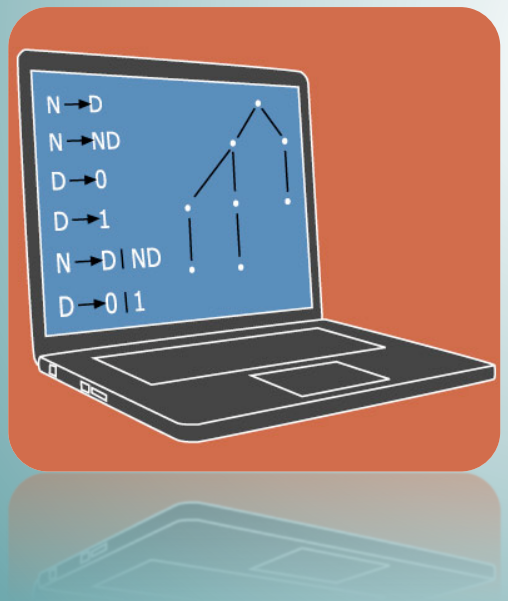
Procesadores de Lenguaje

Bootstrapping de Compiladores y Diagramas en T



Cristina Tirnauca

Domingo Gómez Pérez

Dpto. de Matemáticas,

Estadística y Computación

Este tema se publica bajo Licencia:

Creative Commons BY-NC-SA 3.0

¿Qué es un compilador?

Compilador = un programa informático que traduce un programa escrito en un lenguaje de programación (high-level) a otro lenguaje de programación (lenguaje de máquina, low-level), generando un programa equivalente que la máquina será capaz de interpretar.

Notación (diagramas en T)

Vamos a ver diferentes formas gráficas para representar:

- Compiladores que reciben un lenguaje A y lo transforman en un lenguaje B (están escritos en lenguaje K).
- Programas (están escritos en K).
- Máquinas (que ejecutan lenguaje K).
- Interpretes (traductor de A a B).

Problema I

Nos inventamos un nuevo lenguaje de programación MiJava y tenemos que escribir nosotros el compilador.

Soluciones al problema:

- Escribimos el compilador en MiJava (?).
- Escribimos el compilador en un lenguaje de programación que ya está disponible en la máquina donde vamos a utilizar el compilador (?).

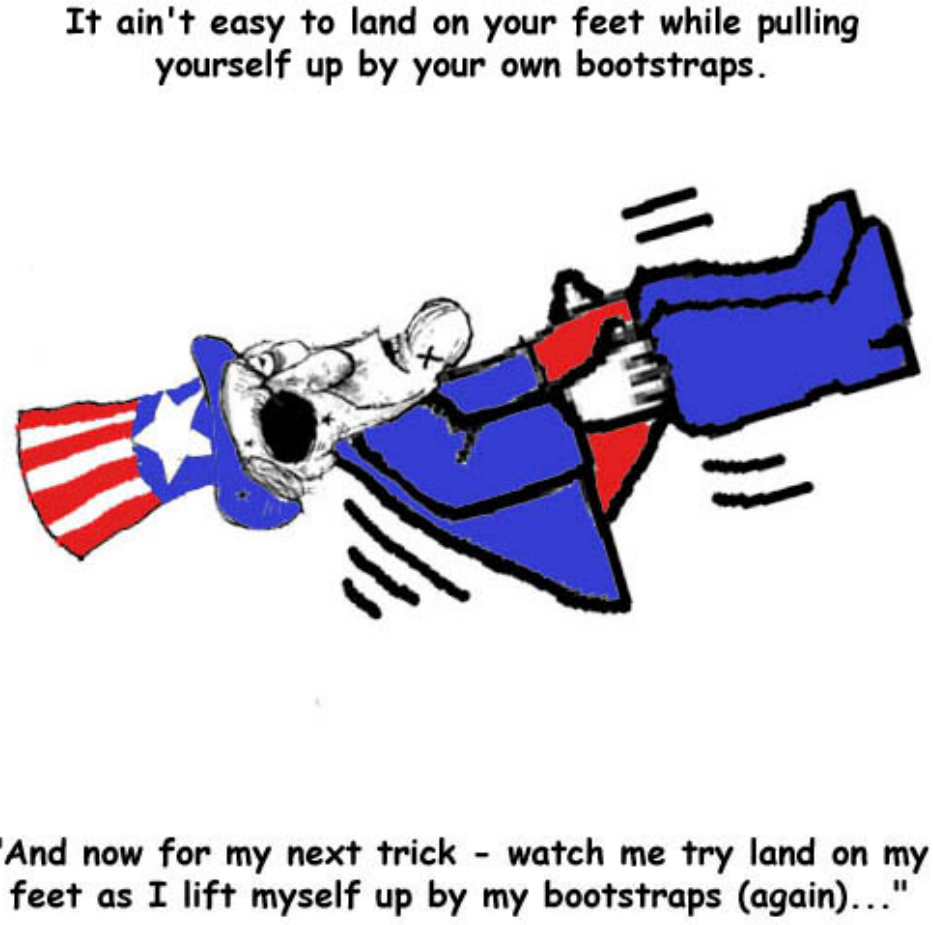
Problema II

Se inventa un nuevo tipo de procesador BetterThanPentium, con su propio lenguaje nativo para el que no existen compiladores de Pascal (por ejemplo).

