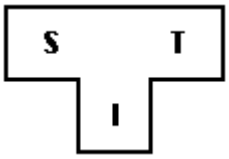
Bootstrapping a compiler

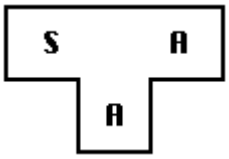
A compiler is characterized by three languages:



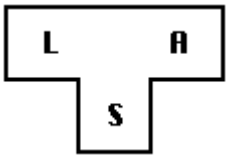
1. Source Language
2. Target Language
3. Implementation Language

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:

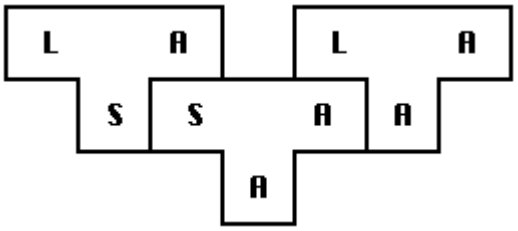


1. Create ${}^{S}C_{A}^{A}$, a compiler for a subset, S, of the desired language, L, using language A, which runs on machine A. (Language A may be assembly language.)



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.

$$ {}^{L}C_{S}^{A} \rightarrow {}^{S}C_{A}^{A} \rightarrow {}^{L}C_{A}^{A} $$



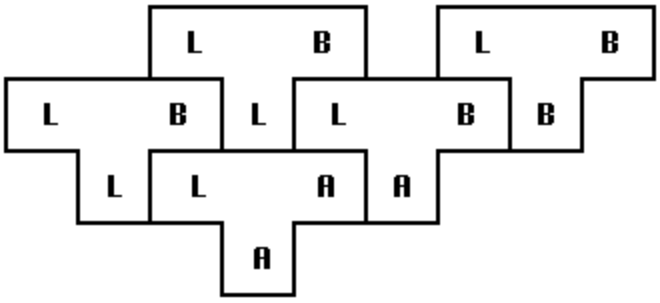
The process illustrated by the T-diagrams is called *bootstrapping* and can be summarized by the equation:

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

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 ${}^{L}C^{B}_{L}$ to produce ${}^{L}C^{B}_{A}$, a *cross-compiler* for L which runs on machine A and produces code for machine B.
3. Compile ${}^{L}C^{B}_{L}$ with the cross-compiler to produce ${}^{L}C^{B}_{B}$, a compiler for language L which runs on machine B.



Difference between compiler and interpreter

• A complier converts the high level instruction into machine language while an interpreter converts the high level instruction into an intermediate form.

• Before execution, entire program is executed by the compiler whereas after translating the first line, an interpreter then executes it and so on.

• List of errors is created by the compiler after the compilation process while an interpreter stops translating after the first error.

• An independent executable file is created by the compiler whereas interpreter is required by an interpreted program each time.