USO DE HEURISTICOS EN COMPILACION DE UN LENGUAJE DE ALTO NIVEL PARA MAQUINADO EXTERNO EN TORNOS DE CONTROL NUMERICO.

OSCAR EDUARDO RUIZ SALGUERO
RODRIGO LOPEZ BELTRAN

GRUPO DE INVESTIGACION DFAC
FACULTAD DE INGENIERIA
UNIVERSIDAD DE LOS ANDES
APARTADO AEREO 4976-BOGOTA,COLOMBIA

RESUMEN: En la programación de máquinas herramientas de comando numerico (CNC) se usa un lenguaje de bajo nivel que es fijado por normas internacionales. La programación de una pieza a ser realizada en este lenguaje es tediosa y propensa al error, por lo cual se han desarrollado lenguajes de alto nivel que liberen al programador de este esfuerzo. Partiendo de una gramática subconjunto de uno de ellos (APT: Automatically Programmed Tools) suficiente para una descripción de geometrías maquinables por torneado externo, se contruyó un compilador , el cual uso en su módulo de generación de código heurísticos que toman en cuenta la geometría de la pieza y de las herramientas involucradas. Este prototipo, construído dentro de un programa a largo plazo del grupo DFAC (Diseño y Fabricación Apoyados por Computador), para desarrollo de software dirigido a apoyar las labores productivas en la industria, contó con la colaboración de los ingenieros José T. Hernández y Alberto García en las etapas de diseño y realización. Este artículo informa sobre las experiencias y espectativas que dicho proyecto representa.

Palabras Clave: Programación CNC, APT.

1. ANTECEDENTES.

Las máquinas herramientas de comando numérico (máquinas CNC) trabajan con una entrada de bajo nivel especificada según normas de la EIA (Electronic Industries Asociation). Tal entrada como todo lenguaje de bajo nivel implica riesgo de error en la programacion y, por el volumen de cálculos implicado, la labor de programación se hace más tediosa y propensa al error. Lo anterior llevó, en la década de los 50's a desarrollar, en MIT, un primer propotipo de APT (Automatically Programmed Tools), un lenguaje de alto nivel para programación de maquinas CNC. Actualmente existe una gran cantidad de tales lenguajes, muchos desarrollados por las propias industrias que describen movimientos en tres dimensiones, y cuyo procesamiento implica validaciones sintácticas y geométricas en una primera aproximación, una posterior generación de un codigo intermedio no adecuado para máquinas CNC reales, y por último, una adecuación de dicho codigo por medio de un post-procesamiento, a una máquina herramienta especifica.

Por la extensión y complejidad del lenguaje y las labores de validación, los ambientes de programación CNC usualmente requieren computadores grandes, lo cual, al lado de la máquina CNC misma, representa altos costos, no accesibles a las empresas del medio colombiano. Una simplificación del problema, con un subconjunto de APT, ha sido abordado en la Universidad de Los Andes, buscando la creación de una herramienta de programación CNC barata y eficaz teniendo en cuenta las necesidades de la industria nacional. Tal herramienta está concebida como ayuda geométrica a la labor de programación CNC en torneado externo. Para simular el uso de diferentes tornos se usa el programa SITOR[1] igualmente construido en la Univesidad de Los Andes, sobre el cual se ejecuta el código producido por el compilador

Este compilador presenta algunas particularidades debidas a las caracteristicas impuestas por el ambiente de programación CNC. Como compilador, podemos decir que sus rutinas de generacion de código no siguen el esquema clásico estático, en el cual, dado un programa fuente y un compilador, hay una sola salida posible como código intermedio o como codigo final[2]. En dichos esquemas clásicos, la generación de un código intermedio es automática, de acuerdo a las instrucciones del programa fuente. Por ejemplo, una instrucción de asignacion, luego de validaciones sintácticas y (en algunos casos) de tipos es traducida en un código intermedio en el

---

[1] El Simulador Gráfico de Tornos de Control Numérico: Una Herramienta de Aprendizaje. Memo de Investigación Nº 16, Diciembre de 1986, Centro de Investigaciones de la Facultad de Ingeniería (CIFI).

[2] Aún si se considera la labor de optimizacion de código, ésta es ejecutada en una forma estática, y funciona casi como un traductor.

cual se expresa la asignación como una transferencia de contenidos entre direcciones en la memoria del computador. y, una posterior pasada, traduce ese codigo intermedio hacia una forma ejecutable en una máquina particular.

A diferencia de ello, en este caso el módulo generador del compilador debe utilizar heuristicos para decidir el código a generar, estando dichos heuristicos directamente relacionados con las caracteristicas geometricas de la pieza a maquinar, y con las herramientas disponibles para ello.

Siendo una herramienta que permite simular las variables geométricas del proceso de torneado, y cuyo código producido se ejecuta sobre un simulador de máquinas virtuales, es posible reducir a dos las tres etapas anteriormente descritas involucradas en programación CNC. Se divide, por lo tanto, el proceso de compilación en dos módulos. El primero de validación sintactica y geometrica, cuyo salida es un conjunto de órdenes de maquinado de alto nivel, que representan la descomposición del problema original en sectores más elementales. El segundo de generacion, aborda en un orden heuristicamente "bueno" la serie de órdenes de maquinado anteriormente producidas y, dentro de cada una de estas tareas elementales emplea otros heuristicos para atacar el problema de generación de una trayectoria que produzca la cavidad deseada dada una o mas herramientas posibles.

En este documento se presentará una visión general de la labor de compilación, comenzando por la gramática base y sus implicaciones sobre la capacidad semántica de las entradas al compilador (seccion 2) y continuando con la labor de generación de código haciendo enfasis en las estrategias y heuristicos usados para tal labor, explicados a través de un ejemplo (seccion 3). Por último se concluirá sobre la experiencia obtenida, (sección 4), dando algunas alternativas interesantes que pueden ser exploradas en el futuro.

