Trabajo Práctico Final

ANÁLISIS DE LENGUAJES DE PROGRAMACIÓN

Lucas Salvatore

S-4828/3

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

FACULTAD DE CIENCIAS EXACTAS, INGENIERIA Y AGRIMENSURA

UNIVERSIDAD NACIONAL DE ROSARIO

1. Lenguaje para modelar escape rooms: Escape

1.1. Introducción

Cada vez son más las actividades sociales en donde se sugiere resolver cierto problema siguiendo pistas. Actividades desde escape rooms hasta enigmas grupales en bares, se vuelven interesantes y más populares porque permite, además de pasar un buen rato con amistades y de divertirse, desarrollar la capacidad para la resolución de problemas, el trabajo en equipo, entre otros beneficios para el bienestar y desarrollo de las personas.

En cuanto a los escape rooms, hay versiones virtuales que se modelan visualmente a través de por ejemplo, Unity. La propuesta de este trabajo es generar un DSL textual para facilitar el modelado de la lógica de escape rooms y similares. Se comenzará diseñando el lenguaje base para posteriores extensiones como por ejemplo, detecciones de botones y sensores físicos.

1.2. Gramática del lenguaje

A continuación definiremos la gramática del lenguaje. El mismo consta de tres elementos principales: Identificadores de objetos (*ObjectName*), códigos de desbloqueo (*UnlockCode*) que en nuestro caso lo modelaremos como naturales, y mensajes (*Message*), también *string*.

La gramática en términos de simbolos terminales y no terminales es la siguiente:

- No terminales:
  game-definition, *elements*, *type*, *objectname*, *declarations*,
  unlockcode, *sentences*, *condition*, *command*, *message*

- Terminales:
  $\mathbb{N} \cup \mathbb{S} \cup \{$ **game**, **{**, **}**, **unlock**, **elements**, **onuse**, **:**, **if**, **->**, **show**,
  locked, **unlocked**, **is**, **and**, **or**, (,), **target**, **item** $\}$

- Simbolo Inicial: *game-definition*

Podemos observar en la gramática definiciones individuales de los objetos, elementos de los objetos, sentencias de uso y el código de desbloqueo. Los elementos de los

objetos y sentencias se separan con una coma, las definiciones de objetos simplemente se separan entre sí mediante mínimo un espacio. Se define que esta última definición sea recursiva a izquierda, es decir que asocie a izquierda, mientras que los objetos y sentencias se definan recursivamente a derecha, debido a la complejidad de armar la lista en Haskell.

A continuación, se muestra las producciones de la gramática:

natural ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | *natural natural*

unlockcode ::= *natural*

objectname ::= *identifier*

message ::= *string*

game-definition ::= **game** **{** *elements* **}**
| *type objectname* **{** *declarations* **}**
| *game-definition game-definition*

declarations ::= **unlock :** *unlockcode*
| **elements {** *elements* **}**
| **onuse {** *sentences* **}**
| *declarations declarations*

sentences ::= **if** *condition* **->** *command*
| *command*
| *sentences* **,** *sentences*

command ::= **show** (*objectname* | *message*)

condition ::= **locked**
| **unlocked**
| *objectname* **is locked**
| *objectname* **is unlocked**
| *condition* **and** *condition*
| *condition* **or** *condition*
| **(** *condition* **)**

type ::= **target** | **item**

elements ::= *objectname*
| *elements* **,** *elements*

Esta gramática es ambigua, por lo que definimos precedencia y asociatividad de operadores en el analizador sintáctico:

- Para la producción *game-definition game-definition* la asociatividad se hace

hacia la izquierda. En la gramática del parser se transforma la producción de esta manera:

definition ::= **game** **{** *elements* **}**
| *type objectname* **{** *declarations* **}**
game-definition ::= *game-definition definition*
| *definition*

- Para la producción *declarations declarations* se resuelve de forma idéntica: Se asocia a izquierda transormando la gramática del parser de la siguiente manera:

declaration ::= **unlock :** *unlockcode*
| **elements {** *elements* **}**
| **onuse {** *sentences* **}**
declarations :: *declarations declaration*
| *declaration*

NOTA: La recursión a izquierda genera que el parser pueda no terminar nunca al encontrarse siempre con el mismo no terminal al principio. Happy se encarga de resolverlo automáticamente.

