What use for macros / compile-time meta-programming?

Laurence Tratt
http://tratt.net/laurie/

King's College London

2012/01/04

A brief history of macros

A brief history of macros

- Lisp (1958).

A brief history of macros

- Lisp (1958).
- Lisp macros (˜1963).

A brief history of macros

- Lisp (1958).
- Lisp macros (˜1963).
- C preprocessor (1973).

A brief history of macros

- Lisp (1958).
- Lisp macros (~1963).
- C preprocessor (1973).
- Hygienic macros (1986).

A brief history of macros

- Lisp (1958).
- Lisp macros (˜1963).
- C preprocessor (1973).
- Hygienic macros (1986).
- Template Haskell (2002) [Compile-Time Meta-Programming (CTMP)].

The Lisp tradition

- Use macros everywhere.

The Lisp tradition

- Use macros everywhere.
- Macros are power. Power is macros.

The C tradition

- Use the preprocessor for performance or portability.

The C tradition

- Use the preprocessor for performance or portability.
- Be careful. Very careful.

The big question

- Should every good language have macros / CTMP?

My perspective

- I designed Converge, which has TH-like CTMP.

My perspective

- I designed Converge, which has TH-like CTMP.
- CTMP is, more-or-less, macros.

My perspective

- I designed Converge, which has TH-like CTMP.
- CTMP is, more-or-less, macros.
- I have used ‘normal’ Converge macros twice.

My perspective

- I designed Converge, which has TH-like CTMP.
- CTMP is, more-or-less, macros.
- I have used ‘normal’ Converge macros twice. Once gratuitously.

My perspective

- I designed Converge, which has TH-like CTMP.
- CTMP is, more-or-less, macros.
- I have used ‘normal’ Converge macros twice. Once gratuitously.
- The other time, usefully, for an HTML library:

```
for elem in ["A", "B", "IMG", ...]:
  yield [|
    class ${aststring(elem)}:
      def init(self, *args):
        ...
  |]
```

My perspective

- I designed Converge, which has TH-like CTMP.
- CTMP is, more-or-less, macros.
- I have used 'normal' Converge macros twice. Once gratuitously.
- The other time, usefully, for an HTML library:

```
for elem in ["A", "B", "IMG", ...]:
  yield [|
    class ${aststring(elem)}:
      def init(self, *args):
        ...
  |]
```

- The only consistent use I have for CTMP is to compile out syntactically distinct DSLs at compile-time. [This has limitations, but is somewhat useful.]

What use for macros?

- My contention: the less syntax in your language, the more useful macros are.

What use for macros?

- My contention: the less syntax in your language, the more useful macros are.
- ‘Modern’ languages are syntax rich so macros add little.

What use for macros?

- My contention: the less syntax in your language, the more useful macros are.
- 'Modern' languages are syntax rich so macros add little.
- Macros add complexity at two levels.
  - At the language level, the manual for 'normal' Converge is the same size as the manual just for CTMP.
  - At the user level, programs that use them are usually harder to understand than those without.

What use for macros?

- My contention: the less syntax in your language, the more useful macros are.
- 'Modern' languages are syntax rich so macros add little.
- Macros add complexity at two levels.
  - At the language level, the manual for 'normal' Converge is the same size as the manual just for CTMP.
  - At the user level, programs that use them are usually harder to understand than those without.
- I'm not saying macros aren't fun. They are.

What use for macros?

- My contention: the less syntax in your language, the more useful macros are.
- 'Modern' languages are syntax rich so macros add little.
- Macros add complexity at two levels.
  - At the language level, the manual for 'normal' Converge is the same size as the manual just for CTMP.
  - At the user level, programs that use them are usually harder to understand than those without.
- I'm not saying macros aren't fun. They are. I even like C preprocessor hacks.

What use for macros?

- My contention: the less syntax in your language, the more useful macros are.
- ‘Modern’ languages are syntax rich so macros add little.
- Macros add complexity at two levels.
  - At the language level, the manual for ‘normal’ Converge is the same size as the manual just for CTMP.
  - At the user level, programs that use them are usually harder to understand than those without.
- I’m not saying macros aren’t fun. They are. I even like C preprocessor hacks.
- But for modern languages, the cost / benefit ratio is questionable.

A final quote

You can indeed "get surprisingly far without macros" when everything, including functions, is a first class value.

There are three reasons for introducing a syntactic abstraction (macro) rather than a function:

1. new binding forms
2. implicit quoting or, more generally, a data sub-language
3. an order of evaluation that is incompatible with evaluation

A lazy language makes macros for 3 unnecessary. A language with first-class functions still needs macros for 1; otherwise you keep writing

```
foo (fn x => ...)
```

all over the place.

Guy Steele, 2001