2. GRAMATICA.

Como gramática básica del compilador, se escogió inicialmente un subconjunto restringido de APT, suficiente únicamente para describir el contorno de la pieza deseada. No se incluyeron instrucciones de loops, subrutinas, etc. Posteriormente se aumentó el conjunto de instrucciones para introducir algunos parámetros de maquinado relevantes para la labor de generación de código.

DEFINICION.

A continuación se define la gramática en la cual se basa el compilador. Esta gramática, aun cuando es prácticamente un subconjunto de APT, tiene modificaciones que la diferencian de el, y cuya justificación se encontrará mas adelante.

```
‹programa_apt› »» partno ‹ident›
                  machin/ ‹ident›
                                ‹bloque_inic›
                                ‹geometria›
                                ‹param-maquinado›
                                ‹movimiento›
                                stop
                                fini

          ‹bloque_inic› »» lenght / ‹numero› ,
                           diam / ‹numero›

          ‹geometria› »» {‹ident› = ‹entidad›}

          ‹entidad› »» ‹punto› | ‹linea› | ‹circulo› | ‹ident› | ( ‹entidad› )

          ‹punto› »» point / ‹descr_punto›

          ‹descr-punto› »»  ‹numero› , ‹numero› |
                            int of , ‹entidad› , ‹entidad› |
                            center , ‹entidad›

          ‹linea› »» line / ‹entidad› , ‹resto_linea›

          ‹resto_linea› »» ‹entidad› |
                           (left | right) , tanto , ‹entidad› |
                           atangl , ‹numero› , ‹entidad›

          ‹circulo› »» circle / ‹descr_circulo›

          ‹descr_circulo› »» ‹entidad› , ‹entidad› , ‹entidad› |
                             center , ‹entidad› , ‹resto_circulo›

          ‹resto_circulo› »» ‹entidad› |
                             radius , ‹numero›
```

‹param_maquinado› »» **coolnt / on**
[**neutral /** ‹herramienta›]
‹herramienta›
‹veloc_cuchilla›
‹arranque›
‹veloc_husillo›
‹margen›

‹herramienta› »» **tool /** ‹ident›

‹veloc_cuchilla› »» **fedrat /** ‹numero›

‹veloc_husillo› »» **spindl /** ‹numero›

‹margen› »» **margin /** ‹numero›

‹arranque› »» **cut /** ‹numero›

‹movimiento› »» **from /** ‹entidad›
‹instr_inic_mov›
{‹instr_interm_mov›}
‹instr_final_mov›

‹instr_inic_mov› »» **go / to ,**
‹entidad› ‹resto_inic›

‹resto_inic› »» □[1] |
, ‹modif_mov› , ‹entidad›

‹instr_interm_mov› »» ‹herramienta› |
‹veloc_husillo› |
‹veloc_cuchilla› |
‹margen› |
‹desplazamiento›

‹desplazamiento› »» ‹direccion› /
‹entidad› ,
‹modif_mov› ,
‹entidad›

‹direccion› »» **gorgt** |
goback |
gofwd |
golft

‹modif_mov› »» **to** |
past |
tanto

‹instr_final_mov› »» **goto /** ‹entidad›

---

[1] □ - nulo

Debido a que la anterior gramática realiza una descripción de la geometría de la pieza a través de una declaración de entidades geométricas (‹entidad›) y de un recorrido de la frontera, pero no arroja ninguna luz sobre el proceso de maquinado, el compilador deberá hacer inferencias sobre la estrategia de maquinado a aplicar.

La gramática anterior garantiza que para cada elemento de la frontera de la pieza existen los siguientes atributos de maquinado:

- Velocidad de rotación del husillo en terminado (**spindl**).
- Velocidad de avance de la herramienta en terminado (**fedrat**).
- Herramienta de terminado asociadada (**tool**).
- Margen de desbastado antes de terminación (**margin**)

Los anteriores elementos estan asociados con una frontera, pero existen otros atributos inherentes a toda la pieza;

- Arranque por pasada (**cut**).
- Herramienta neutra por defecto especificada por el programador si este considera que labores de desbastado no se podrán realizar con la herramienta asociada a las fronteras (**neutral**). Esta especificación es opcional.

SEMANTICA.

Esta sección trata las validaciones semanticas que se hacen sobre el archivo de entrada y sobre la frontera y sus atributos.

Antes de comenzar vale la pena definir algunos términos para su entendimiento en adelante

> -**entidad**: es una figura geométrica; circulo, línea o punto.
>
> -**frontera**: es una porción de una entidad, a la cual además se asocian atributos como velocidad de rotación del husillo, velocidad de terminado, herramienta de terminacion margen entre el desbastado y la terminación y nombre. Al ser una porción de una entidad debe tener información que identifique tanto la entidad de la cual forma parte como la porción de esa entidad que ella abarca.

A continuación se enumeran algunas de las validaciones hechas sobre la frontera. Validaciones sobre la definición de las entidades no se mencionan, aún cuando desde luego, se hacen. Algunos ejemplos de dichas validaciones son: no definir un punto como la intersección de dos rectas paralelas, no definir un circulo con tres puntos colineales, no definir una recta con dos puntos

idénticos,etc.

-Continuidad de la Frontera: Dado que el programa constituye un "viaje" de entidad en entidad, no puede hacerse un paso de una entidad a otra si no se tocan

- Vectores de recorrido no-positivos en el eje X: En cada punto de cada entidad de la frontera, existe un vector tangente que muestra exactamente la direccion del movimiento que ha descrito a esa porción de la frontera. Si un móvil dibujara el contorno de la pieza, tal como lo describe el programa, en cada momento el tendría un vector de velocidad, tangente en cada punto no singular a la frontera sobre la que viaja.Se pide que dicho vector nunca tenga una componente positiva en la dirección X con lo cual se pueden garantizar condiciones de maquinado externo.