- Para las listas de sentencias y elementos (*sentences* , *sentences* y *elements* , *elements*) se optó por utilizar **recursión a derecha**. Esta decisión se fundamenta en la arquitectura del lenguaje anfitrión (Haskell), donde la construcción de listas mediante el operador *cons* (**:**) es más eficiente y natural de implementar con recursión a derecha, garantizando un orden correcto de los elementos con una complejidad lineal $O(N)$, evitando así el costo computacional de la concatenación o inversión de listas.

- Se define asociatividad a izquierda y precedencia de los operadores donde **and** tiene más precedencia que **or**

1.3. Semántica

1.3.1. Introducción

Usaremos una semántica más orientada a *deep-embedding* donde crearemos los elementos del lenguaje en estructuras y clases definidas en el lenguaje anfitrión. A partir de estas estructuras, se puede traducir y definir estáticamente un programa en términos matemáticos para una mejor comprensión.

1.3.2. Elementos del lenguaje

Definiremos los conjuntos básicos que se van a usar para definir la semántica del lenguaje:

- Definiremos los objetos $O : ObjectName \to Elements \times Sentences \times (\mathbb{N} \cup \{\epsilon\}) \times Type$

  En palabras, tenemos un mapa de objetos que asocia cada nombre de objeto a una tupla que contiene un conjunto de elementos que componen el objeto, un conjunto de sentencias a ejecutar cada vez que se usa el objeto, el código de desbloqueo, donde toma el valor vacío si es un item el objeto, y el tipo de objeto.

- $Elements = \mathcal{P}(ObjectName)$

  A continuación, definiremos inductivamente los conjuntos que contienen los diferentes términos del lenguaje

- Si $o \in ObjectName$, luego $o \in Elements$
  Si $o_1 \in Elements$ y $o_2 \in Elements$ luego $o_1, o_2 \in Elements$

- Si $o \in ObjectName$, luego **show** $o \in Command$
  Si $m \in Message$, luego **show** $m \in Command$

- **locked** $\in Conditions$
  unlocked $\in Conditions$
  Si $o \in ObjectName$, luego o **is locked** $\in Conditions$
  Si $o \in ObjectName$, luego o **is unlocked** $\in Conditions$
  Si $b_1 \in Conditions$ y $b_2 \in Conditions$, luego b_1 **and** $b_2 \in Conditions$

Si $b_1 \in Conditions$ y $b_2 \in Conditions$, luego b_1 **or** $b_2 \in Conditions$

Si $b \in Conditions$, luego **(**b**)** $\in Conditions$

- Si $c \in Command$, luego $c \in Sentences$

  Si $b \in Conditions \wedge c \in Command$, luego **if** b **->** $c \in Sentences$

  Si $s_1 \in Sentences$ y $s_2 \in Sentences$, luego s_1 **,** $s_2 \in Sentences$

- Si $s \in Sentences$, luego **onuse {**s**}** $\in Declarations$

  Si $n \in \mathbb{N}$, luego **unlock** $n \in Declarations$

  Si $e \in Elements$, luego **elements {**e**}** $\in Declarations$

  Si $d_1 \in Declarations$ y $d_2 \in Declarations$, luego $d_1\ d_2 \in Declarations$

- **item** $\in Type$

  target $\in Type$

- Si $e \in Elements$, luego **game {**e**}** $\in GameDefinition$

  Si $t \in Type,\ obj \in ObjectName, decls \in Declarations$, luego $t\ obj$ **{**$decls$**}** $\in GameDefinition$

  Si $d_1 \in GameDefinition$ y $d_1 \in GameDefinition$ luego $d_1\ d_2 \in GameDefinition$

Nuestra semántica la dividiremos en 5 análisis:

1 Análisis estático de los objetos. Analizaremos el AST para armar los objetos con la información en la estructura matemática O definida anteriormente. Acá nos enfocaremos en evitar definiciones duplicadas de **unlock**, **elements** y **onuse**.

