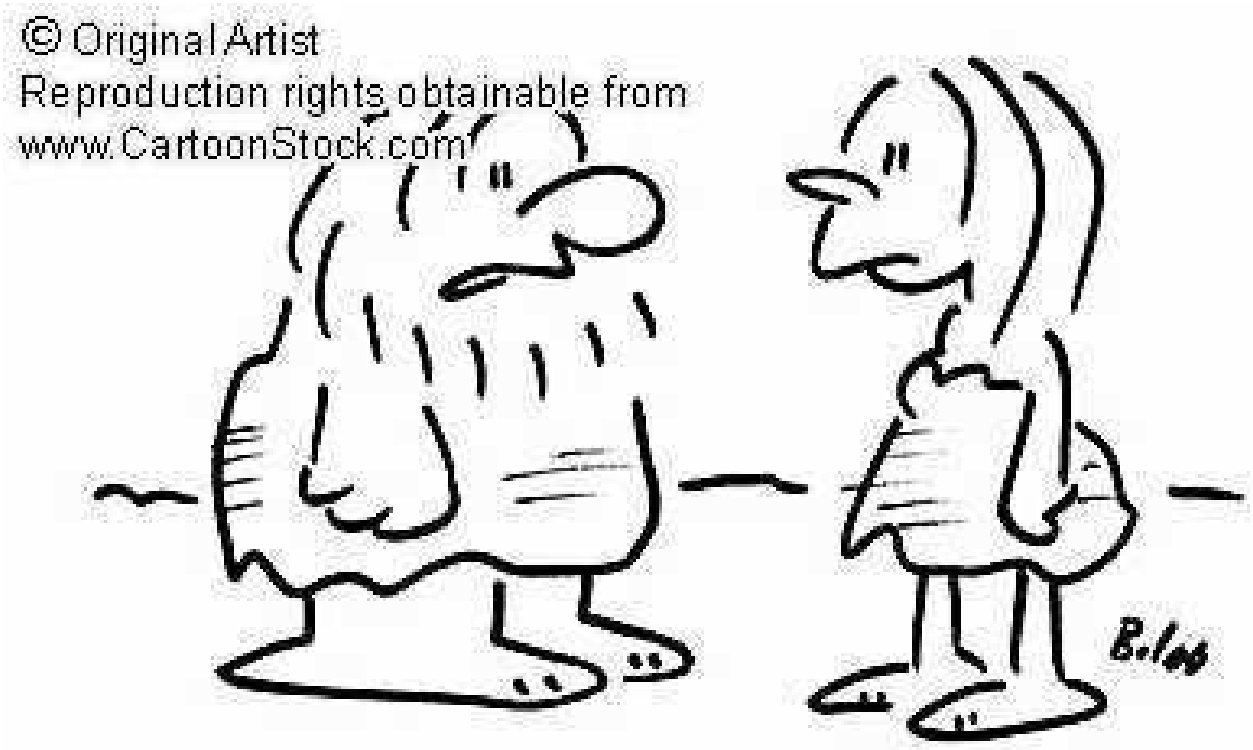
Implementación de una metodología de reparación de errores sintácticos para el generador de analizadores sintácticos GNU Bison

Memoria de Título

Claudio Saavedra

Facultad de Ingeniería
Universidad de Talca
Curicó – Chile

27 de marzo de 2008


© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com
"'Language' is a great invention, but
I keep getting syntax errors."