- El primero y último punto de la frontera no pueden estar en el interior del rectángulo que representa el bloque inicial. La explicación es que si se permitiera esto, un trayecto desconocido entre el borde del bloque inicial y tales puntos quedaría indefinido, sin entidad asociada lo cual equivaldria a decir que la herramienta"apareció" allí sin maquinar nada.

- La descripción del movimiento comienza con un origen, expresado por una entidad un punto, al cual se debe retornar luego de la descripción de la frontera

- En caso de ambigüedad de puntos destino, por ejemplo en la intersección de dos circunferencias, decide inicialmente la dirección (**gorgt**, **gofwd**, **goback**,**golft**), en caso de subsistir ambigüedad, decide el modificador de movimiento (**to**,**past**,**tanto**) Una vez el punto destino se ha definido, se valida según la regla de la componente X no positiva.

Como conclusión de esta sección, podemos decir que el resultado de la validación semantica sobre el programa de entrada arroja una sucesión de porciones diferentes de frontera que respetan continuidad, tangencia correcta de vectores y retorno de la herramienta al final del trabajo sobre una pieza al mismo punto de salida para que las condiciones de comienzo de la siguiente pieza sean idénticas.

3. GENERACION.

Este capitulo pretende explicar el esquema general de generación de codigo, tanto en los heuristicos usados para definir un orden de maquinado como en aquellos empleados para las tareas elementales de generación arrojadas por el orden escogido.

Se explicará la estrategia de generación de código en base a un ejemplo. Antes de comenzar con el vale la pena definir el término "sector", que será usado intensivamente a lo largo de este capitulo

> -**sector** es un área limitada por una sola frontera[1] en su lado izquierdo, y por una sola frontera en su lado derecho. Los niveles superior e inferior de dichas fronteras son iguales respectivamente. La base de un sector puede ser nula, en el caso de que los puntos inferiores de sus fronteras se toquen, o ser una recta horizontal en caso contrario. El tope de un sector siempre es una recta horizontal. Un sector libre es aquel en el cual la frontera derecha coincide con la recta límite derecha del bloque inicial.

3.1 EJEMPLO.

Con este ejemplo se ilustrará la transformación de la información que posee el archivo de entrada hacia una serie de fronteras correctamente definidas desde el punto de vista de maquinado externo, para posteriormente desembocar en una estructura de datos, cuya misma topología conlleva información sobre el proceso de generación.

3.1.1 CODIGO DE ENTRADA.

En seguida se mostrará un programa típico realizado con el conjunto de instrucciones definidas.

```
(zona de instrucciones al postprocesador)
        partno p1
        machin / emco
(zona de declaraciones geométricas)
        length/20 , diam/20
        stpoint = point/21.14
        p1 = point/16.6
        p2 = point/ 12,6
```

---

[1] Para la definición de frontera ver sección de SEMANTICA.

```
p3 = point/16,10
p4 = point/12,2
p8 = point/4,6
p7 = point/8,10
l5 = line/p1,p2
l1 = line/p2,atangl,45,l5
l2 = line/p3,atangl,90,l1
p9 = point/int of, l2,l5
c1 = circle/p2,p3,p9
l0 = line/p7,left,tanto,c1
l4 = line/p4,left,tanto,c1
c3 = circle/p2,p7,p8
p6 = point/center,c3
l6 = line/p7,atangl,90,l0
p5 = point/int of, l1,l6
l11 = line/p8,rigth,tanto,c3
l7 = line/p5, (point/4,10)
l9 = line/(point/int of, l7, l5),atangl,90,l5
l3 = line/p4,left,tanto,c3
l10 = line/(point/2,0),atangl,0,l11
c2 = circle/(point/int of,l7,l5),(point/8,8),(point/10,6)
l8 = line/p5,left,tanto,c2
c4 = circle/center,(point/12,8),radius,2
c6 = circle/center,p2,radius,4
c5 = circle/center,(point/5,6),radius,1
l15= line/(point/0,8),left,tanto,c2
(zona de especificación del maquinado)
coolnt/on
neutral/tool/svvbn3225p16
tool/pdjnr3232m15q
fedrat/90
cut/1
spindl/908
margin/0
(zona de movimientos)
from/ stpoint
go/to,p9
```

```
margin/0.06
fedrat/50
tool/pdjn13232m15q
spindl/890
fedrat/190
gorgt/c1,to,15
spindl/1000
margin/0.4
golft/115,to,12
gorgt/12,to,c1
tool/pdjnr3232m15q
golft/11,to,p2
tool/pdjn13232m15q
gorgt/(line/p2,right,tanto,c2),tanto,c2
gofwd/c2,tanto,15
gofwd/115,to,17
gorgt/17,to,10
(zona de finalización)
goto/stpoint
stop
fini
```

Antes de comenzar con la discusión de la generación se deben notar varias cosas usando el programa mostrado:

1- La zona de instrucciones al postprocesador da las especificaciones de la maquina herramienta sobre la cual se desea ejecutar el programa.

2- Se garantiza, antes de la instrucción from/stpoint , por la existencia de la zona de especificación del maquinado, que hayan sido precisados los parámetros de fabricación que dirigiran la labor de generación de código.

3- La zona de movimientos tiene por objeto describir univocamente el contorno de la pieza. Es claro que, dada una serie de entidades geométricas, innumerables piezas pueden ser constituidas por ellas; por lo tanto es la zona de movimientos la encargada de quitar la ambiguedad.

4- Entre dos especificaciones de movimiento, en la zona de movimientos, los parametros de maquinado pueden ser cambiados cuantas veces se desee, quedando vigente la ultima especificación.

5- El parámetro cut no puede ser establecido más que una vez; la razón es que él no está asociado a

ninguna frontera sino al desbastado global de la pieza. Igual consideración vale para la herramienta neutra por defecto (**neutral/tool..**), la cual es designada para la totalidad de la pieza.

