	<i>val</i> ₁ , <i>val</i> ₂ , <i>val</i> ₃
0	0	0	0	0	1	0	0																																			
0	0	0	0	0	0	<i>t</i>	<i>t</i>																																			
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																																			
0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂																																			
0	0	v ₃	v ₃	v ₃	v ₃	v ₃	v ₃																																			
HEAD	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td><i>t</i></td><td><i>t</i></td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr><tr><td>0</td><td>0</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td></tr></table> <p>sets <i>val</i>₂ = <i>head val</i>₁ <i>t</i> is the list type (00 = int, 01 = float, 02 = addr)</p>	0	0	0	0	0	1	0	1	0	0	0	0	0	0	<i>t</i>	<i>t</i>	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂	<i>val</i> ₁ , <i>val</i> ₂								
0	0	0	0	0	1	0	1																																			
0	0	0	0	0	0	<i>t</i>	<i>t</i>																																			
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																																			
0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂																																			
TAIL	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td><i>t</i></td><td><i>t</i></td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr><tr><td>0</td><td>0</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td></tr></table> <p>sets <i>val</i>₂ = <i>tail val</i>₁ <i>t</i> is the list type (00 = int, 01 = float, 02 = addr)</p>	0	0	0	0	0	1	1	0	0	0	0	0	0	0	<i>t</i>	<i>t</i>	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂	<i>val</i> ₁ , <i>val</i> ₂								
0	0	0	0	0	1	1	0																																			
0	0	0	0	0	0	<i>t</i>	<i>t</i>																																			
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																																			
0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂																																			

INSTRUCTION	BYTE FORMAT	ARGS																																
SELECT	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table> <p>this is a big instruction used to select a specific code block for a node. it is followed by a 4-byte integer indicating the <i>size</i> of the whole instruction, then a 4-byte integer indicating the size <i>N</i> of a simplified hash-table. <i>N</i> represents the number of nodes in the system for efficiency reasons. Next, there is <i>N</i>*4-byte integers, where each integer is the offset to a code block of the corresponding node. The offsets start after the end of the hash table. If the offset is 0, this node has no associated code block, so it should use <i>size</i> to jump to the next instruction. If the offset is positive, you should subtract one byte from it and then jump to the code block. At the end of each code block, there is a RETURN-SELECT.</p>	0	0	0	0	1	0	1	0	<i>size, hsize, htable, codeblocks</i>																								
0	0	0	0	1	0	1	0																											
RETURN-SELECT	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table> <p>This instruction is followed by a 4-byte integer with a jump offset to the next instruction.</p>	0	0	0	0	1	0	1	1	<i>jump</i>																								
0	0	0	0	1	0	1	1																											
COLOCATED	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr><tr><td>0</td><td>0</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table> <p>sets dest = true if nodes n1 and n2 are on the same machine sets dest = false otherwise</p>	0	0	0	0	1	1	0	0	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂	0	0	0	r	r	r	r	r	<i>n1, n2, dest</i>
0	0	0	0	1	1	0	0																											
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																											
0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂																											
0	0	0	r	r	r	r	r																											
DELETE	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr></table> <p>deletes the tuples of type <i>i</i> with the first argument as value <i>v</i>₁</p>	0	0	0	0	1	1	0	1	0	i	i	i	i	i	i	i	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	<i>i, v1</i>								
0	0	0	0	1	1	0	1																											
0	i	i	i	i	i	i	i																											
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																											
REMOVE	<table><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table> <p>deletes tuple reg from the database</p>	1	0	0	0	0	0	0	0	0	0	0	r	r	r	r	r	<i>reg</i>																
1	0	0	0	0	0	0	0																											
0	0	0	r	r	r	r	r																											
RETURN-LINEAR	<table><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>linear fact was used, execution must terminate</p>	1	1	0	1	0	0	0	0																									
1	1	0	1	0	0	0	0																											
RETURN-DERIVED	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>head of rule was derived, return if some linear fact was used</p>	1	1	1	1	0	0	0	0																									
1	1	1	1	0	0	0	0																											

