

## Complete macro expansion algorithm

4 Dec 1986

Here is a complete implementation of macro expansion that meets the requirements of the Standard. It defines a behavior for the two currently unspecified parts of the Standard's macro expansion process. Be assured that these two parts only come into play when the expansion process is abused (when varying *hide sets* are intermixed) and as such do not have any effect on "real" programs.

The superscript sometimes shown for a token is that token's hide set. Initially, each token has an empty hide set.

```

expand(TS) /* recur, substitute, pushback, rescan */
{
    if TS is {} then
        return {};

    else if TS is  $T^{HS} \cdot TS'$  and T is in HS then
        return  $T^{HS} \cdot \text{expand}(TS')$ ;

    else if TS is  $T^{HS} \cdot TS'$  and T is a "()-less macro" then
        return  $\text{expand}(\text{subst}(ts(T), \{\}, \{\}, HS \cup \{T\}, \{\}) \cdot TS')$ ;

    else if TS is  $T^{HS} \cdot (\cdot TS'$  and T is a "()-d macro" then
        check  $TS'$  is  $\text{actuals} \cdot )^{HS'} \cdot TS''$  and actuals are "correct for T"
        return  $\text{expand}(\text{subst}(ts(T), fp(T), \text{actuals}, (HS \sqcup HS')), \{T\}, \{\}) \cdot TS'')$ ;

    note TS must be  $T^{HS} \cdot TS'$ 
    return  $T^{HS} \cdot \text{expand}(TS')$ ;
}

glue(LS, RS) /* paste last of left side with first of right side */
{
    if LS is  $L^{HS}$  and RS is  $R^{HS'} \cdot RS'$  then
        return  $L \& R^{HS \sqcup HS'} \cdot RS'$ ; /* undefined if L&R is invalid */

    note LS must be  $L^{HS} \cdot LS'$ 
    return  $L^{HS} \cdot \text{glue}(LS', RS)$ ;
}

hsadd(HS, TS) /* add to token sequence's hide sets */
{
    if TS is {} then
        return {};

    note TS must be  $T^{HS'} \cdot TS'$ 
    return  $T^{HS \sqcup HS'} \cdot \text{hsadd}(HS, TS')$ ;
}

```

```
subst(IS,FP,AP,HS,OS) /* substitute args, handle stringize and paste */
{
  if IS is {} then
    return hsadd(HS,OS);

  else if IS is #•T•IS' and T is FP[i] then
    return subst(IS',FP,AP,HS,OS•stringize(select(i,AP)));

  else if IS is ##•T•IS' and T is FP[i] then
  {
    if select(i,AP) is {} then /* only if actuals can be empty */
      return subst(IS',FP,AP,HS,OS);
    else
      return subst(IS',FP,AP,HS,glue(OS,select(i,AP)));
  }

  else if IS is ##•THS'•IS' then
    return subst(IS',FP,AP,HS,glue(OS,THS'));

  else if IS is T•##HS'•IS' and T is FP[i] then
  {
    if select(i,AP) is {} then /* only if actuals can be empty */
      return subst(IS',FP,AP,HS,OS);
    else
      return subst(##HS'•IS',FP,AP,HS,OS•select(i,AP));
  }

  else if IS is T•IS' and T is FP[i] then
    return subst(IS',FP,AP,HS,OS•expand(select(i,AP)));

  note IS must be THS'•IS'
  return subst(IS',FP,AP,HS,OS•THS');
}
```

The remaining support functions are:

- |  |  |
|--|--|
| <i>ts</i> ( <i>T</i> )                 | Given a macro-name token, <i>ts</i> returns the replacement token sequence from the macro's definition.  |
| <i>fp</i> ( <i>T</i> )                 | Given a macro-name token, <i>fp</i> returns the (ordered) list of formal parameters from the macro's definition.   |
| <i>select</i> ( <i>i</i> , <i>TS</i> ) | Given a token sequence and an index <i>i</i> , <i>select</i> returns the <i>i</i> -th token sequence using the comma tokens (not between nested parenthesis pairs) in the original token sequence as delimiters. |
| <i>stringize</i> ( <i>TS</i> )         | Given a token sequence, <i>stringize</i> returns a single string literal token containing the concatenated spellings of the tokens.  |