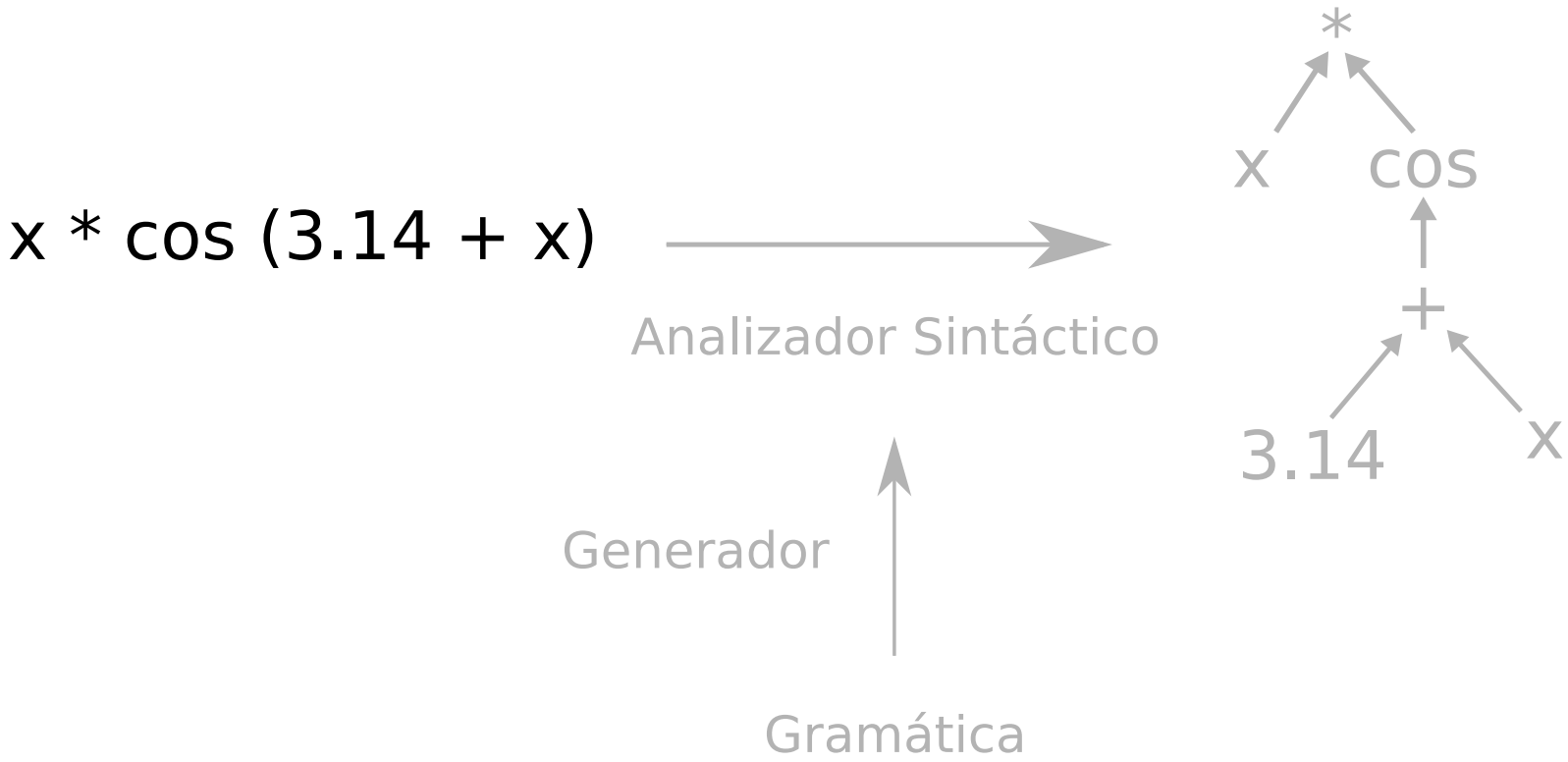
Contenidos

Contenidos

Contenidos

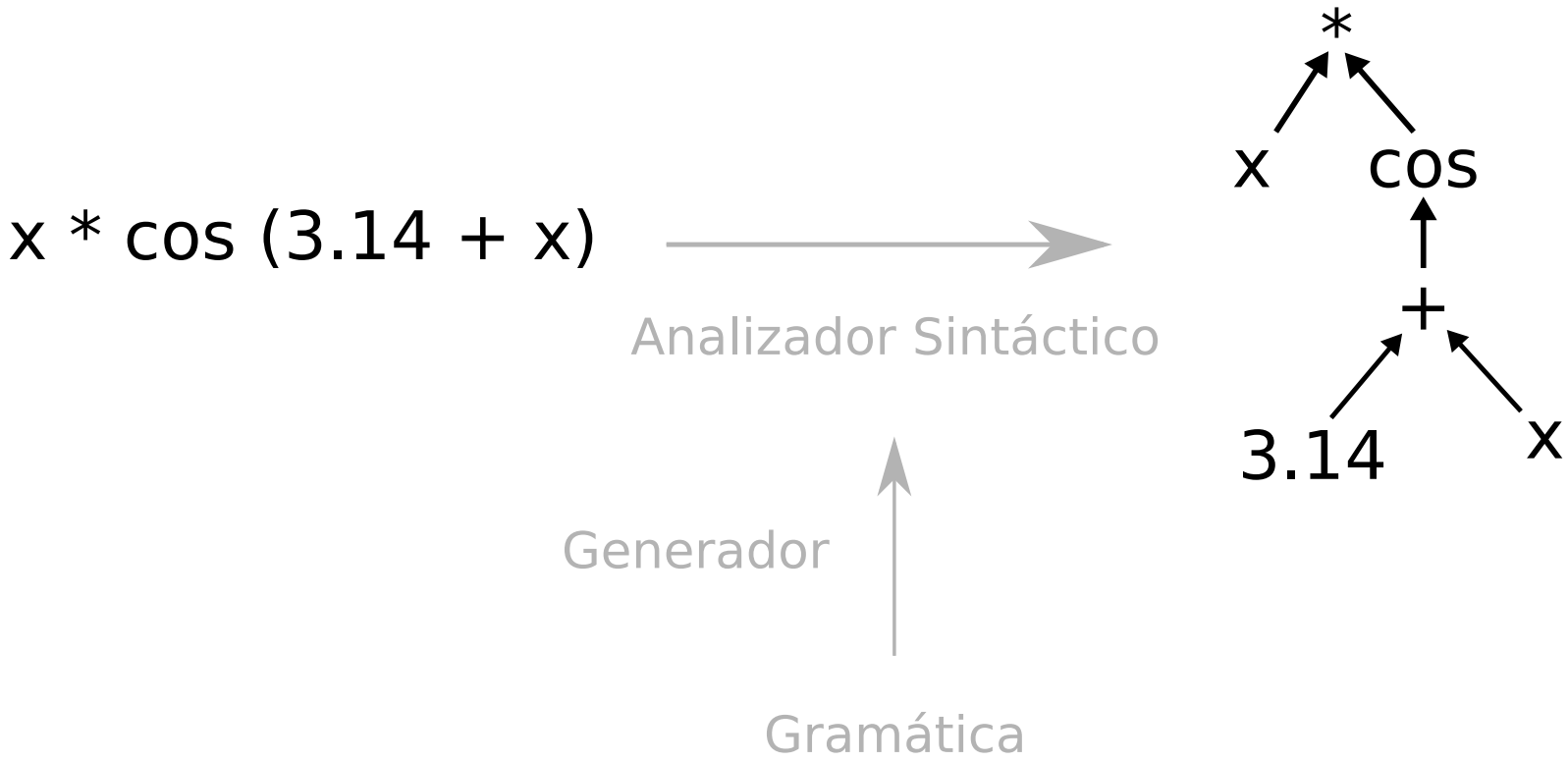
Análisis sintáctico

Visión general



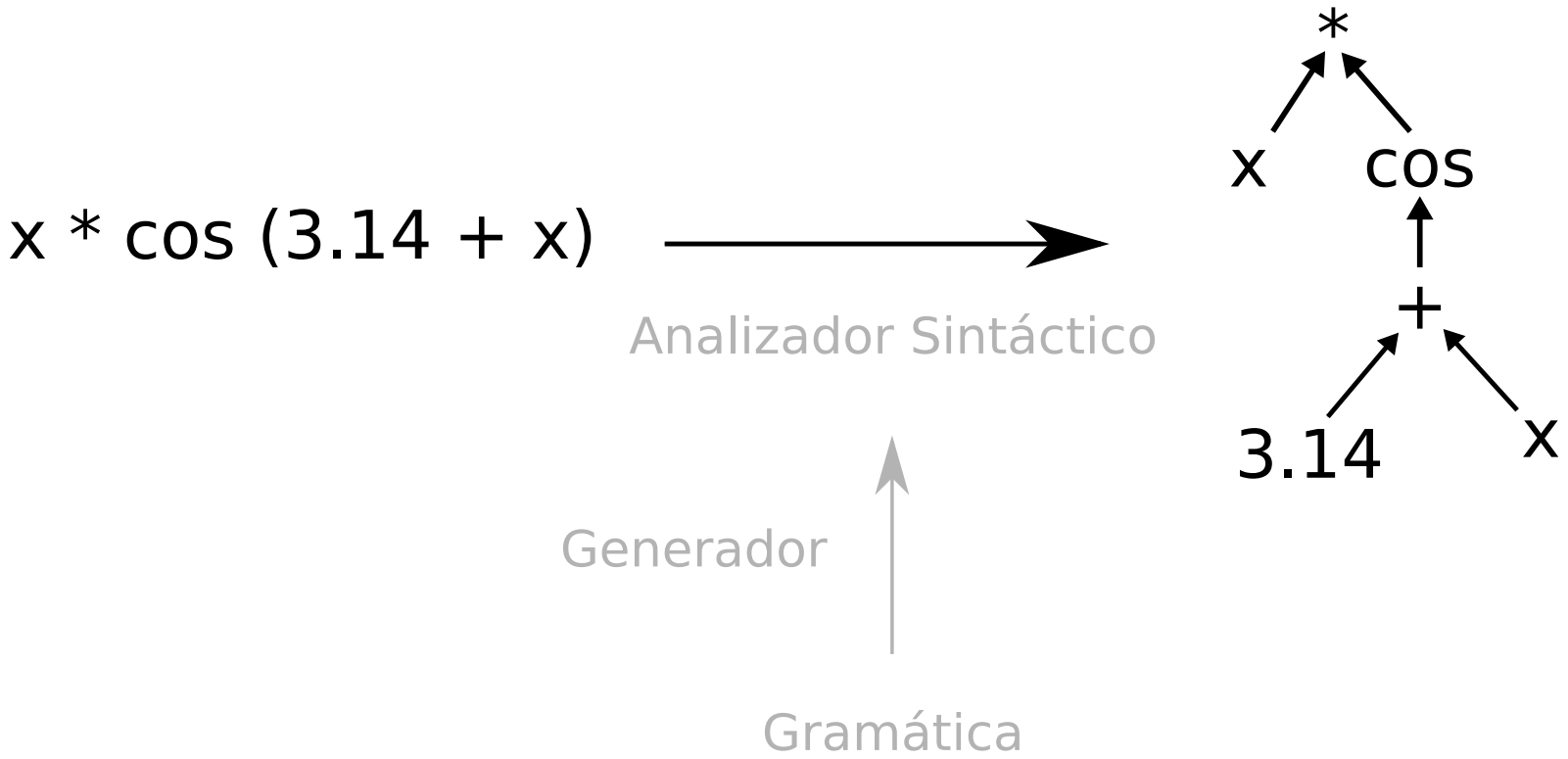