INSTRUCTION	BYTE FORMAT	ARGS																																																								
FLOAT	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td><td>v₁</td></tr><tr><td>0</td><td>0</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td><td>v₂</td></tr></table> <p>sets $val_2 = (float)val_1$</p>	0	0	0	0	1	0	0	1	0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁	0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂	val_1, val_2																																
0	0	0	0	1	0	0	1																																																			
0	0	v ₁	v ₁	v ₁	v ₁	v ₁	v ₁																																																			
0	0	v ₂	v ₂	v ₂	v ₂	v ₂	v ₂																																																			
RULE	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>rule id is gonna be executed</p>	0	0	0	1	0	0	0	0	id																																																
0	0	0	1	0	0	0	0																																																			
RULE DONE	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <p>rule id has been matched</p>	0	0	0	1	0	0	0	0																																																	
0	0	0	1	0	0	0	0																																																			
NEW AXIOMS	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr><tr><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td><td>j</td></tr></table> <p>after the instruction and jump address, there is an arbitrary number of tuples to be instantiated. the first byte of each tuple is the predicate code the tuple arguments follow next. arguments may be integers, floats, node addresses or serialized lists if the list is nil, there is a 0x0 byte, if not, there is 0x1 byte followed by the element of the list and the rest of the list. the jump argument indicates the next instruction and where the tuples end</p>	0	0	0	1	1	1	1	0	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	j	$jump$																
0	0	0	1	1	1	1	0																																																			
j	j	j	j	j	j	j	j																																																			
j	j	j	j	j	j	j	j																																																			
j	j	j	j	j	j	j	j																																																			
j	j	j	j	j	j	j	j																																																			
SEND DELAY	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r₁</td><td>r₁</td><td>r₁</td><td>r₁</td><td>r₁</td></tr><tr><td>0</td><td>0</td><td>0</td><td>r₂</td><td>r₂</td><td>r₂</td><td>r₂</td><td>r₂</td></tr><tr><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td></tr><tr><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td></tr><tr><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td></tr><tr><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td><td>t</td></tr></table> <p>sends the tuple in reg_1 along the path in reg_2 if $reg_1 = reg_2$ then the tuple is stored locally time indicates the delay (in milliseconds) of the derivation</p>	0	0	0	1	0	1	0	1	0	0	0	r ₁	r ₁	r ₁	r ₁	r ₁	0	0	0	r ₂	r ₂	r ₂	r ₂	r ₂	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	$reg_1, reg_2, time$
0	0	0	1	0	1	0	1																																																			
0	0	0	r ₁	r ₁	r ₁	r ₁	r ₁																																																			
0	0	0	r ₂	r ₂	r ₂	r ₂	r ₂																																																			
t	t	t	t	t	t	t	t																																																			
t	t	t	t	t	t	t	t																																																			
t	t	t	t	t	t	t	t																																																			
t	t	t	t	t	t	t	t																																																			

INSTRUCTION	BYTE FORMAT	ARGS																
PUSH	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	0	0	0	1	0	1	1	0									
0	0	0	1	0	1	1	0											
	pushes an uninitialized value into the stack																	
POP	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	0	0	0	1	0	1	1	1									
0	0	0	1	0	1	1	1											
	pops a value from the stack																	
PUSH REGS	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	1	1	0	0	0									
0	0	0	1	1	0	0	0											
	pushes all registers into the stack																	
POP REGS	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	1	1	0	0	1									
0	0	0	1	1	0	0	1											
	pops all registers from the stack and restores their values																	
CALLF	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td></tr></table>	0	0	0	1	1	0	1	0	i	i	i	i	i	i	i	i	<i>id</i>
0	0	0	1	1	0	1	0											
i	i	i	i	i	i	i	i											
	calls a Meld function																	

OP	BYTE FORMAT
----	-------------

<i>float</i> \neq	0	0	0	0	0
<i>int</i> \neq	0	0	0	0	1
<i>float</i> =	0	0	0	1	0
<i>int</i> =	0	0	0	1	1
<i>float</i> <	0	0	1	0	0
<i>int</i> <	0	0	1	0	1
<i>float</i> \leq	0	0	1	1	0
<i>int</i> \leq	0	0	1	1	1
<i>float</i> >	0	1	0	0	0
<i>int</i> >	0	1	0	0	1
<i>float</i> \geq	0	1	0	1	0
<i>int</i> \geq	0	1	0	1	1
<i>float</i> %	0	1	1	0	0
<i>int</i> %	0	1	1	0	1
<i>float</i> +	0	1	1	1	0
<i>int</i> +	0	1	1	1	1
<i>float</i> −	1	0	0	0	0
<i>int</i> −	1	0	0	0	1
<i>float</i> *	1	0	0	1	0
<i>int</i> *	1	0	0	1	1
<i>float</i> ÷	1	0	1	0	0
<i>int</i> ÷	1	0	1	0	1
<i>addr</i> \neq	1	0	1	1	0
<i>addr</i> =	1	0	1	1	1
<i>addr</i> >	1	1	0	0	0
<i>bool</i> or	1	1	0	0	1

