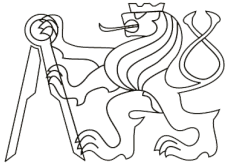
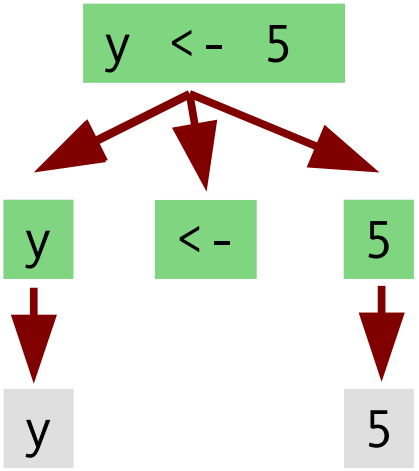
Sustainable Package Development

Tomas Kalibera
R Core, Czech Technical University



CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

Parse data for equal assign

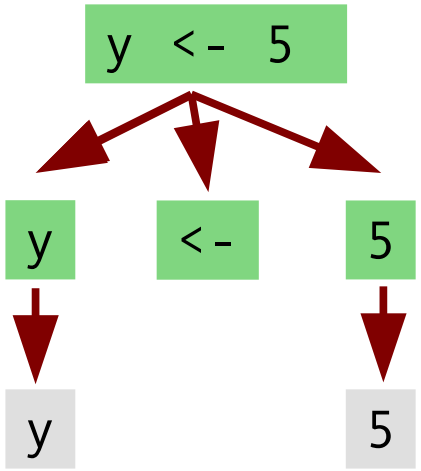



```
y <- 5
```

	line1	col1	line2	col2	id	parent	token	terminal	text
7	1	1	1	6	7	0	expr	FALSE	y <- 5
1	1	1	1	1	1	3	SYMBOL	TRUE	y
3	1	1	1	1	3	7	expr	FALSE	y
2	1	3	1	4	2	7	LEFT_ASSIGN	TRUE	<-
4	1	6	1	6	4	5	NUM_CONST	TRUE	5
5	1	6	1	6	5	7	expr	FALSE	5

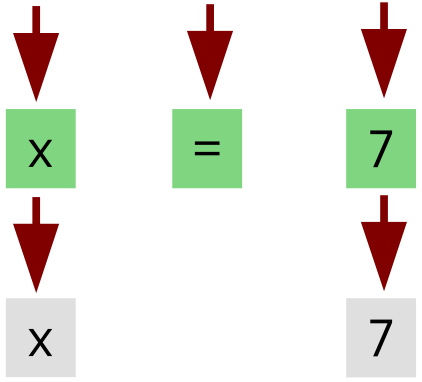
```
parsed <- parse("src.r", keep.source=TRUE)
utils::getParseData (parsed, includeText=TRUE)
```

Bug reported by Barbara Lerner



`y <- 5`

	line1	col1	line2	col2	id	parent	token	terminal	text
7	1	1	1	6	7	0	expr	FALSE	y <- 5
1	1	1	1	1	1	3	SYMBOL	TRUE	y
3	1	1	1	1	3	7	expr	FALSE	y
2	1	3	1	4	2	7	LEFT_ASSIGN	TRUE	<-
4	1	6	1	6	4	5	NUM_CONST	TRUE	5
5	1	6	1	6	5	7	expr	FALSE	5



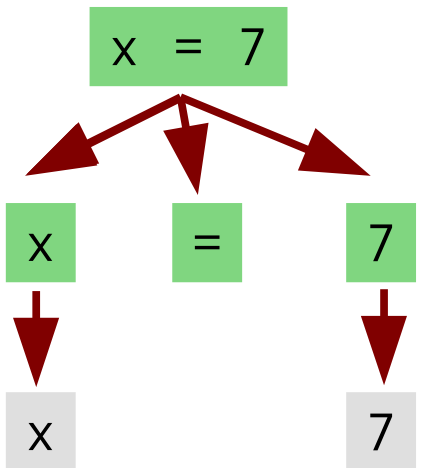
`x = 7`

	line1	col1	line2	col2	id	parent	token	terminal	text
1	1	1	1	1	1	3	SYMBOL	TRUE	x
3	1	1	1	1	3	0	expr	FALSE	x
2	1	3	1	3	2	0	EQ_ASSIGN	TRUE	=
4	1	5	1	5	4	5	NUM_CONST	TRUE	7
5	1	5	1	5	5	0	expr	FALSE	7