Análisis sintáctico

Visión general



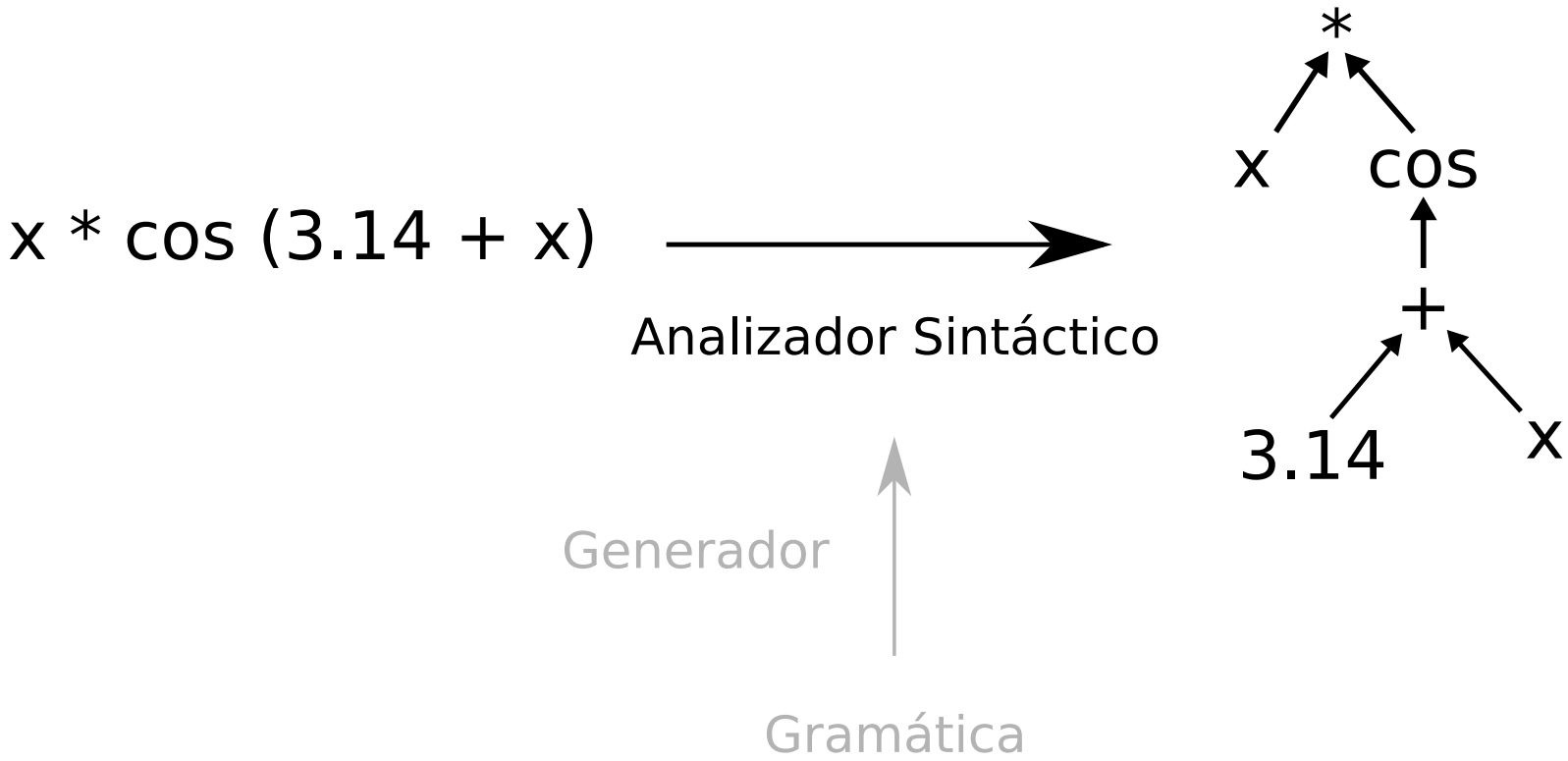
Análisis sintáctico

Visión general



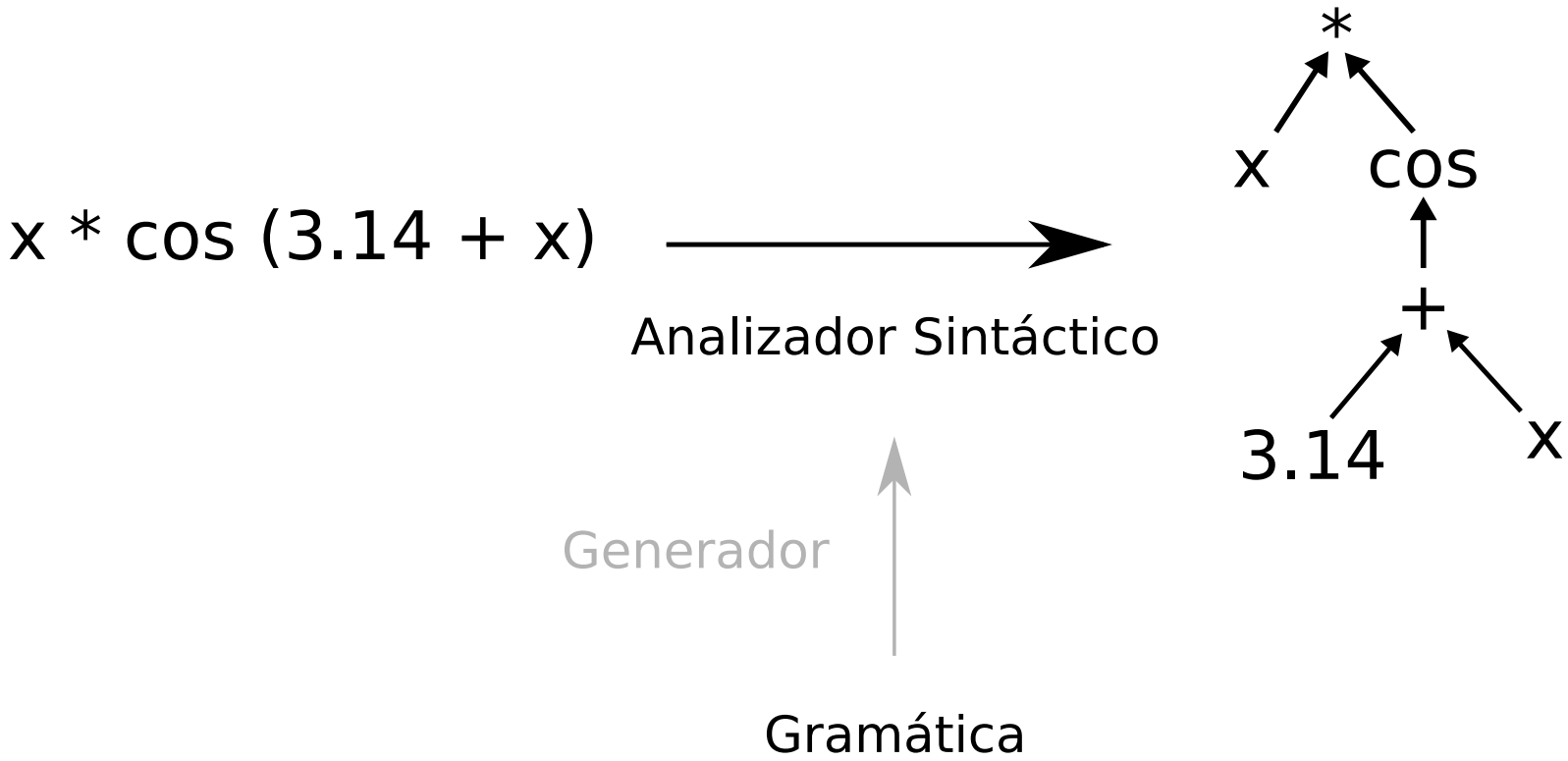
Análisis sintáctico

Visión general



