Este exemplar corresponde à redação final da tese defendida por Joinvile Batista Junior e aprovada pela comissão julgadora em 18/12/1986.

Mario Jino

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA DE CAMPINAS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

SISTEMA AUTOMATIZADO DE DOCUMENTAÇÃO GRÁFICA

TESE DE MESTRADO

AUTOR : JOINVILE BATISTA JUNIOR

ORIENTADOR : PROF. DR. MÁRIO JINO

Dezembro de 1986

"Ainda que eu conheça todos os mistérios e toda a ciência, se eu não tiver amor nada disto se aproveitará." (I Cor. 13)

Dedico este trabalho a meus pais, cujo estímulo não tem me faltado nesta longa caminhada.

AGRADECIMENTOS

Ao professor Mário Jino pela orientação.

Aos técnicos de programação Márcio Brochado Alves da Silva e Sérgio Yoshioka pela implementação de módulos do SADG.

Aos analistas de sistemas Celso Penteado de Barros, Fernando Antonio Vanini e Renata Merino Rodrigues dos Santos e ao programador Orlando Seigi Nakano pelas sugestões ao trabalho.

À técnica de programação Maria Alice Dias pela coleta de dados referente à utilização do SADG no projeto da Central de Comutação Telefônica TRÓPICO-R.

RESUMO

O objetivo deste trabalho é o desenvolvimento de uma ferramenta de software, denominada SADG (Sistema Automatizado de Documentação Gráfica), que permita a geração da forma gráfica, a partir da forma programa, de uma linguagem de especificação (normatizada pelo CCITT) denominada SDL (Specification and Description Language). A saída gráfica é possível em impressora e terminal de vídeo gráficos.

Apresenta-se uma descrição sucinta sobre SDL e as metas para a ferramenta SADG. A seguir são apresentados os módulos do SADG, com ênfase para o módulo responsável pela distribuição de símbolos nas páginas e substituição de conectores e labels por linhas de conexão, que constitui a parte principal deste trabalho. Ao final são apresentadas perpectivas de evolução em relação às funções realizadas pelo SADG e considerações de desempenho do SADG.

Embora a implementação do SADG tenha visado a forma gráfica de SDL, os algorítmos apresentados para distribuição de símbolos e linhas de conexão aplicam-se a fluxogramas genéricos. Técnicas utilizadas para geração automática de fluxogramas e roteamento de circuito integrado são sucintamente apresentadas e comparadas com as soluções adotadas no SADG.

ÍNDICE

ÍNDICE (Continuação)

ÍNDICE (Continuação)

LISTA DE FIGURAS

LISTA DE FIGURAS (Continuação)

LISTA DE FIGURAS (Continuação)

LISTA DE TABELAS

1 - INTRODUÇÃO

1.1 - Motivação do trabalho

A indústria das telecomunicações é uma das maiores produtoras e usuárias de software no mundo. O software assume importância cada vez maior nos projetos de desenvolvimento, tendo o seu custo ultrapassado mais da metade do custo total de desenvolvimento; esta tendência continua a crescer sem demonstrar sinais de inversão deste processo [DELL81]. A busca da solução deste problema tem originado inúmeros esforços no sentido da automação da geração de software, a qual deverá mudar radicalmente nos próximos anos.

A documentação de projetos de desenvolvimento associados a sistemas de comutação telefônica tem tido a necessidade de se atualizar para atender às suas peculiaridades :

- muitas linhas de código;
- muitos programadores;
- a prática corrente de produzir-se a documentação ao final do projeto, para a liberação do software;
- alto grau de disciplina para a produção e a manutenção de documentação, que reflitam a real implementação, em contraste com o comportamento padrão dos programadores;
- a ausência de uma documentação coerente que acompanhe as diversas fases do projeto tem levado a contínuas e indesejáveis realimentações entre fases, dificultando a ação da gerência e ocasionando implementações que demonstram na fase de integração um grau muitas vezes inadmissível de inconsistência e overhead.

A representação na forma gráfica possibilita maior noção de contexto do que a textual devido aos seguintes fatores :

- o texto é unidimensional e não provê separação visual entre a estrutura e o detalhe do programa;

- devido ao problema de retenção na memória, o material lido começa a perder o significado imediatamente após a leitura, o que pode obrigar a repetidos exames do texto.

A tentativa de extrair fluxogramas de listagens tem-se mostrado pouco útil por falhar em evidenciar a estrutura dos programas. Uma solução para contornar os problemas de consistência entre documentação e implementação e tornar a documentação disponível ao longo do projeto tem sido a da geração automática da implementação e da documentação a partir da mesma fonte.

A automação da geração do código fonte da implementação, bem como da conversão entre as duas formas de documentação (textual e gráfica), traz inúmeras vantagens tais como :

- produz documentação e implementação de forma mais rápida, eficiente e padronizada;

- assegura consistência e simultaneidade entre a geração da documentação e da implementação;

- diminui consideravelmente o impopular esforço de geração de documentação;

- viabiliza a heterogeneidade de qualificação entre os implementadores, por reduzir a implementação do sistema a procedimentos menores;

- auxilia a gerência do controle efetivo de versões e da política de alterações durante as fases de integração;

- aumenta a confiabilidade e simplifica a manutenção de produtos com longa vida.

Inúmeras organizações internacionais ligadas a Administrações Telefônicas vêm se utilizando de duas linguagens normatizadas pelo CCITT para o desenvolvimento de seus projetos :

- SDL (Functional Specification and Description Language) [CCIT84a] [CCIT84b] : utilizada para especificação e documentação e

- CHILL (CCITT High Level Language) [CCIT84c] : utilizada para implementação.

A maioria destas organizações tem desenvolvido suas próprias ferramentas para a automação do uso de SDL e CHILL, buscando criar um ambiente de desenvolvimento para os seus projetos.

Dentre as ferramentas, algumas já em uso nas Administrações Telefônicas há alguns anos e outras ainda em desenvolvimento (estando algumas até mesmo disponíveis comercialmente), a maior parte delas implementa a função de geração da forma SDL-GR a partir de SDL-PR. Abaixo estão relacionadas as organizações, os seus respectivos países de origem, nome da ferramenta (se esta existe) e as funções implementadas pelas ferramentas :

- British Telecom, Reino Unido, SX1-CADOS [COAK78] [CORK76] [DELL81] [TINK85], funções : editor gráfico, código esqueleto (linguagens : PASCAL, PL-M, CORAL 66, C e CHILL);

- CSELT e SIP, Itália, [ASS85] e SCAT [GIAR83] [GHIS85], funções : editor gráfico, tradutor PR/GR, tradutor SDL/CHILL, simulador;

- Telelogic, Suécia, SDT [BELI85], funções : editor gráfico, tradutor PR/GR, tradutor SDL/CHILL, simulador;

- Ericsson, Suécia, [BORG82], função : editor gráfico;

- Computas, Noruega, DASOM [VEFS85], funções : editor gráfico e suporte a metodologia DASOM [BRAE79a] [BRAE79b] [BRAE82];

- ELAB, Noruega, SILL [RUST85], função : tradutor SDL/CHILL;

- Siemens, Alemanha Ocidental, SPOCK [CCIT85], funções : editor gráfico, tradutor PR/GR, tradutor C/SDL, trace gráfico;

- Austrália Telecom, Austrália, CADDIE [GERR81] [GERR82] e MELBA [CAIN81], funções : editor gráfico, tradutor PR/GR, tradutor SDL/CHILL;

- AT&T, USA, [CCIT85], funções : editor gráfico, tradutor PR/C;

- PTT Holandes, Holanda, CASA [KONI82], função : editor gráfico;

- Telenokia, Filândia, SDL/PR-GR Compiler [TAIP82] [TAIP85], função : tradutor PR/GR;

- Telefonbau, Alemanha Ocidental, SDLDG [CCIT85], função : tradutor PR/GR.

Dentro deste contexto, o CPqD/Telebrás vem utilizando SDL (desde 1981) para descrição funcional e dos blocos de implementação software do projeto TRÓPICO (CPA Temporal), e prepara-se atualmente para utilizar CHILL (CCITT High Level Language) como linguagem de implementação.

O primeiro passo para a automação da geração da documentação em SDL na forma gráfica foi dado no primeiro semestre de 1983 com a compra de uma ferramenta software do CSELT (Centro Studi E Laboratori Telecomunicazione) de Turim (Itália), denominada EGS (Elaboratori Gestione SDL) [BAGN81a], [BAGN81b], [FERR82], [MACA82] e [GASS85]. Esta ferramenta permite : a conversão nos dois sentidos das duas formas de representação do SDL (forma programa e gráfica), a entrada na forma gráfica (através de editor gráfico) e a saída em impressora e terminal de vídeo gráficos.

No entanto, alguns fatores tais como a indisponibilidade do código fonte (foi adquirido apenas o código objeto) e o fraco desempenho do EGS (sobretudo na geração da forma gráfica a partir da forma programa), impulsionaram a proposta de um desenvolvimento próprio.

Desta forma, em maio de 1984, começou o desenvolvimento do projeto SADG (Sistema Automatizado de Documentação Gráfica), o qual terminou em dezembro de 1985. Esta ferramenta implementa a função de geração automática do SDL na forma gráfica a partir da sua forma programa, com saída em impressora e terminal de vídeo gráficos.

As técnicas utilizadas para geração automática de fluxograma de documentação a posteriori, a partir de listagem de programa, apresentam requisitos distintos das necessidades envolvidas na geração de uma documentação a partir de uma linguagem de implementação (vide item 5.4). Da mesma forma, as técnicas utilizadas para roteamento automático de circuito impresso, que guardam alguma semelhança com a implementação de linhas de conexão em fluxogramas, têm condições de contorno distintas (vide item 5.4). A busca de soluções próprias adequadas à automação de fluxogramas provenientes de uma linguagem de especificação para a implementação do SADG foi a grande motivação deste trabalho.

1.2 - Objetivo e Organização do Trabalho

Neste trabalho, busca-se analisar a conveniência da sua proposta, bem como a solução escolhida para os módulos que compõem o SADG, finalizando com uma comparação de sua performance e o enfoque às perpectivas de sua evolução para uma ferramenta mais abrangente. Procura-se dar ênfase às soluções encontradas para o módulo responsável pela distribuição de símbolos nas páginas, uma vez que nele reside a parte mais complexa e criativa deste trabalho.

O capítulo 2 traz uma descrição sucinta de SDL, com as suas duas formas de representação, e procura analisar as vantagens da automação da geração da forma gráfica a partir da forma programa.

No capítulo 3 é mostrada a estruturação do SADG em módulos, explicitando os recursos de desenvolvimento utilizados, estabelecendo uma escala de prioridade para os seus fatores de desempenho. Propõe-se uma extensão a SDL, para permitir o seu uso de forma estruturada.

No capítulo 4 justifica-se a escolha do método de implementação do módulo Compilador, enfatizando-se particularmente a escolha do mecanismo de recuperação de erros.

O capítulo 5 é o cerne deste trabalho; nele são analisadas soluções para a distribuição otimizada de símbolos nas páginas, e para a distribuição de labels e conectores por linhas de conexão; as soluções são comparadas com as técnicas existentes para a geração automática de fluxogramas e para roteamento automático de circuito integrado.

No capítulo 6 analisa-se o tratamento de textos (hifenação e centralização) e o pacote de rotinas gráficas para criação do arquivo para saída em impressora gráfica.

No capítulo 7, propõe-se uma evolução para o trabalho a partir do acréscimo das funções de tradução SDL/CHILL e edição gráfica.

No capítulo 8, são apresentadas considerações finais, onde se comparam os desempenhos, em termos de qualidade do documento de saída e tempo de processamento, das ferramentas SADG e EGS.

2 - ESPECIFICAÇÃO E DOCUMENTAÇÃO DE SISTEMAS TELEFÔNICOS EM SDL

Neste capítulo são apresentados de forma sucinta os conceitos da linguagem SDL, bem como suas formas de representação programa e gráfica. Analisam-se também as vantagens decorrentes da automação da geração da documentação gráfica em SDL, através da conversão da sua forma programa.

2.1 - Descrição do SDL

SDL (Specification and Description Language) foi normatizada pelo CCITT para a especificação e descrição de sistemas telefônicos. Sua estrutura encontra-se em aperfeiçoamento, e a cada quadriênio novos conceitos e refinamentos lhe são incorporados. SDL utiliza-se de três formas de representação : gráfica (SDL-GR), programa (SDL-PR) e pictórica (que não é objeto de interesse deste trabalho).

Sua estrutura baseia-se numa máquina de estados, de forma que a sua representação mais compacta poderia ser enunciada através de três conceitos básicos : estado, sinal de ativação e ação de transição. Ou seja, a recepção de um sinal ativador num dado estado onde este é reconhecido ocasiona a execução das ações associadas a esta transição e a ida para um novo estado (que pode ser o próprio).

Tais estados agrupam-se em processos que, por sua vez, agrupam-se em blocos para constituir o sistema em descrição. Num dado bloco são especificadas as funções relacionadas com um dado recurso do sistema descrito. Supondo que o sistema seja uma central telefônica, pode-se citar como recursos : assinantes, juntores, equipamentos auxiliares, registradores, etc. A comunicação entre os processos (do mesmo bloco ou de blocos

distintos) dá-se através de sinais. Partindo dos três conceitos básicos citados acima, pode-se derivar as demais estruturas do SDL.

As estruturas em SDL derivadas do conceito de estado são :

- State : Estrutura inicial de uma dada transição. Uma ativação de um processo permanece num estado, até que exista um sinal de entrada (vide input abaixo) que ative a execução de uma transição associada ao respectivo sinal. Sinais que chegam em estados, nos quais não estão previstos, são descartados.

- Nextstate : Estrutura final de uma dada transição, que determina o estado do processo (ou de um indivíduo do processo) ao final da execução de uma transição (o próximo estado pode ser o atual ou outro estado do processo).

- Start : Estado inicial de um processo, ativado pela estrutura Create Request pertencente a um outro processo, ao invés de um sinal de entrada (é opcional e único para um dado processo).

As estruturas em SDL derivadas do conceito de sinal ativador são :

- Input : Sinal de entrada que ativa a execução de uma transição (que é univocamente definida pelo par estado/sinal de entrada). O tratamento de sinais pressupõe uma fila de atendimento por ordem de chegada.

- Save : Salvamento de sinal de entrada que pode chegar num estado, mas só será reconhecido e tratado (existência de transição associada) no estado subsequente no qual não haja declaração de save. Este recurso é muito importante porque todo sinal que é escalado num estado, para o qual não estava previsto, é simplesmente descartado, quando muitas vezes o que se deseja é que o sinal permaneça na fila, aguardando a ocorrência do estado no qual seja reconhecido (ou seja, exista uma transição associada à

recepção de sinal naquele estado). O recurso save evita o descarte do sinal, mantendo-o na fila de recepção de sinais.

- Enable Condition : Condição de ativação de uma dada transição, que impõe uma condição para o tratamento ou retorno à fila (save condicional). Ou seja, além da recepção do sinal de entrada, é necessário que a condição de ativação seja verdadeira, para que o sinal possa ativar a execução da transição a ele associada.

- Signal Continuous : Condição de ativação de uma dada transição, que substitui o sinal de entrada, como se fosse um sinal contínuo. Ou seja, a execução de uma dada transição não está associada à recepção de um sinal, e sim a uma condição de ativação, denominada sinal contínuo (por substituir totalmente a necessidade de recepção de sinal). Sinais contínuos associados a um mesmo estado devem ter prioridades distintas, para evitar ambiguidades.

As estruturas em SDL derivadas do conceito associado às ações de uma transição são :

- Task : Elemento básico de uma transição associado a uma determinada tarefa.

- Decision : Condição que determina opções de continuação para um dado ponto de uma transição (pode ter um número qualquer de ramos).

- Create Request : Solicitação de ativação de um outro processo (iniciado por um Start).

- Call Procedure : Chamada de rotina.

- Option : Semelhante a uma Decision, com a diferença que seus ramos não prescrevem diferentes situações na ativação de uma dada transição, mas sim opções mutuamente exclusivas de implementação de um dado trecho de transição.

- Comment : Informação adicional associada a qualquer ponto da transição (pode também ser associado a qualquer estrutura do processo).

Para completar, tem-se as seguintes estruturas :

- Join : Conector terminador de uma dada transição ou ramo de decisão, associado a um label da mesma transição ou de qualquer outra transição do processo.

- Stop : Terminador de processo cuja ação é oposta à da estrutura Create Request (ou seja, determina o encerramento do processo).

- Label : ponto de convergência de conectores (Join) do mesmo processo numa dada transição.

- Procedure : Definição da subrotina executada devido a ocorrência da estrutura Call Procedure. Deve iniciar e terminar com símbolos de ações de transição, mas pode conter estados e sinais de ativação.

- Return : Indicador de término de procedure e retorno à transição que executou a estrutura Call Procedure.

Qualquer trecho de descrição pode ser substituído por uma chamada de macro, a ser definida à parte tal como as Procedures. Para tal existem as seguintes estruturas : Macro (chamada), Macro Expansion (início da definição) e End Macro (fim da definição). As definições de Macro e de Procedure podem aparecer no escopo do sistema, de um bloco, de um processo, ou de um outro procedimento.

A cada uma destas estruturas associa-se um símbolo gráfico (SDL-GR) e uma declaração (SDL-PR), com textos associados (internos aos símbolos ou componentes de declarações). Tais textos foram inicialmente associados a identificadores e/ou textos livres (Yellow Book [CCITT80]), e ultimamente (Red Book [CCITT84a]) têm incorporado opções de definições mais formais de

expressões, conjuntamente com estruturas de definição de dados acrescentadas à forma PR.

O objetivo deste trabalho está relacionado com a automação da transformação de SDL-PR para SDL-GR. Desta forma, restringe-se a enfocar os conceitos enumerados acima, deixando de lado outros conceitos em SDL-PR, tais como : estrutura de dados, subestruturas (partição de blocos em blocos internos, e de processos em processos internos) e canais (sinalização entre blocos). Ou seja, os textos associados ao interior dos símbolos, serão (como definidos no Yellow Book [CCITT80]) encarados como identificador e/ou texto livre.

2.2 - Formas de Representação de SDL

Há uma correspondência biunívoca entre os símbolos gráficos usados na forma GR, e as palavras chaves usadas na forma PR para representar uma dada estrutura (a menos de declarações de fim de estrutura). Vide Figura 2.1, complementada pelas seguintes observações :

- utiliza-se o mesmo símbolo gráfico que representa State e Nextstate; a diferença ocorre na utilização (State no início de transição e Nextstate no final);

- o símbolo gráfico que representa Signal Continuous, quando utilizado após o símbolo de Input, passa a significar Enable Condition;

- as ramificações associadas ao símbolo de decisão são colocadas sempre abaixo deste, para permitir que o espaço lateral ao símbolo de decisão seja utilizado por comentário associado.

Ambas poderiam ser utilizadas como documentação de projeto, mas a forma gráfica, ao se utilizar de maior aglutinação dos textos (texto interno a um símbolo) e da informação associada a

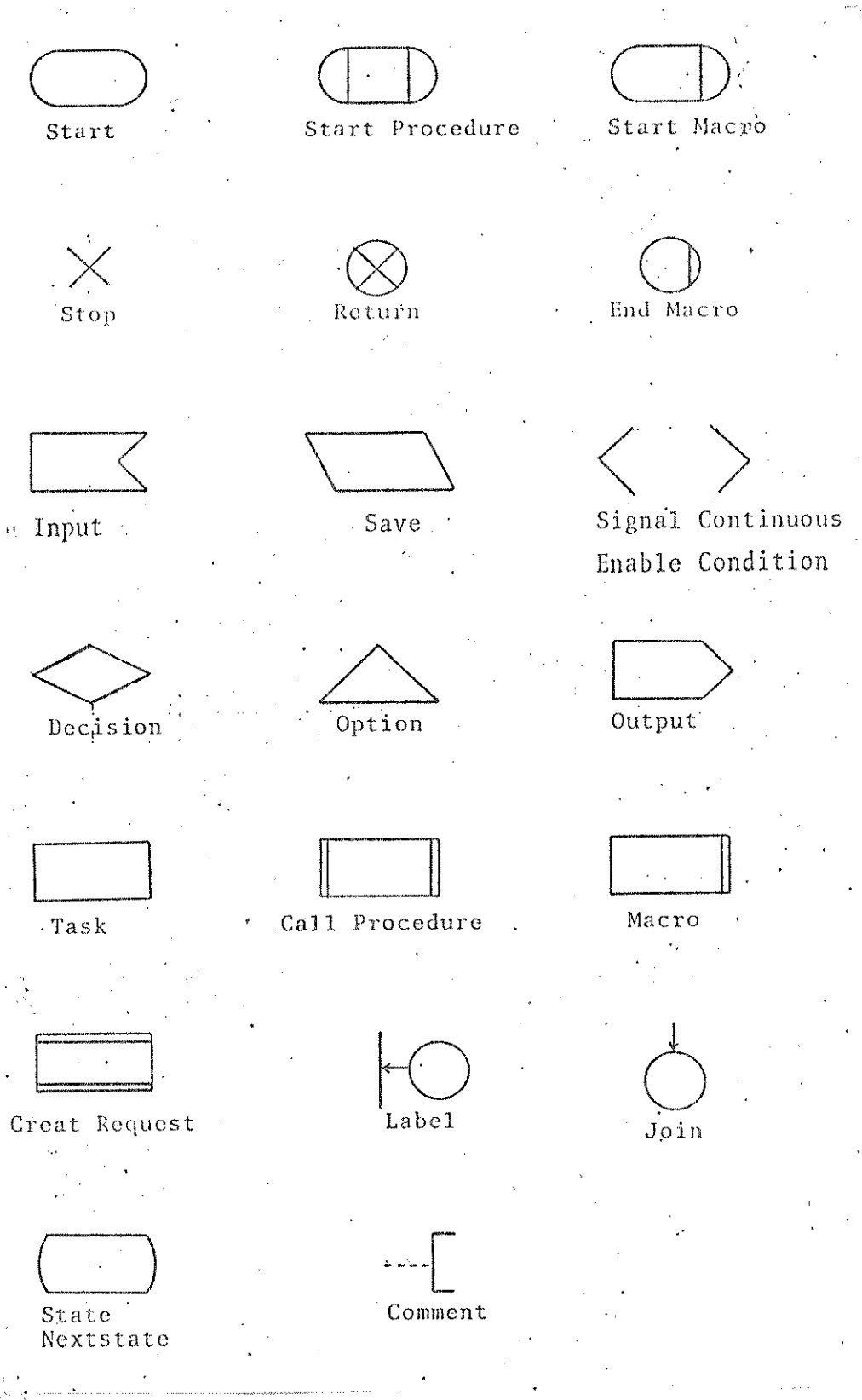


Figura 2.1 - Correspondência entre SDL-PR e SDL-GR

cada símbolo, traz sensíveis vantagens sob o ponto de vista de visualização e noção de contexto, o que torna mais acessível o intercâmbio dos documentos gerados pelo pessoal envolvido no projeto, bem como a geração da documentação final do produto a ser industrializado. A forma programa, no entanto, é mais facilmente gerada e mantida, por implicar apenas edição de textos.

A tradução automatizada de SDL-PR para SDL-GR, minimiza o tempo e os recursos gastos com a documentação de um projeto, que se tornam bastante significativos para um projeto de médio ou grande porte.

2.3 - Automação da documentação em SDL

A automação de documentos em SDL beneficia a todos aqueles que interagem com tais documentos, nas fases de geração, alteração, utilização e manutenção dos mesmos. O armazenamento de tais documentos em memória de massa, suportados por ferramentas que permitam sua criação e alteração, controle de versões e cópias em larga escala, permite disciplinar, sistematizar e reduzir os penosos esforços associados à documentação de um projeto de médio ou grande porte.

Para realizar tal automação são necessárias ferramentas de software adicionais. A geração da forma PR pode ser auxiliada por um analisador, capaz de gerar uma lista do programa analisado com informações dos erros encontrados, de forma a possibilitar a sua correção. A geração da forma GR é bem mais complexa, e pode ser feita através de um editor gráfico ou através da tradução a partir da forma PR.

A geração da forma GR através de um editor gráfico permite ao gerador do documento a distribuição dos símbolos nas páginas

segundo a sua conveniência. No entanto, a sua conveniência está associada a alguns fatores :

- disponibilidade de terminais gráficos eficientes em velocidade e independentes em capacidade de processamento, funcionando como estações de trabalho (workstations), de forma a tornar o processo interativo de edição gráfica eficiente (uma vez que a criação e manutenção de documentos através da forma gráfica é bastante eficiente), e disponível a um grande número de usuários nas diversas fases de desenvolvimento de um dado projeto;

- a conversão GR-PR-GR torna-se uma imposição natural, à medida que tende a automatizar a geração de programas em CHILL através de SDL-PR, e que a opção de entrada na forma PR deve ser fornecida aos usuários; de forma que a eficiência na distribuição de símbolos conseguida através do editor gráfico pode ser anulada se não houver possibilidade de conversão eficiente de PR para GR;

- o editor gráfico deve dispor de funções que minimizem a interação do usuário com o sistema, porque a experiência mostra que editores gráficos onde o usuário é obrigado a detalhar demasiadamente suas operações ou têm alto custo associado a rearranjos e referência de página (por ocasião de remoção ou inserção de uma nova página), acabam desmotivando totalmente a utilização da forma gráfica como entrada.

