WIKIPEDIA

Clean (programming language)

Clean is a general-purpose purely functional computer programming language. It was called the **Concurrent Clean System**,[3] then the **Clean System**,[4][5] later just **Clean**. Clean is being developed by a group of researchers from the Radboud University in Nijmegen since 1987.[6]

Clean

Paradigm	functional
Designed by	Software Technology Research Group of Radboud University Nijmegen
First appeared	1987
Stable release	3.1 / January 5, 2022
Typing discipline	strong, static, dynamic
OS	Cross-platform
License	Simplified BSD[1]
Filename extensions	.icl, .dcl, .abc
Website	clean.cs.ru.nl (http://clean.cs.ru.nl)

Influenced by

Lean (https://foldoc.org/Lean), Miranda, Haskell

Influenced

Haskell, Idris[2]

Contents

Features

The language Clean first appeared in 1987.[7] Although development of the language itself has slowed down, some researchers are still working in the language.[8] In 2018, a spin-off company was founded that uses Clean as well.[9]

Clean shares many properties and syntax with a younger sibling language, Haskell: referential transparency, list comprehension, guards, garbage collection, higher order functions, currying and lazy evaluation. However, Clean deals with mutable state and I/O through a uniqueness typing system, in contrast to Haskell's use of monads. The compiler takes advantage of the uniqueness type system to generate more efficient code, because it knows that at any point during the execution of the program, only one reference can exist to a value with a unique type. Therefore, a unique value can be changed in place.[10]

An integrated development environment (IDE) for Microsoft Windows is included in the Clean distribution.

Examples

Hello world:

```
Start = "Hello, world!"
```

Factorial:

```
fac :: Int -> Int
fac 0 = 1
fac n = n * fac (n-1)

Start = fac 10
```

```
fac :: Int -> Int
fac n = prod [1..n] // The product of the numbers 1 to n

Start = fac 10
```

Fibonacci sequence:

```
fib :: Int -> Int
fib 0 = 1
fib 1 = 1
fib n = fib (n - 2) + fib (n - 1)

Start = fib 7
```

```
fibs :: Int Int -> [Int]
fibs x_2 x_1 = [x_2:fibs x_1 (x_2 + x_1)]

fib :: Int -> Int
fib n = (fibs 1 1) !! n

Start = fib 7
```

Infix operator:

```
(^) infixr 8 :: Int Int -> Int
(^) x 0 = 1
(^) x n = x * x ^ (n-1)
```

The type declaration states that the function is a right associative infix operator with priority 8: this states that `x*x^(n-1)` is equivalent to `x*(x^(n-1))` as opposed to `(x*x)^(n-1)`. This operator is pre-defined in StdEnv, the Clean standard library.

How Clean works

Computation is based on graph rewriting and reduction. Constants such as numbers are graphs and functions are graph rewriting formulas. This, combined with compilation to native code, makes Clean programs which use high abstraction run relatively fast according to the Computer Language Benchmarks Game.[11]

Compiling

1. Source files (.icl) and definition files (.dcl) are translated into Core Clean, a basic variant of Clean, in Clean.
2. Core clean is converted into Clean's platform-independent intermediate language (.abc), implemented in C and Clean.
3. Intermediate ABC code is converted to object code (.o) using C.
4. Object code is linked with other files in the module and the runtime system and converted into a normal executable using the system linker (when available) or a dedicated linker written in Clean on Windows.

Earlier Clean system versions were written completely in C, thus avoiding bootstrapping issues.

The SAPL system compiles Core Clean to JavaScript and does not use ABC code.

The ABC machine

To close the gap between Core Clean, a high-level functional language, and machine code, the ABC machine is used. This is an imperative abstract graph rewriting machine.[12] Generating concrete machine code from abstract ABC code is a relatively small step, so by using the ABC machine it is much easier to target multiple architectures for code generation.

