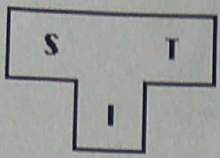
(3)

Bootstrapping

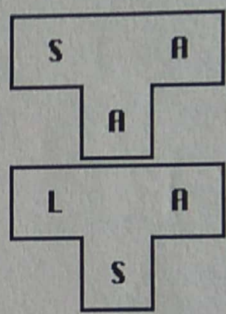


A compiler is characterized by three languages:

1. Source Language
2. Target Language
3. Implementation Language

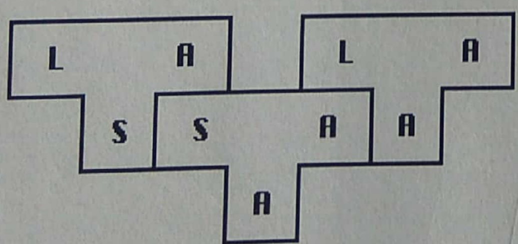
Notation: $^{S}C_I^T$ **represents a compiler for Source** S**, Target** T**, implemented in** I**. The** ***T-diagram*** **shown above is also used to depict the same compiler.**

To create a new language, L, for machine A:



1. Create $^{S}C_A^A$, a compiler for a subset, S, of the desired language, L, using language A, which runs on machine A. (Language A may be assembly language.)
2. Create $^{L}C_S^A$, a compiler for language L written in a subset of L.
3. Compile $^{L}C_S^A$ using $^{S}C_A^A$ to obtain $^{L}C_A^A$, a compiler for language L, which runs on machine A and produces code for machine A.

$$^{L}C_S^A \rightarrow {}^{S}C_A^A \rightarrow {}^{L}C_A^A$$

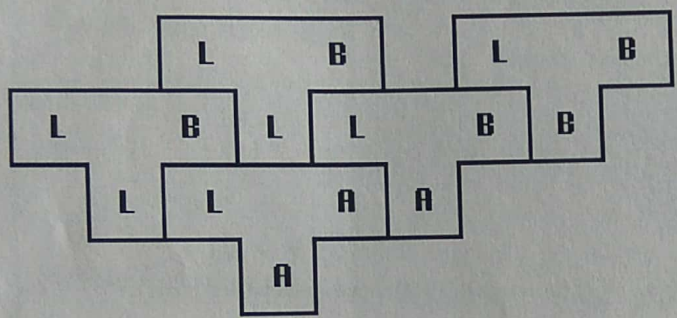


The process illustrated by the T-diagrams is called ***bootstrapping*** **and can be summarized by the equation:**

$$L_S A + S_A A = L_A A$$

To produce a compiler for a different machine B:

1. Convert ${}^{L}C_{S}^{A}$ into ${}^{L}C_{L}^{B}$ (by hand, if necessary). Recall that language S is a subset of language L.
2. Compile ${}^{L}C_{L}^{B}$ to produce ${}^{L}C_{A}^{B}$, a *cross-compiler* for L which runs on machine A and produces code for machine B.
3. Compile ${}^{L}C_{L}^{B}$ with the cross-compiler to produce ${}^{L}C_{B}^{B}$, a compiler for language L which runs on machine B.



Quick & Dirty Compiler

Dirty compilers produce an object program quickly but in this stage code program may be inefficient of its storage consumption & its speed. It also called as Quick Compiler.

Cross- Compiler and Bootstrapping Process:

A cross-compiler is a compiler that runs on one machine and produces object code for another machine. The cross-compiler is used to implement the compiler, which is characterized by three languages:

1. The Source Language,
2. The Target Language (object language),
3. Implementation Language.

If a compiler has been implemented in its own language, then this arrangement is called a "bootstrap" arrangement.

Implementing a Bootstrap compiler:

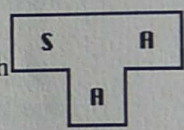
Notation: ${}^{S}C^{T}_{I}$ represents a compiler for Source *S*, Target *T*, implemented in language *I*. The *T-diagram* shown above is also used to depict the same compiler.