Além da tradução otimizada de PR para GR ser importante para a automação completa do ciclo de vida de desenvolvimento de software, ela é vital para projetos que venham a utilizar tais ferramentas numa fase onde os documentos de especificação e detalhamento já se encontram em fase adiantada ou final de geração manual, e a conversão para geração automática requeira entrada maciça de informação no sistema, a exemplo do projeto TRÓPICO-R (CPA Temporal de Pequeno Porte) desenvolvido no CPqD.

segundo a sua conveniência. No entanto, a sua conveniência está associada a alguns fatores :

- disponibilidade de terminais gráficos eficientes em velocidade e independentes em capacidade de processamento, funcionando como estações de trabalho (workstations), de forma a tornar o processo interativo de edição gráfica eficiente (uma vez que a criação e manutenção de documentos através da forma gráfica é bastante eficiente), e disponível a um grande número de usuários nas diversas fases de desenvolvimento de um dado projeto;

- a conversão GR-PR-GR torna-se uma imposição natural, à medida que tende a automatizar a geração de programas em CHILL através de SDL-PR, e que a opção de entrada na forma PR deve ser fornecida aos usuários; de forma que a eficiência na distribuição de símbolos conseguida através do editor gráfico pode ser anulada se não houver possibilidade de conversão eficiente de PR para GR;

- o editor gráfico deve dispor de funções que minimizem a interação do usuário com o sistema, porque a experiência mostra que editores gráficos onde o usuário é obrigado a detalhar demasiadamente suas operações ou têm alto custo associado a rearranjos e referência de página (por ocasião de remoção ou inserção de uma nova página), acabam desmotivando totalmente a utilização da forma gráfica como entrada.

Além da tradução otimizada de PR para GR ser importante para a automação completa do ciclo de vida de desenvolvimento de software, ela é vital para projetos que venham a utilizar tais ferramentas numa fase onde os documentos de especificação e detalhamento já se encontram em fase adiantada ou final de geração manual, e a conversão para geração automática requeira entrada maciça de informação no sistema, a exemplo do projeto TRÓPICO-R (CPA Temporal de Pequeno Porte) desenvolvido no CPqD.

A geração da forma GR através da forma PR requer, além do compilador, um tradutor que realize de maneira otimizada a distribuição dos símbolos nas páginas, que substitua sempre que possível pares de join-label por linhas de conexão, e que associe aos conectores as páginas de origem ou destino referentes aos conectores associados. Utilizando esta forma de gerar e manter documentos em SDL-PR, conta-se com as seguintes vantagens :

- A tradução da forma PR para a forma GR realiza a distribuição dos símbolos de forma totalmente automatizada. Além de minimizar ao máximo a interação do sistema com o usuário na geração do documento, suporta automaticamente qualquer alteração realizada no texto original (forma PR), dispensando rearranjos exigidos pelo editor gráfico.

- Simplicidade na interface entre o gerador e o gerenciador do documento, que implica somente em edição (para geração e atualização) e comparação (para controle de versões) de textos. Desta forma, a aquisição ou alocação de terminais gráficos é desnecessária, e a geração e atualização dos documentos é mais rápida e eficiente, devido ao grau de automação que o sistema provê.

3 - CONCEPÇÃO E METAS DO SADG

A proposta deste trabalho é o projeto de uma ferramenta de software que realize a conversão de um documento em SDL-PR para SDL-GR de forma totalmente automatizada e com requisitos de qualidade que viabilizem a aceitação do documento de saída.

O sistema assim proposto foi denominado SADG (Sistema Automatizado de Documentação Gráfica). Suas metas se fundamentam nas seguintes premissas :

- maior automação possível na geração de documentação em SDL;
- opções prevendo diversos níveis de documentos e usuários;
- saída gráfica com performance comparável à da gerada por editor gráfico;
- modularidade entre seus componentes.

A geração totalmente automatizada da documentação necessita de um compilador e de um mapeador. O compilador, que tem como entrada o SDL-PR, deve efetuar boa recuperação de erros, de forma que o usuário possa corrigir o seu programa em SDL-PR de maneira eficiente. O mapeador deve ser capaz de distribuir em páginas, de forma automática, a representação SDL-GR correspondente ao programa SDL-PR, prevendo saída : substituição de pares label-join por linhas de conexão e referência de página de origem e destino nos conectores e labels.

Posto que a geração é automatizada, o usuário deve ter acesso a parâmetros de entrada que lhe permitam orientar a distribuição e o tamanho dos símbolos SDL-GR, buscando cobrir diversos níveis de documentação ou mesmo diferentes perfis de usuários.

A performance da saída gráfica é vital para a aceitação do documento gerado pela ferramenta proposta. Devem ser estabelecidos critérios para geração das páginas de saída, de forma a equiparar a clareza, legibilidade e estruturação de

documentos gerados pelo SADG com aqueles gerados através do editor gráfico ou manualmente. Obviamente, uma equiparação total exigiria um processamento absurdo; mas a capacidade do ser humano de rearranjar infinitamente os símbolos nas páginas pode ser menos conveniente que uma distribuição automatizada e eficiente que siga critérios estáveis e coerentes em todas as páginas do documento. Tal meta objetiva fornecer ao usuário documentos com boa legibilidade, com a vantagem da automação.

A modularidade do sistema proposto deve ter em mente a definição de módulos com funções bem definidas, evitando partilhar funções entre si. Como pode-se observar do que se segue, tal característica leva à minimização da propagação de alterações (advindas de acréscimos no SDL ou aperfeiçoamentos na própria ferramenta) nos módulos componentes do sistema.

3.1 - A estruturação do SADG

A identificação das fases pelas quais passa o documento em forma PR para chegar à forma GR induz a associar a cada uma delas um módulo componente do SADG, resultando numa estrutura composta de três módulos : Compilador, Paginador e Formatador (vide Figura 3.1). Durante estas fases, ocorre a geração de arquivos auxiliares, tais como :

- Árvore de Símbolos : Contém todos os símbolos do sistema descrito em SDL-PR, caracterizando a relação de hierarquia existente entre eles. Como um dado símbolo pode ter vários filhos, criaram-se três apontadores associados a cada símbolo : pai, primeiro filho e irmão lateral à direita. Desta forma, para obter-se os filhos de um dado símbolo, basta obter o seu primeiro filho, e a partir daí seguir os apontadores aos irmãos deste.

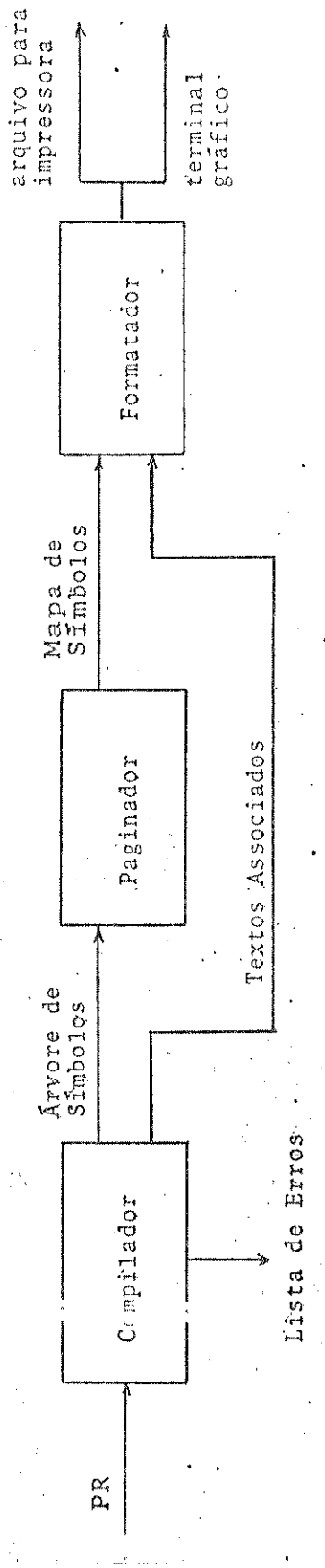


Figura 3.1 - Módulos do SADG

- Textos Associados : Contém os textos associados ao nome dos símbolos, comentários associados, e nomes dos ramos de decisão.

- Lista de Erros : Listagem indentada do fonte SDL-PR acrescida dos erros detetados durante a compilação.

- Mapa de Símbolos : Árvore de Símbolos acrescida de delimitação de páginas, e de informações de posição que os símbolos devem ocupar nas páginas do documento de saída (em impressora ou terminal de vídeo gráficos).

O módulo Compilador tem como função a análise léxica, sintática e semântica do texto em SDL-PR, tendo como saída :

- listagem do programa em PR com os erros encontrados;

- representação hierárquica do sistema descrito em SDL, através do arquivo da árvore de símbolos, auxiliado pelo arquivo complementar de textos associados (ambos criados caso não sejam encontrados erros sintáticos ou semânticos durante a compilação).

O módulo Paginador é responsável pelo mapeamento dos símbolos nas páginas, pela substituição de pares de label-join por linhas de conexão, e pela ordenação e referência das páginas de origem e destino dos joins e labels. Sua saída consiste na inclusão de informações de página, coluna e posição dos símbolos no seu arquivo de entrada (Árvore de Símbolos), vindo a constituir o arquivo do Mapa de Símbolos.

O módulo Formatador realiza a hifenação e a centralização dos textos associados, e provê a geração do arquivo GR para impressão em impressora gráfica ou provê a saída em terminal de vídeo (utilizando notação vetorial e dispensando portanto a necessidade de pré-processamento das páginas de saída). A associação de cada um dos elementos possíveis do desenho de saída com um procedimento específico torna o corpo principal do Formatador independente do terminal de saída.

Pode-se observar que a concentração das funções afins num dado módulo leva a interfaces mais estáveis em relação a possíveis alterações. Tal fato pode ser exemplificado imaginando-se possíveis alterações em cada um dos módulos :

- O módulo Compilador sofre quase todo o impacto advindo do acréscimo de conceitos no SDL-PR, tais como : procedimento e definição de dados. Sua saída é conceitualmente invariável, ou seja, uma árvore com a representação hierárquica dos símbolos componentes do sistema.

- O módulo Paginador é função basicamente dos critérios adotados para distribuição de símbolos e linhas de conexão direta; têm menor relevância os tipos de símbolos que compõem a árvore gerada pelo Compilador. Alterações significativas neste módulo estão ligadas com a sua eficiência no mapeamento das páginas do documento de saída.

- O módulo Formatador sofre acréscimo de novos procedimentos específicos (sem alterar as já existentes) se novos símbolos ou elementos são acrescentados ao desenho. O conteúdo de tais procedimentos é função dos símbolos acrescentados e do pacote gráfico utilizado.

3.2 - Recursos de desenvolvimento e fatores de desempenho

O SADG se utiliza dos seguintes recursos :

- máquina para desenvolvimento : VAX-750;

- linguagem de programação : PASCAL, suportada por compilador com geração de código otimizado e debug através de referência simbólica;

- terminais de saída : printer plotter LXY-11 e terminal de vídeo VT-125.

Existe disponibilidade de utilização do pacote gráfico de suporte PLXY11, mas o desenvolvimento de um pacote próprio permite otimizações consideráveis em relação à legibilidade e tempo de processamento.

Dentre os fatores de desempenho, poderíamos enumerar :

- qualidade do produto final;
- tempo de processamento;
- expansão dinâmica de dados;
- tamanho de código gerado.

Face ao que foi exposto, busca-se otimizar os dois fatores de desempenho mais importantes : qualidade do produto final e tempo de processamento.

A qualidade do produto final é vital para a viabilização da utilização do SADG, como já foi exposto, e alicerça-se em quatro fatores :

- estruturação : a ordem de aparecimento de estruturas externas e internas do documento PR deve ser seguida no documento GR;
- ordem prevista pelo usuário : o ramo principal de uma decisão deve ser aquele que o usuário descreveu primeiro no arquivo PR, de forma a permitir que o usuário possa prescrever sua ordem preferencial de importância no aparecimento dos ramos;
- legibilidade : a fragmentação do desenho deverá ser minimizada, e os conectores e labels deverão conter referência de página de origem ou destino;
- ocupação da página : deve-se buscar a maior ocupação possível da página durante a distribuição dos símbolos, sem contudo comprometer a minimização da fragmentação.

A conversão do software para executar em micro computadores, utilizados como workstations, deve pressupor a existência de um compilador PASCAL que suporte acesso direto e permita contornar

possíveis restrições de tamanho contíguo de módulos menores, com visibilidade de dados estendida a todos os módulos de código, e com a possibilidade de alocar grandes espaços de memória de dados através da utilização de memória dinâmica.

3.3 - Proposta de SDL-PR estruturado

Uma linguagem estruturada compõe-se em geral de dois grupos de comandos que ocasionam saltos condicionais : os de decisão e os de loop condicional. Os comandos de decisão (exemplificados em PASCAL pelas palavras chave : IF-THEN-ELSE e CASE-OF) são perfeitamente substituíveis pela estrutura Decision do SDL-PR. No entanto, os comandos de loop condicional (exemplificados em PASCAL pelas palavras chave : FOR-TO/DOWNTO, WHILE-DO e REPEAT-UNTIL), ao serem substituidos por estruturas do SDL-PR, dão origem a uma notação que não corresponde necessariamente a uma linguagem estruturada.

Este fato ocorre porque um comando de loop condicional é representado em SDL-PR através de três estruturas em conjunto : Decision, Join e Label. Desta forma, o usuário é obrigado a utilizar labels e conectores, o que acabará inevitavelmente por desestruturar a linguagem. Vide na figura 3.2, a representação em SDL-GR dos comandos do PASCAL citados acima, cuja correspondência em SDL-PR é a seguinte :

- IF THEN

```
DECISION c1;
(s): TASK t1;
(n):
ENDDECISION;
```

- IF THEN ELSE

```
DECISION c1;
(s): TASK t1;
(n): TASK t2;
ENDDECISION;
```

- CASE OF

```
DECISION c1;
(r1): TASK t1;
(r2): TASK t2;
(r3): TASK t3;
ENDDECISION;
```

- WHILE

```
l1: DECISION c1;
    (s): TASK t1;
         JOIN l1;
    (n):
    ENDDECISION;
```

- REPEAT UNTIL

```
l1: TASK t1;
    DECISION c1;
    (n): JOIN l1;
    (s):
    ENDDECISION;
```

SDL-PR é estruturado através de estados, como linguagem descritora de sistemas. No entanto, no nível mais baixo de descrição, a partir do qual faz-se a codificação do bloco funcional, é útil a opção de uma descrição onde labels e joins

possam ser evitados, pois tal forma poderia passar facilmente para uma codificação estruturada. Esta estruturação torna-se fundamental quando se utiliza ferramentas que geram CHILL através de SDL-PR.

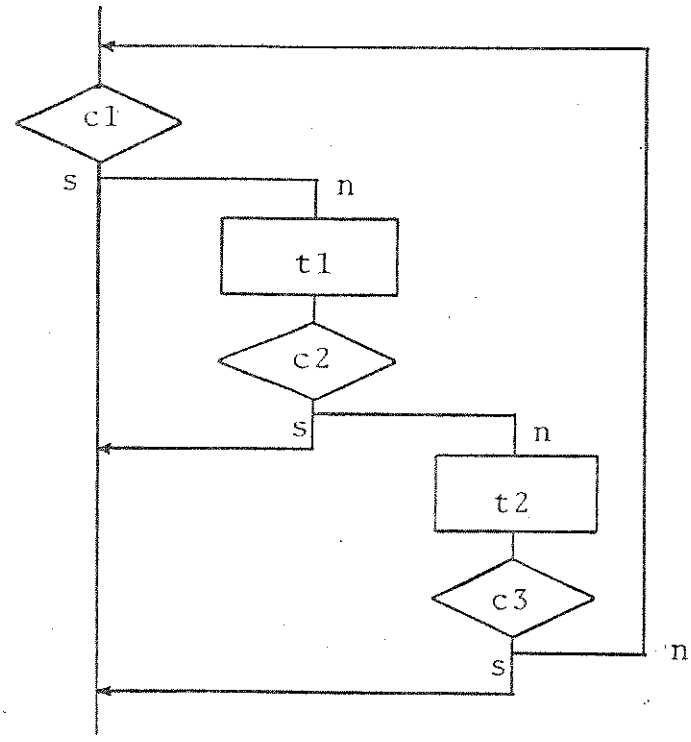
Para tal, é desejável a disponibilidade em SDL-PR de uma única estrutura que possa corresponder a loops estruturados, cuja conversão para SDL-GR redunda automaticamente nos três símbolos gráficos : decision, label e join. Tal estrutura existe na implementação de alguns compiladores da linguagem PASCAL e se chama loop, sendo caracterizada pelas palavras chaves : LOOP, EXITIF e ENDLOOP. A irrestrição no número de condições de saída (exitif) configura o caso mais geral; com apenas um exitif no início do loop tem-se uma estrutura semelhante ao WHILE, e uma semelhante ao REPEAT com um exitif no final do loop. Vide a Figura 3.3, onde estão representados em SDL-GR o loop genérico e o loop substituindo WHILE e REPEAT, cuja correspondência no SDL-PR estruturado proposto é a seguinte :

- loop genérico

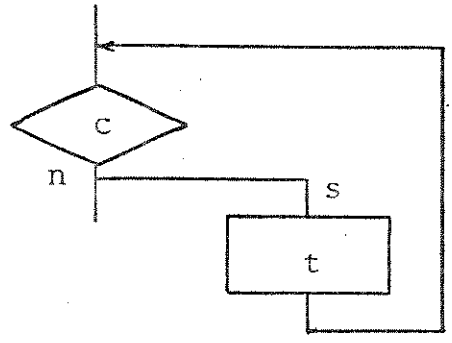
```
LOOP;
EXITIF c1;
TASK t1;
EXITIF c2;
TASK t2;
EXITIF c3;
ENDLOOP;
```

- WHILE

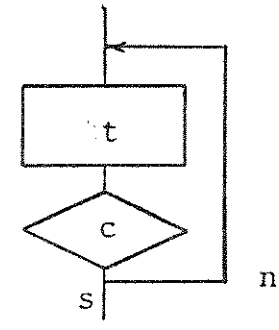
```
LOOP;
EXITIF not c;
TASK t;
```



Loop Genérico



Loop para WHILE



Loop para REPEAT

Figura 3.3 - Representação de loops em SDL-GR

```
ENDLOOP;
```

- REPEAT UNTIL

```
LOOP;
TASK t;
EXITIF c;
ENDLOOP;
```

A estrutura loop, foi por mim acrescentada ao SDL-PR como uma opção para uma descrição estruturada. Sua conversão para SDL-GR utiliza os símbolos : decision, label e join. Com o acréscimo da estrutura loop, é possível [illegible] todos os casos de utilização de labels e conectores, como norma de descrição de um dado projeto, o que se torna particularmente útil para o documento de nível mais detalhado da partição, a partir do qual faz-se a codificação dos blocos funcionais de software.

4 - MÓDULO COMPILADOR

Neste capítulo são analisados aspectos relevantes em relação à escolha do método de implementação do compilador [BAUE76] [BARR79], de sua recuperação de erros e geração da representação hierárquica do sistema (denominada árvore de símbolos). São apresentados vários métodos existentes para recuperação de erros, culminando com a escolha do método julgado mais conveniente para esta aplicação e com a definição das premissas e soluções adequadas ao subconjunto SDL utilizado (vide item 2.1). São discutidos também alguns vínculos relacionados com a geração da árvore de símbolos.

4.1 - Escolha do método de implementação do compilador

A estruturação de um compilador deve levar em conta os parâmetros significativos e os compromissos resultantes das metas estabelecidas. Num projeto genérico, pode-se apontar como parâmetros básicos :

- confiabilidade : se o compilador não funciona, as medidas de eficiência perdem o sentido;
- eficiência : na compilação (quantidade de memória e tempo de processamento) e no código gerado;
- disponibilidade num curto espaço de tempo;
- generalidade e adaptabilidade;
- diagnóstico de erros eficiente;
- documentação completa.

Existem compromissos entre fatores que usualmente se otimizam em sentidos opostos, tais como :

- eficiência de compilação x eficiência do código gerado;

- tempo de processamento x diagnose;

- eficiência x generalidade;

- confiabilidade x complexidade.

O compilador implementado não tem vínculos de generalidade e a sua rápida disponibilidade é um fator preponderante. O sub-conjunto da linguagem para o qual será implementado (sem definição de dados e com estruturas bem definidas por palavras chaves) e a saída que se deseja do compilador (árvore com os símbolos do sistema descrito em PR) conferem-lhe características bem peculiares se comparado com um compilador para uma linguagem estruturada genérica. A maior parte do esforço no seu desenvolvimento concentrar-se-á em :

- verificação de referências não declaradas, duplicidade de declaração ou declarações não referenciadas (num dado escopo do programa em SDL-PR);

- recuperação de erros confiável com descarte mínimo de símbolos;

- redução do tempo de compilação.

É importante observar que o código gerado não requer otimização, pois os símbolos da árvore têm correspondência biunívoca com os comandos de SDL-PR e a relação de hierarquia entre as estruturas está perfeitamente caracterizada, de tal forma que o compilador não tem nenhum grau de liberdade, e gerará uma árvore de símbolos bem definida.

O subconjunto da linguagem SDL-PR a ser implementado, ao se excluir a definição de dados e expressões, concorre para uma considerável simplificação da análise léxica e semântica, tornando a análise sintática e a recuperação de erros, os únicos parâmetros significativos na escolha do método de construção do compilador.

A análise sintática é responsável pelo reconhecimento das sentenças (e suas estruturas) utilizadas no programa fonte; a sua

complexidade depende basicamente do tipo de regras de produção (regras de transcrição aplicadas a símbolos não terminais) usadas para definir a linguagem. O CCITT fornece o diagrama sintático da linguagem SDL-PR, o que nos sugere a implementação de um analisador descendente recursivo. Para minimizar o esforço computacional, buscar-se-á um método que se limite a um passo de análise e utilize apenas um símbolo de análise, para que o seu custo se aproxime de uma função linear do comprimento do texto de entrada (n), ou nlogn no pior caso.

A maneira mais simples de evitar análise com retrocesso é fazer com que o algorítmo sempre tome a decisão correta quanto à produção a ser aplicada. Uma classe muito simples de gramáticas para as quais isto pode ser feito é obtida impondo-se as seguintes restrições [KOWA83] :

- toda produção é da forma A ::= XQ, onde X é terminal;

- se A ::= X1Q1/X2Q2/.../XnQn, são todas as alternativas para o não terminal A, então os terminais Xi são distintos entre si.

A classe de gramáticas que satisfaz esta restrição é denominada LL(1), por executar análise sintática da esquerda para a direita, gerando uma árvore de derivação a partir de símbolos à esquerda; e toda gramática pertencente a esta classe é não ambígua.

As gramáticas LR(1) (com árvore de derivação à direita), também possibilitam recuperação sem análise com retrocesso e com um símbolo de análise, mas os compiladores ascendentes são implementados por tabela de análise, e sua aplicabilidade está na generalidade com que podem ser construídos geradores de analisadores sintáticos ascendentes, que aceitam outras linguagens como entrada, com modificações apenas nas tabelas de análise. No entanto, o interesse se volta para a implementação de um compilador específico e otimizado, utilizando-se do grafo

sintático disponível da linguagem, cuja gramática se adapta às condições das gramáticas LL(1).

O método descendente pode ser implementado através de pilha explícita [KOWA83] (para armazenar a árvore em tratamento) ou da associação de procedimentos recursivos aos símbolos terminais da linguagem. O procedimento recursivo presta-se bem à implementação de analisadores sintáticos descritos por grafos sintáticos; além de ser muito flexível, permitindo executar ações em pontos intermediários que não correspondem às produções (por inserção de comandos apropriados nos pontos correspondentes aos procedimentos recursivos).Em busca de maior eficiência, a recursividade pode ser conjugada com outros métodos, tais como a análise de precedência de operadores para o reconhecimento de expressões, o que elimina grande parte das chamadas recursivas, otimizando o analisador sintático. A gramática em questão não contém expressões e permite a implementação puramente recursiva, sem perda de eficiência.

A implementação em pilha explícita pode favorecer a recuperação de erros nas gramáticas LL(k) (k símbolos de análise), mas existem métodos de recuperação de erros simples e eficientes que dispensam tal utilização, os quais serão vistos adiante.

Seria possível formalizar o reconhecimento da linguagem através de sintaxe dirigida por tabela, que possui um alto grau de flexibilidade não presente em esquemas de analisadores sintáticos específicos (usados nas gramáticas LR). Tal flexibilidade é em geral requerida nas linguagens extensíveis e não será importante na implementação do conjunto total do SDL-PR, uma vez que a definição de dados e expressões é bem específica, e altera mais profundamente a análise léxica e semântica.

Sob o ponto de vista de recuperação de erros, a comparação entre as gramáticas LR(1) e LL(1) (supondo implementação com pilha explícita, pois a recursiva não se presta a LR), caracteriza uma

vantagem no caso da LL(1), devido a informação a mais que este tipo de gramática utiliza. No caso LR, não se sabe a sentença à qual a redução levará, até que todos os termos desta sejam encontrados; enquanto que na LL, sabe-se a árvore que está sendo seguida. Esta característica confere maior facilidade de recuperação à LL (embora a detecção do erro seja idêntica em ambos).

4.2 - Análise Léxica, Sintática e Semântica

A validação do programa de entrada é feita sob três aspectos :

- análise léxica : reconhecimento dos símbolos da gramática e preenchimento da tabela de símbolos;
- análise sintática : reconhecimento das sentenças;
- análise semântica : validação das regras adicionais que não são englobadas pela gramática.

