- Al salir del bloque "se pierden" los procedimientos declarados en él, "recuperandose" el acceso a posibles "redefiniciones" más externas de los mismos.
- Observese que en la semántica el entorno env_p, "necesario" para la ejecución, "puede ser cualquiera". Esto "significa" que por lo general "el significado" definido va mucho más allá de lo "esperado", pues la semántica de cualquier instrucción que contenga llamadas a procedimientos nos indica "lo que haría", sea cual sea el cuerpo en la "especificación" de los mismos.
- Observese que la definición "aparentemente sencilla" de la semántica con "alcance dinámico" precisa intrinsecamente la generalidad comentada en el punto anterior. En otras palabras, no tendría sentido pediros que "generalizaseis" de nuevo el Ejercicio 3, para definir "el significado esperado" de un procedimiento, en cuanto contenga llamadas a otros, pues el efecto de éstas dependerá del "punto" desde el que se haya hecho la llamada al procedimiento que los contiene.

Ejercicio 5: Definir un "programa" (o sea, un bloque "complejo") que contenga un procedimiento cuyo cuerpo incluya llamadas a otros procedimientos "todavía desconocidos". Completar la definición del programa incluyendo los elementos que hagan que su cuerpo, incluyendo una llamada al procedimiento en estudio, funcione "correctamente".

Ejercicio 6: a) Indicar qué limitaciones os deberíais poner al programar con un lenguaje con alcance dinámico para que los significados de los procedimientos no se vean afectados nunca por los "efectos perversos" del alcance dinámico discutidos más arriba.

b) La principal utilidad del alcance dinámico, utilizando precisamente los efectos que lo diferencian del estático, es conseguir la "genericidad" que

se refleja precisamente en la definición de su semántica. En cierta forma, los nombres de procedimientos invocados en el cuerpo de un procedimiento funcionan como "parámetros formales", que se instanciarán su significado en el punto en que se llama a este último. Esto corresponde en buena medida a lo que perseguimos al utilizar un lenguaje orientado a objetos. Reflexionar sobre ello, y buscar algún ejemplo sencillo que ilustre adecuadamente la situación.

- Observar que con alcance dinámico la recursión funciona "automáticamente", pues si hemos invocado a un procedimiento ha sido porque su nombre aparecía asociado a su cuerpo en el entorno de procedimientos vigente en el momento en que se le llamó. Y allí seguirá si tenemos una llamada recursiva dentro de su cuerpo. Naturalmente, podríamos hablar de una excepción que se produciría si en el cuerpo aparece un bloque "anidado" en el que se nos ocurre la bastante perversa idea de "volver a declarar" ese procedimiento (o sea, declarar otro con su mismo nombre. En tal caso, para cuando llegáramos a una llamada al mismo dentro del cuerpo interno estaríamos llamando al "nuevo", sin poder hacerse llamadas recursivas al antiguo hasta que terminara la ejecución del bloque interno.

Alcance estático para los procedimientos

- Para eliminar la genericidad observada antes hemos de definir la semántica de los procedimientos de manera que al introducirlos quede cerrado su significado, incluyendo el de todos los elementos que invoque. Ello incluye (eventualmente) procedimientos y variables, pero por el momento nos ocupamos sólo de los primeros.
- La solución es bastante natural: si al ligar al nombre de un procedimiento sólo su cuerpo, dejábamos "abierto" el significado

de las llamadas que pudiera incluir, ahora los "cerramos" añadiendo en el "significado" del procedimiento a su cuerpo el entorno vigente en el momento en que se declara. Ahora "sacaremos" de ese significado el de toda llamada a otro procedimiento que se realice en su cuerpo, sin importarnos en absoluto lo que pudiera indicarse en el entorno vigente en el punto en que se realizó el "primer" procedimiento.