Compilers are of two kinds;

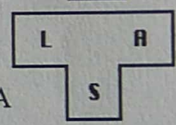
1. Native Compiler
2. Cross Compiler

1. ***Native Compiler:*** Native compilers are written in the same langage as the target language. E.g. SMM is a compiler for the language S that is in a language that runs on machine M and generates output code that runs on machine M.

Steps involving to create a new language, L, for machine A (or Native Compiler):

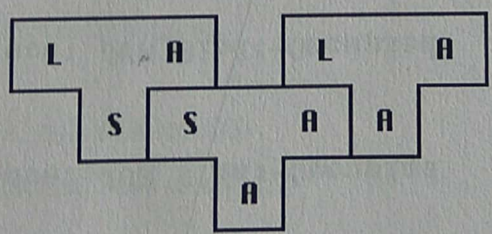


1. Create ${}^{S}C^{A}_{A}$, a compiler for a subset, S, of the desired language, L, using language A, which runs on machine A. (Language A may be assembly language.)



2. Create ${}^{L}C^{A}_{S}$, a compiler for language L written in a subset of L.
3. Compile ${}^{L}C^{A}_{S}$ using ${}^{S}C^{A}_{A}$ to obtain ${}^{L}C^{A}_{A}$, a compiler for language L, which runs on machine A and produces code for machine A.

$$ {}^{L}C^{A}_{S} \rightarrow {}^{S}C^{A}_{A} \rightarrow {}^{L}C^{A}_{A} $$



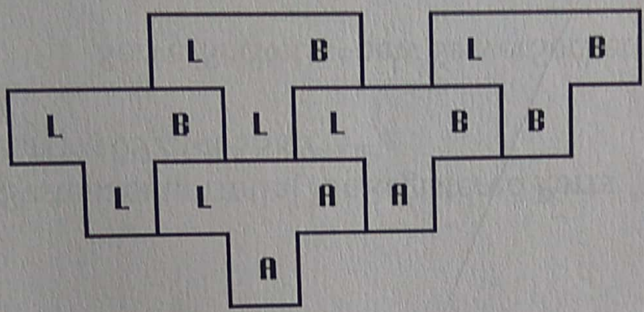
The process illustrated by the T-diagrams is called *bootstrapping* and can be summarized by the equation:

$$ L_S A + S_A A = L_A A $$

2. ***Cross Compiler:*** Cross compilers are written in different language as the target language. E.g. SNM is a compiler for the language S that is in a language that runs on machine N and generates output code that runs on machine M.

Steps involving to produce a compiler for a different machine B (or cross compiler):

1. Convert ${}^{L}C_{S}^{A}$ into ${}^{L}C_{L}^{B}$ (by hand, if necessary). Recall that language S is a subset of language L.
2. Compile ${}^{L}C_{L}^{B}$ to produce ${}^{L}C_{A}^{B}$, a *cross-compiler* for L which runs on machine A and produces code for machine B.
3. Compile ${}^{L}C_{L}^{B}$ with the cross-compiler to produce ${}^{L}C_{B}^{B}$, a compiler for language L which runs on machine B.



Finite state machine:

A *finite automaton* is an abstract machine that serves as a recognizer for the strings that comprise a regular language. The idea is that we can feed an input string into a finite automaton, and it will answer "yes" or "no" depending on whether or not the input string belongs to the language that the automaton recognizes.

A finite auto-mata consists of a finite number of states and a finite number of transition, and these transitions are defined on certain, specific symbols called input symbols. A finite automate (M) is defined by using five (5) tuples that are;

$$M = (Q, \sum, \delta, q_0, F)$$

Where;

- Q is a finite non empty set of states,
- $\sum$ is a finite non empty set of input alphabet (symbols)
- δ is a transitions functions in the automata.
- i.e. ; $\delta : Q * \sum \rightarrow Q$
- q_0 is the initial state (i.e.: q_0 is a subset of Q)
- F is the set of final states (i.e.: F is a subset of Q)