6- La definición geométrica dada es exagerada para la pieza finalmente hecha; el proposito era únicamente mostrar las capacidades de definición y anidamiento de definiciones, tanto en la zona de declaración geométrica, como en la zona de movimientos o ejecucion.

7- La definición de herramienta neutra por defecto (**neutral/tool**.....) no obliga a que alguna frontera tenga asociada dicha herramienta, ni tampoco lo impide.

8- La zona de finalización marca el final del contorno de la pieza, el retorno al punto inicial y la entrada a un estado latente de la máquina, en espera de la siguiente ejecucion del programa

3.1.2 INFORMACION EXTRAIDA DEL CODIGO FUENTE.

El código fuente provee una descripción de entidades geométricas (figura 3.1) y una forma de recorrido sobre ellas, la cual engendra una serie de fronteras con sus correspondientes atributos (figura 3.2), una lista de herramientas propuestas para el maquinado y unos parametros generales como el arranque por pasada (**cut/**) y la herramienta neutra por defecto (**neutral/tool**).

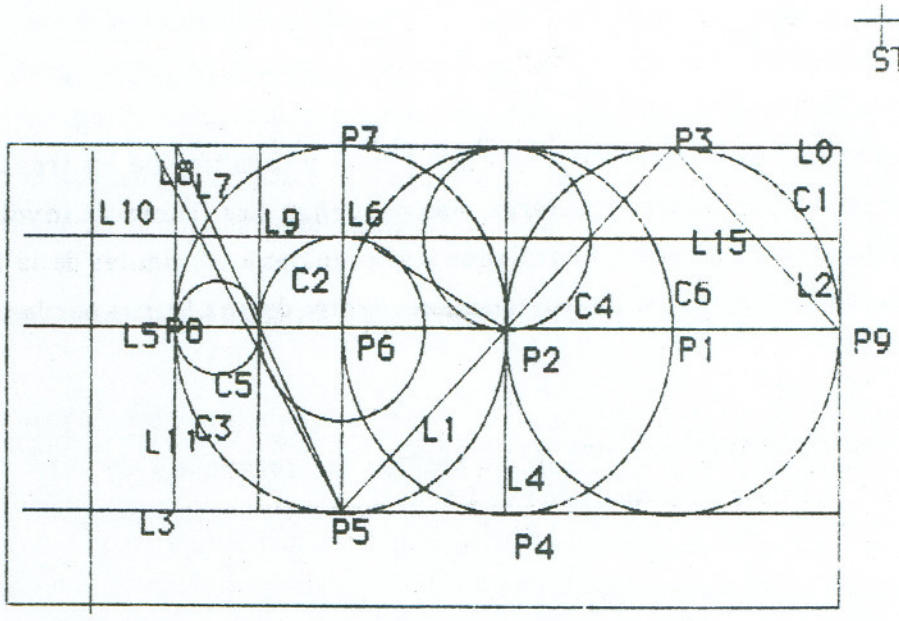

fig 3.1

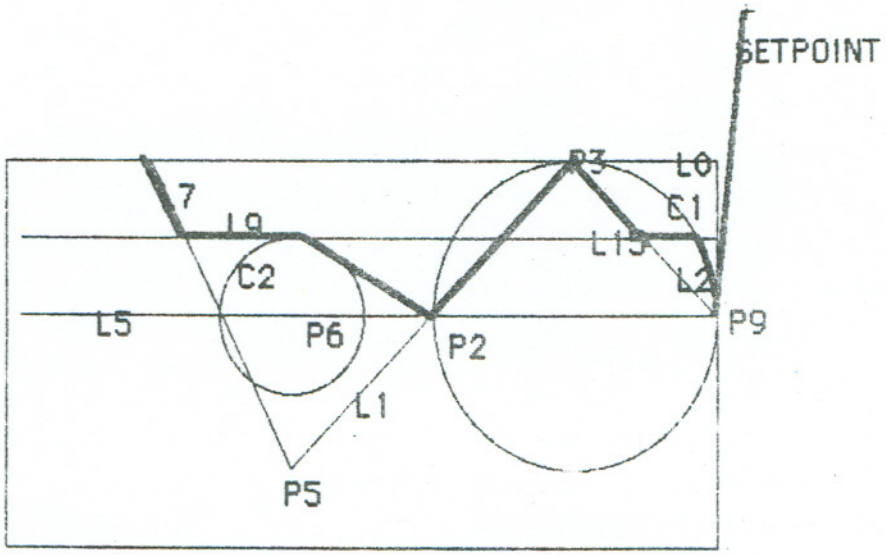

fig 3.2

3.1.3 PARTICION HORIZONTAL DEL TRABAJO DE GENERACION.

Conceptualmente, el trabajo de partición horizontal tiene como entrada una sucesion de fronteras, las cuales forman un contorno válido en cuanto a continuidad y en cuanto a los supuestos de maquinado externo explicados en la sección de semántica del programa fuente. Su salida es una serie de franjas horizontales, cuyos límites son coordenadas Y en las cuales se ha producido un cambio de frontera. Cada franja secciona a todas y cada una de las fronteras con las que se involucra, engendrando nuevas fronteras, más pequeñas. Las fronteras involucradas con una franja son aquellas cuyos límites en Y contienen o son iguales a los límites de la franja (note que una frontera nunca tendrá sus límites estrictamente dentro de una franja por la forma misma de concebir las franjas).

En al figura 3.3 se pueden ver las siguientes tres franjas:

-Limites de la franja.Y_superior = coordenada Y de P3

Y_inferior = coordenada Y de intersección de L15 con L2.

Fronteras involucradas: L2, L1 (seccionada), L7

-Limites de la franja: Y_superior = coordenada Y de intersección de L15 con L2.

Y_inferior = coordenada Y de intersección de C2 con "?".

Fronteras involucradas: C1 (seccionada), L1(seccionada), C2.

-Limites de la franja: Y_superior = coordenada Y de intersección de C2 con "?".