2 Análisis estático de ciertas caracteristicas que tienen que cumplir los objetos. Acá nos enfocaremos lo siguiente:

   - Que el **unlock** únicamente aparezca en los objetos target.
   - Que cada objeto esté definido, es decir, pertenezca al dominio del mapa de objetos construido en el punto 1.
   - Que los objetos usados en las sentencias (comandos y condiciones) existan bajo el contexto del objeto contenedor, es decir que estén incluidos en los elementos del objeto contenedor.
   - Que las condiciones **locked** y **unlocked** se usen en los objetos target.

- Que el tipo del objeto *obj* sea objetivo (**target)** para ser usadas en las condiciones *obj* **is locked** y *obj* **is unlocked**
- Que los objetivos tengan un statement **unlock**

3 Análisis y evaluación dinámica de las condiciones de las sentencias. Acá introducimos el concepto de estado, que incluye el estado de los objetivos (desbloqueado/bloqueado), la pila de navegación de los objetos durante el juego y un buffer de salida que en Haskell será representado por la mónada IO.

4 Análisis y ejecución dinámica de las sentencias de los objetos. Para evaluar el segmento **onuse**, necesitaremos unas reglas small-step para definir cómo se ejecutan.

5 Análisis y evaluación de dinámica del estado en función de la entrada del usuario. El juego en tiempo de ejecución. En este punto entra la interacción con el usuario, por lo que se requerirá una notación para representar la entrada del usuario

1.3.3. PUNTO 1: Semántica denotacional para especificar la recolección de objetos y sus tipos

La recolección de objetos y los tipos implica armar el entorno del juego en el cual nos basaremos para la búsqueda de objetos y las validaciones. Definimos las siguientes funciones de evaluación:

$$[\![]\!] : \mathit{game\text{-}definition} \to O$$

$$[\![]\!]_d : \mathit{declarations} \to Type \to Elements \times Sentences \times (\mathbb{N} \cup \{\epsilon\}) \times Type$$

donde notamos $[\![]\!]_d\ decl\ t$ como $[\![decl]\!]_d^t$

$$[\![]\!]_e : \mathit{elements} \to Elements$$

- $[\![\textbf{game } \{e\}]\!] = \{\texttt{game} \mapsto [\![\textbf{elements } \{\ e\ \}]\!]_d^{\textbf{item}}\}$
- $[\![\textbf{item } objectname\ \{decs\}]\!] = \{objectname \mapsto [\![decs]\!]_d^{\textbf{item}}\}$
- $[\![\textbf{target } objectname\ \{decs\}]\!] = \{objectname \mapsto [\![decs]\!]_d^{\textbf{target}}\}$

- Si $O_1 = [\![gdf_1]\!], O_2 = [\![gdf_2]\!]$

$$[\![gdf_1\ gdf_2]\!] = \begin{cases} O_1 \ \cup \ O_2 & \text{si } dom(O_1) \cap dom(O_2) = \emptyset \\ \texttt{error} & \text{caso contrario (definición de objeto duplicado)} \end{cases}$$

- $[\![\textbf{unlock :}\ n]\!]^t_d = (\emptyset, [], n, t)$

- $[\![\textbf{elements \{}\ e\ \textbf{\}}]\!]^t_d = ([\![e]\!]_e, [], \epsilon, t)$

- $[\![\textbf{onuse \{}\ s\ \textbf{\}}]\!]^t_d = (\emptyset, s, \epsilon, t)$

- Si $(E_1, S_1, n_1) = [\![d_1]\!]^t_d, (E_2, S_2, n_2) = [\![d_2]\!]^t_d$

$$[\![d_1\ d_2]\!]^t_d = \begin{cases} (E_1 \cup E_2, S_1 +\!\!+ S_2, max(n_1, n_2)) & \text{si } (E_1 = \emptyset \ \vee \ E_2 = \emptyset) \\ & \wedge\ (S_1 = [] \ \vee \ S_2 = []) \\ & \wedge\ (n_1 = \epsilon \ \vee \ n_2 = \epsilon) \\ \texttt{error} & \text{caso contrario (declaración duplicada)} \end{cases}$$