- Observesé que la regla [callns] "ordinaria" nunca podría generar llamadas recursivas, ya que trata de reflejar un significado 100% composicional. Una llamada "recursiva" aparece "dentro" del procedimiento al que queremos dar significado con el entorno que "ya" tenemos al ir a declarlo, pero éste no contendrá "todavía" el significado del propio procedimiento, que "estamos" generando. Así que con esta regla la llamada correspondería siempre a un hipotético procedimiento con el mismo nombre ya "previamente definido" en un bloque más externo, en el que se anide la declaración actual.
- Parecería inevitable algún sofisticado mecanismo de "auto-referencia" (que en realidad se podría conseguir emulando la "generación" de una lista infinita de valores iguales, con un "puntero" cíclico que apunta al propio nodo donde aparece), pero en su lugar podemos utilizar el "truco" de "recopiar" el entorno en que se "generó" la llamada "inicial" al procedimiento que ahora pretendemos que sea recursivo. Eso sí, sólo copiamos la entrada correspondiente al procedimiento llamado, pues en caso contrario volveríamos a tener alcance dinámico.

Ejercicio 7: Indicar cómo deberíamos proceder para conseguir que algunos procedimientos (con llamadas "a sí mismos") sean recursivos y otros no, dependiendo de "cómo los declaremos".

Alcance estático para las variables

- En un principio podría parecer que para lograr que el acceso a las variables pase a ser también dinámico basta "copiar" lo que hemos hecho con los procedimientos, pero un análisis más detallado nos indica que ello no será suficiente. El acceso dinámico "congela" el valor de las variables declaradas en un bloque cuando éste comienza a ejecutarse. Durante su ejecución accedemos (¡y modificamos!) el valor en el "renovado" estado. Sin embargo, en lo que se refiere a las variables no "redeclaradas", el efecto es que sigue accediendo (¡y modificando!) su valor en el estado de partida.
- Se genera así una "pila virtual" para la gestión de todas las variables, con acceso en cada momento a las cimas, y apilamientos y desapilamientos sólo de un elemento en cada variable declarada en cada bloque, cuando entramos y salimos de él, cada vez que ello suceda durante una ejecución.
- Para definir el alcance estático hemos de asociar a cada procedimiento un entorno fijo que nos defina "el valor" de partida de cada variable cuando vayamos a ejecutarlo. Pero ese valor fijo no puede ser el "valor numérico" de la susodicha variable, que podría variar llamada a llamada.
- Surge así la necesidad de introducir un intermediario entre la variable y su valor numérico. La forma "física" en que se materializa todo esto en las implementaciones de los lenguajes imperativos nos ofrece la solución: hemos de introducir una "memoria abstracta" cuyas direcciones (Loc) nos facilitarán en efecto la solución a nuestro problema. Ello es así porque la dirección que asignemos a cada variable al crearla, permanecerá constante durante toda su existencia. En cambio, podremos ir variando dinámicamente el valor "contenido" en la misma, representando satisfactoriamente la "variabilidad" de las variables.

- En definitiva, la semántica ahora maneja como argumento "principal" la memoria (sto ∈ Store), y como "auxiliares" los entornos de procedimientos y variables, que nos indican el significado de estos "vigente" cuando nos encontremos a éstas últimas, o llamadas a los primeros.

Ejercicio 8 : Versionar el Ejercicio 4 definiendo una categoría adecuada de programa, que sólo podrá manejar (inicialmente) las variables y procedimientos definidos en su "cabecera".

Ejercicio 9 : Presentar algún ejemplo sencillo, pero al tiempo suficientemente representativo que ilustre el hecho de que las tres semánticas estudiadas producirán en general significados radicalmente diferentes.

Ejercicio 10 : Extender el lenguaje con procedimientos con un parámetro "por variable", que se deberá "unificar" con la variable con la que se haga cada llamada al mismo, durante toda la ejecución del cuerpo del procedimiento.

Pequeño proyecto : Incorporar al lenguaje "funciones definidas por el programa" al uso, dotándolas para empezar de sólo un argumento, manejado "por valor". Introducirlas de la forma adecuada en cada uno de los tres marcos semánticos que hemos desarrollado.

Indicación : Intentar concebir una "simulación" de las funciones y sus llamadas utilizando bloques "ordinarios" y llamadas a procedimientos (que ni siquiera necesitarán utilizar parámetros explícitos de ningún tipo), y a partir de la misma tratar de "sintetizar" las definiciones explícitas que se piden.

Observación / recomendación : Naturalmente, resultaría ideal que quien vaya a afrontar la realización de los ejercicios más complejos contenidos en estos "apuntes complementarios", lo intente primero por sí mismo, pero si en principio no se obtuvieran resultados, lo más operativo sería buscar en textos más avanzados que el que estamos utilizando como texto.