VALUE	BYTE FORMAT	ARGS																						
REG	<table><tr><td>1</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table>	1	r	r	r	r	r	<i>reg</i>																
1	r	r	r	r	r																			
TUPLE	<table><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table> refers to the tuple currently being processed	0	1	1	1	1	1	—																
0	1	1	1	1	1																			
HOST ID	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table> refers to the node currently being processed	0	0	0	0	1	1	—																
0	0	0	0	1	1																			
NIL	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table> the empty list	0	0	0	1	0	0	—																
0	0	0	1	0	0																			
INT	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> the next 4 bytes after the current instruction are an immediate integer to which this refers	0	0	0	0	0	1	<i>int</i>																
0	0	0	0	0	1																			
FLOAT	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> the next 4 bytes after the current instruction are an immediate float to which this refers	0	0	0	0	0	0	<i>float</i>																
0	0	0	0	0	0																			
ADDR	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table> the next 4 bytes after the current instruction are the address to which this refers	0	0	0	1	0	1	<i>addr</i>																
0	0	0	1	0	1																			
FIELD	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table> the next two bytes after the current instruction indicate a field of a register in the following format: <table><tr><td>X</td><td>X</td><td>X</td><td>X</td><td>f</td><td>f</td><td>f</td><td>f</td></tr><tr><td>X</td><td>X</td><td>X</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table> with <i>reg</i> indicating a register with a tuple value and <i>field</i> indicating the tuple's field number.	0	0	0	0	1	0	X	X	X	X	f	f	f	f	X	X	X	r	r	r	r	r	<i>field, reg</i>
0	0	0	0	1	0																			
X	X	X	X	f	f	f	f																	
X	X	X	r	r	r	r	r																	
STRING	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table> the next 4 bytes indicate the length of the string which are followed by the string itself	0	0	0	1	1	0	<i>size, content</i>																
0	0	0	1	1	0																			
ARG	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table> the next byte indicates the argument id	0	0	0	1	1	1	<i>id</i>																
0	0	0	1	1	1																			
CONST	<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table> the next 4 bytes indicates the constant id	0	0	1	0	0	0	<i>const id</i>																
0	0	1	0	0	0																			
STACK	<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table> use the element at offset from the top of the stack (4 byte unsigned int)	0	0	1	0	0	1	<i>offset</i>																
0	0	1	0	0	1																			
PC-COUNTER	<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table> refers to the program counter	0	0	1	0	1	0																	
0	0	1	0	1	0																			

ARGS BYTE FORMAT

VALUE

X	X	v	v	v	v	v	v
---	---	---	---	---	---	---	---

value

MATCHLIST BYTE FORMAT

MATCHLIST

f	f	f	f	f	f	f	f
m	m	v	v	v	v	v	v

field, marker, value
 requires that the tuple's field *field* match *value*
 mm=11 if the match list is empty and mm=01 for the last entry in the list.

AGGREGATE BYTE FORMAT

<i>none</i>	0	0	0	0
<i>first</i>	0	0	0	1
<i>int max</i>	0	0	1	0
<i>int min</i>	0	0	1	1
<i>int sum</i>	0	1	0	0
<i>float max</i>	0	1	0	1
<i>float min</i>	0	1	1	0
<i>float sum</i>	0	1	1	1
<i>int set_union</i>	1	0	0	0
<i>float set_union</i>	1	0	0	1
<i>int list sum</i>	1	0	1	0
<i>float list sum</i>	1	0	1	1

TYPE BYTE FORMAT

<i>int</i>	0	0	0	0
<i>float</i>	0	0	0	1
<i>addr</i>	0	0	1	0
<i>int list</i>	0	0	1	1
<i>float list</i>	0	1	0	0
<i>addr list</i>	0	1	0	1
<i>int set</i>	0	1	1	0
<i>float set</i>	0	1	1	1
<i>type</i>	1	0	0	0
<i>string</i>	1	0	0	1

PROPERTY BYTE POSITION

<i>aggregate</i>	1
<i>persistent</i>	2
<i>linear</i>	3
<i>delete</i>	4
<i>schedule</i>	5

NOTES:

All offsets and lengths are given in bytes.