- $[\![objectname]\!]_e = \{objectname\}$

- $[\![e_1\ \textbf{,}\ e_2]\!]_e = [\![e_1]\!]_e \ \cup \ [\![e_2]\!]_e$

1.3.4. PUNTO 2: Validación de Expresiones

Definamos todos los términos del lenguaje como la unión: $Terminos = Elements \cup Command \cup Conditions \cup Sentences \cup Declarations \cup GameDefinition \cup Type$
Definimos el predicado de validez $\mathcal{V} : O \to ObjectName \to Terminos \to \mathbb{B}$ donde $\mathbb{B}$ son los booleanos, que determina si una construcción es semánticamente correcta respecto al entorno de objetos O y al objeto contexto:

$$\mathcal{V}^{f,o}(\textbf{game } \{e\}) = \mathcal{V}^{f,o}(e)$$

$$\mathcal{V}^{f,o}(\textbf{item } objectname\ \{decs\}) = \mathcal{V}^{f,objectname}(decs)$$

$$\mathcal{V}^{f,o}(\textbf{target } objectname\ \{decs\}) = \mathcal{V}^{f,objectname}(decs)$$

$$\mathcal{V}^{f,o}(d_1\ d_2) = \mathcal{V}^{f,o}(d_1) \wedge \mathcal{V}^{f,o}(d_2)$$

$$\text{Si } f(o) = (E,S,n,t) \quad , \quad \mathcal{V}^{f,o}(\textbf{unlock : } n) = \begin{cases} \texttt{true} & \text{si } o \in dom(f) \wedge t = \textbf{target} \\ \texttt{false} & \text{caso contrario (no existe o no es target)} \end{cases}$$

$$\mathcal{V}^{f,o}(\textbf{elements \{ } e \textbf{ \}}) = \mathcal{V}^{f,o}(e)$$

$$\mathcal{V}^{f,o}(objectname) = (objectname \in dom(f)) \quad \text{Definición de objeto}$$

$$\mathcal{V}^{f,o}(e_1 \textbf{ , } e_2) = \mathcal{V}^{f,o}(e_1) \wedge \mathcal{V}^{f,o}(e_2)$$

$$\mathcal{V}^{f,o}(\textbf{onuse \{ } s \textbf{ \}}) = \mathcal{V}^{f,o}(s)$$

$$\text{Si } f(o) = (E,S,n,t) \quad , \quad \mathcal{V}^{f,o}(\textbf{show } x) = \begin{cases} \texttt{true} & \text{si } x \text{ es un string} \\ \texttt{true} & \text{si } x \in ObjectName \wedge x \in dom(f) \wedge x \in E \\ \texttt{false} & \text{si } x \in ObjectName \wedge x \notin E \text{ (No es un elemento del objeto)} \\ \texttt{false} & \text{(Referencia Desconocida)} \end{cases}$$

$$\mathcal{V}^{f,o}(\textbf{if } c \textbf{ ->} cmd) = \mathcal{V}^{f,o}(c) \wedge \mathcal{V}^{f,o}(cmd)$$

$$\mathcal{V}^{f,o}(s_1 \textbf{ , } s_2) = \mathcal{V}^{f,o}(s_1) \wedge \mathcal{V}^{f,o}(s_2)$$

$$\text{Si } f(o) = (E,S,n,t) \quad , \quad \mathcal{V}^{f,o}(\textbf{locked}) = \begin{cases} \texttt{true} & \text{si } t = \textbf{target} \\ \texttt{false} & \text{caso contrario (no es target)} \end{cases}$$

$$\text{Si } f(o) = (E,S,n,t) \quad , \quad \mathcal{V}^{f,o}(\textbf{unlocked}) = \begin{cases} \texttt{true} & \text{si } t = \textbf{target} \\ \texttt{false} & \text{caso contrario (no es target)} \end{cases}$$

$$\text{Si } f(o) = (E,S,n,t) \quad , \quad \mathcal{V}^{f,o}(obj \textbf{ is locked}) = \begin{cases} \texttt{true} & \text{si } obj \in dom(f) \wedge t = \textbf{target} \wedge obj \in E \\ \texttt{false} & \text{caso contrario (no existe o no es target o no es un elemento del objeto)} \end{cases}$$

$$\text{Si } f(o) = (E,S,n,t) \quad , \quad \mathcal{V}^{f,o}(obj \textbf{ is unlocked}) = \begin{cases} \texttt{true} & \text{si } obj \in dom(f) \wedge t = \textbf{target} \wedge obj \in E \\ \texttt{false} & \text{caso contrario (no existe o no es target)} \end{cases}$$

$$\mathcal{V}^{f,o}(c_1 \textbf{ and } c_2) = \mathcal{V}^{f,o}(c_1) \wedge \mathcal{V}^{f,o}(c_2)$$

$$\mathcal{V}^{f,o}(c_1 \textbf{ or } c_2) = \mathcal{V}^{f,o}(c_1) \wedge \mathcal{V}^{f,o}(c_2)$$