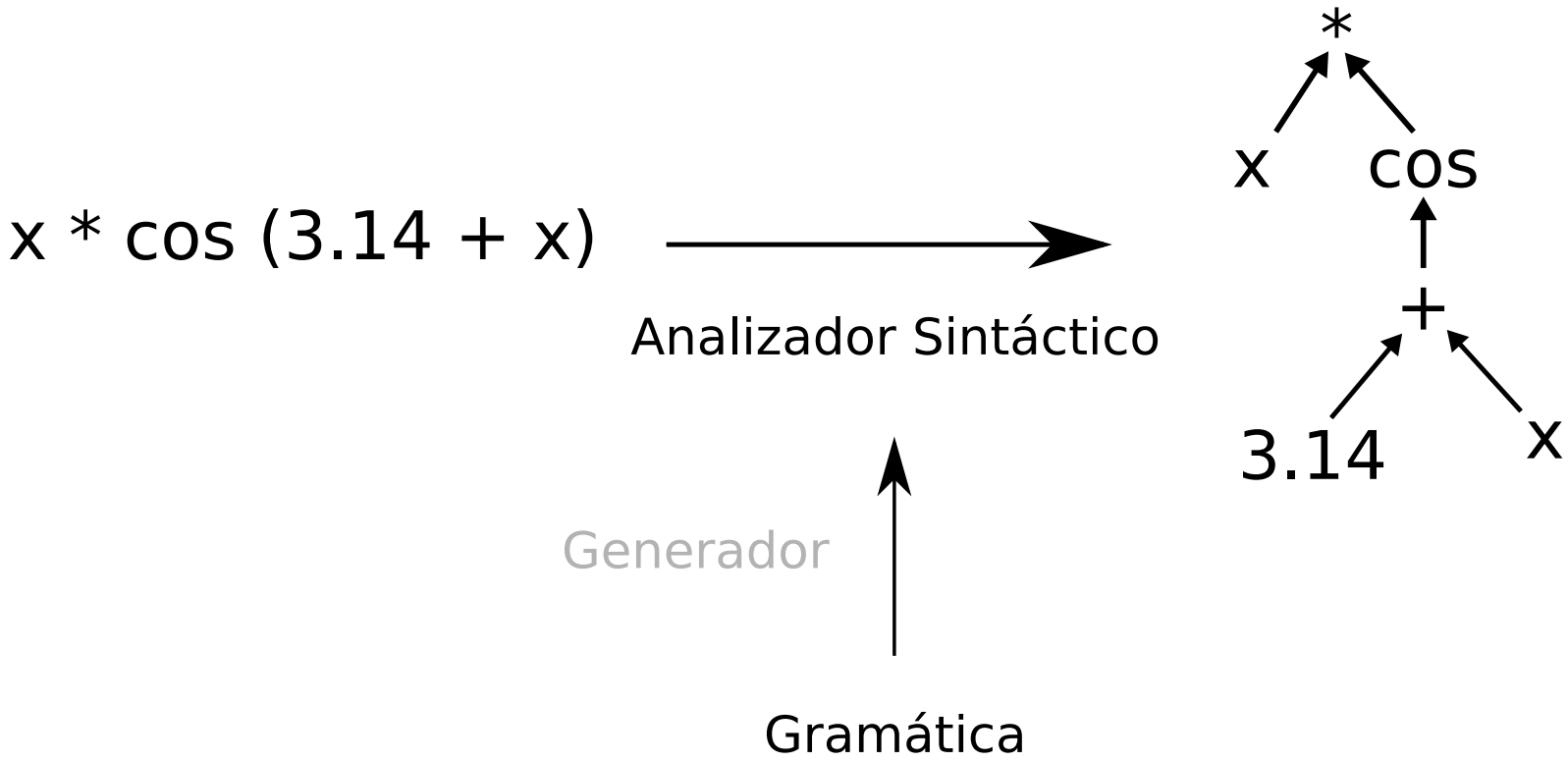
Análisis sintáctico

Visión general



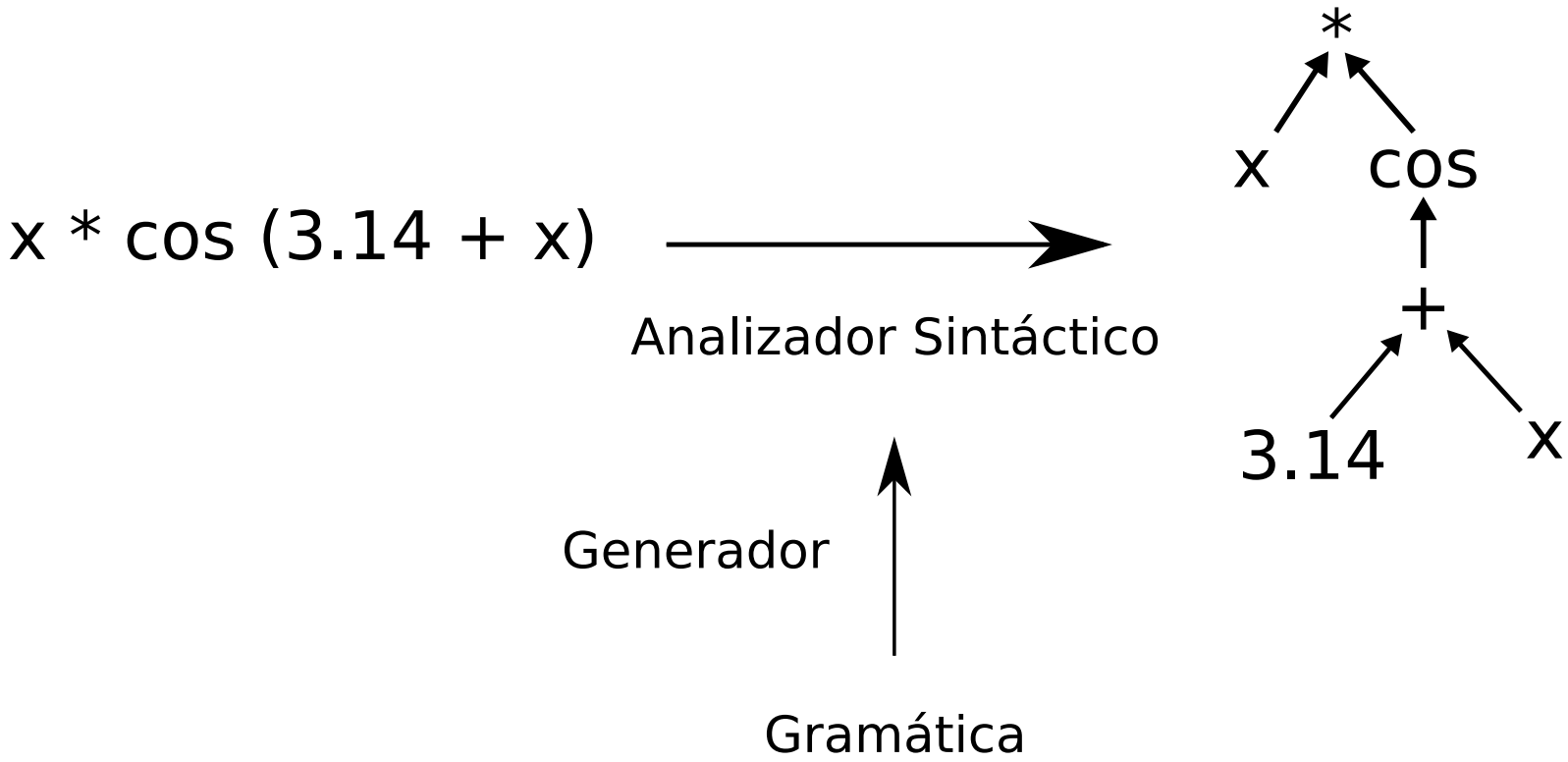
Análisis sintáctico

Visión general



Análisis sintáctico

Visión general



Contenidos

Errores sintácticos

Un error sintáctico se produce cuando se detecta una construcción que no pertenece al lenguaje

Ejemplo

`x * (/ 12 - x)` no es una expresión algebraica válida

¿Qué hacer ante la presencia de un error?