Soluciones al problema:

- Llamar al que hizo ese procesador.
- Escribir un compilador, desde cero, de Pascal en el lenguaje de máquina BetterThanPentium.
- Hacer un pequeño traductor de otro lenguaje máquina (por ejemplo, Pentium) a BetterThanPentium.

Dos problemas, una solución: Bootstrapping



Fuente: `http://www.nationalview.org/thumbnails.htm`

¿Qué fue primero: el huevo o la gallina?

La idea del bootstrapping es simple: escribes el compilador del lenguaje A al lenguaje B en A, y luego dejas el compilador compilarse a sí mismo.

El resultado es un compilador de A a B escrito en B.

Puede sonar un poco paradójico que el compilador se compile a sí mismo. En breve veremos cómo se resuelve esta aparente paradoja.

Ventajas del bootstrapping

- es un test no trivial para el lenguaje que está siendo compilado;
- los desarrolladores del compilador sólo necesitan saber el lenguaje de programación que está siendo compilado;
- se trata de una comprobación de coherencia global, ya que debe ser capaz de reproducir su propio código objeto.

Compilando compiladores (half bootstrapping)

Volvamos otra vez al problema II. Supongamos que tenemos:

- una máquina Pentium (que ejecuta lenguaje nativo Pentium)
- una máquina BetterThanPentium (que ejecuta lenguaje nativo BetterThanPentium)
- un compilador que transforma Pascal en código nativo Pentium y que está escrito en Pentium.

Queremos hacer un compilador de Pascal a BetterThanPentium escrito en BetterThanPentium.

Solución ineficiente

Hacemos un traductor binario de código Pentium a código BetterThanPentium que se ejecute en Pentium, y lo utilizamos para traducir nuestro compilador.
Problemas que surgen:

- Genera código Pentium. Lo resolvemos utilizando nuestro traductor.
- Ahora tenemos que hacer dos operaciones: compilar nuestro programa escrito en Pascal a código Pentium, y traducir el resultado de Pentium a BetterThanPentium. Seguramente se pierda eficiencia.
- Pero lo más gordo de todo, todavía necesitamos la máquina Pentium para traducir de Pentium a BetterThanPentium.

Una solución mejor

Empezamos escribiendo un compilador de Pascal a BetterThanPentium en Pascal.

Después lo compilamos con nuestro compilador de Pascal a Pentium (en nuestra máquina Pentium).

El resultado es un compilador de Pascal a BetterThanPentium escrito en código nativo Pentium.

Ahora podemos ejecutar este último compilador en la máquina Pentium, dejándole compilarse "a sí mismo".

Hemos obtenido el compilador deseado.

Compilando compiladores (full bootstrapping)

Lo anterior se basa en que tenemos un compilador del lenguaje deseado en una máquina diferente. ¿Pero que pasa si diseñamos un lenguaje desde cero?

Tres maneras distintas:

- escribiendo un compilador QAD (quick and dirty) en un lenguaje de programación existente
- utilizando un interprete QAD
- bootstrapping incremental

El compilador QAD

Dado un nuevo lenguaje de programación MiJava, vamos a utilizar uno ya conocido (por ejemplo, Java) para construir un compilador para MiJava que se ejecute en nuestra máquina (vamos a suponer que tenemos un Pentium).

Empezamos construyendo un compilador muy sencillo, de MiJava a Java, escrito en Java. Lo compilamos para obtener un ejecutable (hemos obtenido lo que se llama el compilador QAD).

Escribimos otro compilador, de MiJava a Pentium (esta vez en MiJava) y lo compilamos con el QAD.

El resultado es un programa en Java, que vamos a compilar, obteniendo un compilador que hace lo que queremos (pero puede ser demasiado lento)

El interprete QAD

Empezamos escribiendo un traductor de MiJava a Java y lo compilamos. Obtenemos un traductor de MiJava a Pentium.

Como en el caso anterior, escribimos un compilador de MiJava a Pentium en MiJava y lo dejamos compilarse a sí mismo (vamos a necesitar el traductor de MiJava a Pentium).

Bootstrapping incremental

Empezamos escribiendo un compilador para un pequeno subconjunto del lenguaje, utilizando sólo este subconjunto para escribirlo.

Bootstraping (con uno de los métodos anteriores) para compilar este compilador.

Aumentamos un poco el subconjunto, y añadimos al primer compilador más funcionalidad, sin utilizar las nuevas características. Lo compilamos con el compilador obtenido en el paso anterior. Repetimos el proceso.