Usualmente a análise léxica é implementada na forma de um procedimento utilizado pela análise sintática. Esta estruturação traz as seguintes vantagens :

- pode ser otimizada em separado (possibitando até o uso de linguagem de montagem), o que é muitas vezes desejável, devido ao tempo substancial consumido pela análise léxica durante a compilação;
- as regras de formação dos átomos permitem técnicas de análise mais simples e eficientes;
- solução de problemas que possam ocorrer devido a idiossincrasias da linguagem, tais como palavras chaves como identificadores, tornando o analisador sintático mais simples;
- concentração dos símbolos terminais numa única rotina, tornando mais simples eventuais modificações de representação;

- realização de tarefas adicionais tais como : leitura e impressão do programa fonte (incluindo processamento de fim de linha e fim de arquivo), criação da tabela de símbolos, indentação, etc.

Em geral, a análise léxica preenche, através das declarações, a tabela de símbolos para testar futuras referências, e realiza a consistência das palavras reservadas e dos simbolos pré-declarados, dentro de um dado escopo. No subconjunto que vamos implementar não haverá pré-declaração de variáveis, mas será necessário construir uma tabela de símbolos ao longo da compilação para verificar declaração e referência a labels, sinais de entrada, estados, processos e blocos, dentro de um dado contexto (ou seja processos de blocos diferentes podem ter o mesmo nome, idem para estado e sinais de entrada). Essa consulta à tabela para reconhecer palavras reservadas e validar referências representa na realidade uma dependência de contexto, que as linguagens de implementação têm em geral; isto não impede, no entanto, o uso de um modelamento de gramáticas livres de contexto, que simplifica bastante o analisador sintático.

No Compilador do SADG, a análise léxica realiza as seguintes funções :

- retirar brancos separadores;
- reconhecer palavras reservadas, nomes (associados a símbolos ou comentários), identificadores (associados a labels), texto livre (entre delimitadores de texto) e caracteres especiais;
- editar o programa fonte, com as mensagens de erro fornecidas pela recuperação de erro;
- detectar fim de linha e fim de arquivo.

A análise sintática orienta-se a partir do grafo sintático fornecido pelo CCITT, sua implementação top-down recursiva é simples e já foi suficientemente avaliada acima.

A análise semântica restringe-se a :

- garantir que toda transição terminará com um símbolo pertencente ao conjunto dos finalizadores de transição;

- garantir que uma decisão com todos os ramos terminados funciona como finalizador de transição.

A rotinas de recuperação de erro e geração de código podem ser chamadas de qualquer ponto da análise.

4.3 - Recuperação de Erros

A recuperação de erros pode se realizar a nível léxico, sintático ou semântico; vamos enumerar as situações usuais em que ela pode ocorrer em cada uma destas fases, num compilador genérico.

Na fase léxica os erros mais comuns são :

- identificador com número de caracteres não permitido pela implementação;

- identificador ou número mal escrito, ou presença de caracter estranho ao vocabulário;

- não detecção do fechamento de comentário;

- arquivo termina ao procurar um símbolo.

A recuperação na fase léxica em geral consiste em :

- pular caracteres errados até achar um átomo (símbolo indivisível);

- tentar achar outro prefixo que combine com uma lista de átomos possíveis naquele momento.

Na fase semântica os erros mais comuns são :

- inconsistência dos atributos de um identificador com o seu uso no programa fonte (ex : [illegible] usado como variável, variável como index de array, procedimento como função);

- ordem de declaração de parâmetros em procedimentos;

- limitações de implementação para tamanhos de arrays e números de variáveis;

- overflow em operações aritméticas.

Em geral os erros semânticos não são difíceis de diagnosticar (porque o programa está estruturalmente correto; sendo possível, em geral, identificar a intenção do programador), e os erros léxicos levam a procedimentos simples. A preocupação com os métodos de recuperação de erros está mais voltada para os erros sintáticos, cuja recuperação pode se tornar complexa :

- às vezes é impossível determinar a intenção do programador (ex : parêntesis ausente em expressão, que pode ser colocado em vários lugares);

- às vezes um erro pode não ser detectado quando ocorre (ex : concatenação de palavras chave com identificador gerando outro tipo de identificador com lado direito da expressão que sucede a palavra chave), e se houver uma diagnose prematura poderá ocorrer uma recuperação incorreta (ex : IFA = B THEN A := A + B; pode ser diagnosticado apressadamente como IFA := B).

Apesar dos exemplos genéricos utilizados não condizerem com os casos reais encontrados no subconjunto utilizado de SDL, é possível ocorrerem situações idênticas onde não se poderá ter certeza da intenção do programador ou casos nos quais a diagnose precoce poderá levar a erros na recuperação. Tais casos podem ocorrer, por exemplo, devido à omissão de uma palavra chave inicial de uma dada estrutura, que poderá ser precedida por um nome idêntico a uma outra palavra chave, sugerindo uma estrutura errada, ou deixando o compilador sem condições de decidir se a estrutura que se inicia é um bloco, um procedimento ou uma macro.

Qualquer esquema de recuperação de erros [WIRT76], implementado com esforço razoável, não será adequado para

manipular determinadas construções erradas. As características importantes de um bom compilador são :

- nenhuma sequência de entrada o levará ao colapso;
- todas as construções ilegais, segundo a definição da linguagem, serão detectadas e assinaladas;
- erros que podem ocorrer com frequência, causados por falta de atenção ou compreensão errônea da linguagem, deverão ser diagnosticados corretamente, sem causar a emissão de mensagens espúrias;
- o encadeamento de mensagens de erro deve ser minimizado e todo trecho descartado deve ser assinalado.

A arte de adivinhar o que o programador pretende, diagnosticando corretamente e recuperando-se do erro, é muitas vezes intrincada, pois depende fortemente dos fatores que influenciam a mente humana (capazes de atribuir pesos diferentes a diferentes símbolos não terminais, como por exemplo o terminador [;]). Antes de definir a recuperação de erros utilizada, serão vistas algumas técnicas a respeito.

4.3.1 - Métodos de Recuperação de Erros

Neste item comenta-se sucintamente um método de recuperação de erros útil em qualquer tipo de compilador, conhecido como correção de ortografia (spelling correction), e alguns métodos relacionados com analisadores sintáticos top-down, citados em [SILV81]. O objetivo não é detalhar o funcionamento de tais métodos, e sim colher subsídios para comparar o custo, a eficiência e a aplicabilidade dos mesmos nesta implementação, de forma a definir o esquema de recuperação de erros mais conveniente.

4.3.1.1 - Correção de ortografia

Existem situações nas quais o compilador pode suspeitar que um identificador foi escrito com sílabas erradas e, neste caso, comparar com a tabela de símbolos e decidir qual o símbolo correto. Tal método é conhecido como correção de ortografia, e os candidatos a este tipo de correção podem ser detectados em vários níveis :

- durante a análise sintática, quando o próximo símbolo esperado é constituído por um conjunto de palavras chaves, ou em uma expressão booleana, quando o símbolo esperado é um operador lógico;

- durante a análise semântica, quando um símbolo é usado num contexto que requer outro tipo de símbolo;

- referências não declaradas ou declarações não referenciadas.

Após selecionar o candidato à correção de ortografia, deve-se determinar a qual identificador da tabela de símbolos ele corresponde.

O primeiro trabalho neste sentido foi de Freeman (63), com uma função que utiliza informações tais como : número de letras que combinam, número de letras que combinam após uma ou duas transposições de caracteres, e número de letras que combinam após levar em conta erros comuns (ex : número 0 à letra O, e número 1 à letra l).

Esta técnica foi substituída por Morgan (70) por um algorítmo mais eficiente, embora menos poderoso, baseado na evidência de que 80 por cento dos erros em sílabas ocorrem em uma destas quatro classes :

- uma letra errada;

- uma letra omitida;

- um caracter extra inserido;

- dois caracteres transpostos.

A correção se baseia em dois passos :

- selecionar o subset de identificadores que podem substituir o identificador a ser corrigido;

- testar o identificador candidato com cada elemento do conjunto, tentando torna-lo idêntico através das 4 operações acima descritas.

4.3.1.2 - Método do Pânico

Os métodos apresentados a seguir são aprimoramentos do método do pânico, cujo procedimento é o mais simples possível. Ao detectar um erro, o compilador despreza os símbolos seguintes, até encontrar o início de uma nova estrutura, para que possa prosseguir. Apesar de sua simplicidade, tem o inconveniente de descartar muitos símbolos, uma vez que todas as estruturas internas a uma dada estrutura não reconhecida são descartadas. Além disso sua recuperação pode ser prematura, ou seja, realizada num ponto incorreto.

4.3.1.3 - Turner (1977)

Este método procura corrigir o erro, supondo que os erros encontrados são de primeira ordem (inserção, omissão ou erro na transcrição de um símbolo). Quando o símbolo esperado for omitido, todo o restante será pulado; mas o método supõe que isto ocorra com baixa frequência.

Pode se tornar ineficiente quando dois símbolos vizinhos estão incorretos, o que não permitirá a correção e poderá implicar

em alto descarte de símbolos, ou recuperação em ponto errado, introduzindo erros (neste caso pode ser pior que o método do pânico). Ao detectar um erro, assinala-o e ativa uma indicação para inibir mensagens de erro espúrias, até a recuperação do sincronismo.

4.3.1.4 - Wirth (1976)

Verifica se o símbolo em análise pertence ao conjunto dos possíveis sucessores ao último símbolo analisado, ao percorrer a árvore de produção. Se o símbolo não pertencer ao referido conjunto detecta um erro, e os símbolos seguintes serão descartados até que apareça um símbolo pertencente ao conjunto dos símbolos de parada, naquele ponto da análise [WIRT76].

O conjunto dos símbolos de parada, num dado ponto da análise, é constituído de palavras chaves e símbolos terminadores referentes às estruturas externas àquela na qual o erro foi detectado, de tal forma que o número de símbolos de parada vai sendo aumentado à medida que se penetra em procedimentos mais internos. Este aumento dos símbolos de parada pode ocasionar recuperação prematura, e o analisador pode se comportar como se um símbolo esquecido (ou esperado) tivesse sido encontrado, o que pode ser desastroso (ex : palavras chaves em textos livres cuja abertura não foi reconhecida).

Pode se tornar pouco eficiente se o erro for omissão de símbolo; no entanto, a omissão em geral está restrita a símbolos que possuem função puramente sintática (ex : [;]), e portanto não representam ações e podem ser previstos pelo método, que passa a funcionar satisfatoriamente.

Aimman (1977) ajustou o conjunto de símbolos de parada de acordo com a experiência do uso da linguagem ou características da mesma, estendendo a maneira sistemática que Wirth propõe.

4.3.1.5 - Hartman (1977) e Pemberton (1980)

Esta técnica é uma variação do método de Wirth, fundamentada na derivação dos símbolos de parada através do grafo sintático da linguagem, tornando-se desta forma um método específico para compiladores top-down recursivos.

O conjunto dos símbolos de parada irá conter operadores do grafo analisado e palavras chaves de qualquer outro grafo que contenha o grafo em tratamento. Suponha, por exemplo, as produções : S ::= Ab, e A ::= Ca. Pelo método de Wirth, ao ser chamado [A], a partir de [S], o conjunto dos símbolos de parada é acrescido de [b], e caso não seja encontrado algum símbolo que inicie [C], é feito o retorno a [S] sem procurar encontrar [a], o que não ocorre no método de Hartman.

Pemberton, mais tarde, generalizou o método para os três casos de construções possíveis no grafo : simbolos em série, em paralelo ou em realimentação (supondo a possibilidade de símbolos não terminais nos três casos).

4.3.1.6 - Irons (1963)

O método busca completar cadeias incompletas com uma cadeia intermediária até permitir a continuação da análise do símbolo atual. Sua aplicação não é inteiramente possível com implementação recursiva, porque nem sempre a árvore sintática está disponível.

4.3.1.7 - Fisher, Milton e Quiring (1977)

Este método produz um programa sintaticamente correto para uma gramática LL(1) corrigível por inserção (desde que a detecção do erro seja imediata à sua ocorrência, o que é comum em tais gramáticas), apesar de ser necessário fornecer, a priori, um custo de inserção para cada símbolo terminal.

Armazena todas as construções possíveis que possam derivar o símbolo errado e escolhe a de menor custo. Nada do texto é descartado e a cadeia inserida é a de menor custo; mas não a única, o que poderá ocasionar erros espúrios.

As tabelas de custos podem ser muito grandes (necessitando armazenamento em memória secundária) e de difícil formalização (pois varia conforme a linguagem compilada).

4.3.1.8 - Ghezzi (1976)

Este algoritmo pode ser gerado automaticamente a partir da descrição formal da linguagem e tem como filosofia geral tentar primeiro uma ação de reparo local (considerando apenas omissão, inserção ou má escrita de símbolos); se esta falhar, tomará uma ação de recuperação com descarte de símbolos.

A recuperação tem como objetivo descartar o menor número de símbolos possível para continuar a análise. O reparo local pode ocasionar inserção de erros espúrios, pois as possibilidades não são únicas para um dado erro.

4.3.1.9 - Setzer (1981)

Este método é análogo ao de Ghezzi, diferindo nos seguintes pontos :

- critério de aplicação das estratégias de correção;

- maneira de verificá-las (Setzer representa a gramática sob forma de listas, e Ghezzi, sob forma de tabelas).

Considere a cadeia de entrada B = A1 A2 ... Aj Aj+1 ... An, suponha detecção de erro no símbolo Aj e como conteúdo da pilha de análise X = X1 X2 ... Xi, sendo Xi o topo da pilha. O método executa as seguintes estratégias :

- E : elimina o símbolo se Aj+1 pertence a PRI(Xi);

- I : insere símbolo se entre Aj-1 e Aj deveria haver um símbolo de PRI(Xi) e, para tal, verifica se Aj pertence ao conjunto de símbolos sucessores de alguns símbolos de PRI(X);

- T : símbolo trocado Aj, aplicar estratégia I com Aj+1 ao invés de Aj;

- D : busca de delimitador para todo não terminal contido na pilha de análise, verifica se Aj pertence ao conjunto de símbolos sucessores, ou seja para todo não terminal Xk verifica-se se Aj pertence a SUC(Xk); caso pertença a SUC(Xk), faz X = X1 ... Xk-1.

Estas estratégias são testadas sequencialmente e, se nenhuma delas corrigir o erro, ignora-se Aj e passa-se a testar Aj+1.

Há uma preocupação evidente de se descartar o menor número de símbolos de entrada, sendo que a quarta estratégia é a que garante menor eliminação de símbolos. Este método é restrito para uma sub-classe de LL(1), denominada ESLL(1).

4.3.1.10 - Pai (1978) e Kieburtz (1980)

Segue a filosofia adotada por Ghezzi e Setzer, buscando inicialmente a correção local, partindo para a recuperação global caso a primeira não tenha obtido sucesso.

A recuperação de contexto global é feita em dois estágios :

- a cadeia de entrada é percorrida e os átomos são descartados até que seja achado um átomo pertencente a um conjunto específico, denominado conjunto de símbolos fiduciais;

- o contexto da pilha de análise é substituído por uma cadeia, a partir da qual o analisador pode aceitar o restante do texto de entrada, que agora se inicia com o símbolo fiducial encontrado no primeiro estágio.

Existem dois problemas decorrentes de se olhar para frente :

- a cadeia verificada pode ter outro erro;

- não existe um número fixo de símbolos que possa resolver todas as ambiguidades (ex : expressão longa).

Na recuperação é útil saber :

- se dois símbolos terminais podem ocorrer contíguos;

- se um símbolo terminal pode aparecer em uma cadeia derivada de um símbolo não terminal.

A definição dos símbolos fiduciais é insatisfatória e imprecisa, necessitando-se recorrer ao conhecimento prático da linguagem; portanto não é conseguida a automação desejada, mas é o caminho mais inovador tentado até agora no campo de recuperação de erros. Tal método pressupõe um analisador sintático sem análise com retrocesso e com uso de pilha explícita.

4.3.2 - Definição do esquema adotado de recuperação de erros

Um método de recuperação de erros muito elaborado, envolvendo a retenção de mais de um símbolo para análise ou buscando corrigir os erros cometidos na maioria dos casos, pode requerer um esforço computacional inaceitável ou aumentar substancialmente o grau de complexidade da sua implementação.

É conveniente escolher um método simples mas que traga um compromisso de eficiência com a linguagem utilizada. Tal compromisso se alicerça em dois itens básicos :

- o não reconhecimento da palavra chave que inicia uma dada estrutura externa (blocos, processos, estados, macros e procedimentos) não poderá ocasionar o descarte da análise das estruturas internas a ela, o que levaria a desperdícios substanciais devido à estruturação do SDL-PR;

- a perda do símbolo de fechamento ou abertura de texto livre deverá ter pelo menos mais um caracter especial terminador (que de preferência tenha alta frequência de ocorrência no texto) que determine a sua recuperação, de forma a minimizar a recuperação das declarações seguintes.

Fica claro que a meta não é corrigir os erros em sua maioria, mas recuperar o sincronismo da análise o mais próximo possível do erro detectado, evitando descartar a análise das declarações subsequentes ou das estruturas internas, de tal forma que o erro fique tão localizado que o programador possa facilmente identificá-lo. Desta forma, a meta prioritária será a não propagação da ausência de análise, e a secundária será a caracterização mais adequada do erro ocorrido.

Neste contexto, o conceito de símbolos de parada de Wirth, com os acréscimos propostos por Hartman e Pemberton é o que mais se adapta a estas necessidades, pelos seguintes motivos :

- presta-se à implementação top-down recursiva;

- é simples e de baixo custo;

- fornece uma recuperação de erros que, com algumas adaptações, atenderá as premissas da meta estabelecida.

Os demais métodos são menos adequados para uma implementação recursiva ou proporcionam um baixa taxa de custo/benefício adicionais. O método de correção de ortografia poderá ser

utilizado conjuntamente, em implementações futuras, nas situações em que for possível inferir, através da transposição de caracteres, a palavra chave que inicia uma dada estrutura opcional entre estruturas em paralelo (o que representa um ganho justificável, pois capacita o analisador a identificar qual das estruturas em paralelo deve escolher).

Em relação ao analisador léxico pode-se observar que, devido à alta taxa de incidência do símbolo terminal separador [;], a sua exclusão do conjunto de caracteres especiais permitidos no texto livre não se torna uma restrição relevante (devido ao grande número de caracteres especiais), e passa a se constituir num eficiente símbolo de parada para a detecção de fechamento de texto livre. Este procedimento cumpre a segunda premissa das duas que compõem a meta proposta.

O cumprimento da primeira premissa implicará a utilização do conceito dos símbolos de parada, o que permitirá analisar estruturas internas a uma dada estrutura cuja palavra chave de abertura não tenha sido identificada.

Para simplificar inicialmente a análise, as declarações de macros e procedimentos são excluídas e a seguir adaptadas ao método proposto.

Para a análise que se segue são conceituadas como estruturas mais internas aquelas que não contêm estruturas internas a si (ex : task, output, comment, create, etc); e como estruturas mais externas, aquelas que contêm estruturas internas a si (ex : system, block, state, decision, etc).

Desta forma pode-se notar que nas estruturas mais externas do SDL-PR, tais como bloco, processo e estado, não há em geral opções em paralelo no grafo sintático; ou seja, um bloco deve conter um processo (ou outro bloco, que é a única exceção) e um processo deve conter um estado. O estado terá como opções em paralelo :

save, sinal de entrada e sinal contínuo; o sinal de entrada e o sinal contínuo têm basicamente a mesma estrutura subsequente.

As opções em paralelo ocorrem em geral nas ações de transição, uma vez que qualquer uma delas é possível após um dado sinal de entrada ou sinal interno a uma decisão.

Considere o exemplo do não reconhecimento da palavra chave TASK que define a estrutura interna tarefa. Como todos os elementos de transição são possíveis, será impossível descobrir que a estrutura é tarefa, sem o reconhecimento de sua palavra chave. No entanto, o descarte da estrutura tarefa está associado a apenas uma declaração, sem nenhuma estrutura interna. Isto tampouco viola a primeira premissa, que está preocupada basicamente com o descarte de uma dada estrutura, cuja recuperação leve ao descarte de outras estruturas internas a ela. É interessante observar que o não reconhecimento da palavra chave DECISION não leva ao descarte dos elementos de transição internos à decisão não reconhecida (idem para option e loop); pois tanto a decisão, como os demais comandos internos a ela, são alternativas para elementos componentes de uma transição na carta sintática de SDL-PR.

Desta forma, o cumprimento da primeira premissa significa impedir que o não reconhecimento das palavras chaves BLOCK, PROCESS e STATE leve ao descarte das estruturas internas às estruturas que representam. Isto não se torna uma tarefa difícil, pois essas estruturas mais externas estão dispostas em série; ou seja, se num dado ponto é esperado um processo e sua palavra chave não é reconhecida, o analisador pode assumir que está dentro de um processo e invocar o procedimento process, que acusará a ausência da palavra chave PROCESS e continuará analisando as estruturas internas ao processo. O fato de que um bloco possa ter internamente um outro bloco ou um processo não trará problemas de

recuperação, pois se não conseguir reconhecer as palavras chaves BLOCK e PROCESS, o analisador assumirá que está dentro de um processo, o que não o impedirá de analisar as estruturas internas, que são comuns a ambas as estruturas externas, bloco e processo.

Para que tal esquema possa funcionar, toda vez que não é reconhecida uma determinada palavra chave, os símbolos de parada serão constituídos por todas as palavras chaves que iniciam uma dada estrutura. Se a próxima palavra chave reconhecida não for de fechamento da estrutura não reconhecida ou de abertura de uma outra estrutura mais externa ou do mesmo nível, será uma estrutura interna da estrutura não reconhecida (que será suposta), e portanto será analisada ao invés de descartada.

Na análise das estruturas internas e no redimensionamento do conjunto do símbolo de parada num dado ponto, está a diferença básica da adaptação do método de Wirth, para que a primeira premissa seja cumprida.

Neste ponto, deve-se voltar a incluir a possibilidade das declarações de procedimento e macro no sistema. As alternativas de estruturas externas em paralelo poderão, então, ocorrer em três níveis :

- nível de sistema : macro, procedimento e bloco;
- nível de bloco : macro, procedimento e processo;
- nível de processo : macro, procedimento e estado.

O método de recuperação de erros utilizado é eficiente quando opções em paralelo são encontradas no grafo sintático nas estruturas mais internas, pois neste caso a estrutura pode ser desprezada (pois é constituída de uma única declaração) sem necessidade de assumir nenhuma delas como verdadeira. No caso de macro ou procedimento em paralelo com bloco, processo ou estado, deve-se assumir uma das estruturas como verdadeira (e neste caso será, respectivamente, bloco, processo ou estado), uma vez que a

primeira premissa impõe a análise das estruturas internas a uma dada estrutura não reconhecida. Este mecanismo poderá acarretar a emissão de mensagens espúrias, mas o usuário poderá associá-las facilmente ao não reconhecimento da palavra chave que define a estrutura mais externa, de forma que o tipo de erro e a sua localização continuarão bem definidos, o que é o objetivo primordial da proposta.

Quando um dado símbolo em análise é comparado com o símbolo esperado e ambos são distintos, o compilador pode escolher um dos três procedimentos :

- descartar o símbolo em análise e manter o símbolo esperado, supondo que o erro foi de inserção;

- descartar o símbolo esperado e manter o símbolo em análise, supondo que o erro foi de omissão;

- descartar ambos, supondo que houve erro de grafia.

Qualquer um dos três procedimentos funcionam bem, quando o erro corresponde à suposição associada, e são danosos quando isto não ocorre. Com o objetivo de não descartar palavras chaves que iniciem uma dada estrutura, que desempenham papel preponderante na nossa recuperação de erros, optou-se por descartar o símbolo esperado, mantendo o símbolo em análise (até o final da declaração).

Com o objetivo de simplificar o esquema de verificação de referência e declaração de labels, conectores e nextstates (que difere das linguagens de programação que exigem pré-declaração de identificadores), as mensagens de erros, relativas aos erros de referência sem declaração ou declaração múltipla, serão colocadas quando do término da respectiva estrutura (processo, procedimento ou macro), que constitue o escopo de validade de tais referências. A verificação de duplicidade de nomes de blocos, processos, estados, sinais, procedimentos, macros e ramos de decisão é feita

na ocorrência de cada nome, por comparação com os nomes anteriores de mesmo nível (ex : um dado nome de estado é comparado com os nomes dos estados definidos antes dele no mesmo processo).

A múltipla declaração de estados é assinalada apenas com mensagem de advertência, sem impedir a geração da árvore de símbolos. Desta forma, o usuário poderá caracterizar através do SDL-PR a sequência de eventos que deseja na descrição do sistema.

Os demais erros são assinalados na linha imediatamente abaixo da linha do programa fonte na qual o erro foi detectado e, se possível, na posição relativa à posição do erro na frase. Os símbolos ou caracteres (texto livre) descartados serão assinalados da mesma forma, através de um caractere especial. Os erros são assinalados por números e a mensagem com o seu significado é inserida a seguir.

4.4 - Geração de Código

Nesta aplicação, o equivalente à geração de código constitui-se na geração de uma árvore com os símbolos componentes do sistema descrito em SDL-PR; tarefa relativamente simples, uma vez que a cada declaração do SDL-PR há um símbolo associado na árvore. Em geral, a geração de código está associada com o processador utilizado e para um dado comando numa linguagem de alto nível, existem múltiplas possibilidades para a composição dos comandos em assembly correspondentes; esta é a parte mais complexa e menos formalizada nos compiladores e que determina diretamente a eficiência do módulo compilado (exigindo muitas vezes, alto esforço computacional para otimização do código gerado).

A geração da árvore de atributo dos símbolos que compõem o sistema descrito em SDL-PR é quase imediata a menos de algumas considerações.