The ABC machine has an uncommon memory model. It has a graph store to hold the Clean graph that is being rewritten. The A(rgument)-stack holds arguments that refer to nodes in the graph store. This way, a node's arguments can be rewritten, which is needed for pattern matching. The B(asic value)-stack holds basic values (integers, characters, reals, etc.). While not strictly necessary (all these elements could be nodes in the graph store as well), using a separate stack is much more efficient. The C(ontrol)-stack holds return addresses for flow control.

The runtime system, which is linked into every executable, has a `print` rule which prints a node to the output channel. When a program is executed, the `Start` node is printed. For this, it has to be rewritten to root normal form, after which its children are rewritten to root normal form, etc., until the whole node is printed.

Platforms

Clean is available for Microsoft Windows (IA-32 and X86-64), macOS (X86-64), and Linux (IA-32, X86-64, and AArch64).

Some libraries are not available on all platforms, like ObjectIO which is only available on Windows. Also the feature to write dynamics to files is only available on Windows.

The availability of Clean per platform varies with each version:[13][14]

Version	Date	Linux			macOS			Oracle Solaris	Windows		Miscellaneous
		IA-32	X86-64	AArch64	Motorola 68040	PowerPC	X86-64	SPARC	IA-32	X86-64	
3.1	January 5, 2022	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	
3.0	October 2, 2018	Yes	Yes	No	No	No	Yes	No	Yes	Yes	
2.4	December 23, 2011	Yes	Yes	No	No	No	Yes	No	Yes	Yes	
2.3	December 22, 2010	Yes	Yes	No	No	No	No	No	Yes	Yes	
2.2	December 19, 2006	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	
2.1.1	May 31, 2005	Yes	No	No	No	Yes	No	Yes	Yes	No	
2.1.0	October 31, 2003	Yes	No	No	No	Yes	No	Yes	Yes	No	
2.0.2	December 12, 2002	Yes	No	No	No	Yes	No	Yes	Yes	No	
2.0.1	July 4, 2002	Yes	No	No	No	Yes	No	Yes	Yes	No	
2.0	December 21, 2001	No	No	No	No	No	No	No	Yes	No	
1.3.3	September 13, 2000	Yes	No	No	No	Yes	No	Yes	Yes	No	
1.3.2	July 1, 1999	No	No	No	Yes	Yes	No	Yes	Yes	No	
1.3.1	January 1999	Yes	No	No	No	Yes	No	Yes	Yes	No	
1.3	May 22, 1998	Yes	No	No	No	Yes	No	Yes	Yes	No	
1.2.4	June 1997	No	No	No	Yes	Yes	No	No	Yes	No	
1.2.3	May 1997	No	No	No	Yes	Yes	No	No	Yes	No	
1.2	January 13, 1997	No	No	No	Yes	Yes	No	No	No	No	
1.1.3	October 1996	No	No	No	No	No	No	Yes	No	No	OS/2 (i80386)
1.1.2	September 1996	Yes	No	No	No	No	No	Yes	No	No	SunOS 4 (SPARC)
1.1	March 1996	Yes	No	No	Yes	No	No	No	No	No	
1.0.2	September 1995	Yes	No	No	Yes	No	No	Yes	No	No	OS/2 (i80386); SunOS 4 (SPARC)
1.0	May 1995	No	No	No	Yes	No	No	No	No	No	OS/2 (i80386)
0.8.4	May 11, 1993	Yes	No	No	Yes	No	No	No	No	No	Experimental T800 transputer release
0.8.3	February 26, 1993	No	No	No	Yes	No	No	No	No	No	
0.8.1	October 19, 1992	No	No	No	Yes	No	No	No	No	No	
0.8	July 13, 1992	No	No	No	Yes	No	No	No	No	No	OS/2 (i80386); SunOS 3–4 (SPARC)
0.7	May 1991	No	No	No	Yes	No	No	No	No	No	SunOS 3–4 (SPARC)

Comparison to Haskell

A 2008 benchmark showed that Clean native code performs roughly equally well as Haskell (GHC), depending on the benchmark.[15]