Tratamiento de errores sintácticos

Diagnóstico: Informar al usuario sobre la presencia del error

Recuperación: Intentar continuar el análisis para detectar otros errores

Reparación: Idealmente, corregir el error automáticamente

Estado del arte en generadores

Tratamiento de errores sintácticos

Software	diagn.	recup.	repar.
yacc	$\surd$	$\surd$	$\times$
Bison	$\surd$	$\surd$	$\times$
ml-yacc	$\surd$	$\surd$	$\surd$
JavaCC	$\surd$	$\surd$	$\times$
ANTLR	$\surd$	$\surd$	$\times$

Tratamiento de errores en Bison

Tratamiento de errores sintácticos

- Consiste en una recuperación en modo de pánico
- Para un mejor funcionamiento, el usuario necesita adaptar su gramática de entrada
- No hay corrección automática de errores

Algunas preguntas

Tratamiento de errores sintácticos

- ¿Puede el diagnóstico ser exacto si se desconoce cómo corregir el error?
- ¿Es cómodo para los usuarios adaptar sus gramáticas para permitir un mejor manejo de los errores?
- ¿Podemos hacer algo para mejorar la situación actual?

Corrección automática de errores

Tratamiento de errores sintácticos

- Existen metodologías que permiten reparar automáticamente los errores
- La reparación consiste en la modificación de la entrada en un punto alrededor del token de error
- Burke y Fisher (1987) propusieron una metodología interesante: eficiente, flexible, independiente del lenguaje

Contenidos

En esta memoria..

- Implementamos reparación automática de errores para los analizadores generados por Bison, inspirados en la metodología de Burke-Fisher
- Proveemos de un diagnóstico de errores de alta calidad
- Añadimos interfaces apropiadas para el uso de estas características

Contenidos

Ejemplo de análisis sintáctico

Consideremos la gramática definida por las siguientes producciones

Gramática

```
Expr → Suma ;
Suma → Num + Suma
Suma → Num
```

y el análisis de la entrada `45 + 18 ;`

Ejemplo de análisis sintáctico

Entrada

```
45 + 18 ;
```

Pila de estados

s_0

Desplazamiento de `45`
Desplazamiento de `+`
Desplazamiento de `18`
Reducción de `45 + 18`
Desplazamiento de `;`
Reducción de `Suma ;`
Aceptación

Gramática

```
Expr → Suma ;
Suma → Num + Suma
Suma → Num
```

Ejemplo de análisis sintáctico

Entrada

```
45 + 18 ;
```

Pila de estados

$s_0 s_1$

Desplazamiento de `45`

Desplazamiento de `+`

Desplazamiento de `18`

Reducción de `45 + 18`

Desplazamiento de `;`

Reducción de `Suma ;`

Aceptación

Gramática

```
Expr → Suma ;
Suma → Num + Suma
Suma → Num
```

Ejemplo de análisis sintáctico

Entrada

```
45 + 18 ;
```

Pila de estados

$s_0\, s_1\, s_4$

Desplazamiento de `45`

Desplazamiento de `+`

Desplazamiento de `18`

Reducción de `45 + 18`

Desplazamiento de `;`

Reducción de `Suma ;`

Aceptación

Gramática

```
Expr → Suma ;
Suma → Num + Suma
Suma → Num
```

Ejemplo de análisis sintáctico

Entrada

```
45 + 18 ;
```

Pila de estados

$s_0\, s_1\, s_4\, s_1$

Desplazamiento de `45`

Desplazamiento de `+`

Desplazamiento de `18`

Reducción de `45 + 18`

Desplazamiento de `;`

Reducción de `Suma ;`

Aceptación

Gramática

```
Expr → Suma ;
Suma → Num + Suma
Suma → Num
```

Ejemplo de análisis sintáctico

Entrada

`45 + 18 ;`

Pila de estados

$s_0 s_3$

Desplazamiento de `45`
Desplazamiento de `+`
Desplazamiento de `18`
Reducción de `45 + 18`
Desplazamiento de `;`
Reducción de `Suma ;`
Aceptación

Gramática

`Expr` $\longrightarrow$ `Suma ;`
`Suma` $\longrightarrow$ `Num + Suma`
`Suma` $\longrightarrow$ `Num`

Ejemplo de análisis sintáctico

Entrada

`Suma ;`

Pila de estados

$s_0\, s_3\, s_6$

Desplazamiento de `45`
Desplazamiento de `+`
Desplazamiento de `18`
Reducción de `45 + 18`
Desplazamiento de `;`
Reducción de `Suma ;`
Aceptación

Gramática

`Expr` $\longrightarrow$ `Suma ;`
`Suma` $\longrightarrow$ `Num + Suma`
`Suma` $\longrightarrow$ `Num`

Ejemplo de análisis sintáctico

Entrada

`Suma ;`

Pila de estados

$s_0 s_2$

Desplazamiento de `45`
Desplazamiento de `+`
Desplazamiento de `18`
Reducción de `45 + 18`
Desplazamiento de `;`
Reducción de `Suma ;`
Aceptación

Gramática

`Expr` $\longrightarrow$ `Suma ;`
`Suma` $\longrightarrow$ `Num + Suma`
`Suma` $\longrightarrow$ `Num`

Ejemplo de análisis sintáctico

Entrada

`Expr`

Pila de estados

$s_0\, s_2$

Desplazamiento de `45`
Desplazamiento de `+`
Desplazamiento de `18`
Reducción de `45 + 18`
Desplazamiento de `;`
Reducción de `Suma ;`
Aceptación

Gramática

`Expr` $\longrightarrow$ `Suma ;`
`Suma` $\longrightarrow$ `Num + Suma`
`Suma` $\longrightarrow$ `Num`

Contenidos

Contenidos

Recuperación de errores de Burke-Fisher

Características generales

- Consiste en tres técnicas de reparación: simple, de ámbito, secundaria
- La recuperación simple consiste en la eliminación, inserción o sustitución de un token por otro
- La recuperación de ámbito consiste en la inserción de una secuencia de tokens que cierre un ámbito (e.g., `end if ;` o `end while ;`)
- La recuperación secundaria consiste en la eliminación de un fragmento de código

Postergación del análisis

- Algunas correcciones requieren deshacer parte del análisis (costoso, complejo)
- Burke-Fisher introduce una metodología de postergación del análisis sintáctico para simular la capacidad de deshacer el análisis de k tokens
- k es un parámetro ajustable por el usuario

Búsqueda de una reparación simple

- Para la reparación simple, buscamos reparaciones candidatas sobre los k tokens postergados y el token de error
- Las reparaciones candidatas deben permitir un análisis por sobre cierta cantidad de tokens
- La reparación a aplicar se elige mediante una heurística de selección sobre las candidatas
- Esta heurística requiere información adicional sobre el lenguaje: sustituciones preferidas; tokens preferidos para eliminación e inserción; y palabras claves del lenguaje