A implementação da estrutura loop (vide Figura 4.1) traz a seguinte associação entre declaração (representado pela sua palavra chave) e símbolo :

- LOOP - símbolo de label (início do loop);

- EXITIF - símbolo de decisão com dois ramos, um deles iniciando-se com o próximo símbolo dentro do loop e o outro iniciando-se com o próximo simbolo após o loop (se o exitif for único) ou com conector apontando para label (ambos colocados automaticamente) que será inserido de forma a preceder o próximo símbolo após o loop;

- ENDLOOP - símbolo conector (para o label que inicia o loop).

A implementação da decisão deve permitir ao usuário a definição de ramos não terminados (cujo último símbolo não é terminador) ou vazios (ramo não terminado sem símbolos), para os quais deverá ser provida a inserção automática de label no primeiro ramo não terminado e conectores finalizando os demais labels não terminados. Se a decisão só tiver um ramo não terminado, não será necessária a inserção de label e conectores automáticos, mas deve prever-se o uso de decisões encadeadas onde uma decisão mais interna pode definir a terminação de um ramo de uma decisão mais externa (vide Figura 4.2).

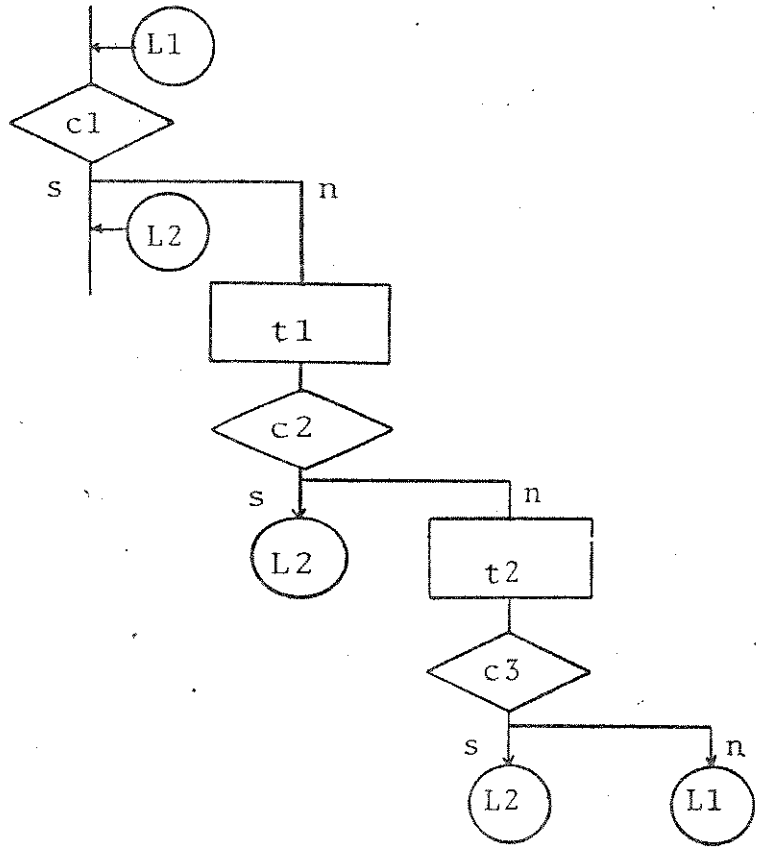


Figura 4.1 - Labels e Joins gerados automaticamente pelo Compilador na representação do loop genérico

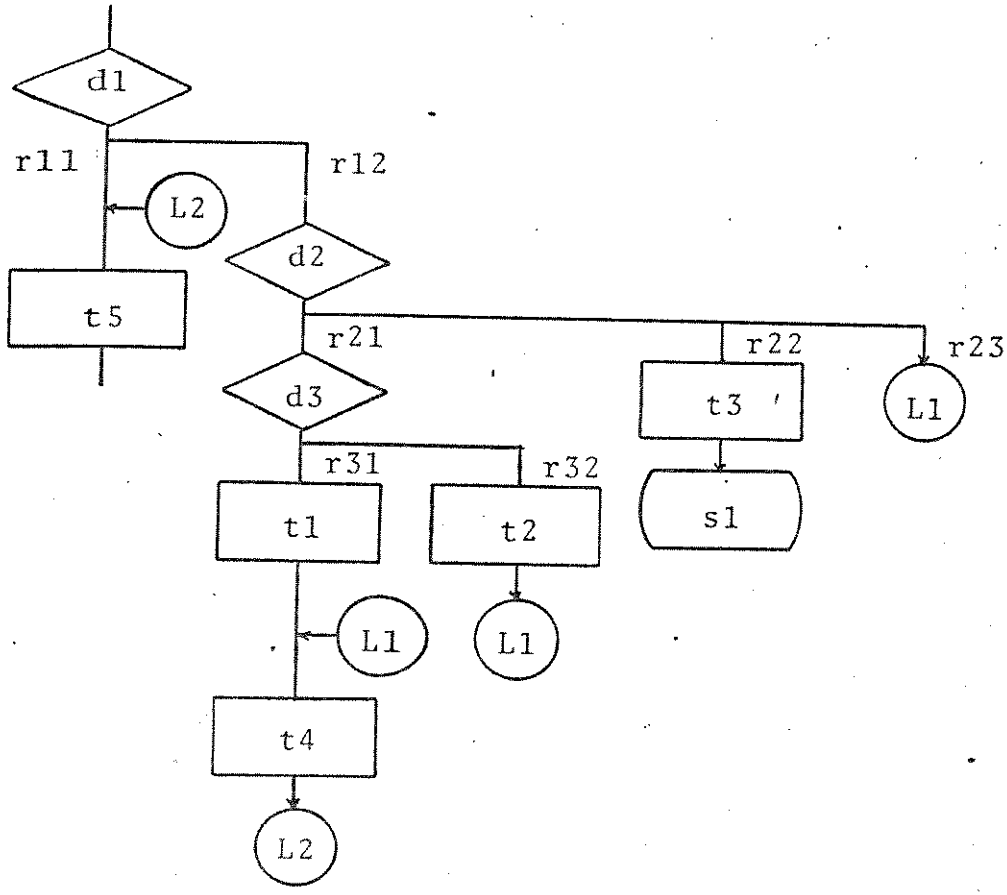


Figura 4.2 - Labels e Joins gerados automaticamente pelo Compilador na representação de decisões com ramos não terminados

5 - MÓDULO PAGINADOR

Este capítulo é o cerne deste trabalho. Nele se encontram a análise dos problemas e as propostas de solução para a distribuição de símbolos nas páginas e a substituição de labels e conectores por linhas de conexão. Todos os conceitos e as definições descritos neste capítulo, sem referência explícita de bibliografia, foram originados neste trabalho.

5.1 - Caracterização do problema

Inicialmente são definidas as condições de contorno do problema a ser equacionado, examinando-se as restrições impostas pela sua própria natureza e padronizando-se as dimensões da página de saída e os tamanhos dos símbolos gráficos adequados à utilização do SADG. A seguir são levantadas as premissas cuja análise e detalhamento levam às soluções apresentadas.

5.1.1 - Representação gráfica de fluxogramas em páginas

O projeto deste módulo define a performance do sistema no que se refere à saída gráfica. Existem limitações naturais associadas à representação em forma gráfica, seja ela feita manualmente ou de forma automatizada, que são definidas basicamente por dois fatores : limitação física da página e natureza do desenho.

A limitação física da página é que cria a necessidade da existência do Paginador; quanto menor for a página, em relação ao tamanho do sistema que deverá ser representado, maior a fragmentação e a descontinuidade da representação resultante.

A natureza do desenho, ou seja, a forma como o sistema está dividido em blocos, processos e estados, e o número de decisões

(bem como decisões encadeadas), labels e joins, constitui outro importante fator de fragmentação. O usuário traça as linhas mestras da estruturação do sistema que pretende descrever, particionando-o convenientemente e utilizando procedimentos e macros para maior simplicidade de representação, clareza e legibilidade. No entanto, a natureza da função que pretende representar é um fator que foge ao seu controle e que determina, em função de sua complexidade, fluxogramas mais ou menos complexos (sob o ponto de vista de encadeamento de ramos de decisão e conexões).

Ao desenhar manualmente o sistema na forma gráfica, o usuário é obrigado a optar pela ordem e a disposição na qual os ramos de decisão e as conexões aparecerão nas diversas páginas componentes do desenho. Nem sempre será óbvia a escolha da ordem e da disposição que minimizem a fragmentação do desenho e otimizem sua clareza.

O Paginador terá que fazer escolha semelhante; uma das metas deste trabalho é mostrar que os critérios definidos para a fragmentação automática do desenho podem aproximar sua eficiência à da fragmentação obtida manualmente, em relação à apresentação do produto (forma gráfica de saída), com a vantagem da automação e da padronização da documentação.

A escolha de critérios para distribuição dos símbolos nas páginas deverá considerar dois pontos básicos :

- saída gráfica com boa estruturação, legibilidade e boa ocupação das páginas;

- obtenção de algorítmos que não se tornem proibitivos em termos de tempo de processamento.

5.1.2 - Dimensionamento de páginas e símbolos

A dimensão da página na qual será representada a forma gráfica de saída não é arbitrária. Uma vez que o objetivo do SADG é gerar documentos em SDL-GR, a partir do SDL-PR, tal escolha deve recair entre dois formatos padrões : A4 (21 x 29,7 cm) e A3 (42 x 29,7 cm). Dentre ambos, o formato A3 é mais conveniente porque fixa limites físicos menos rígidos para a expansão das ramificações, concorrendo para uma menor fragmentação do desenho (fator de grande importância no produto final).

O CCITT fixou uma relação de dimensões entre os símbolos de 1 para 2; para o símbolo tarefa sugere os seguintes tamanhos : pequeno (1,0 x 2,0 cm), médio (1,4 x 2,8 cm) e grande (2,0 x 4,0 cm).

A fixação do tamanho dos símbolos define o número máximo de caracteres do texto interno aos mesmos e deve ter a preocupação de não ser restritivo quanto ao tipo de documento que representará. Textos técnicos devem ser concisos, mas deve ser permitido ao usuário um número adequado de caracteres para que seja mínima a necessidade de comprimir os textos que seriam naturalmente concebidos. O usuário poderia ser levado a dividir uma tarefa, que funcionalmente deveria ser única, em duas devido à limitação do texto (ou utilizar o comentário como continuação do texto interno ao símbolo quando, na realidade, o objetivo do comentário é fornecer informação complementar). No símbolo de decisão, que é bem mais crítico em relação ao espaço interno, esta divisão não é possível, tornando a restrição do texto bastante incômoda.

O tamanho pequeno é inviável do ponto de vista de texto interno aos símbolos e deve-se optar entre os tamanhos médio e grande. O número máximo de caracteres do texto interno para os símbolos de tarefa e decisão nos tamanhos grande e médio são os seguintes : 75 e 33 (para tarefa), e 24 e 12 (para decisão).

A utilização dos símbolos de tamanho grande cobre satisfatoriamente as necessidades do usuário quanto aos textos internos aos símbolos; no entanto, o número de símbolos por página fica reduzido em relação aos símbolos de tamanho médio. Desta forma, visando obter uma melhor distribuição dos símbolos nas páginas e cobrir os mais variados níveis de documentos e perfis de usuários, o SADG permite as seguintes opções :

- fixar um tamanho único para todos os símbolos;
- fixar um tamanho para cada grupo de símbolos com as mesmas características;
- escolher automaticamente o menor símbolo capaz de conter seu texto interno ou comentário associado.

Na definição dos critérios para distribuição de símbolos procurou-se ficar próximo dos critérios que um projetista usaria para desenhar manualmente a especificação do seu sistema em SDL-GR. No entanto, a expansão lateral dos ramos (oriundos de uma decisão, opção ou estado) à direita ou à esquerda, embora seja natural quando se desenha manualmente, não é conveniente para a automação (como veremos adiante). A restrição de expansão lateral somente à direita, que significa apenas um grau de liberdade na expansão horizontal de símbolos, não fere critérios de clareza, legibilidade e distribuição uniforme de símbolos na página, mas simplifica extraordinariamente o algorítmo que a automatiza e, portanto, será adotada.

A quase todo símbolo pode ser associado um comentário que, muito provavelmente, deverá aparecer pelo menos uma vez em uma dada coluna de símbolos, com tamanho médio. A reserva de uma coluna especialmente para comentários, ao lado de cada coluna de símbolos, dividiria por dois o número de colunas disponíveis para os símbolos (uma vez que o espaço reservado para um comentário será equivalente ao de um símbolo tarefa), o que é bastante

indesejável uma vez que o recurso coluna é escasso. Desta forma, o comentário associado ao símbolo é tratado como um ponto de ramificação, que só exige o deslocamento de uma coluna quando há conflito entre símbolos e comentário (vide figura 5.1 : não há conflito; conflito por comentário e deslocamento para a direita).

O SADG é capaz de trabalhar com qualquer dimensão de página, tendo como única restrição o espaçamento entre colunas (fixado em 5,0 cm para comportar um símbolo de tamanho grande e o espaço necessário para a passagem de linhas de conexão). Desta forma, quando a largura da página é escolhida, o número de colunas está automaticamente determinado. Naturalmente, existe um limite mínimo para as dimensões da página, para não comprometer irremediavelmente a legibilidade do documento de saída.

5.2 - Critérios e análises associadas à paginação

A paginação busca transformar a árvore de símbolos, que representa toda a hierarquia do sistema, desvinculada de uma distribuição espacial de seus símbolos, em subconjuntos de símbolos distribuídos em páginas, visando a qualidade da documentação gerada e a eficiência na sua obtenção.

Partindo das premissas estabelecidas para o SADG, são enunciados critérios para o Paginador sobre os quais devem se fundamentar as alternativas de solução para o problema da paginação.

São apresentados e comparados, face aos critérios estabelecidos, dois tipos de análise para a solução do problema de distribuição de símbolos.

5.2.1 - Conceitos Básicos

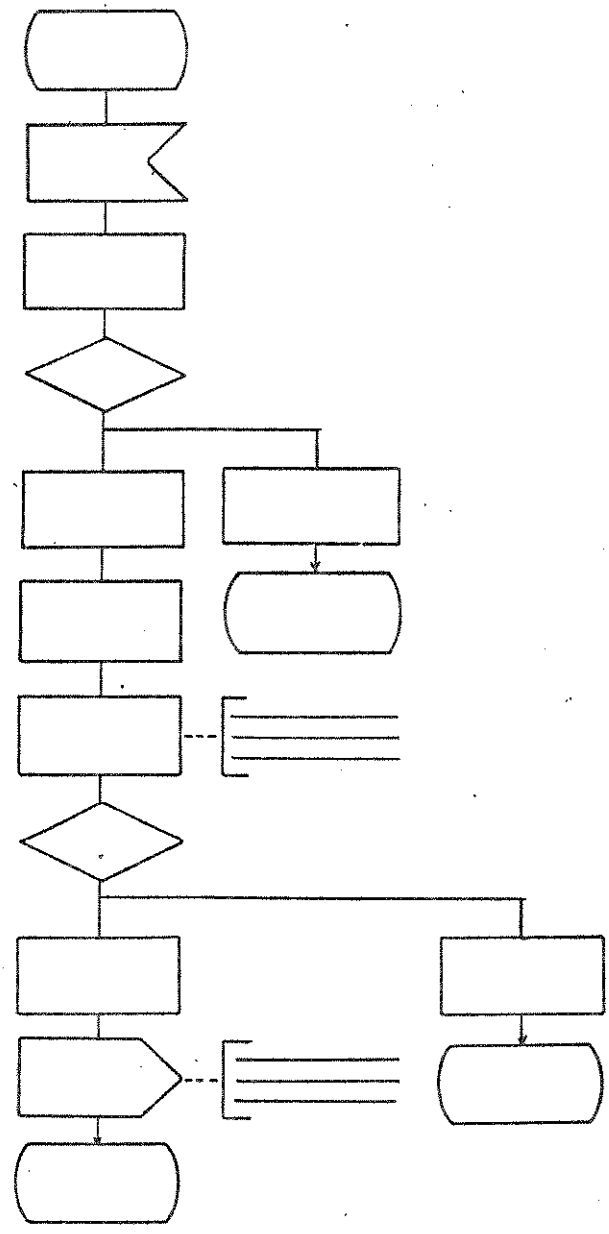


Figura 5.1 - Comentários associados a símbolos não têm coluna reservada a priori

Inicialmente, são definidos conceitos que serão utilizados nos ítens subsequentes, a saber :

- Ramo : Trecho de símbolos da árvore, onde a ramificação à direita é desconsiderada (vide figura 5.2 : sub-árvore com ramos demarcados).

- Sub-Árvore de Símbolos : Sub-conjunto da árvore de símbolos composto de um ramo principal, de todos os ramos que a ele se filiam, e dos filhos destes, estendendo-se até os ramos sem filhos (folhas da árvore) (vide figura 5.3 : as sub-árvores associadas a um dado ramo).

- Conector Automático : Devido ao limite de espaço físico da página, ocorrem descontinuidades na representação de um trecho de ramo ou de uma sub-árvore. Ao ponto de descontinuidade associa-se um conector automático de saída, que deverá conter um identificador associado ao seu ponto de continuação, que se iniciará com um conector automático de entrada. Cada conector automático de entrada terá como identificador um número inteiro associado à sua ordem de aparecimento nas páginas. Cada conector trará também o número da página onde está o seu conector associado (vide figura 5.4 conectores de saída e os seus respectivos conectores de entrada).

- Pendência : Os ramos ou ramificações descontinuados e substituídos por conectores automáticos de saída tornam-se pendências a serem resolvidas em outras páginas (ou em outras colunas da própria página). A solução de tal pendência introduz o conector automático de entrada. Além das pendências internas a transições (que geram conectores automáticos), existem as pendências associadas a uma transição inteira (iniciada por um sinal de entrada) de um dado estado. A resolução de uma pendência de transição necessitará da repetição do símbolo de estado ao qual pertence o sinal de entrada que inicia tal transição (vide figura

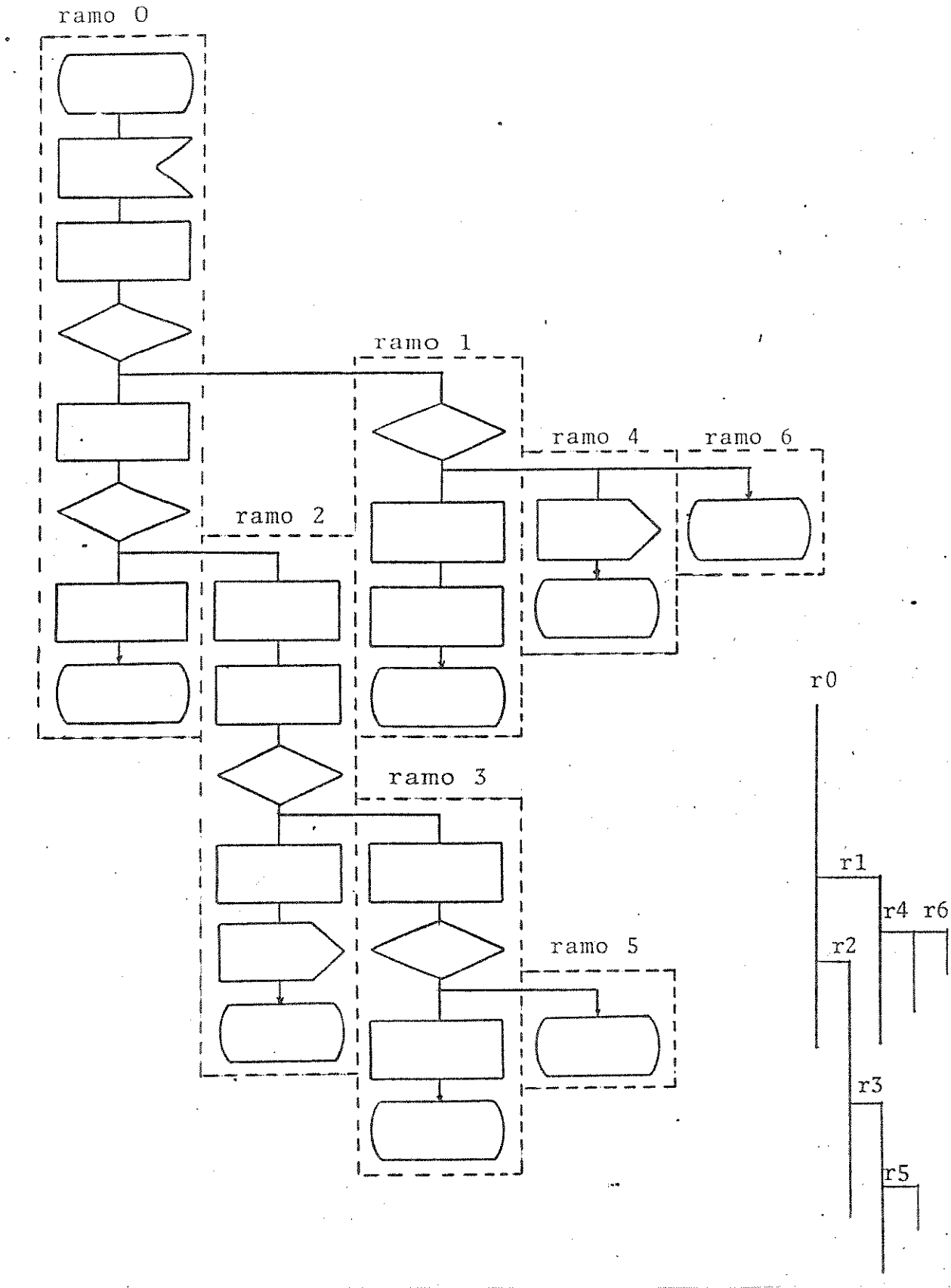


Figura 5.2 - SA com seus ramos demarcados

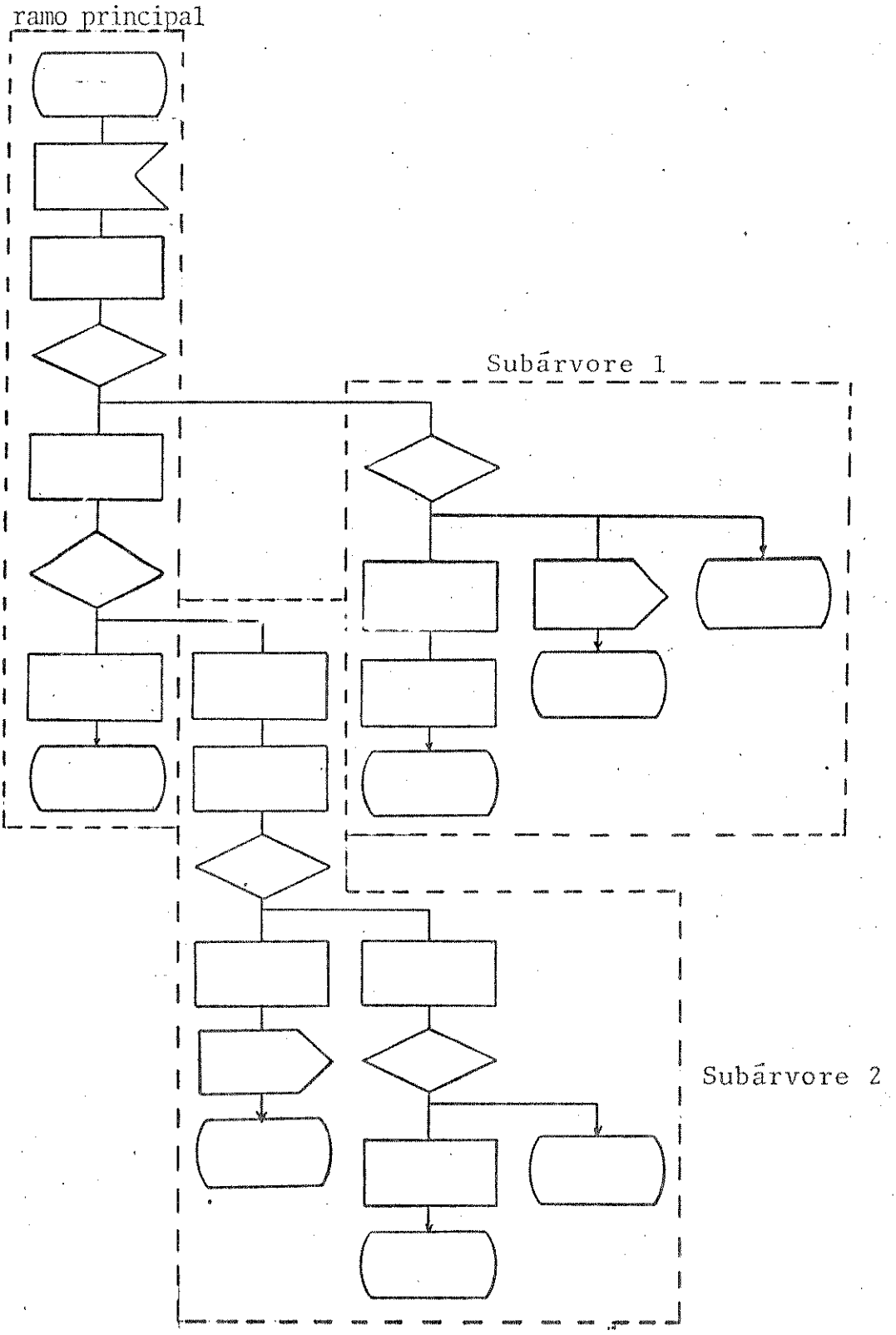


Figura 5.3 - SA principal com o seu ramo principal e as suas SAs filhas demarcadas

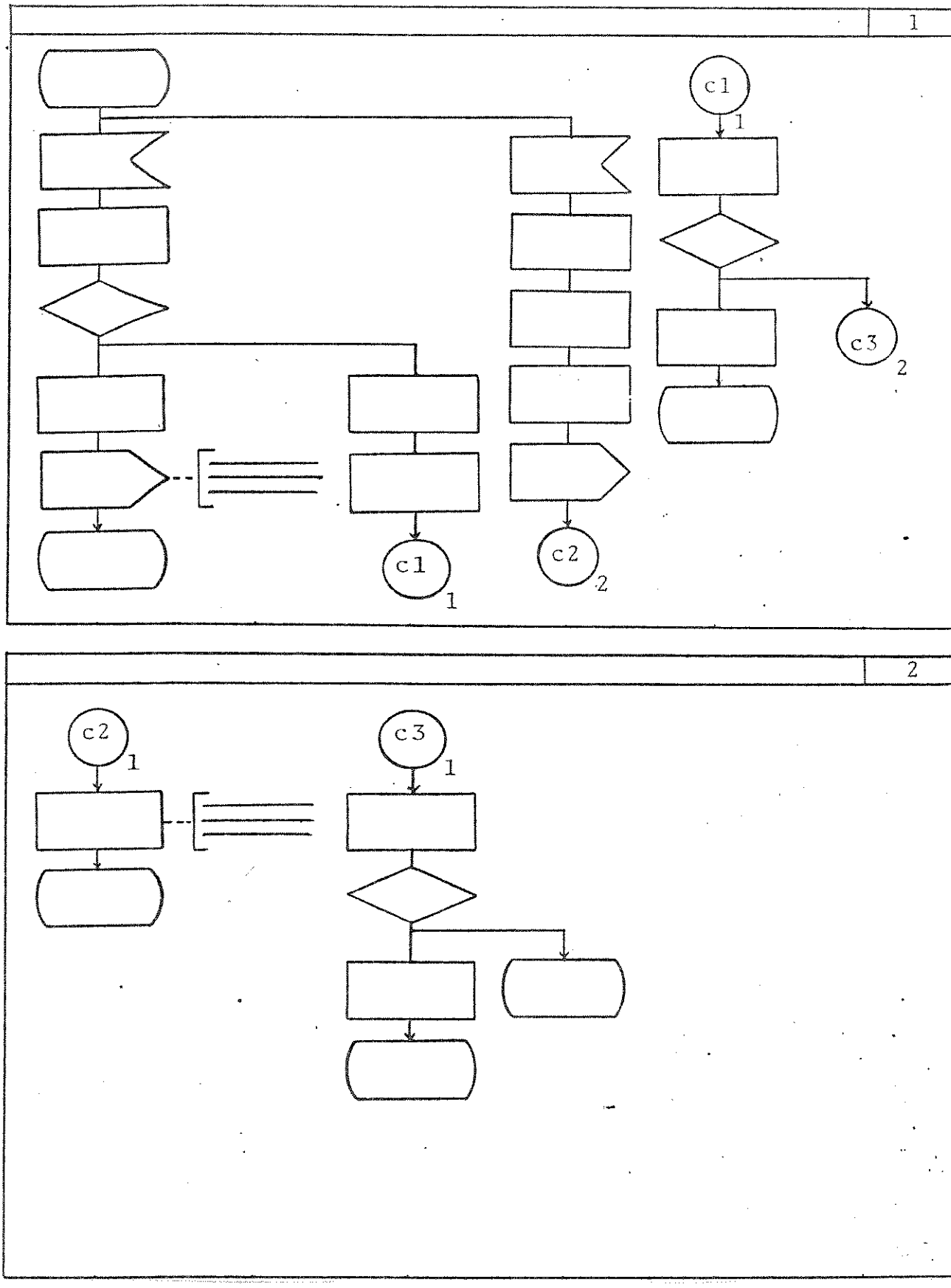


Figura 5.4 - Conectores de saída e os seus respectivos conectores de entrada com referência de página

5.5 : repetição de estado no tratamento de pendência de transição).