1.3.5. PUNTO 3: Semántica denotacional para las condiciones

Sea un entorno de objetos $f \in \mathcal{O}$, definimos el conjunto de objetivos como

$$\Theta = \{o \in dom(f) \ / \ \pi_4(f(o)) = \textbf{target}\}$$

Luego definimos el conjunto Σ como el conjunto de todos los estados σ donde

$$\sigma = (\mathcal{L}, \mathbf{O}, Out)$$

donde:

- $\mathcal{L} : \Theta \to \{\texttt{locked}, \texttt{unlocked}\}$ es el mapa de bloqueos.
- $\mathbf{O}$ es la pila de navegación de objetos donde $\texttt{top}(\mathbf{O})$ es el objeto actual en cuestión
- $Out = List(String \cup ObjectName)$ es una lista de salida para los mensajes generados por `show`

El estado inicial es $\sigma_0 = (\mathcal{L}_0, \mathbf{O}_0, Out_0)$ donde

- $\mathcal{L}_0 = \{t \mapsto \texttt{locked} \mid \forall t \in \Theta\}$
- $\mathbf{O}_0 = [\texttt{game}]$
- $Out_0 = []$

Primero definimos la función $\mathcal{B}[\![]\!] : \Sigma \to Conditions \to \mathbb{B}$ que evaluan las condiciones como

- $\mathcal{B}[\![obj \textbf{ is locked}]\!]_{(\mathcal{L},\mathbf{O},Out)} = \mathcal{L}(obj) == \texttt{locked}$
- $\mathcal{B}[\![obj \textbf{ is unlocked}]\!]_{(\mathcal{L},\mathbf{O},Out)} = \mathcal{L}(obj) = \texttt{locked}$
- $\mathcal{B}[\![\textbf{locked}]\!]_{(\mathcal{L},\mathbf{O},Out)} = \mathcal{B}[\![\texttt{top}(\mathbf{O})\ \texttt{is locked}]\!]$
- $\mathcal{B}[\![\textbf{unlocked}]\!]_{(\mathcal{L},\mathbf{O},Out)} = \mathcal{B}[\![\texttt{top}(\mathbf{O})\ \texttt{is unlocked}]\!]$
- $\mathcal{B}[\![b_1\ \texttt{and}\ b_2]\!]_{(\mathcal{L},\mathbf{O},Out)} = \mathcal{B}[\![b_1]\!]_{(\mathcal{L},\mathbf{O},Out)} \wedge \mathcal{B}[\![b_2]\!]_{(\mathcal{L},\mathbf{O},Out)}$
- $\mathcal{B}[\![b_1\ \texttt{or}\ b_2]\!]_{(\mathcal{L},\mathbf{O},Out)} = \mathcal{B}[\![b_1]\!]_{(\mathcal{L},\mathbf{O},Out)} \vee \mathcal{B}[\![b_2]\!]_{(\mathcal{L},\mathbf{O},Out)}$

1.3.6. PUNTO 4: Semántica operacional small-step para las sentencias

Para la ejecución de las sentencias, usaremos la semántica operacional small-step. Definimos la relación de transición $\rightarrow_S : Sentences \times \Sigma \rightarrow Sentences \times \Sigma$ como sigue:

$$\frac{\sigma = (\mathcal{L}, \mathbf{O}, Out) \quad \mathcal{B}[\![b]\!]_\sigma = \texttt{true}}{\langle (\textbf{if } b \textbf{ ->} cmd) :: S_{tail}, \sigma \rangle \rightarrow_S \langle cmd :: S_{tail}, \sigma \rangle} \text{ SS-If-True}$$