Reparación de ámbito

- Si la reparación simple falla, invocamos la reparación de ámbito
- Intentamos cerrar algún ámbito mediante inserción de una secuencia cerradora en los mismos puntos donde se buscó una reparación simple
- Cada secuencia cerradora es especificada por el usuario mediante una directiva añadida a Bison
- Si falla, invocamos la recuperación secundaria en modo de pánico

Contenidos

Parámetros y directivas

Parámetros

`YYDEFERRAL_LEVEL`: Nivel de postergación k (por defecto, 2)

`YYERROR_MIN_THRESHOLD`: Equivalente a t_{min} (por defecto, 2)

`YYERROR_THRESHOLD`: Equivalente a t_e (por defecto, 5)

Directivas

`%keyword`: Para indicar las palabras claves del lenguaje

`%prefer`: Tokens preferidos para inserción o eliminación

`%subst`: Sustitución preferida para un token

`%closer`: Especifica cada secuencia de tokens cerradores

Funciones

Diagnóstico

```
void yydiagnosis (const char const *message);
```

Funciones

Diagnóstico

```
void yydiagnosis (const char const *message);
```

Con `message` una especificación textual de la corrección.

Contenidos

Contenidos

Evaluación

- Construimos tres analizadores sintácticos para distintos lenguajes: Una calculadora, Pascal y Ada 95
- Corrimos los analizadores sobre distintos programas erróneos
- Categorizamos las reparaciones como excelentes, buenas, pobres y errores no corregidos

Calculadora sencilla

Ejecutadas 20 pruebas, con un total de 24 errores sintácticos:

Excelente	Buena	Pobre	No Corregido
13	8	0	3
54.2 %	33.3 %	0 %	12.5 %

Analizador para Pascal

Ejecutadas 20 pruebas, con un total de 45 errores sintácticos:

Excelente	Buena	Pobre	No Corregido
38	1	2	4
84.4 %	2.2 %	4.4 %	8.9 %

Analizador para Ada 95

Ejecutada una prueba, con un total de 32 errores sintácticos:

Excelente	Buena	Pobre	No Corregido
17	2	1	12
53.3 %	6.2 %	3.1 %	37.5 %

Observaciones

- Un grado de postergación $k > 3$ no mejora la calidad de las reparaciones
- Múltiples errores consecutivos pueden ser problemáticos para la recuperación de ámbito y secundaria

Contenidos

Calculadora sencilla

Expresión con un error

```
) (234 + 1);
```

Calculadora sencilla

Expresión con un error

```
) (234 + 1);
```

Calculadora sencilla

Expresión con un error

```
(234 + 1);
```

```
Unexpected ')' ignored
Operation in 1.3-1.13: result is 235
```

En Pascal

Código con 2 errores simples

```
program p(input, output);
var key, record : array [1..limit] if integer;
begin
    x := 1
end.
```

En Pascal

Código con 2 errores simples

```
program p(input, output);
var key, record : array [1..limit] if integer;
begin
    x := 1
end.
```

En Pascal

Código con 2 errores simples

```
program p(input, output);
var key, some_id : array [1..limit] if integer;
begin
    x := 1
end.
```

```
Warning: IDENTIFIER expected instead of RECORD
```

En Pascal

Código con 2 errores simples

```
program p(input, output);
var key, some_id : array [1..limit] if integer;
begin
    x := 1
end.
```

```
Warning: IDENTIFIER expected instead of RECORD
```

En Pascal

Código con 2 errores simples

```
program p(input, output);
var key, some_id : array [1..limit] of integer;
begin
   x := 1
end.
```

```
Warning: IDENTIFIER expected instead of RECORD
Warning: OF expected instead of IF
```

En Ada 95

Código con error de ámbito

```
procedure P is
begin
   loop
      if Y < 0 then Z := 3;
   end loop;
end P;
```

En Ada 95

Código con error de ámbito

```
procedure P is
begin
   loop
      if Y < 0 then Z := 3;
   end loop;
end P;
```

En Ada 95

Código con error de ámbito

```
procedure P is
begin
   loop
      if Y < 0 then Z := 3;
      end if;
   end loop;
end P;
```

```
line 5: END IF ';' inserted
```

Contenidos

Metodología

Tiempo de ejecución (mediante `gprof` y `gcov`):

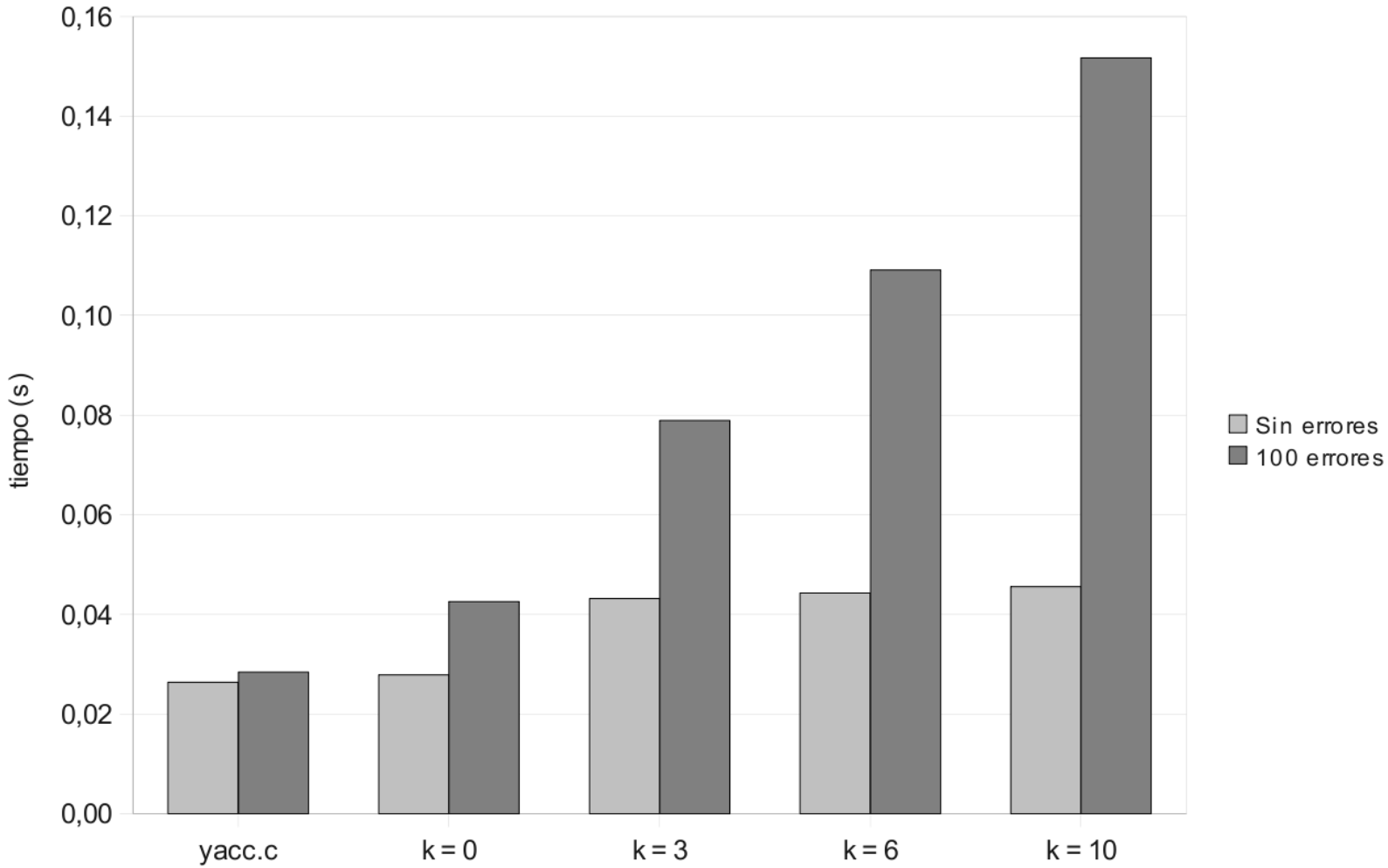
- Comparación entre código correcto y erróneo
- Distintos niveles de postergación del análisis
- Comparación de la implementación original y la nuestra