- Linha de Conexão : Após a distribuição dos símbolos em uma dada página, deverá ser tentada a substituição dos pares label-join associados que ocorrem na página por linhas compostas por segmentos de retas. Tal linha recebe o nome de linha de conexão e não poderá atravessar os símbolos e as linhas presentes na página (vide figura 5.6 : página com os pares label-join e outra com as respectivas linhas de conexão).

5.2.2 - Premissas e Critérios de Paginação

A grande ênfase da proposta do SADG é a automação mais completa possível da geração de documentos SDL-GR com legibilidade e estruturação compatíveis com os gerados manualmente ou por editor gráfico. O módulo Paginador, que responde pela qualidade do produto a ser gerado, deve cumprir premissas que traduzam a meta a que se propõe :

- estruturação por estados;
- a ordem prescrita em SDL-PR define a continuação do ramo principal na ocorrência de derivações;
- eficiência na distribuição de símbolos;
- conectores e labels com referência de página;
- substituição, sempre que possível, de pares label-join por linhas de conexão que não podem se cruzar;
- sequência para percorrer caminhos na árvore;
- opções para escolha de tamanho dos símbolos;
- cobertura de diferentes níveis de especificação do sistema.

Pode-se resumir as tarefas do Paginador em três pontos básicos :

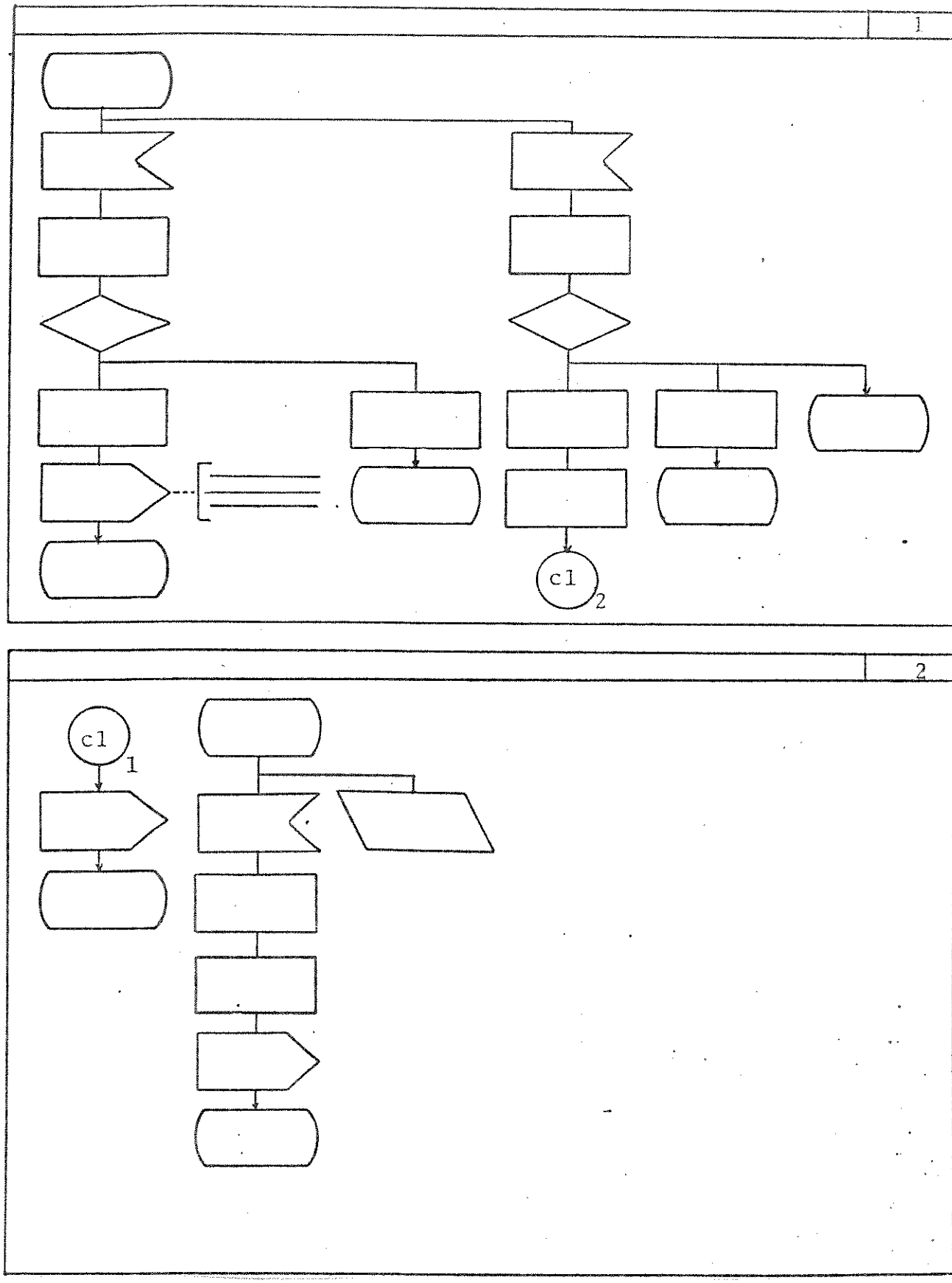


Figura 5.5 - Repetição do símbolo de estado no tratamento de uma transição específica

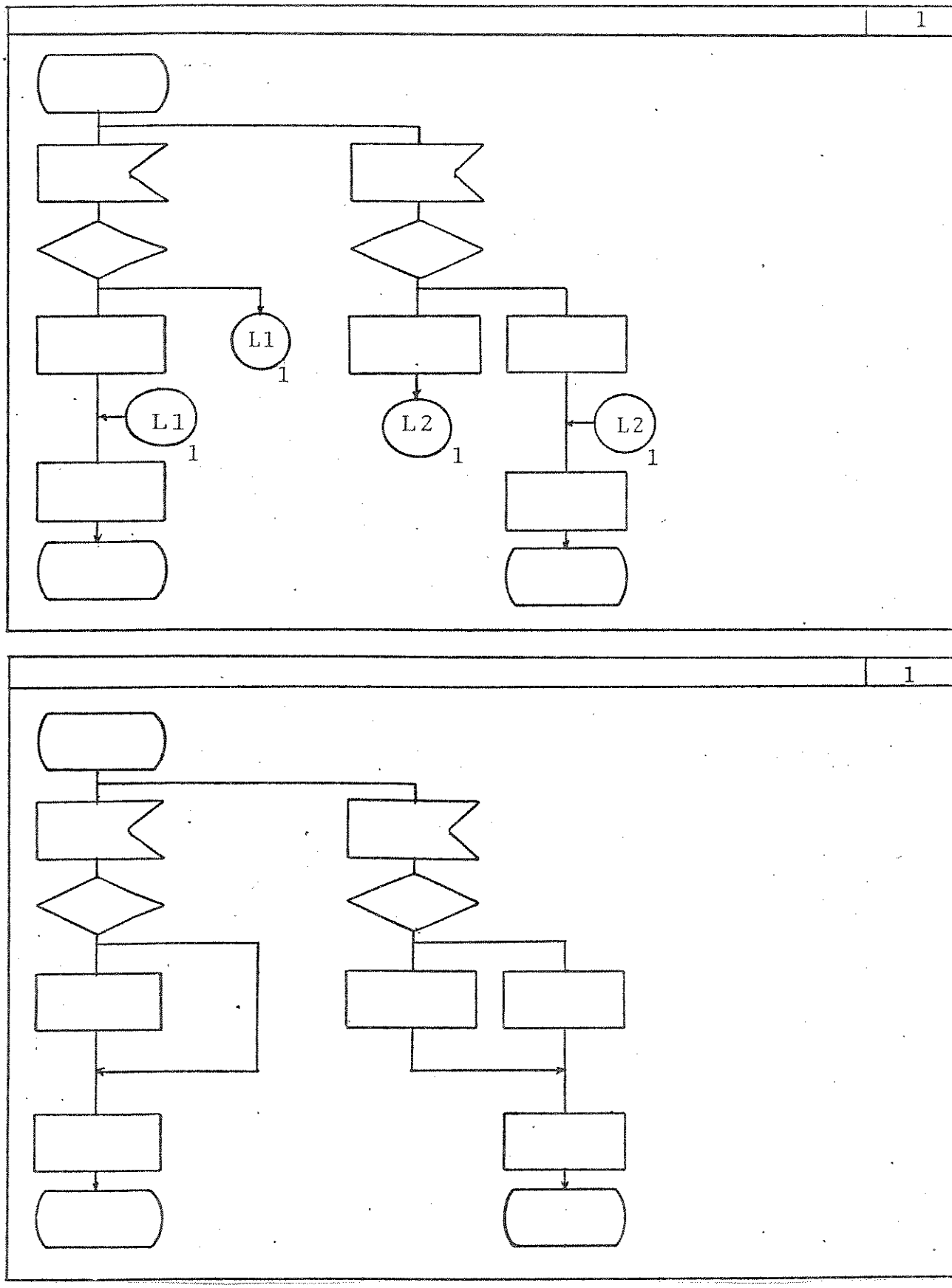


Figura 5.6 – Página com pares de Label-Join e página com linhas de conexão

- distribuição de símbolos;

- substituição de pares label-join por linhas de conexão;

- enumeração e assinalamento de referência de página nos labels e conectores.

Destes três pontos, os dois primeiros necessitam de uma análise dos problemas que trazem consigo e de alternativas que viabilizem a sua solução. Para sistematizar a análise que se segue nos próximos itens, as premissas do SADG serão detalhadas em alguns critérios que permitam maior objetividade para derivar soluções que atendam à meta estabelecida de forma concreta e que também forneçam condições de comparação entre possíveis alternativas, ou ainda, entre o SADG e outras ferramentas correlatas.

Fundamentando-se nas premissas iniciais, podem ser enunciados os seguintes critérios :

- paginação de todos os símbolos da sub-árvore associada a um dado estado antes da paginação de qualquer símbolo pertencente ao estado seguinte (derivado da estruturação de estado);

- ordenação da paginação dando preferência aos símbolos pertencentes ao ramo principal (derivado da ordem dos ramos no SDL-PR);

- redução de conectores automáticos para diminuir a fragmentação do desenho (derivado da eficiência na distribuição de símbolos);

- tratamento das pendências dando prioridade à paginação completa dos símbolos de uma sub-árvore antes de passar ao tratamento de outra sub-árvore filha do mesmo ramo (derivado da sequência para percorrer caminhos na árvore);

- boa ocupação da página, buscando evitar a má distribuição do espaço da página sem, contudo, ferir os três primeiros critérios (derivado da eficiência na distribuição de símbolos);

- conversão de nextstate para state, para continuar no ramo principal a representação dos símbolos do estado seguinte, desde que : o estado referenciado pelo nextstate seja o estado seguinte, não haja pendências no estado anterior e a opção do usuário que lhe permite escolher diferentes níveis de documentos (a concepção do sistema é feita em vários níveis de detalhe) determine tal conversão (derivado da opção para cobrir diversos níveis de documentos);

- prioridade entre candidatos a linha de conexão para resolver os conflitos que possam ocorrer numa página (derivado da implementação de linhas de conexão, sem cruzamentos entre si).

5.2.3 - Enfoques alternativos para a distribuição de símbolos

A árvore de símbolos estabelece as relações de hierarquia entre eles desconhecendo-se, no entanto, qualquer vínculo de distribuição espacial. A distribuição espacial será necessária para a paginação dos símbolos e poderá ser feita de duas formas :

- análise global : quando a distribuição é feita considerando todos os símbolos do sistema como componentes de uma página infinita;

- análise local : quando a distribuição ocorre apenas para um subconjunto de símbolos da árvore, limitado pelas dimensões de uma página real.

5.2.3.1 - Análise global

A distribuição da árvore de símbolos como um todo é, na realidade, feita de forma independente para cada uma das sub-árvores de símbolos associados a cada um dos estados

componentes da árvore de símbolos, devido ao critério de estruturação por estados. Pode-se enunciá-la da seguinte forma :

- a expansão dos ramos da sub-árvore de símbolos associados a um dado estado é realizada sem levar em conta as sub-árvores associadas aos demais estados e considerando-se existir à disposição uma página de dimensão infinita;

- assim sendo, a cada estado associa-se uma página cuja área final de ocupação dependerá da expansão dos ramos de cada uma das sub-árvores que representam um dado estado;

- o conflito entre ramos será evitado através do deslocamento para a direita dos ramos superiores à sub-árvore associada a um dado ramo em expansão;

- inicialmente, caracteriza-se como ramo principal da sub-árvore do estado o ramo descrito em primeiro lugar na forma SDL-PR para tal estado;

- os símbolos do ramo principal são distribuídos na primeira coluna da página;

- se o ramo principal tiver pontos de derivação, será escolhido o seu ponto inferior (mais ao pé da página), de tal forma que toda a sub-árvore associada a este ponto seja expandida, para que as sub-árvores filiadas aos pontos superiores sejam deslocadas à sua direita para evitar conflitos entre símbolos;

- desta forma a precedência das sub-árvores associadas aos pontos de derivação do ramo principal é de baixo para cima;

- a cada sub-árvore encontrada, o tratamento aplicado é análogo ao da sub-árvore principal;

- a expansão dos ramos caminha então na direção das folhas da sub-árvore de ramos (ramos sem filhos) e retorna depois ao ramo pai, tratando sucessivamente as sub-árvores associadas aos pontos de derivação acima do último ponto tratado deste ramo;

- quando ocorre o retorno ao ramo principal, todos os ramos da sub-árvore associada a um dado estado estarão expandidos.

Feita a distribuição espacial, considerando para cada estado do sistema uma página infinita, resta a tarefa de fracioná-la em páginas reais de dimensão finita. A forma mais simples de realizar este fracionamento é considerar uma envoltória retangular para a página infinita, e dividi-la em retângulos iguais (com a dimensão da página real) (vide figura 5.7).

Este método, que será denominado fracionamento simples, não cumpre vários dos critérios estabelecidos, a saber :

- redução do número de conectores automáticos : pode-se inferir da figura 5.7 a ocorrência de uma alta fragmentação (a expansão sem critério das sub-árvores, leva à incidência de uma alta taxa de conectores automáticos nas mesmas);

- tratamento das pendências percorrendo uma sub-árvore até o fim : a divisão das páginas mistura sub-árvores distintas no tratamento das pendências (na figura 5.7, pode-se ver tal mistura);

- boa ocupação da página : pode-se observar a ocorrência de páginas com utilização de apenas um símbolo efetivo, sendo os demais conectores automáticos (na figura 5.7, tal fato é mostrado);

- conversão de nextstate para state : impossível de ser coberta pela análise global, devido às páginas separadas por estados; e ainda que tal separação não ocorresse, seria impraticável por não se poder garantir que as páginas anteriores tivessem tratado todas as pendências do estado anterior, uma vez que não pode prevê-las quando é realizada a distribuição espacial.

Pode-se modificar o método de fracionamento simples para o que será denominado fracionamento otimizado. As páginas são divididas em quatro quadrantes, e se acrescenta a informação de

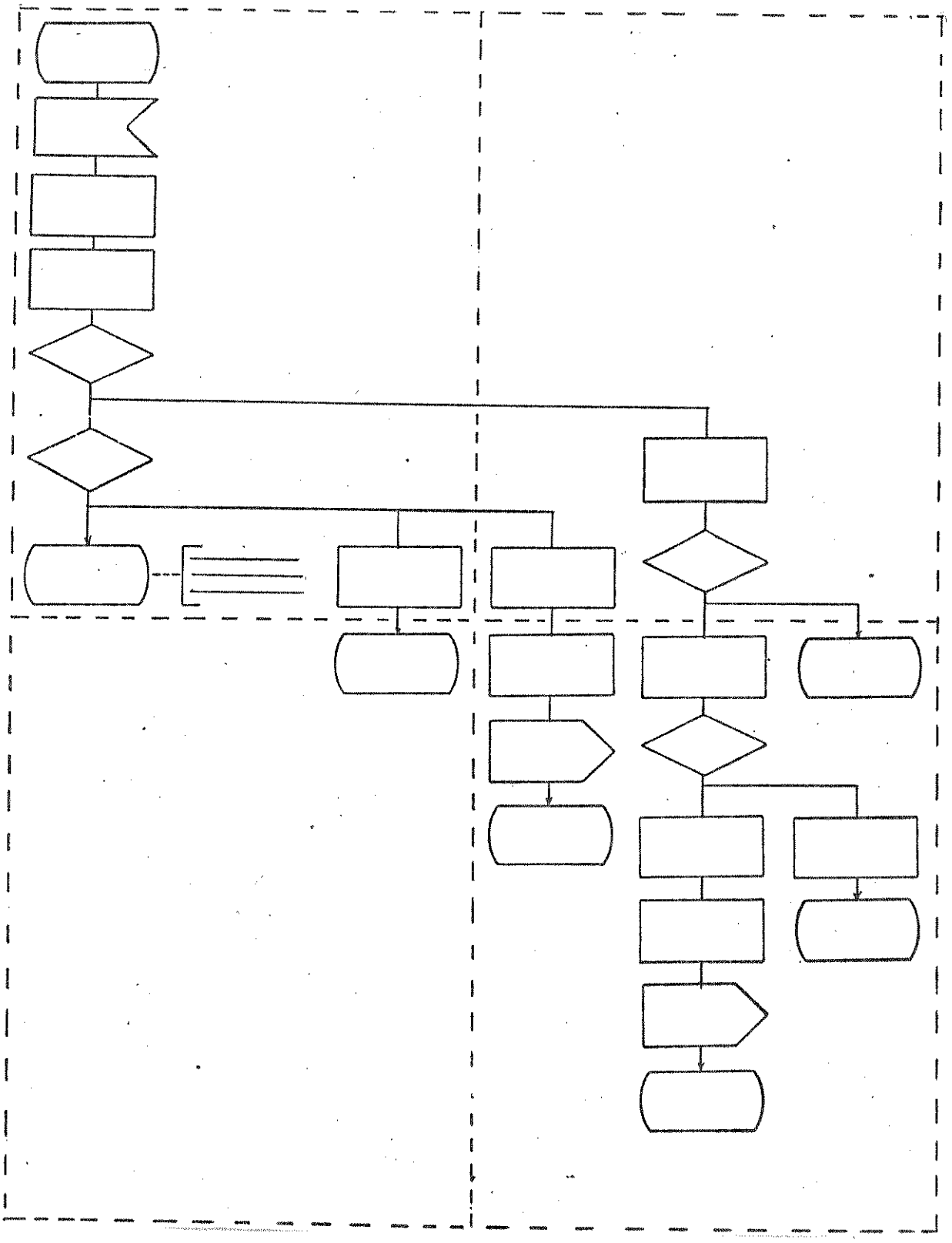


Figura 5.7 - Particionamento de uma página infinita em páginas finitas

ocupação de cada um deles. De posse desta informação pode-se localizar pares de páginas complementares (com ocupação de quadrantes distintos), de forma a concatenar duas páginas em uma. Na figura 5.7 as páginas 2 e 3 são complementares.

O método do fracionamento otimizado consegue cumprir o critério de boa ocupação da página mas permanecem desrespeitados os demais critérios que o fracionamento simples não cumpria. Em relação ao critério de tratamento das pendências percorrendo uma sub-árvore até o fim, a violação é mais acentuada, uma vez que as páginas complementares só levam em conta a ocupação dos quadrantes.

A análise global parece dificultar a obtenção de um método viável em termos de complexidade e tempo de processamento, capaz de cumprir conjuntamente os critérios estabelecidos.

5.2.3.2 - Análise local

Este tipo de análise fundamenta-se no algorítmo natural utilizado pelo projetista para gerar a forma gráfica manualmente ou através de um editor gráfico. Na prática, o projetista não escolhe regiões do desenho com maior densidade de símbolos para colocar em uma determinada página. Primeiro porque, em geral, não dispõe a priori do sistema representado como um todo para tal avaliação e, segundo porque outros parâmetros (tais como sequência na representação) não são observados por tal algorítmo. Na realidade, a distribuição é feita página a página, buscando seguir uma sequência coerente na apresentação dos ramos da árvore de símbolos.

A análise dos símbolos que poderiam ocupar uma página define um objeto menor de manipulação. As opções se reduzem e propiciam a aplicação de algorítmos que tornem possível o cumprimento dos

critérios pré-estabelecidos, através de uma distribuição local que analisa uma quantidade restrita de informação. Desta forma, critérios de boa ocupação da página e de redução da fragmentação do desenho, segundo uma ordem coerente de paginação dos ramos, emergem naturalmente da análise local, uma vez que o universo de manipulação é mais restrito e não há vínculos de subordinação a nenhum pré-processamento para distribuição espacial global (onde tais critérios não são observados).

Pelos motivos acima, o projeto do Paginador se fundamenta na análise local.

5.3 - Soluções para o Paginador face aos critérios adotados

O Paginador, buscando cumprir os critérios pré-estabelecidos, adota três procedimentos básicos :

- Otimização de bordas : Visa evitar a ramificação abortiva de símbolos nas colunas (ao pé da página) ou na extremidade direita da página. Conceitua-se como ramificação abortiva a inclusão de símbolos ou ramos que não poderão expandir seus ramos filhos devido ao limite físico da página (vide figura 5.8, onde aparecem as duas situações de ramificação abortiva de símbolo e de ramo). Nestas situações, o símbolo ou o ramo que causam as ramificações abortivas tornarão-se-ão pendências para páginas posteriores. A otimização de bordas concorre para a minimização dos conectores automáticos, ao evitar as ramificações abortivas.