$$\frac{\sigma = (\mathcal{L}, \mathbf{O}, Out) \quad \mathcal{B}[\![b]\!]_\sigma = \texttt{false}}{\langle (\textbf{if } b \textbf{ ->} cmd) :: S_{tail}, \sigma \rangle \rightarrow_S \langle S_{tail}, \sigma \rangle} \text{ SS-If-False}$$

$$\frac{\sigma = (\mathcal{L}, \mathbf{O}, Out)}{\langle (\textbf{show } x) :: S_{tail}, \sigma \rangle \rightarrow_S \langle S_{tail}, (\mathcal{L}, \mathbf{O}, Out \mathbin{+\!\!+} [\text{str}(x)]) \rangle} \text{ SS-Cmd}$$

Observemos que la ejecución termina si llega a un estado de la forma $\langle [], \sigma \rangle$ para algún σ, es decir que la lista de sentencias fué consumida en su totalidad.

1.3.7. PUNTO 5: Semántica operacional small-step para los comandos que tiene que realizar el usuario

Definimos la gramática para los comandos que tiene que realizar el usuario:

user_input ::= **unlock** *unlockcode*
| **use**
| **select** *objectname*
| **back**

Sea $f \in O$ un mapa de objetos. Primero necesitamos funciones para poder lller el mapa de objetos f:

elements$(f, obj) = E$ donde $f(obj) = (E, \_, \_, \_)$

sentences$(f, obj) = S$ donde $f(obj) = (\_, S, \_, \_)$

code$(f, obj) = n$ donde $f(obj) = (\_, \_, n, \_)$

objecttype$(f, obj) = t$ donde $f(obj) = (\_, \_, \_, t)$

Para las transiciones, necesitamos que el usuario ingrese un comando, es decir que ingrese un *user_input*. Para esto definimos la transición $\rightarrow_U$: $\Sigma \rightarrow user\_input \rightarrow \Sigma$ para los comandos del usuario, donde $(\sigma, cmd, \sigma') \in \rightarrow_U$ si σ' es obtenido luego de ejecutar el comando *cmd* en el estado σ, y lo notamos $\sigma \overset{cmd}{\rightarrow}_U \sigma'$.

$$\frac{c = \texttt{top}(\mathbf{O}) \quad E = \text{elements}(f, c) \quad obj \in E}{(\mathcal{L}, \mathbf{O}, Out) \overset{\texttt{select}\ obj}{\rightarrow}_U (\mathcal{L}, \texttt{push}(\mathbf{O}, obj), Out)}\ \textsc{Select-OK}$$

$$\frac{c = \texttt{top}(\mathbf{O}) \quad E = \text{elements}(f, c) \quad obj \notin E}{(\mathcal{L}, \mathbf{O}, Out) \overset{\texttt{select}\ obj}{\rightarrow}_U (\mathcal{L}, \texttt{push}(\mathbf{O}, obj), Out])}\ \textsc{Select-Fail}$$

$$\frac{c = \texttt{top}(\mathbf{O}) \quad \text{objecttype}(f, c) = \textbf{target} \quad n_c = \text{code}(f, c) \quad n = n_c}{(\mathcal{L}, \mathbf{O}, Out) \overset{\texttt{unlock}\ n}{\rightarrow}_U (\mathcal{L} \oplus \{c \mapsto \texttt{unlocked}\}, \mathbf{O}, Out)}\ \textsc{Unlock-Success}$$

$$\frac{c = \texttt{top}(\mathbf{O}) \quad \text{objecttype}(f, c) \neq \textbf{target}}{(\mathcal{L}, \mathbf{O}, Out) \overset{\texttt{unlock}\ n}{\rightarrow}_U (\mathcal{L}, \mathbf{O}, Out)}\ \textsc{Unlock-Not-Target}$$

$$\frac{c = \texttt{top}(\mathbf{O}) \quad \text{objecttype}(f, c) = \textbf{target} \quad n_c = \text{code}(f, c) \quad n \neq n_c}{(\mathcal{L}, \mathbf{O}, Out) \overset{\texttt{unlock}\ n}{\rightarrow}_U (\mathcal{L}, \mathbf{O}, Out)}\ \textsc{Unlock-Fail}$$