Trivial fix in R: change a single line

setId($$, @$);

equal_assign: expr EQ_ASSIGN expr_or_assign { $$ = xxbinary($2,$1,$3); }



x = 7

	line1	col1	line2	col2	id	parent	token	terminal	text
8	1	1	1	5	8	0	equal_assign	FALSE	x = 7
1	1	1	1	1	1	3	SYMBOL	TRUE	x
3	1	1	1	1	3	8	expr	FALSE	x
2	1	3	1	3	2	8	EQ_ASSIGN	TRUE	=
4	1	5	1	5	4	5	NUM_CONST	TRUE	7
5	1	5	1	5	5	8	expr	FALSE	7

Testing change with CRAN/BIOC

“Woken-up” segfault: quickReg

Non-deterministic installation problems:

- ~2 packages, race condition in parallel install

Sophisticated workarounds that broke:

- Styler: 129 LOC (46 true code)
- Linter: 104 LOC (71 true code)

Fixed in R 3.6.0 (R-devel 75386)

Helping to get the packages fixed

Had to debug to identify it is a problem to be fixed in packages (not R)

Provided patch for one package

One package had a new maintainer not yet familiar with the code

One maintainer chose to wait for the commit as he had difficulty building R on Windows

15 emails; commited after 6 weeks, informed CRAN of breakage, packages fixed 6 weeks later

Best practice: File a good bug report first.

Rchk PROTECT bug report

```
for (r_ssize i = 0; i < r_length(elt); ++i) {
    sexp* value = r_list_get(elt, i);
    r_list_poke(out, count, value);

    sexp* name = r_nms_get(names, i);
    if (name != r_string("")) {
        name = KEEP(r_str_unserialise_unicode(name));

        if (out_names == r_null)
            out_names = KEEP_N(init_names(out),
                               n_protect);
        r_chr_poke(out_names, count, name);
        FREE(1);
    }
    ++count;
}
```

"Unprotected variable out_names while calling allocating function r_nms_get"

Rchk PROTECT bug report

```
for (r_ssize i = 0; i < r_length(elt); ++i) {
    sexp* value = r_list_get(elt, i);
    r_list_poke(out, count, value);

    sexp* name = r_nms_get(names, i);
    if (name != r_string("")) {
        name = KEEP(r_str_unserialise_unicode(name));

        if (out_names == r_null)
            out_names = KEEP_N(init_names(out),
                               n_protect);
        r_chr_poke(out_names, count, name);
        FREE(1);
    }
    ++count;
}
```

sexp*		SEXP
r_length		Rf_xlength
r_list_get		VECTOR_ELT
r_list_poke		SET_VECTOR_ELT
r_nms_get	~	STRING_ELT
r_string	~	Rf_mkChar
KEEP		PROTECT
KEEP_N		PROTECT
		n_protect++
r_chr_poke		SET_STRING_ELT
FREE		UNPROTECT

Rchk PROTECT bug report

```
for (r_ssize i = 0; i < r_length(elt); ++i) {
    sexp* value = r_list_get(elt, i);
    r_list_poke(out, count, value);

    sexp* name = r_nms_get(names, i);
    if (name != r_string("")) {
        name = KEEP(r_str_unserialise_unicode(name));

        if (out_names == r_null)
            out_names = KEEP_N(init_names(out),
                               n_protect);
        r_chr_poke(out_names, count, name);
        FREE(1);
    }
    ++count;
}
```

"Unprotected variable out_names while calling allocating function r_nms_get"

out_names PROTECTed

out_names UNPROTECTed

Best practice: use R API directly.

Risks to maintainability: native code

LOW RISK

HIGH RISK

R

C/Fortran with .C/.Fortran

C with .Call

C++

C++ with .Call

C memory corruption
Windows

PROTECT errors
R heap corruption
Constants corruption

Missed destructors
Error handling
Indirect use of APIs

Best practice: only use native code when absolutely necessary.

Maintainability of packages impacts maintainability of R

R changes must be checked against packages

Problems discovered must be debugged

- Requires knowledge of R internals
- Few people willing and able to do this

Debugging memory corruption can take days