Lenguajes de Programación 2022-1
Nota de clase 13: Excepciones
Procedimental

Javier Enríquez Mendoza

13 de enero de 2022

En los programas desarrollados en un ambiente real podemos encontrarnos en muchas situaciones en donde los cómputos no pueden terminar o completarse correctamente y en esos casos el lenguaje reporta a la persona que desarrolla en él la ocurrencia de esta situación. Algunos ejemplos de estas situaciones son: errores aritméticos como divisiones entre cero, referencias no encontradas, indices fuera de rango en estructuras de datos, desborde de la pila de ejecución, entre muchos otros.

Algunas de estas situaciones pueden evitarse programativamente, es decir lidiar con ellas cambiando o restringiendo los programas que desarrollamos. Sin embargo, hay situaciones realmente excepcionales que no somos capaces de reconocer al momento de escribir el programa. Para lidiar con estas excepciones es necesaria una herramienta dentro del lenguaje que nos permita reconocerlas y nos brinde de una forma de lidiar con su ocurrencia, como lo es la transferencia del control del programa a un manejador de excepciones ya definido en el lenguaje.

En esta nota de clase estudiaremos estos conceptos así como tres mecanismos de manejo de errores y excepciones, así como las correspondientes extensiones a la máquina $\mathcal{C}$ para agregarlos a TinyC .

1 Errores

La forma mas simple de introducir el manejo de excepciones al lenguaje es mediante un constructor `error` que al ser evaluado detenga completamente la ejecución del programa, terminando el proceso de transición en la máquna $\mathcal{C}$.

Definición 1.1 (Errores en TinyC)**.** A continuación se presentan las extensiones necesarias para agregar errores al lenguaje.

Sintaxis

$$e ::= \ldots \mid \texttt{error}$$

Semántica Operacional Para definir la semántica dinámica en la máquina $\mathcal{C}$ de define una nueva clase de estados llamados de propagación. Estos estados se encargan, como su nombre lo dice, a propagar un error a la pila de control de tal forma que el error se propague hasta llegar aun estado final, es decir, hasta vaciar la pila de control. Estos estados se denotan

como

$$\mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\!\prec \texttt{error}$$

Con lo que definimos las reglas de transición para el constructor de error como sigue:

$$\frac{}{\mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ \texttt{error} \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\!\prec \texttt{error}}$$

$$\frac{}{\mathfrak{m}; \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\!\prec \texttt{error} \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\!\prec \texttt{error}}$$

En las reglas anteriores el error se propaga sin importar el marco en el tope de la pila para detener la ejecución del programa, de esta forma la propagación de error terminará con el estado:

$$\blacklozenge \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\!\prec \texttt{error}$$

Que representa un estado final.

Semántica estática La regla de tipado para el constructor de `error` es la siguiente:

$$\frac{}{\Gamma \vdash \texttt{error} : \mathsf{T}}$$

La regla anterior permite que la constante de error tenga cualquier tipo, esto es necesario para preservar el tipado de los programas, sin embargo, la propiedad de unicidad de tipos se pierde, que no es importante desde un punto de vista teórico, sin embargo, no contar con esta propiedad se traduce en dificultades para la implementación de algoritmos de inferencia de tipos.

Si bien la presencia de un constructor de error es la forma mas simple de tratar con las excepciones en los programas, también es la forma mas agresiva pues la presencia de alguna excepción termina completamente con la ejecución del programa, en las siguientes secciones presentaremos una alternativa mas amigable parta manejar excepciones.

2 Manejo de excepciones con bloques

Otra alternativa para el manejo de errores de una forma menos radical y mas útil para el control del flujo de la ejecución del programa consiste en no sólo agregar una constante de error sino también contar con un manejador de excepciones que sea capaz de lidiar con el error generado en la ejecución volviendo a un punto estable del programa. De esta forma la propagación del error no llega hasta un estado final sino que se detiene en cuanto encuentra dentro de la pila de control el manejador de excepciones correspondiente, en este sentido un error define un salto en el control del programa y no la terminación de la ejecución. Para lograr esto se definen los constructores `try` y `catch` que definen los bloques de interés para el manejo de excepciones.