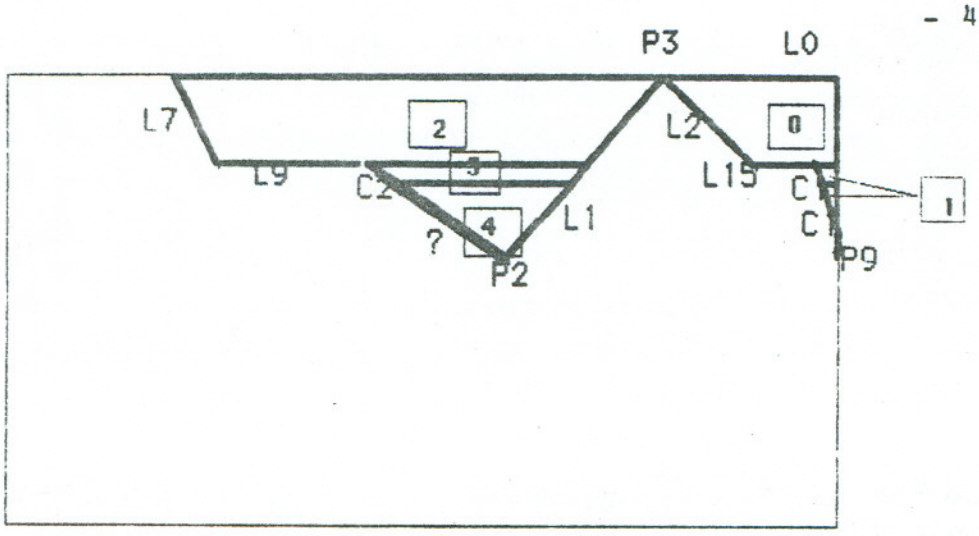

fig 3.3

Y_inferior = coordenada Y de P2.

Fronteras involucradas: C1(seccionada), L1(seccionada), "?".

El propósito del anterior seccionamiento es tener lugares geométricos definidos por **una** frontera al lado izquierdo y por **una** frontera al lado derecho, es decir sectores (ver definicion). Por ello se engendraron las fronteras más pequeñas necesarias para que todos los lugares geométricos así formados tuvieran propiedades de sector.

3.1.4 CONSTRUCCION DEL ARBOL DE MAQUINADO

El árbol de maquinado se construye en base a la definición de **hijo de un sector**:

> **hijo de un sector** es un sector, situado en la franja inmediatamente inferior, y cuyas coordenadas X están contenidas en el rango de coordenadas X del padre

Lo anterior implica que un sector puede tener varios hijos, que serán hermanos entre sí, y se organizarán según coordenadas X descendentes.

El árbol de maquinado es un árbol n-ario, aún cuando en este caso su grado máximo sea 2. El árbol construído con este ejemplo se muestra en la figura 3.3. La numeración dada muestra el orden de recorrido del árbol. El nodo raíz no se identifica con ningun sector. Las relaciones entre los sectores dibujados son:

- **0** y **2** son hijos de la raiz principal y hermanos entre sí.

- **1** es el único hijo de **0**.
- **3** es el único hijo de **2**.
- **4** es el único hijo de **3**.

Como ilustración a las ideas sobre sectores dadas arriba, notamos que **0** y **1** son sectores libres. **1** y **4** tienen base nula (es un punto), mientras que **0**,**2** y **3** no. Todos los sectores tienen una horizontal como tope. Un sector siempre tiene una frontera asociada con su lado izquierdo y una frontera asociada con su lado derecho. La creación de la malla por cada punto singular garantiza esto, aún cuando atomize exageradamente el árbol.

Note que, al haber un punto de quiebre de la frontera anónima ("?") hacia C2, este nivel divide innecesariamente el sector**1**. La labor de normalización del árbol de maquinado incluye reconocer que esos dos sectores en que se ha dividido el sector **1** se pueden fundir en uno solo, puesto que cumplen la definición de sector dada arriba.

El recorrido en pre-orden garantiza que no se intentará maquinar primero el sector **1** que el **0**, o el **3** que el **2**. La prioridad derecha provoca que **0** sea maquinado antes que **2**, aún cuando físicamente ello no es obligatorio. Lo anterior hace que los sectores maquinables por cilindrado tengan prioridad.

3.1.5 ESTRATEGIAS DE GENERACION.

Habiendo establecido la forma en la cual la frontera se transforma en una definición de sectores, organizados en forma de árbol para definir el orden de maquinado,resta explicar cuáles con las estrategias propuestas para cada sector.

3.1.5.1 ESTRATEGIA 1.

Esta se usa para sectores libres; incluye ciclos de cilindrado cuyo arranque es establecido por el parámetro **cut/** del programa, hasta que se llegue a la profundidad suficiente para que solo reste el terminado final. Durante todo el proceso, la herramienta usada es la asociada a la frontera izquierda del sector, puesto que la frontera derecha no existe para el programador; es únicamente un concepto dentro del compilador. Las figuras 3.4 y 3.5 muestran ejemplos de esta estrategia usada para maquinar los sectores **0** y **1**.

Las comprobaciones realizadas para determinar si el maquinado de un sector de este tipo es posible son:

- Tangencia correcta en los puntos superior e inferior de la frontera izquierda.
- Corte izquierdo de la herramienta suficiente para realizar el arranque especificado por **cut**.

3.1.5.2 ESTRATEGIA 2.

La estrategia 2 se usa cuando las herramientas de ambos lados son correctas para terminar su respectiva frontera, pero no son capaces de realizar el desbastado, ni terminar la frontera opuesta. En este caso se usa la herramienta neutra por defecto (**neutral/tool**...) para realizar el desbastado, y cuando se ha llegado la fondo del sector, se reemplaza la herramienta neutra por la correspondiente a cada lado, para, alternativamente, terminar la frontera izquierda, y la derecha. La figura 3.6 muestra el esquema de generación usado.

