PLAN Grammar*

for PLAN version 3.1

Jonathan T. Moore with Pankaj Kakkar

July 25, 2001

This is intended to be a human-readable form of the grammar; it is not intended to indicate precedence or associativity of operators. Terminals are specified in **bold**, non-terminals in *italics*. ϵ denotes the empty expression (reduction on no input).

<table>
<tr><td>program</td><td>::=</td><td>def-list</td></tr>
<tr><td>def-list</td><td>::=</td><td>def | def def-list</td></tr>
<tr><td>def</td><td>::=</td><td>fundef | exndef | valdef | svcdef</td></tr>
<tr><td>fundef</td><td>::=</td><td>fun var (paramlist) opt-type-expr = expr</td></tr>
<tr><td></td><td>|</td><td>fun var ( ) opt-type-expr = expr</td></tr>
<tr><td>param</td><td>::=</td><td>var opt-type-expr</td></tr>
<tr><td>paramlist</td><td>::=</td><td>param | param paramlist</td></tr>
<tr><td>exndef</td><td>::=</td><td>exception var</td></tr>
<tr><td>valdef</td><td>::=</td><td>val var opt-type-expr = expr</td></tr>
<tr><td>svcdef</td><td>::=</td><td>svc var : type-expr</td></tr>
<tr><td>opt-type-expr</td><td>::=</td><td>type-expr | ϵ</td></tr>
<tr><td>type-expr</td><td>::=</td><td>void -> type-expr</td></tr>
<tr><td></td><td>|</td><td>(type-params) -> type-expr</td></tr>
<tr><td></td><td>|</td><td>type-expr -> type-expr</td></tr>
<tr><td></td><td>|</td><td>tuple-type</td></tr>
<tr><td>type-params</td><td>::=</td><td>type-expr , type-params</td></tr>
<tr><td></td><td>|</td><td>type-expr , type-expr</td></tr>
<tr><td>list-type</td><td>::=</td><td>type-atom</td></tr>
<tr><td></td><td>|</td><td>list-type list</td></tr>
<tr><td></td><td>|</td><td>list-type chunk</td></tr>
<tr><td>tuple-type</td><td>::=</td><td>list-type</td></tr>
<tr><td></td><td>|</td><td>list-type * tuple-type</td></tr>
<tr><td>type-atom</td><td>::=</td><td>base-type | ' var | (type-expr)</td></tr>
<tr><td>base-type</td><td>::=</td><td>unit | int | char | string | bool | host | port</td></tr>
<tr><td></td><td>|</td><td>key | blob | exn | dev</td></tr>
<tr><td>expr</td><td>::=</td><td>value</td></tr>
<tr><td></td><td>|</td><td>op-expr</td></tr>
<tr><td></td><td>|</td><td>if expr then expr else expr</td></tr>
<tr><td></td><td>|</td><td>raise id</td></tr>
<tr><td></td><td>|</td><td>try expr handle id => expr</td></tr>
<tr><td></td><td>|</td><td>let def-list in expr end</td></tr>
<tr><td></td><td>|</td><td>(expr-list)</td></tr>
</table>

*This work was supported by DARPA under Contract N66001-96-C-852.

| (*arg-list*)
| |*id*|(*arg-list*)
| |*id*|( )
arg-list ::= *expr* | *expr* , *arg-list*
expr-list ::= *expr* | *expr* ; *expr-list*

value ::= *var* | **true** | **false** | **()** | **[]** | [*expr-list*]
| *int-literal* | *char-literal* | *string-literal*

op-expr ::= *id* **( )** | *id* **(** *arg-list*)
| *unary-op expr*
| *expr binary-op expr*
| *nary-op-expr*

unary-op ::= **~** | **not** | **hd** | **tl** | **fst** | **snd** | **#** *int-literal* | **noti**
| **explode** | **implode** | **ord** | **chr**
binary-op ::= **/** | **%** | ∗ | **+** | − | **and** | **or** | < | <= | > | >= | = | <> | **::** | **^**
| << | >> | **xori** | **andi** | **ori**
nary-op-expr ::= **OnRemote** **(** *expr* , *expr* , *expr* , *expr*)
| **OnNeighbor** **(** *expr* , *expr* , *expr*)
| **RetransOnRemote** **(** *expr* , *expr* , *expr* , *expr* , *expr* , *expr*)
| **foldr** **(** *expr* , *expr* , *expr*)
| **foldl** **(** *expr* , *expr* , *expr*)

int-literal ::= *digit* | *nonzero-digit digit-string*
digit-string ::= *digit* | *digit digit-string*
nonzero-digit ::= [**1** - **9**]
digit ::= [**0** - **9**]

char-literal ::= ' *character* '
character ::= ~[', \] | \\ | **\n** | **\t** | **\b** | **\r** | \' | \"

string-literal ::= "" | " *strchar-list* "
strchar-list ::= *strchar* | *strchar strchar-list*
strchar ::= ~[", \] | \\ | **\n** | **\t** | **\b** | **\r** | \' | \"

id ::= *var* | *var* **.** *id*
var ::= *varstartchar* | *varstartchar varchar-list*
varstartchar ::= [**a** - **z**, **A** - **Z**]
varchar ::= [**a** - **z**, **A** - **Z**, **0** - **9**, _]
varchar-list ::= *varchar* | *varchar varchar-list*