Definición 2.1 (Bloques de excepción en TinyC)**.** Se define la extensión de la máquina $\mathcal{C}$ para agregar bloques de manejo de excepciones a TinyC .

Sintaxis Concreta

$$e ::= \ldots \mid \texttt{error} \mid \texttt{try}\, e_1\, \texttt{catch}\, e_2$$

Sintaxis Abstracta

$$a ::= \ldots \mid \texttt{error} \mid \texttt{catch}(e_1, e_2)$$

Marcos de la pila de control

$$\frac{}{\texttt{catch}(\square, e, \mathcal{L}, \mathcal{G}) \;\; \textsf{marco}}$$

Observemos como en el marco se almacenan también los ambientes en los cuales se declara el bloque de manejo de excepciones.

Semántica operacional Para la semántica operacional, las transiciones del constructor de error funcionan de la misma forma que en la extensión presentada en la sección anterior y solo cambia cuando hay un `catch` se encuentra como tope de la pila, lo que se define con las siguientes transiciones

$$\frac{}{\mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ \texttt{catch}(e_1, e_2) \longrightarrow_{\mathcal{C}} \texttt{catch}(\square, e_2, \mathcal{L}, \mathcal{G}); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ e_1}$$

$$\frac{}{\texttt{catch}(\square, e_2, \mathcal{L}', \mathcal{G}'); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec v \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec v}$$

$$\frac{}{\texttt{catch}(\square, e_2, \mathcal{L}', \mathcal{G}'); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\prec \texttt{error} \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L}' \mid \mathcal{G}' \succ e_2}$$

De esta forma, en lugar de detener la ejecución del programa, cuando el error encuentra una sentencia `catch` en la pila se detiene la propagación y se ejecuta el bloque de código definido para lidiar con el error.

Semántica estática

$$\frac{\Gamma \vdash e_1 : \mathsf{T} \qquad \Gamma \vdash e_2 : \mathsf{T}}{\Gamma \vdash \texttt{catch}(e_1, e_2) : \mathsf{T}}$$

Como `catch` puede regresar tanto e_1 como e_2 dependiendo de la ocurrencia de un error, ambas deben tener el mismo tipo para garantizar el tipado de los programas.

Observación. Es de importancia mencionar que el alcance en una expresión `try`...`catch` se resuelve de manera dinámica en contraste con el alcance del resto de los constructores.

3 Excepciones con valor

En esta sección estudiamos una forma más general del manejo de errores. Esta técnica continua con la ejecución del programa pasando el control a un punto estable de la ejecución, al igual que la propuesta en a sección anterior, pero la diferencia es que en este caso se busca no sólo atrapar el error sino que este error venga acompañado de un valor que nos sirva para comprender la procedencia del error y también a continuar con la ejecución de este. Esto se logra con los constructores `raise` y `handle`.

Definición 3.1 (Excepciones con valor en TinyC). Se define la extensión de la máquina $\mathcal{C}$ agregando los constructores de manejo de excepciones con valores.

Sintaxis concreta

$$e ::= \ldots \mid \texttt{raise}\, e \mid \texttt{handle}\, e_1\, \texttt{with}\, x \Rightarrow e_2$$

Sintaxis abstracta

$$a ::= \ldots \mid \texttt{raise}(a) \mid \texttt{handle}(a_1, x.a_2)$$

Marcos de la máquina $\mathcal{C}$

$$\frac{}{\texttt{raise}(\square)\ \ \mathsf{marco}} \qquad \frac{}{\texttt{handle}(\square, x.e, \mathcal{L}, \mathcal{G})\ \ \mathsf{marco}}$$