- Escolha das sub-árvores filhas do ramo principal que ficarão na página : O primeiro critério a ser observado é o da estruturação, que impõe a paginação de todos os símbolos de um dado estado antes dos símbolos pertencentes a outro estado. No âmbito de uma página, no entanto, a presença de todas as sub-árvores derivadas do trecho do ramo principal na página pode

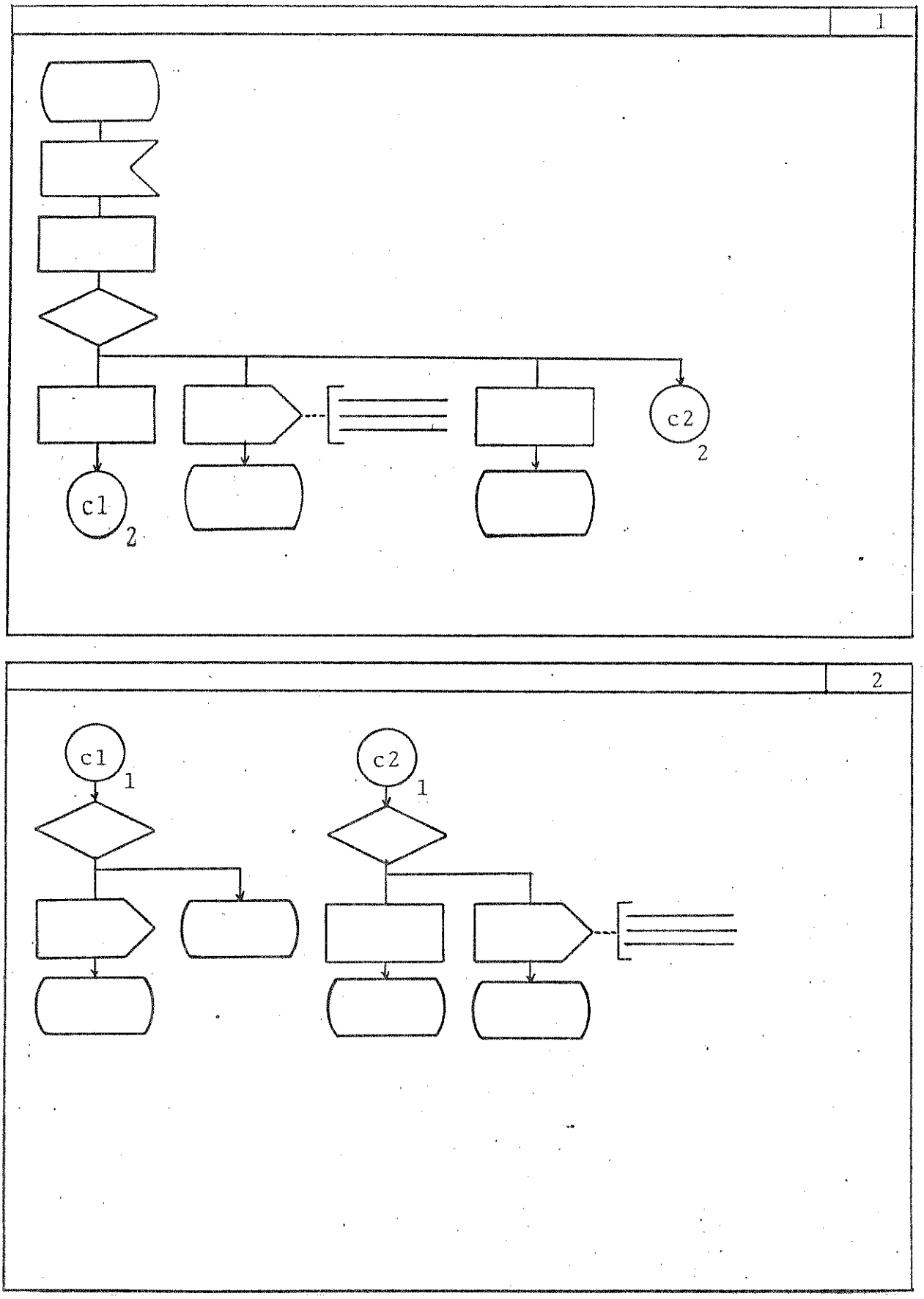


Figura 5.8 - Situações onde a ramificação abortiva foi evitada

ser conflitante, o que obrigará o Paginador a escolher algumas sub-árvores para permanecer na página, tornando as demais sub-árvores pendências para tratamento posterior. Quando uma ramificação do ramo principal origina várias transições (ramificação derivada de símbolo de estado), a extensão dos critérios de estruturação leva a dar prioridade aos símbolos da transição em tratamento podendo as demais transições coexistir na página se houver espaço. No entanto, entre as sub-árvores pertencentes à mesma transição e conflitantes entre si, deverá haver uma escolha que leve em conta os critérios de minimização dos conectores automáticos e de ocupação da página. Tal escolha será analisada no ítem que se segue.

- Ordenação das pendências : Os critérios que norteiam a prioridade de tratamento entre as pendências existentes são : ordenação da paginação em função do ramo principal e tratamento completo de uma dada sub-árvore. Desta forma, a pendência de continuação do ramo principal é prioritária em relação às pendências originárias das derivações laterais do ramo principal. Entre as pendências das sub-árvores filiadas ao ramo (originárias das derivações laterais) não há prioridade, mas ao ser escolhida uma delas para o tratamento, todas as suas pendências resultantes serão prioritárias às sub-árvores irmãs (filhas do mesmo ramo). Este enfoque se estende recursivamente à medida que se obtém sub-árvores filhas, ao caminhar na direção das folhas da árvore.

A substituição de pares label-join por linhas de conexão é realizada em cada página após a distribuição dos símbolos na página, nos espaços disponíveis (não ocupado pelos símbolos ou conexões realizadas a priori).

5.3.1 - Distribuição de símbolos nas páginas

Dos três procedimentos básicos enunciados acima para a distribuição de símbolos, o mais complexo é o de escolher de forma eficiente e segundo os critérios adotados, as sub-árvores que deverão permanecer na página quando ocorre conflito espacial entre as possíveis sub-árvores candidatas à página. Neste item são apresentados conceitos, definições e analisadas soluções para tal procedimento, concluindo-se pela solução adotada no Paginador.

Quando se considera a distribuição dos símbolos das sub-árvores de um dado estado numa página infinita, não se impõem limites ao comprimento dos ramos nem ao número de colunas necessárias. Quando duas sub-árvores filiadas ao mesmo ramo pai entram em conflito espacial devido à expansão de seus ramos, a sub-árvore superior é deslocada para a direita (ocupando novas colunas) até que o conflito entre ramos desapareça.

Numa página finita, o comprimento de qualquer ramo é limitado pelo comprimento da página, de forma que para o melhor aproveitamento do espaço, todos os ramos deverão ter como espaço disponível para expansão : o trecho vertical que se inicia na cota do ramo (ponto de início do ramo em relação ao ramo pai) e termina ao pé da página. Toma-se como referência zero de cota na página a horizontal superior, crescendo a cota até o seu valor máximo na horizontal inferior (largura útil da página) (vide figura 5.9). Para que um dado ramo possa ocupar o espaço que lhe cabe, deve deslocar-se para a direita (sentido crescente das colunas) enquanto ocorrer conflito com os ramos de cota inicial maior que a sua. No entanto, a limitação do número de colunas da página ocasionará situações de conflito espacial entre ramos de sub-árvores pertencentes a um dado ramo principal. Surge então, a necessidade de escolher dentre as sub-árvores conflitantes aquelas que sejam mais vantajosas para permanecer na página sob o ponto de

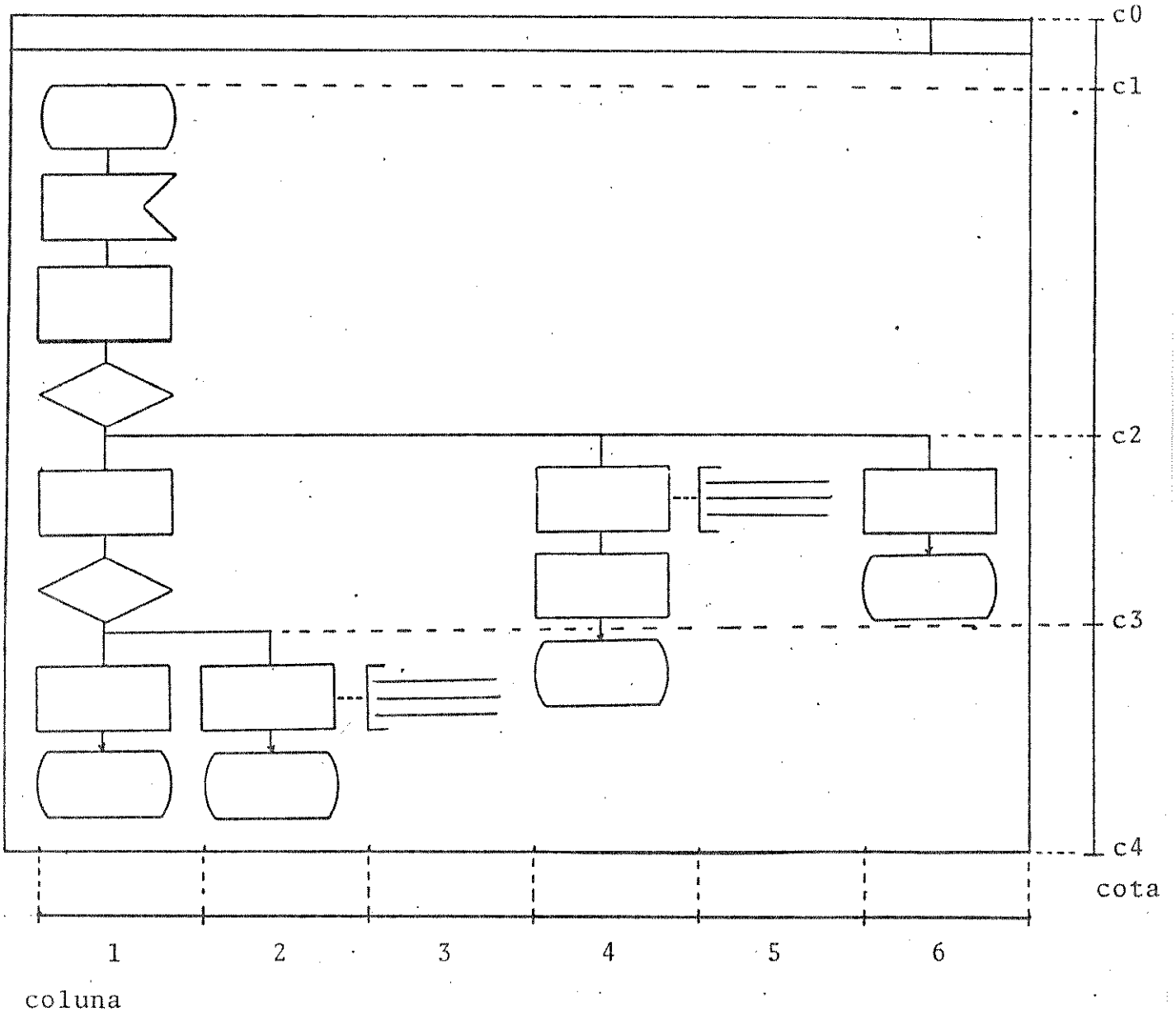


Figura 5.9 - Espaço utilizado pelos ramos na vertical e cotas associadas a ramos

vista dos critérios de redução de conectores automáticos e boa ocupação da página.

5.3.1.1 - Definição recursiva de sub-árvore

A distribuição espacial de símbolos, numa página finita ou infinita, utiliza-se de alguns conceitos básicos como : ramo principal, sub-árvores filhas e deslocamento à direita de ramos superiores. A seguir são analisadas algumas particularidades das sub-árvores utilizadas, de forma a derivar com maior simplicidade e clareza as estruturas de dados e os algorítmos propostos para a distribuição de símbolos da árvore, através de escolha entre as sub-árvores conflitantes.

A expansão de ramos de uma sub-árvore necessita basicamente de informações associadas aos seus ramos. A representação de uma sub-árvore através de seus ramos sintetiza a informação necessária para a expansão dos símbolos da árvore. A conveniência de se utilizar algorítmos recursivos para percorrer os ramos de uma sub-árvore leva à sua definição de forma recursiva.

Pode-se definir recursivamente a sub-árvore em função de seus ramos da seguinte forma :

SA = Rp + Conjunto [k = 1 .. nf(Rp)] SAfk

onde :

Rp : ramo principal da SA;

nf(Rp) : número de SAs filhas de Rp;

SAfk : k ésima SA filha de Rp;

k : número da SA filha (1 para a SA filha superior e nf(Rp) para a SA filha inferior).

Esta definição é recursiva, pois necessita avançar nos níveis da SA até que nf(Rp) = 0 e portanto SA = Rp, ou seja, a definição de uma SA depende da definição de suas SAs filhas até chegar-se às

folhas da SA. A univocidade de uma SA depende do caminho percorrido do ramo principal da árvore até o ramo principal da SA, ou seja, a cada nível da árvore escolhe-se um ramo filho (ramo principal da SA do próximo nível) para percorrer a árvore. Pode se definir tal caminho como (vide figura 5.10) :

C(SAn) = Conjunto [k = 1 .. n] fk

Ou seja, o conjunto de índices de ramos escolhidos em cada nível, do nível 1 até o nível n, é o caminho que define a SA de nível n. Os algorítmos recursivos são bastante convenientes para este tipo de definição, uma vez que num dado nível n, as escolhas feitas nos níveis anteriores são salvas em uma pilha.

5.3.1.2 - Conceito de envoltória

A sistematização dos procedimentos de espalhamento dos ramos de uma dada SA induz à introdução do conceito de envoltória.

Os símbolos, ao serem distribuídos nas páginas, deverão ter associados a si dois parâmetros : coluna e cota. A coluna cresce da esquerda para a direita, e a cota tem valor igual a 0 no eixo superior da página e valor igual ao limite inferior da página no eixo inferior da mesma.

A representação de uma SA através de seus ramos utiliza-se de informações como : cota inicial do ramo (cota do primeiro símbolo do ramo) e espaço do ramo (somatória do espaço ocupado por cada um dos símbolos do ramo mais o espaço entre os símbolos e possíveis espaços adicionais, como o necessário para nome de ramo de decisão). Com estas duas informações é possível deslocar ramos superiores (menor cota) á direita, de forma a resolver o conflito espacial com ramos inferiores de maior cota (vide figura 5.11, onde as linhas tracejadas representam a envoltória da SA cujo ramo principal tem cota2 (ramo inferior) e que deve servir de limite

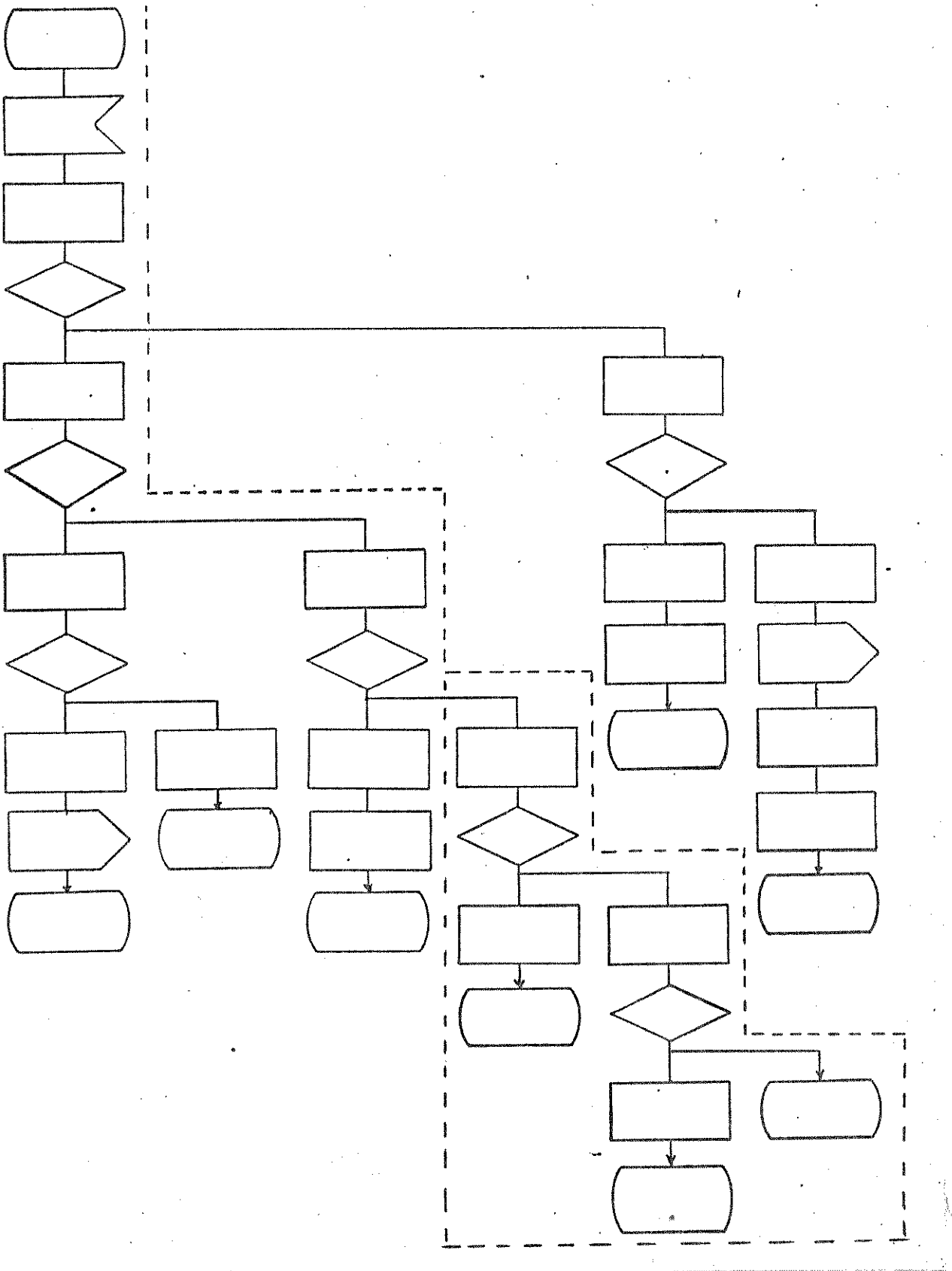


Figura 5.10 - SA definida pelo caminho percorrido até o ramo principal da SA principal

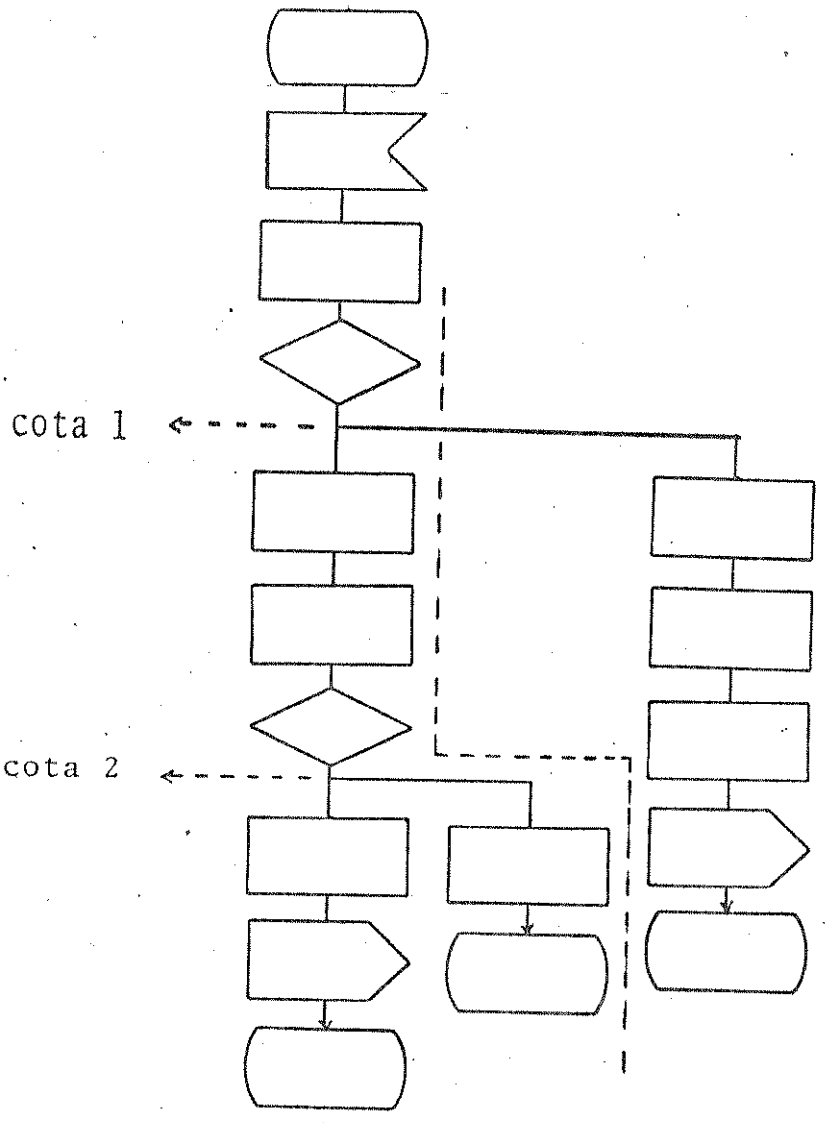


Figura 5.11 - SA inferior (cota2) serve de limite inferior para a SA superior (cota1)

para a expansão dos ramos da SA cujo ramo principal tem cota1 (ramo superior)). No caso mais geral de uma SA cujo ramo principal tem nível n, a expansão de seus ramos terá que considerar uma envoltória que pode receber contribuição de todas as SAs já expandidas (vide figura 5.12).

Pode-se definir uma envoltória de forma genérica como tendo um valor de cota para cada coluna : E(c). Pode-se inferir da figura 5.13 as seguintes propriedades :

- propriedade 1 : E(c1) <= E(c2), para c1 <= c2;
- propriedade 2 : E1(c) + E2(c) = menor_cota(c).

Define-se então envoltória para ramo e para SA, mantidas as propriedades enunciadas acima, sendo a envoltória de ramo um caso particular em que :

- E(c) = cota_ramo, para c <= coluna_ramo, ou
- E(c) = limite_inferior_página para c > coluna_ramo.

Aplicando-se a propriedade de soma de envoltórias na definição da SA (como somatória de seus ramos) tem-se :

ESA(c) = ERp(c) + Somatória [k = 1 .. nf(Rp)] ESAfk(c)

O cálculo de envoltória também é recursivo, pois para nf(Rp) = 0 tem-se que ESA(c) = ERp(c), e chega-se por indução que ESA se compõe através da somatória das envoltórias de todos os seus ramos.

Mas o que importa determinar para a expansão dos ramos de uma SA não é a sua envoltória, e sim a envoltória das SAs inferiores a ela. Considerando-se uma SA de nível n, pode-se constatar que a envoltória das SAs inferiores, que lhe serve de base para expansão, é a somatória de todos os ramos expandidos anteriores a ela.

O deslocamento de um de seus ramos sobre uma dada envoltória só é possível sob as seguintes condições :

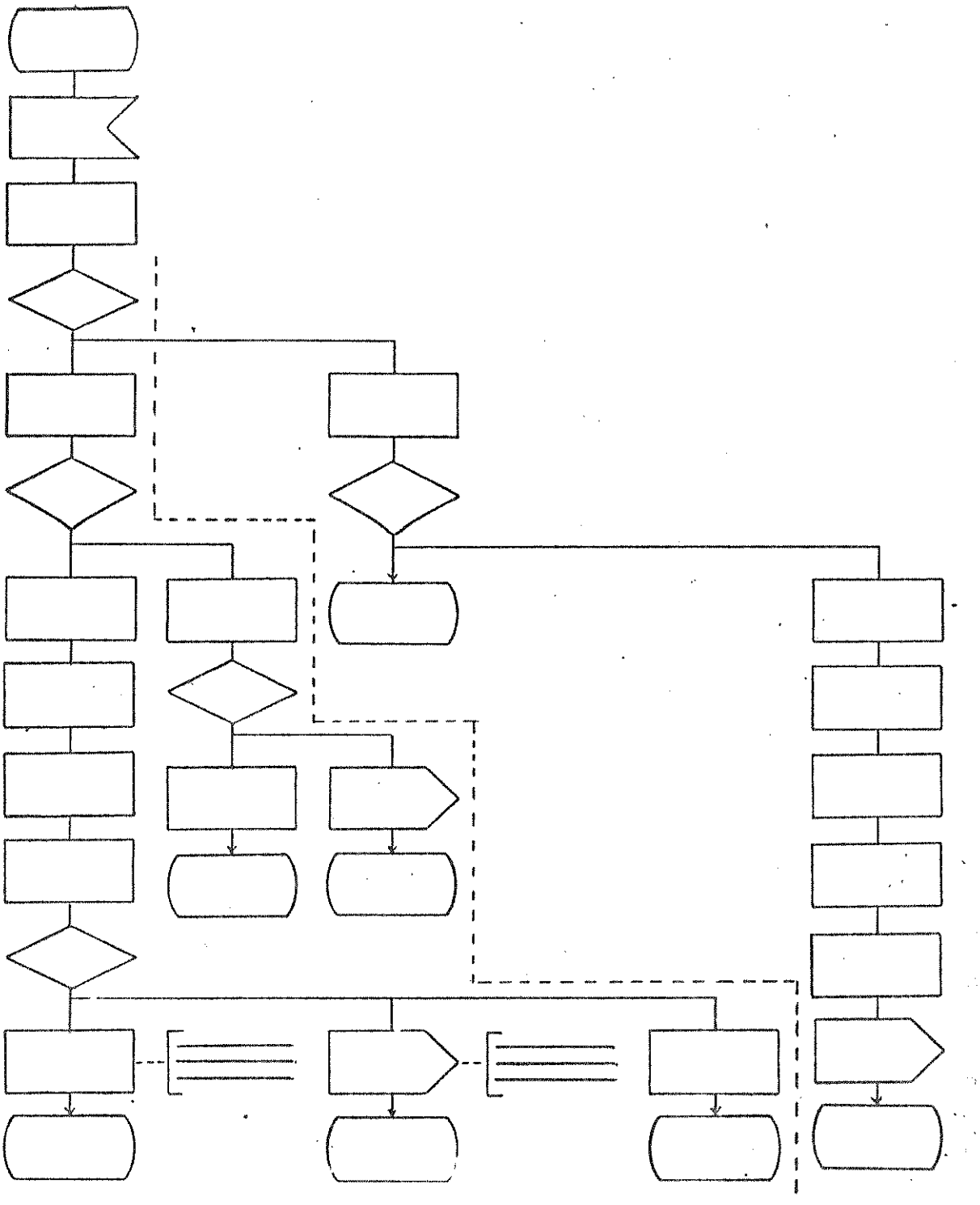
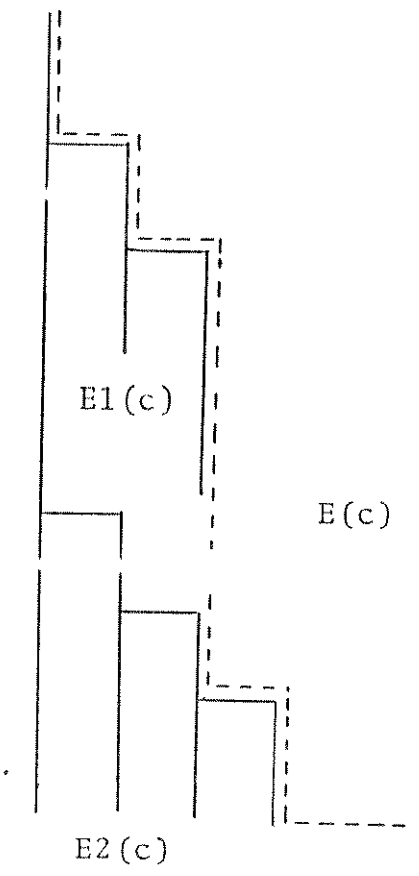


Figura 5.12 - Envoltória das SAs à esquerda da SA superior



$$E(c) = E1(c) + E2(c)$$

Figura 5.13 - Soma das envoltórias

- coluna_mínima_ramo = coluna_ramo_pai + 1;

- cota_ramo < E(c), para c = coluna_ramo.