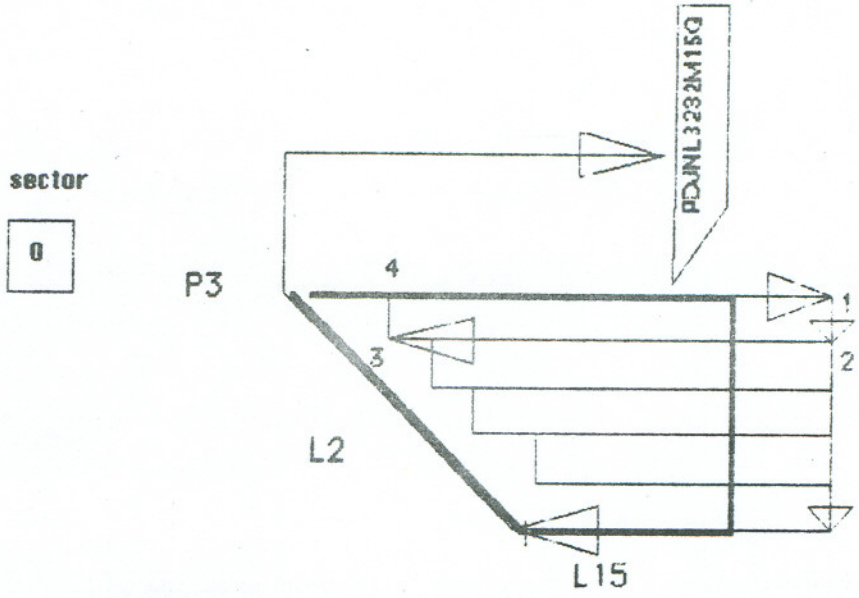

fig 3.4

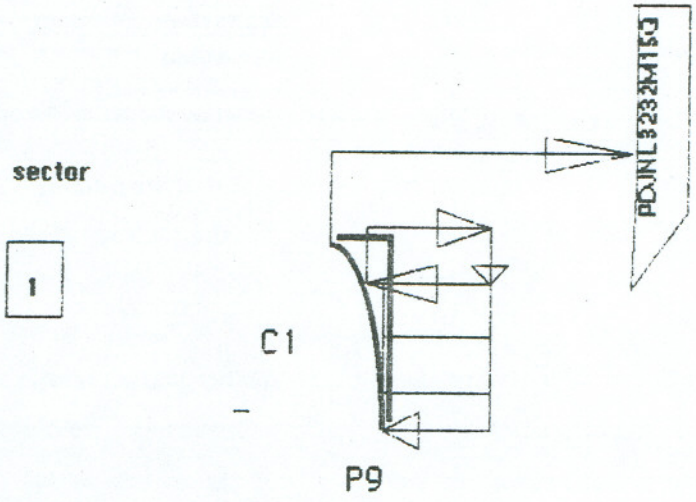

fig 3.5

Cuando se ha hecho el desbastado , un sector parecido al original pero más estrecho aparece por efecto de la especificación de márgenes a lado y lado.

Las condiciones necesarias para que este maquinado resulte exitoso son:

- Tangencia y posicionabilidad correctas tanto en el sector final como en el sector intermedio más estrecho que queda por efecto del desbastado.

- La herramienta de desbastado tiene que poder llegar hasta el fondo mismo del sector. La razon es que las dos herramientas de terminación harán su labor de abajo hacia arriba . Por lo tanto deben poder llegar hasta el fondo. La herramienta de desbastado sigue el camino mostrado determinado por los margenes (**margin**) y el arranque por pasada (**cut**) Cuando los puntos de

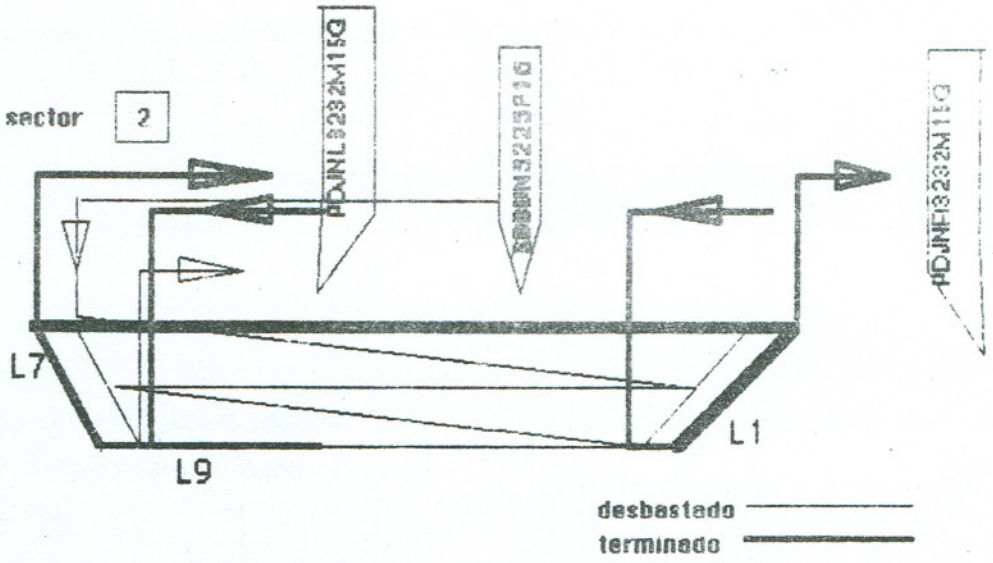

fig 3.6

zig-zag del desbastado están demasiado cerca, la relación distancia-horizontal/arranque se hace muy pendiente, lo cual viola el angulo permisible de maquinado para la herramienta En este caso se desisto de esta estrategia.

- La herramienta de desbaste debe tener suficiente corte para garantizar el arranque por pasada (**cut**) especificado.