Syntactic differences

The syntax of Clean is very similar to that of Haskell, with some notable differences:[10]

Haskell	Clean	Remarks
`[ x | x <- [1..10] , isOdd x]`	`[ x \\ x <- [1..10] | isOdd x]`	list comprehension
`x:xs`	`[x:xs]`	cons operator

`data Tree a`
`  = Empty`
`  | Node (Tree a) a (Tree a)`
	`:: Tree a`
	`  = Empty`
	`  | Node (Tree a) a (Tree a)`
		algebraic data type

`(Eq a, Eq b) => ...`	`... | Eq a & Eq b`	class assertions and contexts
`fun t@(Node l x r) = ...`	`fun t=:(Node l x r) = ...`	as-patterns
`if x > 10 then 10 else x`	`if (x > 10) 10 x`	if

In general, Haskell has introduced more syntactic sugar than Clean.

References

1. "Download Clean" (https://clean.cs.ru.nl/Download_Clean#Clean_3.0_License). *Clean*. Retrieved 23 July 2019.
2. "Idris - Uniqueness Types" (http://docs.idris-lang.org/en/latest/reference/uniqueness-types.html). Retrieved 2018-11-20.
3. "Clean 0.7: README" (https://web.archive.org/web/20190524121044/https://ftp.cs.ru.nl/Clean/old/Clean07/Sun4/README). Archived from the original (https://ftp.cs.ru.nl/Clean/old/Clean07/Sun4/README) on 2019-05-24.
4. "Clean 1.0: README" (https://web.archive.org/web/20190505113256/https://ftp.cs.ru.nl/Clean/old/Clean10/README). Archived from the original (https://ftp.cs.ru.nl/Clean/old/Clean10/README) on 2019-05-05.
5. "Clean 1.3: README" (https://web.archive.org/web/20190427194714/https://ftp.cs.ru.nl/Clean/Clean13/README). Archived from the original (https://ftp.cs.ru.nl/Clean/Clean13/README) on 2019-04-27.
6. "Radboud University Nijmegen: Department of Software Science: Software" (https://www.mbsd.cs.ru.nl/Software).
7. "FAQ" (http://wiki.clean.cs.ru.nl/FAQ). *Clean*. Retrieved 2021-11-26.
8. "Publications" (https://clean.cs.ru.nl/Publications). *Clean*. Retrieved 2021-11-26.
9. "Home" (https://www.top-software.nl/index.html). *TOP Software Technology*. Retrieved 26 November 2021.
10. ftp://ftp.cs.ru.nl/pub/Clean/papers/2007/achp2007-CleanHaskellQuickGuide.pdf

11. "Which programming languages are fastest?" (https://web.archive.org/web/20110628185627/http://shootout.alioth.debian.org/u32/which-programming-languages-are-fastest.php). *Computer Language Benchmarks Game*. Archived from the original on 28 June 2011.
12. Koopman, Pieter (December 10, 1990). *Functional Programs as Executable Specifications* (PhD). Katholieke Universiteit Nijmegen. p. 35. ISBN 90-9003689-X.
13. "Release history" (https://clean.cs.ru.nl/Release_history). *Clean*. Retrieved 7 January 2022.
14. "Index of /Clean" (https://ftp.cs.ru.nl/Clean/). Retrieved 7 January 2022.
15. Jansen, Jan Martin; Koopman, Pieter; Plasmeijer, Rinus (2008). "From Interpretation to Compilation" (ftp://ftp.cs.ru.nl/pub/Clean/papers/2008/janj08-CEFP07-InterpretationToCompilation.pdf) (PDF). Retrieved 2016-05-21.

External links

- Clean Wiki (http://wiki.clean.cs.ru.nl/)
- Cloogle: Clean function search engine (http://cloogle.org)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Clean_(programming_language)&oldid=1064252581"

This page was last edited on 7 January 2022, at 12:01 (UTC).