$$\frac{|\mathbf{O}| > 1}{(\mathcal{L}, \mathbf{O}, Out) \overset{\texttt{back}}{\rightarrow}_U (\mathcal{L}, \texttt{pop}(\mathbf{O}), Out)}\ \textsc{Back-OK}$$

$$\frac{|\mathbf{O}| \leq 1}{(\mathcal{L}, \mathbf{O}, Out) \xrightarrow{\texttt{back}}_U (\mathcal{L}, \mathbf{O}, Out)} \textsc{Back-Fail}$$

$$\frac{c = \texttt{top}(\mathbf{O}) \quad S = \text{sentences}(f, c) \quad \langle S, (\mathcal{L}, \mathbf{O}, \text{Out})\rangle \rightarrow_S^* \langle [], (\mathcal{L}, \mathbf{O}, \text{Out}')\rangle}{(\mathcal{L}, \mathbf{O}, \text{Out}) \xrightarrow{\texttt{use}}_U (\mathcal{L}, \mathbf{O}, \text{Out}')} \textsc{Use-Current}$$

Hay varias combinaciones de estados finales. Un estado final $\sigma_!$ tiene que cumplir que lo siguiente:

$$\text{Si } \sigma_! = (\mathcal{L}, \mathbf{O}, Out) \text{ luego } \forall obj \in dom(\mathcal{L}), \mathcal{L}(obj) = \texttt{unlocked}$$

1.4. Ejemplo

Se adjunta como ejemplo el siguiente código

```
1
2  game {
3      door,
4      oldbook,
5      box
6  }
7
8  target door {
9      unlock: 1234
10 }
11
12 target oldbook {
13    onuse {
14       if locked -> show "Charles was born on /|/",
15       if unlocked -> show "Charles was born on 1234"
16    }
```

```
17
18    unlock: 232
19 }
20
21 item box {
22     elements {
23        scraper
24     }
25
26     onuse {
27       show scraper
28     }
29 }
30
31 item scraper {
32      onuse {
33         show "use this number to unlock an object: 232"
34      }
35 }
```

Donde tenemos el mapa de objetos $f \in O$ como

$$
\begin{aligned}
f = \{\ & \texttt{game} \mapsto (\{\texttt{door}, \texttt{oldbook}, \texttt{box}\}, [], \epsilon, \textbf{item}), \\
& \texttt{box} \mapsto (\{\texttt{scraper}\}, [\textbf{show}\ \texttt{scraper}], \epsilon, \textbf{item}), \\
& \texttt{oldbook} \mapsto (\emptyset, [s_1, s_2], 232, \textbf{target}), \\
& \texttt{door} \mapsto (\emptyset, [], 1234, \textbf{target}), \\
& \texttt{scraper} \mapsto (\emptyset, [s_3], \epsilon, \textbf{item})\ \}
\end{aligned}
$$

y donde el estado inicial es:

$$\sigma_0 = (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{game}], [])$$

La traza de ejecución es la siguiente:

$$\sigma_0 \xrightarrow{\texttt{select}\ \textit{oldbook}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{oldbook}, \texttt{game}], [])$$

$$\xrightarrow{\textit{use}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{oldbook}, \texttt{game}],$$

$$[\texttt{Charles was born on /|/}])$$

$$\xrightarrow{\textit{back}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{game}],$$

$$[\texttt{Charles was born on /|/}])$$

$$\xrightarrow{\texttt{select}\ \textit{box}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{box}, \texttt{game}],$$

$$[\texttt{Charles was born on /|/}])$$

$$\xrightarrow{\textit{use}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{box}, \texttt{game}],$$

$$[\textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\texttt{select}\ \textit{scraper}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{scraper}, \texttt{box}, \texttt{game}],$$

$$[\textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\textit{use}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{scraper}, \texttt{box}, \texttt{game}],$$

$$[\texttt{use this number to unlock an object: 232}, \textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\textit{back}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{box}, \texttt{game}],$$

$$[\texttt{use this number to unlock an object: 232}, \textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\textit{back}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{game}],$$

$$[\texttt{use this number to unlock an object: 232}, \textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\textit{back}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{game}],$$