Se a página é suposta infinita, o incremento da coluna do ramo não sofre restriçõ s; caso contrário, é limitado pelo número máximo de colunas da página.

Estendendo o conceito de deslocamento de ramo para uma SA, observa-se que todos os seus ramos devem ser deslocados caso a caso. Ou seja, não basta deslocar os ramos das SAs filhas em função do deslocamento do ramo principal, devido à composição em degraus da envoltória das SAs inferiores, que serve de base para o deslocamento.

5.3.1.3 - Procedimentos Básicos associados à definição de SA

Como foi exposto, uma SA pode ser representada como composta de símbolos ou ramos. Pode-se agora definir uma estrutura de dados que simplifique a sua manipulação e, consequentemente, os procedimentos básicos para percorrer a SA e realizar a distribuição espacial de seus símbolos.

Na representação da SA através de seus símbolos, pode-se associar a cada um deles três apontadores, com os quais será possível acessar os seus símbolos : pai, filho e irmão. Existem alguns símbolos em SDL que podem ter mais que um filho (exs. : decision, state, etc) e, neste caso, o símbolo aponta para o seu primeiro filho, obtendo-se os demais ponteiros através da cadeia de ponteiros de símbolos irmãos.

Na representação da SA em ramos são definidos os mesmos apontadores mas, para permitir o acesso aos ramos filiados a um dado ramo pai através do seu ramo inferior até o superior (acesso descendente) ou vice-versa (acesso ascendente), os apontadores de filho e irmão se dividem em : primeiro filho e último filho, e

irmão inferior e irmão superior. O acesso descendente inicia-se pelo primeiro filho e percorre a cadeia de ponteiros de irmãos inferiores, e o ascendente inicia-se pelo último filho e segue a cadeia dos irmãos superiores.

Exemplifica-se abaixo a criação da SA de ramos, percorrendo recursivamente a SA de símbolos, e convencionando-se os apontadores como variáveis inteiras que indicam o fim de cadeia quando assumem valor igual a 0.

```
    PROCEDURE CriaSARamos;
    VAR nramcs : INTEGER; {número total de ramos}

    PROCEDURE TrataSA (simb, ramopaiSA : INTEGER);
    VAR ramo : INTEGER; {número do ramo em tratamento}

    PROCEDURE AjustaApontadoresRamo;
{1}BEGIN {AjustaApontadoresRamo}
    nramos := nramos + 1; {incrementa número de ramos}
    ramo := nramos; {número do novo ramo}
    WITH R[ramo] DO
{2}BEGIN {inicia apontadores do novo ramo}
    nsimbramo := 0; {inicia número de símbolos do ramo}
    primsimbramo := simb; {primeiro símbolo do ramo}
    primramofilho := 0;
    {inicia apontador para primeiro ramo filho}
    ultramofilho := 0;
    {inicia apontador para último ramo filho}
    ramoirmaoinf := 0;
    {inicia apontador para ramo irmão inferior}
    ramopai := ramopaiSA; {inicia apontador para ramo pai}
    IF (ramopai = 0) THEN ramoirmaosup := 0
```

```
    {não há ramo pai : não há apontador para irmão superior}
    ELSE ramoirmaosup := R[pairamo].ultramofilho;
    {irmão superior = último ramo filho do ramo pai}
{2}END;
    WITH R[ramopaiSA] DO
{2}BEGIN {atualiza apontadores do ramo pai}
    ultramofilho := ramo;
    {novo ramo é último filho do ramo pai}
    IF (primsimbramo = 0) THEN primsimbramo := ramo
    {ramo pai sem filhos : novo ramo é o primeiro ramo filho}
    ELSE R[ultramofilho].ramoirmaoinf := ramo;
    {novo ramo é ramo inferior do último ramo filho}
{2}END;
{1}END; {AjustaApontadoresRamo}

    PROCEDURE TrataSimbolo;
{1}BEGIN {TrataSimbolo}
    WITH R[ramo] DO nsimbramo := nsimbramo + 1;
    {incrementa número de símbolos do ramo em tratamento}
{1}END {TrataSimbolo}

{1}BEGIN {TrataSA}
    AjustaApontadoresRamo; {inicia apontadores do novo ramo
          e atualiza apontadores do ramo pai}
    WHILE (simb > 0) DO WITH S[simb] DO
{2}BEGIN {tratar símbolo do ramo}
    IF (simbirmao > 0) THEN TrataSA (simbirmao, ramo);
    {se símbolo tem irmão : tratar SA filha}
    TrataSimbolo; {incrementa número de símbolos do ramo}
    simb := simbfilho; {aponta símbolo filho}
{2}END;
```

```
{1}END; {TrataSA}

{1}BEGIN {CriaSARamos}
   nramos := 0; {inicia número total de ramos}
   TrataSA (primsimbSA, 0);
   {trata SA a partir do seu primeiro símbolo}
{1}END; {CriaSARamos}
```

O procedimento TrataSA evidencia (se forem excluídas os procedimentos utilizadas para gerar a estrutura de ramos : AjustaApontadoresRamo e TrataSimbolo) que a recursividade é bastante adequada para percorrer a SA de símbolos.

A expansão de ramos de uma SA é exemplificada abaixo supondo uma página infinita e utilizando os conceitos previamente definidos de SA de ramos, envoltória e deslocamento à direita. Mais uma vez, o procedimento TrataSA evidencia a conveniência do uso da recursividade em decorrência da própria notação recursiva da SA de ramos. O exemplo ilustra o acesso ascendente (mais conveniente para a expansão dos ramos devido ao deslocamento à direita). Para obter o acesso descendente, basta substituir ramo_ultimo_filho por ramo_primeiro_filho e ramo_irmao_superior por ramo_irmao_inferior.

```
PROCEDURE ExpandeRamosSA;
VAR cont1 : INTEGER; {contador}

PROCEDURE TrataSA (r : INTEGER);

PROCEDURE TrataRamo;
VAR cont2, {contador}
    c : INTEGER; {coluna}
```

```
{1}BEGIN {TrataRamo}
   WITH R[r] DO
{2}BEGIN {desloca ramo à direita até desaparecer conflito}
   c := colunaramo; {inicia variável auxiliar com coluna do ramo}
   WHILE (cotaramo + espacoramo >= E[c]) DO c := c + 1;
   {enquanto ramo conflitar com envoltória : deslocar à direita}
   {não há limite de colunas (hipótese de página infinita)}
   FOR cont2 := colunaramo + 1 TO c DO E[cont2] := cotaramo;
   {atualiza envoltória no intervalo de colunas
    entre ramo pai e novo ramo}
   colunaramo := c; {atualiza coluna do ramo}
{2}END;
{1}END; {TrataRamo}

{1}BEGIN {TrataSA}
   REPEAT WITH R[r] DO
{2}BEGIN {trata todos os ramos da SA}
   IF (ramopai = 0) THEN colunaramo := colunaramoprinc
   {inicia coluna do ramo principal da SA}
   ELSE colunaramo := R[ramopai].colunaramo + 1;
   {coluna do ramo é a próxima em relação à do ramo pai}
   TrataRamo; {ajusta coluna do ramo em relação à envoltória}
   IF (ramoultimofilho > 0) THEN TrataSA (ramoultimofilho);
   {inicia tratamento pelo último ramo filho}
   r := ramoirmaosuperior; {aponta ramo irmão superior}
{2}END UNTIL (r = 0);
{1}END; {TrataSA}

{1}BEGIN {ExpandeRamosSA}
   FOR cont1 := 1 TO numerocolunaspagina
   DO E[cont1] := limitecotapagina;
```

```
{inicia envoltória com cota do pé da página}
TrataSA (ramoprinc); {trata SA definida pelo ramo principal}
{1}END; {ExpandeRamosSA}
```

Na prática, o deslocamento à direita é limitado pelo número máximo de colunas de uma página finita e a expansão dos ramos na vertical é limitada pela cota limite ao pé da página. Como foi visto anteriormente, escolhe-se um ramo (denominado ramo principal) para realizar a distribuição dos símbolos nas páginas, selecionando-se (em caso de conflito) dentre as SAs filhas aquelas que permanecerão na página e colocando-se as demais na pilha de pendências. Aos procedimentos acima descritos, são acrescentadas restrições de cota limite de página e número máximo de colunas bem como os procedimentos de escolha de SAs filhas, os quais serão discutidos a seguir.

5.3.1.4 - Método de inserção ascendente

Em algumas situações é possível evitar os conflitos entre as SAs filhas de um dado ramo através de superposição e deslocamento à direita das SAs superiores (vide figura 5.14 (SAs convivem superpostas através de deslocamento à direita)). Esta solução é bastante simples, mas não é capaz de resolver a maioria das situações de conflito além de que as SAs superiores estarão sendo tratadas como menos prioritárias. Isso leva a distorções que se contrapõem aos critérios de :

- Redução de conectores automáticos : SAs que poderiam ser colocadas na página sem gerar conectores automáticos podem conduzir a uma grande fragmentação devido aos deslocamentos à direita (vide figura 5.15). Esta fragmentação é reduzida se a prioridade de escolha entre as SAs for função do número de

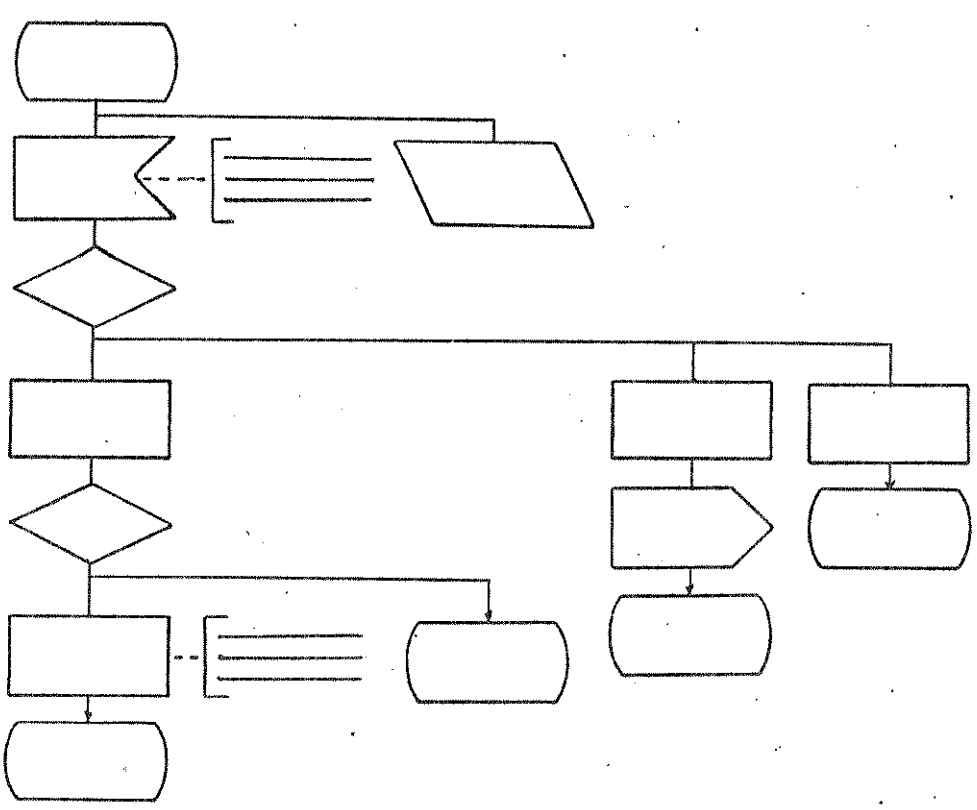


Figura 5.14 - Deslocamento à direita resolve conflito entre SAs irmãs

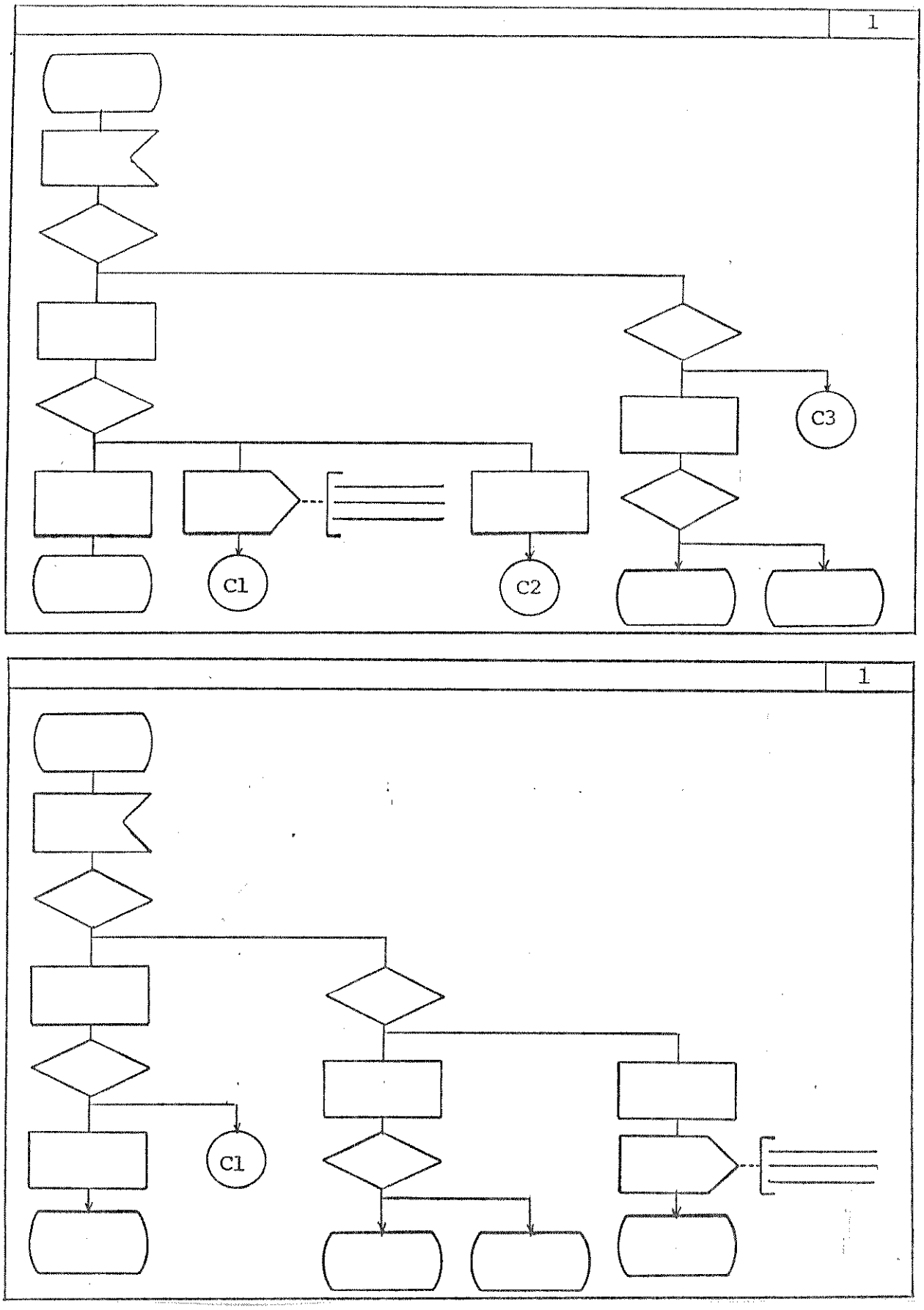


Figura 5.15 - Escolha entre SAs filhas para reduzir o número de conectores automáticos

conectores automáticos que uma SA teria se não fosse deslocada para a direita devido ao conflito com suas SAs irmãs inferiores.

- Ocupação da Página : A retirada completa das SAs superiores em conflito (uma vez que sua colocação parcial aumenta consideravelmente a fragmentação) pode conduzir a um baixo fator de ocupação da página. Logo, a escolha das SAs também deve ser função da área ocupada da página.

É importante observar que o método de inserção ascendente apenas fixa a prioridade das SAs para ocupação da página em função da cota inicial de cada SA (quanto maior a cota, maior a prioridade) sem, no entanto, evitar que todas as SAs irmãs sejam testadas. Seria possível contabilizar um fator de ocupação, como percentual da soma dos tamanhos dos ramos colocados em relação ao total de espaço disponível na página (soma dos espaços úteis das colunas) que, ao atingir um dado limiar inibiria a tentativa de colocação das SAs restantes. No entanto, este índice seria muito falho por não ter relação alguma com um importante parâmetro na expansão dos ramos de uma SA, que é o posicionamento de seus diversos ramos. Desta forma, todas as SAs irmãs devem ser testadas e o que importa definir é a ordem de prioridade das mesmas na ocupação da página.

Como foi visto no procedimento da expansão dos ramos de uma SA, a envoltória das SAs colocadas anteriormente na página é determinante para o deslocamento de cada um dos ramos da SA em expansão. Pode-se inferir, portanto, que a distribuição dos símbolos de uma dada SA (ou a expansão de seus ramos) numa página finita caracteriza-se basicamente por dois fatores :

- dependência da envoltória calculada a partir de todas as SAs já colocadas na página;

- cada ramo da SA candidata deve ser computado separadamente, independentemente do deslocamento sofrido pelo respectivo ramo

pai, em decorrência do fato da envoltória ser função da coluna (que cresce com o deslocamento à direita).

Estes fatores, que decorrem da natureza do problema à luz dos critérios expostos anteriormente, constituem-se na justificativa mais consistente em defesa da análise local, porque caracterizam a dependência da distribuição de símbolos em uma página do particular arranjo da mesma.

Assim sendo, os procedimentos CriaSARamos e ExpandeRamosSA são computados página a página e a melhoria da distribuição de símbolos por análise local paga o tributo do reprocessamento, em páginas posteriores, da informação associada às SAs tornadas pendentes em uma dada página.

5.3.1.5 - Método de inserção ascendente com retirada de SA

Como consequência da necessidade de associar uma prioridade a cada uma das SAs candidatas a entrar na página, em função do número de seus conectores automáticos e do seu fator de ocupação da página, surge a necessidade de definir um valor que seja função de ambos os parâmetros. A cada SA associam-se os seguintes parâmetros : número de conectores automáticos e tamanho da SA (espaço total ocupado pelos ramos). Pode-se então, calcular o valor associado a cada SA pela seguinte expressão :

```
valor = espaço_sa / espaço_página * peso_fator_ocupação
        - n_conectores_automáticos * peso_fragmentação
```

A constante espaco_pagina é o produto do espaço da coluna (diferença entre as cotas dos limites superior e inferior da coluna) pelo número máximo de colunas da página.

O método de inserção ascendente poderia se utilizar da informação do valor associado a cada SA, resultando numa alternativa mista aqui denominada método ascendente com retirada de SA. Neste caso, as SAs seriam inseridas na página na ordem decrescente de cota (método ascendente) até que houvesse conflito. Se dentre as SAs já colocadas, existissem algumas com prioridade menor que a da candidata, estas seriam retiradas uma a uma em ordem crescente de prioridade até solucionar o conflito da candidata ou concluir-se que esta deveria tornar-se pendente. É necessário garantir que, do conjunto das SAs já inseridas e menos prioritárias que a candidata, será retirado o menor sub-conjunto, levando em conta a prioridade entre tais elementos, que solucione o conflito. Para exemplificar, poder-se-ia imaginar a situação de elementos de prioridade decrescente de 1 a 4 onde o conflito é resolvido com a saída de 2 e 4, sem a necessidade da saída de 1 e 3. Conclui-se que o método ascendente com retirada de SA é ineficiente, pois o conjunto de n SAs menos prioritárias que a candidata pode, no pior caso, levar a rearranjos (porque cada subconjunto possível de SAs retiradas leva a um rearranjo de toda a página) cujo número é dado pela expressão :

nrearranjos = Somátoria [p = 1 .. n] An,p
onde An,p = n! / (p! * (n - p)!)

5.3.1.6 - Método de inserção de SA por prioridade

A solução adotada foi o método de inserção por prioridade, dividido em dois procedimentos : OrdenaSaIrmas e EscolheSAIrmas.

O procedimento OrdenaSAIrmas deve considerar três níveis distintos de SAs irmãs, em escala decrescente de prioridade (vide figura 5.16) :

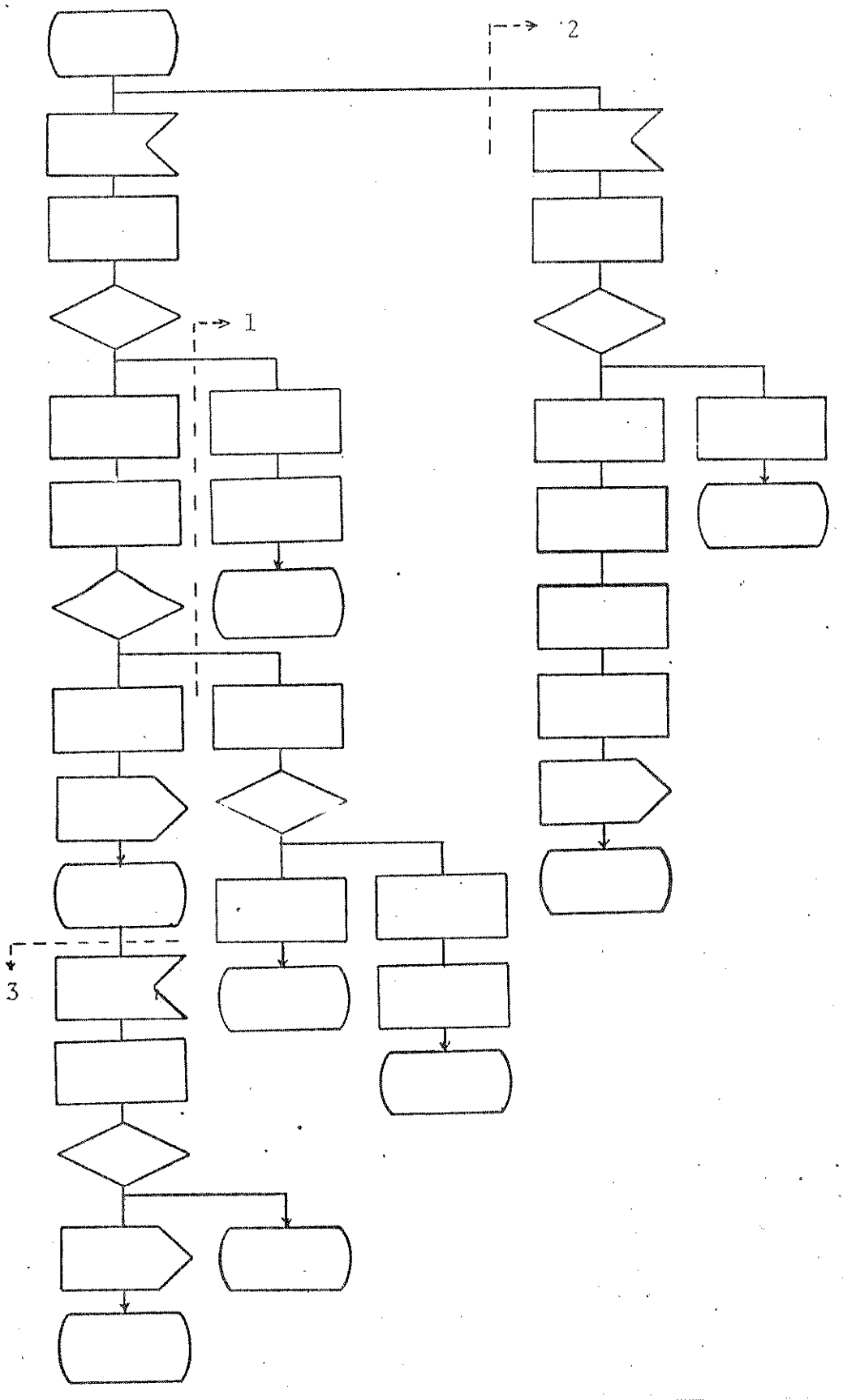


Figura 5.16 - Os três níveis de precedência entre possíveis pendências

- SAs pertencentes à transição e ao estado correntes : as SAs neste nível serão ordenadas segundo o seu valor (maior valor significa maior prioridade);

- SAs pertencentes a novas transições (iniciadas por símbolos filhos do estado corrente) : as SAs deste nível terão sua prioridade associada à sua ordem de ocorrência;

- SAs pertencentes a novo estado : serão reenquadradas nos três níveis e ordenadas segundo proposto acima.