3.1.5.3 ESTRATEGIA 3.

Esta estrategia busca realizar el desbaste y el terminado final de un sector encajonado, con la misma herramienta. Ello implica que la herramienta escogida debe poder terminar tanto el lado izquierdo como el derecho del sector, y, tener un corte suficiente de acuerdo a lo especificado por

CUT. Un ejemplo de ella se muestra en las figuras 3.7 y 3.8.Esta estrategia ensaya tanto con la herramienta de la frontera derecha como con la herramienta de la frontera izquierda, y con el movimiento de "caracol", tanto en el sentido del reloj como al contrario.

Las comprobaciones que se tienen que hacer dentro de este esquema son las siguientes:

- La herramienta escogida debe ser posicionable y de tangente correcta en los puntos superior e inferior tanto de la frontera izquierda como de la derecha. Posicionable quiere decir que colocada allí no choca ninguna frontera. De tangencia correcta, quiere decir que sus ángulos tanto del lado del filo como del contrario permiten llegar a dichos puntos realizando la operación de contorneado necesaria sin dañar la frontera. Aún cuando los dos conceptos parezcan el mismo, no lo son; generalmente hay problemas en los límites superiores de un sector por motivo de la tangencia pero no de la posicionabilidad; la herramienta se puede colocar en tal sitio sin danar ninguna frontera; sin embargo su movimiento a lo largo de la frontera la daña.

-El ciclo de desbastado puede terminar antes de llegar al fondo del sector. En tal caso, para que se tenga exito debe restar una profundidad menor al corte especificado por **cut**.

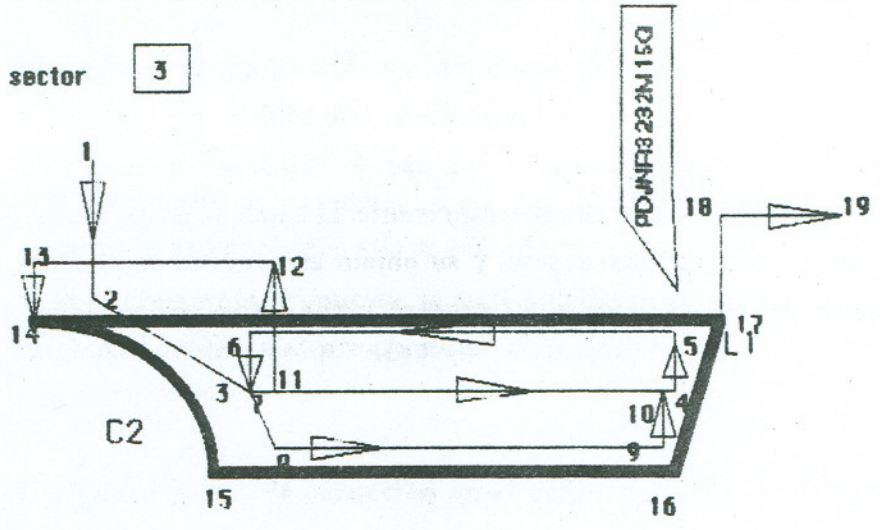

fig 3.7

-Aún cuando la tangencia de la herramienta y su posicionabilidad sean correctas,la trayectoria de un punto a otro puede cortar la frontera. En este caso, lo que ocurre generalmente es que el parámetro de margen (**margin**) es demasiado pequeño. Aumentarlo puede solucionar el problema. Un ejemplo en el cual esto podría pasar es el movimiento 2-3 de la figura 3.7. Allí si el margen de la frontera izquierda fuera demasiado pequeño, el movimiento podría cortar la frontera.

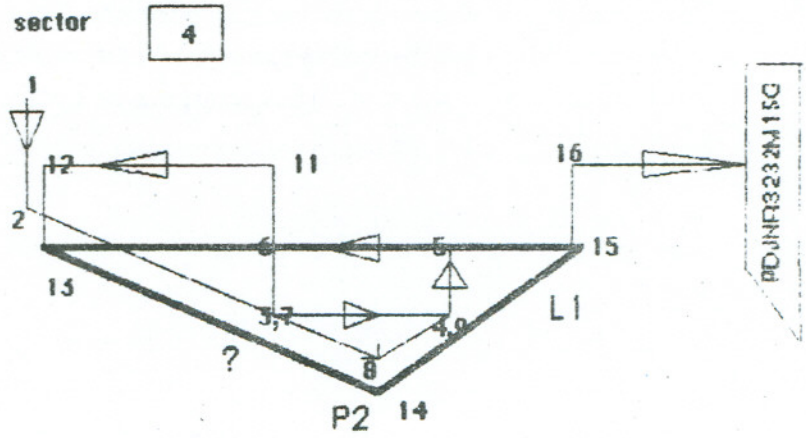

fig 3.8

Para finalizar la sección de generación se hace una precisión sobre la especificación de la herramienta en el programa. En ambientes de programación CNC normales se puede dar una especificación genérica de la herramienta en compilación, pues su validación final puede y debe ser hecha en el postproceso, cuando se trata la máquina herramienta particular. Aquí, por el contrario, se espera la especificación de la herramienta desde el comienzo. Lamentablemente, los códigos normalizados para especificación de herramientas no describen unívocamente la geometría de una herramienta, por lo cual, una representación de ella se debe basar en los datos de algún fabricante. El programa generador permite hacer uso de catálogos definidos por el programador, con el fin de no comprometerse con ningún fabricante. El catálogo es un archivo de texto que es procesado para su validación y rápido acceso, y su objeto es precisar totalmente los parámetros geométricos relevantes de la herramienta. Su proceso y contenido, sin embargo no serán tratados en este artículo.

4. CONCLUSIONES.

Este capítulo trata dos aspectos; el primero de ellos establece la comparación entre los objetivos impuestos y las realizaciones finales del proyecto; el segundo de ellos esboza conclusiones acerca de la realización misma y traduce esas conclusiones en sugerencias para ulteriores proyectos.