Uso de memoria (mediante `valgrind`):

- Uso de memoria para distintos niveles de postergación v/s analizador original

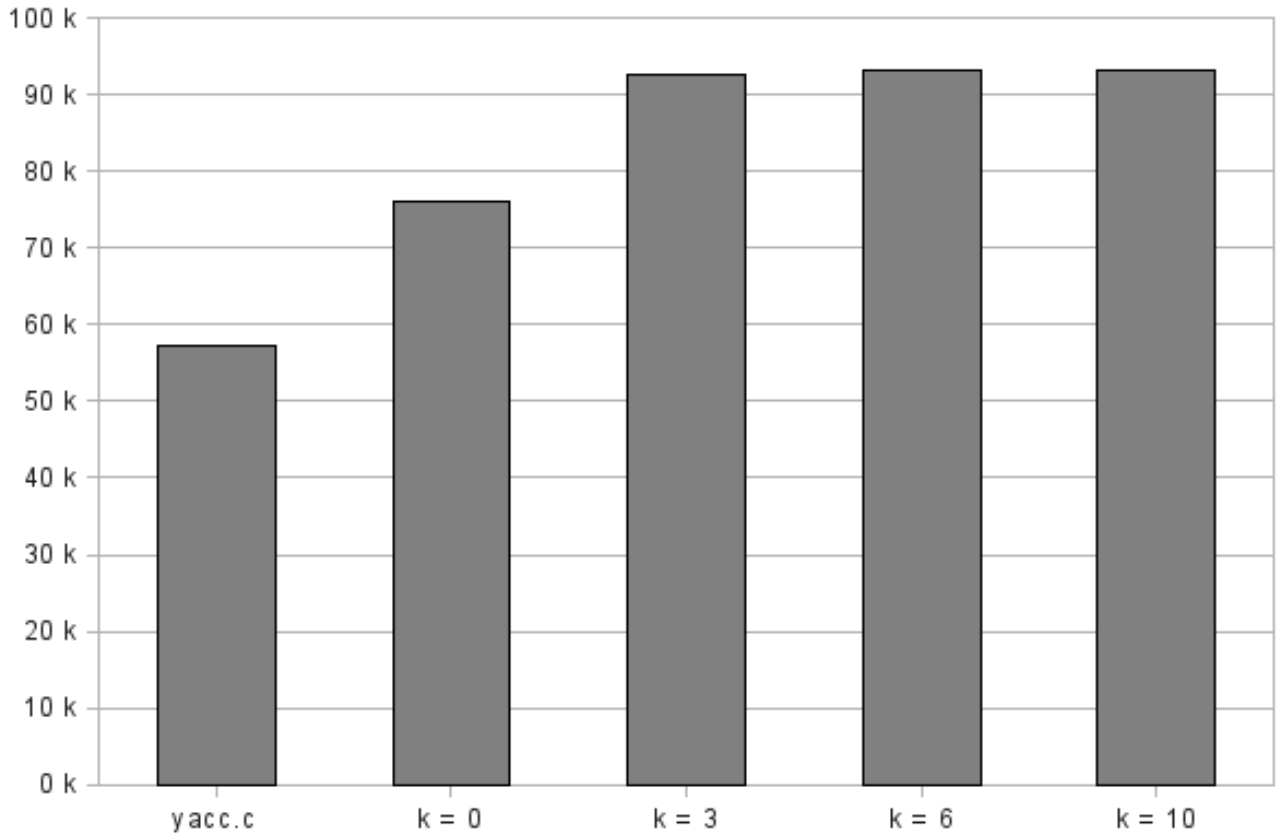
Tiempo de ejecución

Desempeño para distintas versiones de un analizador para Ada 95



Uso de memoria

Comparación para distintas versiones de un analizador para Ada 95



Contenidos

Lo que conseguimos

- Adaptar la técnica de Burke-Fisher a la naturaleza de Bison
- Corregir satisfactoriamente la mayoría de los errores sintácticos presentes en programas para distintos lenguajes
- Posibilitar la entrega de un diagnóstico de errores de alta calidad
- Separar el ajuste de la metodología de recuperación de errores de la especificación del lenguaje

Trabajo futuro

- Permitir la búsqueda (¿recursiva?) de múltiples errores de ámbito consecutivos
- Implementar una reparación secundaria más precisa
- Optimizar la implementación
- Generar automáticamente las secuencias cerradoras de tokens (Charles, 1991)
- Publicar los detalles de la implementación y resultados en un reporte técnico útil para desarrolladores y usuarios de Bison
- Integrar el código con una próxima versión de Bison

Discusión..