Semántica operacional Se define la semántica operacional con las siguientes reglas de transición para la máquina $\mathcal{C}$.

$$\frac{}{\mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ \texttt{raise}(e) \longrightarrow_{\mathcal{C}} \texttt{raise}(\square); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ e}$$

$$\frac{}{\texttt{raise}(\square); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec v \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\prec \texttt{raise}(v)}$$

$$\frac{}{\mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ \texttt{handle}(e_1, x.e_2) \longrightarrow_{\mathcal{C}} \texttt{handle}(\square, x.e_2, \mathcal{L}, \mathcal{G}); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ e_1}$$

$$\frac{}{\texttt{handle}(\square, x.e_2, \mathcal{L}', \mathcal{G}'); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec v \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec v}$$

$$\frac{}{\mathtt{m}; \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\prec \texttt{raise}(v) \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\prec \texttt{raise}(v)}$$

$$\frac{}{\texttt{handle}(\square, x.e_2, \mathcal{L}', \mathcal{G}'); \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\prec \texttt{raise}(v) \longrightarrow_{\mathcal{C}} \mathcal{P} \mid \mathcal{L}' \mid x \leftarrow v; \mathcal{G}' \succ e_2}$$

En estas reglas se observa que para este manejo de excepciones la propagación no es sobre un error sino sobre un valor envuelto en el constructor `raise` y detiene su propagación cuando encuentra un `handle`, si la ejecución del bloque `handle` termina sin problema entonces simplemente se quita ese marco de la pila para continuar con la ejecución del programa.

Semántica estática Se definen las reglas de tipado de los nuevos constructores del lenguaje.

$$\frac{\Gamma \vdash e : \mathsf{S}}{\Gamma \vdash \texttt{raise}(e) : \mathsf{T}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{T} \qquad \Gamma, x : \mathsf{S} \vdash e_2 : \mathsf{T}}{\Gamma \vdash \texttt{handle}(e_1, x.e_2) : \mathsf{T}}$$

Puesto que el lanzamiento de una excepción $\texttt{raise}(e)$ surge en cualquier contexto su tipo deber ser arbitrario. Adicionalmente el valor de excepción puede ser cualquiera así que el tipo de e también es arbitrario y sin relación directa con el anterior.

Observación. Observemos como esta última versión del manejo de excepciones tiene una estructura muy similar a los constructores usados para el manejo de continuaciones en las notas de clase anteriores. Esto no es casualidad y en algunos lenguajes de programación reales las excepciones de este tipo se implementan usando continuaciones las cuales por su naturaleza permiten volver al punto deseado en la ejecución del programa.

4 Seguridad del lenguaje

En la nota de clase anterior se definió la semántica estática de TinyC mediante un conjunto de reglas de tipado para cada uno de los constructores del lenguaje. En esta sección estudiaremos la

seguridad del lenguaje respecto a sus sistema de tipos considerando las extensiones que se definieron en las secciones anteriores. Como se discutió anteriormente al agregar el constructor de error la propiedad de unicidad deja de ser válida pues este programa puede tener más de una derivación de tipos válida según lo requiera el contexto del programa que lanza el error. Si bien esta propiedad es relevante y muy útil en la práctica se puede probar la seguridad de un sistema de tipos sin ella, probando las propiedades de preservación y progreso.

Hasta ahora no hemos relacionado la semántica estática con las máquinas abstractas, sin embargo para las propiedades de preservación y progreso que relacionan los dos niveles semánticos (estático y dinámico) es necesario hacerlo. Lo que requiere de la definición de nuevos juicios, que son los siguientes:

- El estado s de la máquina $\mathcal{C}$ es correcto, denotado como $s\ \ \mathsf{ok}$.
- La pila $\mathcal{P}$ espera un valor de tipo T y se obtiene como resultado final un valor de tipo S, denotado como $\Gamma \vdash \mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S}$. En donde Γ es el contexto de tipado para variables.
- El marco $\mathfrak{m}$ espera un valor de tipo T para calcular un valor de tipo S denotado como $\Gamma \vdash \mathfrak{m} : \mathsf{T} \Rightarrow \mathsf{S}$.
- Sea $\mathcal{E}$ un ambiente de declaraciones, definimos $\mathcal{E}_\mathsf{T}$ como el contexto en el que se almacenan las mismas variables que en $\mathcal{E}$ pero en lugar de guardar su valor almacenan su tipo. Como los ambientes guardan puros valores recuperar el tipo de ellos no es una tarea difícil.

Definición 4.1 (Juicios de tipado para pilas de control)**.** Se definen los juicios anteriores como sigue:

- $$\frac{\Gamma \vdash \mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S} \quad \Gamma \vdash e : \mathsf{T} \quad \Gamma = \mathcal{L}_\mathsf{T} \cup \mathcal{G}_\mathsf{T}}{\Gamma \vdash \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \succ e\ \ \mathsf{ok}}$$
- $$\frac{\Gamma \vdash \mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S} \quad \Gamma \vdash v : \mathsf{T} \quad \Gamma = \mathcal{L}_\mathsf{T} \cup \mathcal{G}_\mathsf{T}}{\Gamma \vdash \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec v\ \ \mathsf{ok}}$$
- $$\frac{}{\Gamma \vdash \blacklozenge : \mathsf{T} \Rightarrow \mathsf{T}}$$
- $$\frac{\Gamma \vdash \mathcal{P} : \mathsf{R} \Rightarrow \mathsf{S} \quad \Gamma \vdash \mathfrak{m} : \mathsf{T} \Rightarrow \mathsf{R}}{\Gamma \vdash \mathfrak{m}; \mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S}}$$
- De igual forma es necesario definir las reglas de tipado para cada uno de los marcos de la máquina $\mathcal{C}$, estas reglas se siguen de la semántica estática de los constructores a los que corresponden. Como ejemplo mostramos la regla correspondiente al marco de `if`.

$$\frac{\Gamma \vdash e_1 : \mathsf{T} \quad \Gamma \vdash e_2 : \mathsf{T}}{\Gamma \vdash \mathtt{if}(\square, e_1, e_2) : \mathsf{Bool} \Rightarrow \mathsf{T}}$$

Proposición 4.2 (Preservación de tipos)**.** Si $s\ \ \mathsf{ok}$ y en la máquina $\mathcal{C}$ existe la transición $s \longrightarrow_\mathcal{C} s'$ entonces $s'\ \ \mathsf{ok}$

Proposición 4.3 (Progreso de la relación $\longrightarrow_\mathcal{C}$)**.** Si $s\ \ \mathsf{ok}$ entonces sucede una y sólo una de las siguientes situaciones:

- s es final, es decir, es de la forma: $\blacklozenge \mid \mathcal{L} \mid \mathcal{G} \prec \bot$.
- Existe s' tal que $s \longrightarrow_{\mathcal{C}} s'$.
- s es un error, es decir, es una de las formas:

$$\mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\!\prec \texttt{error} \qquad\qquad \mathcal{P} \mid \mathcal{L} \mid \mathcal{G} \prec\!\!\!\prec \texttt{raise}(v)$$

Referencias

[1] Miranda Perea F., González Huesca L., Nota de Clase del curso de Lenguajes de Programación, Facultad de Ciencias UNAM, Semestre 2021-1.

[2] Keller G., O'Connor-Davis L., Class Notes from the course Concepts of programming language design, Department of Information and Computing Sciences, Utrecht University, The Netherlands, Fall 2020.

[3] Harper R., Practical Foundations for Programming Languages. Working draft, 2010.

[4] Mitchell J., Foundations for Programming Languages. MIT Press 1996.

[5] Krishnamurthi S., Programming Languages Application and Interpretation; Version 26.04.2007.