El propósito inicial de definir un subconjunto de APT para maquinado externo se vio afectado por dos circunstancias: la primera de ellas fué inherente al hecho de que en este proyecto se debía condensar la labor de compilación y postprocesamiento; la segunda fué debida al interés de usar un subconjunto los más pequeño posible, con el fin de explorar hasta dónde se podía llegar en la tendencia de cargar al compilador con la responsabilidad de definir estrategias tanto generales como específicas de maquinado.

El primer punto obedece a lo siguiente; en ambientes normales de programación CNC al disponerse de un postprocesador se deja en el programa de entrada la posibilidad de trabajar con una herramienta genérica, definida únicamente por algunos parámetros relevantes en ese momento, para hacer una validación de cortes y ángulos más exhaustiva en la labor de postproceso. En el caso presente, se requiere que la definición de la herramienta se haga completa desde la entrada inicial. Tal definición se realizó de acuerdo a los standards de maquinado externo, pero ello de todas formas implicó pedir al programador especificar el nombre de la herramienta propuesta, lo cual no ocurre en un preproceso normal APT. Sin embargo esta exigencia no carga al programador con ningún dato que no debiera saber ya en el momento de planear una pieza.

El segundo punto, de reducción en el conjunto de instrucciones básicas, llevó a transladar parte del trabajo que debiera enfrentar el programador hacia el compilador y el generador de código. La posibilidad de ello reside en el hecho de que con el conjunto definido, lo único que el programador hace es describir la frontera asignándole a sus componentes atributos tales como velocidad de maquinado, velocidad de avance, etc. Esto representa un "salto" muy grande entre el programa de entrada y el código generado, que debe ser asumido por el compilador, ya que para una misma descripción de la frontera, existen innumerables programas en código de máquina que la ejecutan. La conveniencia de ello será discutida más adelante.

Esta segunda desviación se manifiesta en tres modificaciones hechas al lenguaje: introducción de un parámetro de arranque por pasada, **cut**; un parámetro de terminación **margin**; y la declaración opcional del programador de una herramienta **neutral** para el caso de requerirse

para la ejecución de la pieza de una herramienta de desbastado neutra además de aquellas de terminación de la frontera en cuestión.

Como se puede ver, las adiciones no comprometen la filosofía de descargar de trabajo al programador que persigue el compilador, puesto que son factores muy comunes de maquinado que él como persona con algún conocimiento en el tema debe saber.

El hecho de que el compilador asuma totalmente la tarea de generación de código sin preguntar nada al usuario tiene consecuencias sobre la consecución de objetivos iniciales del proyecto en lo que tiene que ver con maquinabilidad de la pieza. En este momento, el compilador genera código a través de tres estrategias distintas y cuando no puede hacerlo con ellas declara que la generacion fué imposible. Estas tres estrategias cubren la mayor parte de las piezas torneadas externamente, pero desde luego, pueden existir piezas para las cuales el compilador se declara impotente, sin que ello quiera decir que la pieza , con otra estrategia más sofisticada no se pueda realizar. Sin embargo, por las pruebas hechas se llegó a la conclusion de que tales piezas son verdaderamente infrecuentes.

Habiendo tratado los logros y limitaciones del proyecto, conviene dedicar algun espacio a las conclusiones que trabajar en él ha producido.

En primer lugar vale la pena tratar las consecuencias que un subconjunto muy reducido de instrucciones tiene sobre la facilidad en la generación de código. Como se dijo anteriormente, esta deficiencia produce de inmediato un salto muy grande entre el programa, y lo que finalmente serán las estrategias usadas. El problema no es una cuestión de aumentar el repertorio de generación para que el compilador no "se dé por vencido" pronto, aunque tal acción es deseable. El problema estriba en que no hay una parte del lenguaje que exprese de alguna forma lo que el programador quiere. Tal y como están las cosas ahora el define la frontera las herramientas asociadas, los parámetros de maquinado, y en ese momento pierde el control del proceso. Por otro lado, el compilador realiza todo el trabajo de identificación de primitivas trabajables, ordenamiento sobre ellas y escogencia de la estrategia a seguir. Juzgar correcta la ultima aproximación no es del todo razonable, y obligar al programador a especificar el movimiento de las herramientas lo carga de un trabajo del que , se ha visto en las condiciones de maquinado externo, podría estar liberado. Si lo que interesa es proveer una herramienta que haga el trabajo con una direccion del programador se puede llegar a compromisos como proponer una lista de estrategias para cada sector ordenadas segun razonamiento del programa para que el programador escoja, rechace, o modifique el orden propuesto, o , posiblemente presentar ordenamientos sobre los sectores a maquinar, y luego , dentro de ellos ordenamientos

sobre las estretegias a usar, o , que el programa pida ayuda al experto que tiene al frente cuando sus conocimientos fallaron del todo para un sector, o cuando nota que las estrategias que faltan por probar están muy atrás en la lista de prioridades, lo cual parece indicar poca eficiencia en el caso en particular, etc.

El proceso de generación se puede hacer más "razonable" en el sentido de que conserve su eficiencia en la transformación entrada/salida, pero se sirva de un sistema flexible de heuristicos para la escogencia de la linea de acción a seguir; dicho de otro modo, es posible abordar la labor de generación de código no de una forma estática, sino con un comportamiento adaptable por parte del programa, asimilable a un sistema experto.

5.BIBLIOGRAFIA

[AHO77] Aho Alfred, Ullman Jeffrey, "Principles of Compiler Desing".
Addison-Wesley 1977.

[KOR83] Korem, Yoram, "Computer Control of Manufacturing Systems".McGraw-Hill Book,1983.

[KRA86] Kral Irving, "Numerical Control Programming APT".
Prentice-Hall, 1986

[ROD86] Rodriguez German, "SITOR, Manual del Usuario".
Uniandes 1986.