$$[\texttt{use this number to unlock an object: 232}, \textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\texttt{select}\ \textit{oldbook}}_U (\{\texttt{oldbook} \mapsto \texttt{locked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{oldbook}, \texttt{game}],$$

$$[\texttt{use this number to unlock an object: 232}, \textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\texttt{unlock}\ 232}_U (\{\texttt{oldbook} \mapsto \texttt{unlocked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{oldbook}, \texttt{game}],$$

$$[\texttt{use this number to unlock an object: 232}, \textit{scraper}, \texttt{Charles was born on /|/}])$$

$$\xrightarrow{\textit{use}}_U (\{\texttt{oldbook} \mapsto \texttt{unlocked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{oldbook}, \texttt{game}],$$

$$[\texttt{Charles was born on 1234}, \texttt{use this number to unlock an object: 232}, \ldots])$$

$$\xrightarrow{\textit{back}}_U (\{\texttt{oldbook} \mapsto \texttt{unlocked}, \texttt{door} \mapsto \texttt{locked}\}, [\texttt{game}],$$

$$[\texttt{Charles was born on 1234}, \texttt{use this number to unlock an object: 232}, \ldots])$$

$\xrightarrow{\texttt{select } door}_U$ ({oldbook ↦ unlocked, door ↦ locked}, [door, game],

[Charles was born on 1234, use this number to unlock an object: 232, ...])

$\xrightarrow{\texttt{unlock } 1234}_U$ ({oldbook ↦ unlocked, door ↦ unlocked}, [door, game],

[Charles was born on 1234, use this number to unlock an object: 232, ...])

A partir de aquí, la traza de ejecución termina y en el interprete desarrollado en Haskell termina el programa una vez que todos los objetos son desbloqueados. Se omitió detalles de la ejecución de las sentencias, pero cada vez que se ejecuta el *use*, se ejecutan todas las sentencias especificadas en cada objeto.

1.5. Archivos de ejemplo

En el código fuente de Haskell se encuentran varios archivos en los cuales representamos las principales características del lenguaje. Nombraremos los archivos de los casos de errores,

- Que el **unlock** únicamente aparezca en los objetos target.
  En el archivo `1.escape`
- Que cada objeto esté definido, es decir, pertenezca al dominio del mapa de objetos construido en el punto 1.
  En el archivo `2.escape`
- Que los objetos usados en las sentencias (comandos y condiciones) existan bajo el contexto del objeto contenedor, es decir que estén incluidos en los elementos del objeto contenedor.
  En el archivo `3.escape`
- Que las condiciones **locked** y **unlocked** se usen en los objetos target.
  En el archivo `4.escape`
- Que el tipo del objeto *obj* sea objetivo (**target)** para ser usadas en las condiciones *obj* **is locked** y *obj* **is unlocked**
  En el archivo `5.escape`

- Que los objetivos tengan un statement **unlock**

  En el archivo `6.escape`

- Declaración múltiple de sentencias:

  En los archivos `7.escape`, `8.escape` y `9.escape`

1.6. Extensiones y Mejoras

- Posibilidad de interacción entre dos objetos. Podemos incorporar desbloqueos de objetos si se relacionan con otros objetos, por ejemplo *scraper* y *wall*. Se podría agregar un comando de usuario donde el usuario quiere usar el *scraper* sobre el *wall* haciendo **use** *scraper* **with** *wall*.

- Podemos incorporar desbloqueos de objetos si estos tienen determinados objetos. Para esto podemos agregar el concepto de inventario, donde podemos extraer objetos, agregarlos al inventario, y usarlos en determinados casos. Sin embargo debemos debilitar las restricciones mencionadas en el punto 2, debido a que estaríamos haciendo dinámico el conjunto de elementos de cada objeto, donde pasaría a ser parte del estado del juego en lugar de un modelo estático.

- Un comando nuevo que desbloquee objetos. En general se puede extender a comandos nuevos que modifiquen el estado.

- Manejo de dispositivos físicos como pad numéricos los cuales envíen señales cada vez que se presiona un código. La modularización del proyecto permitiría que la sentencia **unlock** *n* lo haga. Se cambiaría la mónada *IO* por una mónada que capture eventos de dispositivos