O procedimento EscolheSAIrmas busca inserir as SAs na página segundo a ordem de prioridade traçada pelo procedimento anterior. Sua ação se fundamenta nas seguintes asserções :

- como a tentativa de inserção das SAs é feita por ordem de prioridade, se uma SA é aceita não poderá ser mais retirada;

- uma dada SA candidata separa as SAs já colocadas na página em dois grupos (no caso geral) : as SAs com cotas superiores e as SAs com cotas inferiores à cota da candidata;

- as SAs de cota inferior não podem ser deslocadas para a direita e a envoltória a elas associada serve de base para a expansão dos ramos da SA candidata;

- a persistência de conflito (após o deslocamento à direita) com a envoltória das SAs inferiores torna a SA candidata pendente;

- o ajuste da SA candidata sobre a envoltória das SAs inferiores cria uma nova envoltória que serve de base para o rearranjo das SAs superiores;

- cada SA superior, na ordem decrescente de cota (superior significa menor cota), sofre rearranjo tendo como base a envoltória da SA candidata (para a SA superior logo acima da candidata) e redefinindo-se a envoltória para o rearranjo da SA imediatamente superior à ultima SA rearranjada;

- a persistência de conflito com qualquer SA superior torna pendente a SA candidata;

- se nenhum conflito persistir, a SA candidata é considerada inserida na página;

- se uma SA candidata associada a uma nova transição é tornada pendente, todas as demais SAs menos prioritárias (novas transições ou novos estados) deverão passar a ser pendentes;

- a tentativa de inserção de SA associada a um novo estado deve ser precedida da verificação da não existência de pendências do estado anterior (execução do procedimento OrdenaPendenciasSA).

Pode-se observar que, enquanto o método de inserção ascendente pode provocar um rearranjo (entre as SAs já colocadas na página) para cada configuração possível de retirada de SA (como foi visto anteriormente), o método de inserção por prioridade só provoca um rearranjo entre as SAs já inseridas (no caso mais geral) sendo, portanto, muito mais eficiente.

Descarta-se a possibilidade de reavaliar as prioridades das SAs candidatas à página, face à alteração das envoltórias na página após a colocação de uma SA, pelos seguintes motivos :

- nos dois métodos apresentados, que levam em conta a prioridade das SAs, a expansão de ramos de uma SA até os limites da página para definir os seus possíveis ramos na página ocorre uma só vez, pois os rearranjos resultantes de sucessivas reavaliações da possível composição das SAs (decorrente da reavaliação das prioridades) levaria a um alto custo adicional em tempo de processamento para um ganho insignificante (na grande maioria dos casos) na distribuição de símbolos;

- a continua reavaliação da composição das SAs levaria à tentativa de colocar na página SAs cada vez menores em relação às irmãs colocadas anteriormente, aumentando substancialmente a fragmentação devido à diminuição dos espaços disponíveis.

A estrutura de dados que simplifica o procedimento EscolheSAIrmas é a seguinte :

- coluna corrente e reserva para cada ramo;

- envoltória (indexada por coluna) para cada uma das SAs irmãs colocadas na página, com valor corrente e reserva.

As posições de reserva para coluna de ramo e valor de envoltória são utilizadas para o deslocamento das SAs superiores que precede a rejeição ou aceitação da SA candidata. Se a SA candidata é aceita, as posições correntes são atualizadas pela cópia das respectivas posições reservas; caso contrário, não são alteradas.

O procedimento OrdenaPendenciasSA salva na pilha de pendências o ponteiro do primeiro símbolo de cada uma das pendências de uma dada SA, contidas no seguinte intervalo :

- do início da SA na página até o próximo estado ou até o seu fim (se não houver próximo estado);

- do início do estado da SA na página até o próximo estado ou fim da SA (se não houver próximo estado).

Esta parada ao encontrar um novo estado é exigida pelo procedimento EscolheSAIrmas que deve respeitar o critério de conversão de state/nextstate (que restringe tal conversão à não existência de pendências no estado anterior). A ordem de salvamento de pendências na pilha leva a SA de símbolos a ser percorrida de maneira análoga à descrita no procedimento TrataSA (interno ao procedimento CriaSARamos). Desta forma, a colocação na pilha de pendências obedece à seguinte sequência de prioridades na sua ordem de entrada :

- SAs superiores têm maior prioridade;

- numa dada SA, a prioridade decresce dos ramos folha para o ramo principal;

- num dado ramo, a sua continuação é menos prioritária que as continuações das SAs que dele derivam.

Esta forma de percorrer a SA de símbolos torna-se bastante sintética devido à recursividade e ordena as pendências na pilha de forma que elas sejam consumidas segundo os critérios de ordenação da paginação em função do ramo principal (a primeira pendência a ser consumida será a de continuação do ramo principal) e de tratamento completo de uma dada SA (as pendências dos ramos filhos de um dada SA são consumidos antes das pendências das suas SAs irmãs).

Resta comentar as adaptações nos procedimentos CriaSARamos e ExpandeRamosSA (descritos anteriormente) para incluir as restrições impostas pela análise local numa página finita e pelo interfaceamento com os procedimentos OrdenaSAIrmas e EscolheSAIrmas.

O procedimento CriaSARamos sofre as seguintes alterações :

- a expansão dos ramos na vertical é limitada pela cota do limite inferior da página, resguardado o espaço para colocação de conectores automáticos de saída para ramos truncados;

- a coluna de um ramo filho, que é a priori igual a coluna do ramo pai mais 1, passa a ser limitada pelo número máximo de colunas na página;

- a cada ramo é associado um parâmetro que mensura seu tamanho, que é a soma dos espaços ocupados pelos símbolos, espaços entre símbolos ou destinados a nome de ramo (ramo de decisão), etc;

- se um dado ramo filho tem um ramo pai com comentário (associado a símbolo) ou outro ramo filho no intervalo de cotas compreendido pelo ramo filho em questão, este terá como coluna mínima a coluna do ramo pai mais 2, para que haja entre eles o espaço mínimo necessário para comentário associado ou para conector de saída (em substituição ao outro ramo filho), conforme figura 5.17.

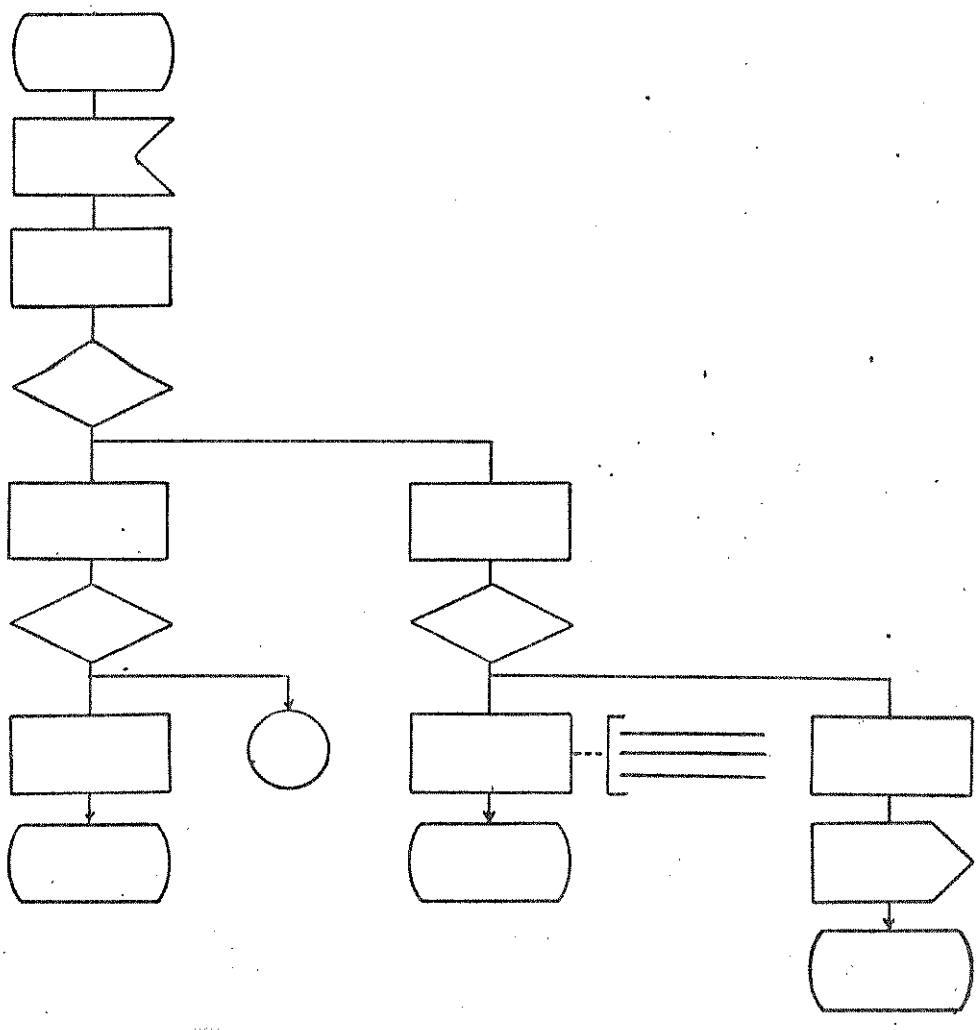


Figura 5.17 - Espaço mínimo entre ramo pai e filho é função de comentário associado ou ramo irmão

O procedimento ExpandeRamosSA sofre as seguintes alterações :

- restrições idênticas de cota do limite inferior da página e número máximo de colunas na página;

- no processamento que realiza ao percorrer as SAs dos ramos folha em direção ao ramo principal, chama os procedimentos OrdenaSAIrmas e EscolheSAIrmas (que, por sua vez, chama o procedimento OrdenaPendenciasSA quando um novo estado é achado ou quando finaliza a expansão da SA principal) a cada nível que regride na SA em direção ao seu ramo principal;

- a cada nível contabiliza o número de conectores automáticos e o tamanho da SA, de forma que uma dada SA totalize os conectores automáticos e o tamanho de todos os seus ramos.

Claramente, a descrição dos procedimentos utilizados na distribuição de símbolos foi feita de forma a salientar as ações básicas realizadas para que todos os aspectos fundamentais do problema fossem abordados. O enfoque buscou enfatizar também a conveniência do tratamento recursivo e o perfil das estruturas de dados adequadas.

Existe um efeito de borda que não é tratado nem pela otimização de bordas, nem pela escolha das SAs na página. Muitas vezes um ramo pendente tem poucos símbolos, e a sua representação isolada do seu ramo pai pode ser inconveniente sob o ponto de vista de representação contígua do desenho. Neste caso pode-se forçar a pendência do ramo ao qual o ramo pequeno se filia, de forma a representar contiguamente um número maior de símbolos.

5.3.2 - Linhas de Conexão

Inicialmente são definidos alguns conceitos, e analisadas as diversas formas de ocorrência de linhas de conexão. A seguir, é proposta uma sistematização para o tratamento das mesmas.

A substituição de símbolos de labels e joins por linhas de conexão contribue para maior clareza na representação e concretiza o que o usuário faria manualmente. A ligação entre os dois pontos deve se utilizar de um número máximo bem definido de segmentos de retas, que deverá cobrir a grande maioria dos casos e tornar viável a sua automação. Para isto é suficiente a utilização de linhas de conexão formadas por um segmento (de reta) vertical e dois segmentos horizontais (algumas conexões necessitam apenas de um segmento horizontal). A figura 5.18 mostra três tipos de conexões.

Os segmentos verticais (paralelos às colunas) não devem ocupar uma nova coluna para sua implementação, uma vez que o recurso coluna é escasso e deve ser poupado. Definem-se desta forma espaços entre colunas a serem utilizados pelos segmentos verticais. Estas retas paralelas, por onde podem passar os segmentos verticais, são chamadas vias de conexão e numeradas de 0 a 3. A via 0 é coincidente com a linha que centra os símbolos na coluna e as vias de 1 a 3 ocupam o espaço que sobra entre o final de um símbolo grande e o início da nova coluna. A caracterização completa de uma dada via de conexão é feita através do par número da coluna e número da via na coluna. A figura 5.19 mostra a utilização das vias associadas a uma dada coluna.

Pode-se ainda caracterizar dois tipos de conexões especiais :

- conexões paralelas : quando duas ou mais conexões têm seus segmentos verticais paralelos (vide figura 5.20);

- conexões múltiplas : quando duas ou mais conexões têm o mesmo label como terminador (vide figura 5.21).

5.3.2.1 - Análise dos diferentes tipos de conexões possíveis

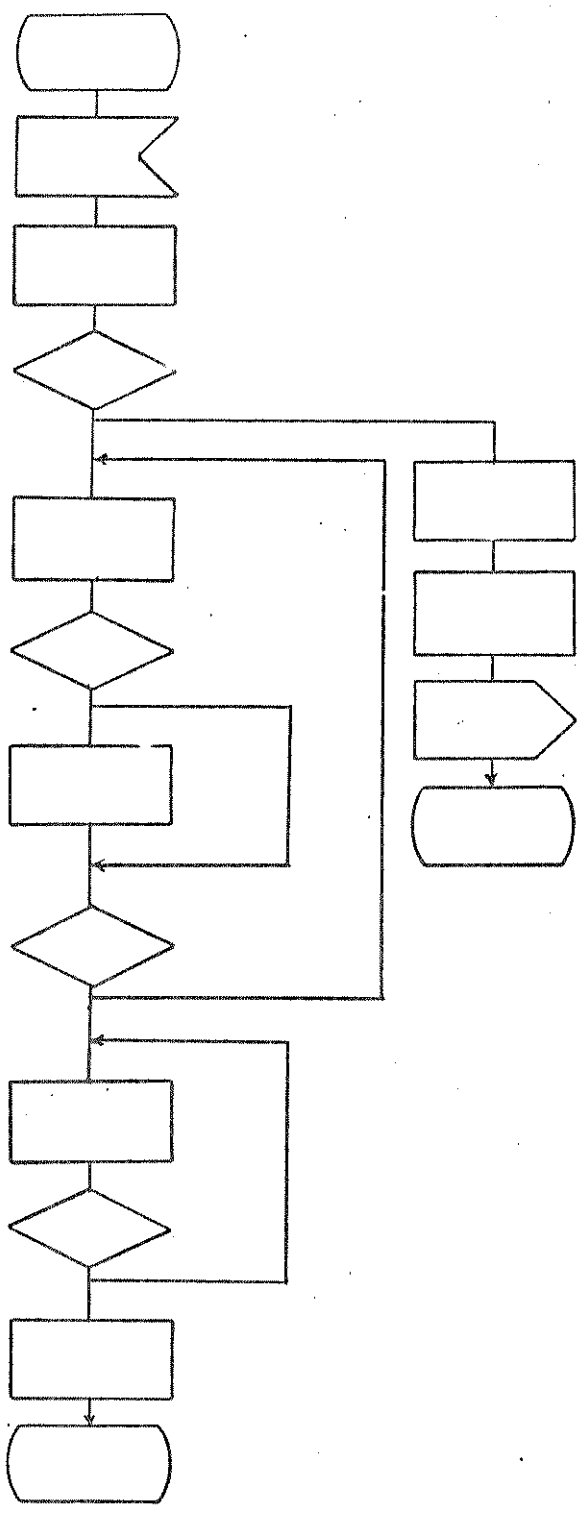


Figura 5.18 - Utilização de vias de conexão

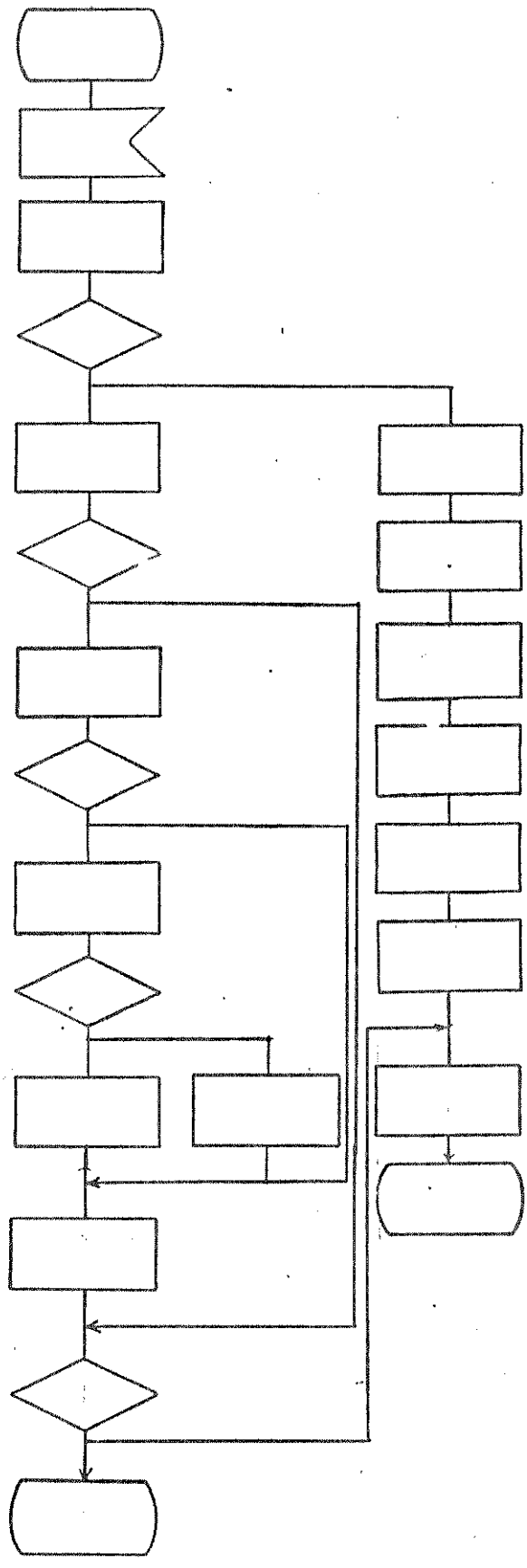


Figura 5.19 - Utilização das 4 vias de conexão associadas a uma dada coluna

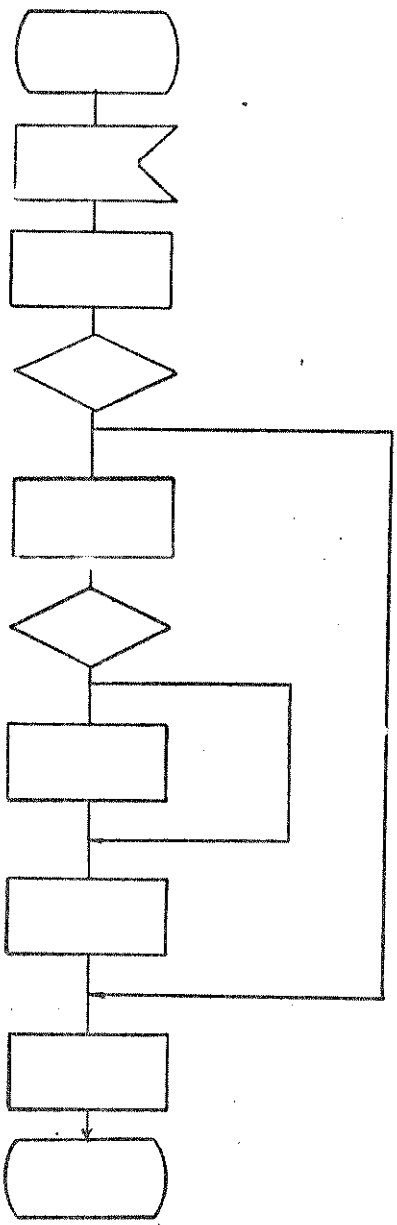


Figura 5.20 - Conexões paralelas

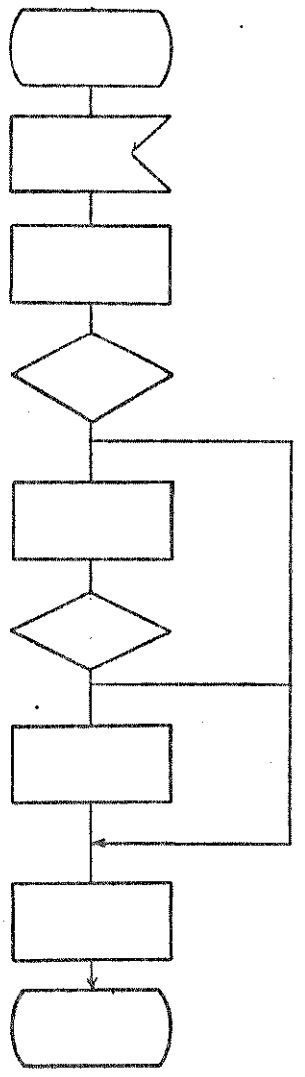


Figura 5.21 - Conexões múltiplas

Existem dois tipos de conectores (originadores de conexões) quanto à ocorrência dos mesmos nos ramos. O conector (join) pode aparecer no final de um ramo ou como único elemento de um ramo. A figura 5.22 exemplifica as conexões originadas por um join como único símbolo do ramo e a figura 5.23 exemplifica as conexões originadas por join fim de ramo.

Pode-se classificar as conexões quanto à ocupação da via de conexões em grupos com a mesma característica. Pode-se enumerar cinco grupos, a saber :

- conexões simples que podem ocupar o mesmo caminho na via (vide figura 5.24);

- conexões múltiplas que ocupam o mesmo caminho na via (vide figura 5.25);

- conexões que ocupam caminhos paralelos na via (vide figura 5.26);

- conexões que ocupam apenas a via de conexão central (via 0) da coluna (vide figura 5.27);

- situações de conflito ou cruzamento entre candidatas à linha de conexão, quando é necessário escolher aquela que será implementada; este último grupo é discutido abaixo.

Conflitos podem ocorrer quando as extremidades dos segmentos verticais se interceptam ou quando um segmento vertical é interno ao outro. Conflitos que ocorrem quando extremidades de segmentos verticais se interceptam, podem ser exemplificados de duas formas : labels numa extremidade (vide figura 5.28) e label no centro (vide figura 5.29). Conflitos com um segmento vertical interno ao segmento vertical da conexão paralela são exemplificados pela figura 5.30.

Face à generalidade de tipos nos quais as conexões podem ser enquadradas, seria desejável a busca de um denominador comum entre

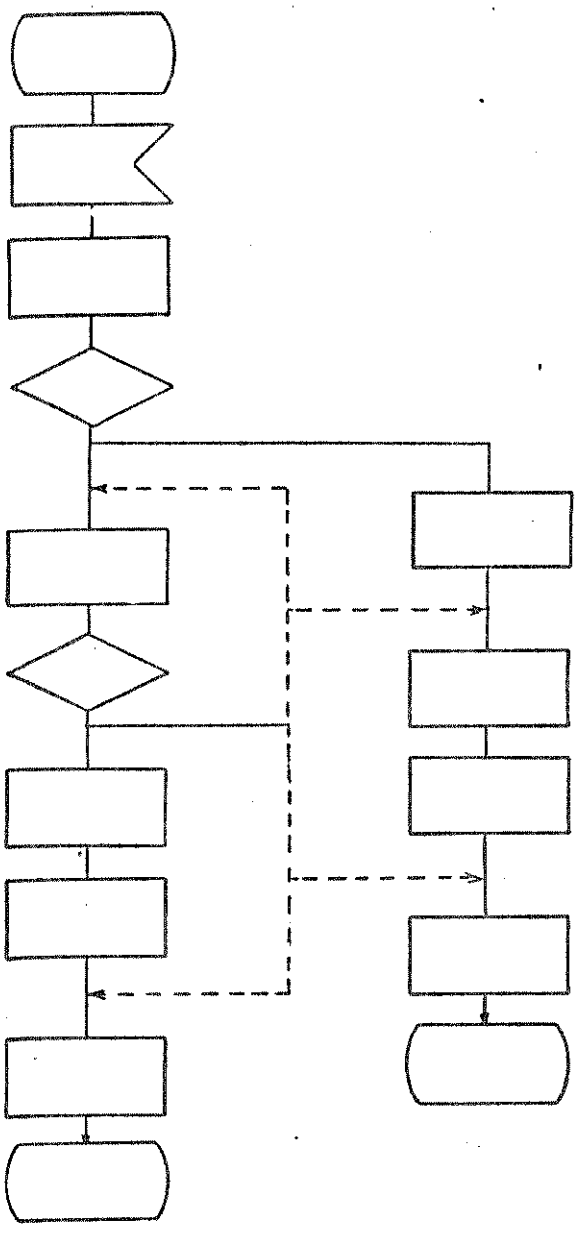


Figura 5.22 - Conexões geradas por um Join único no ramo

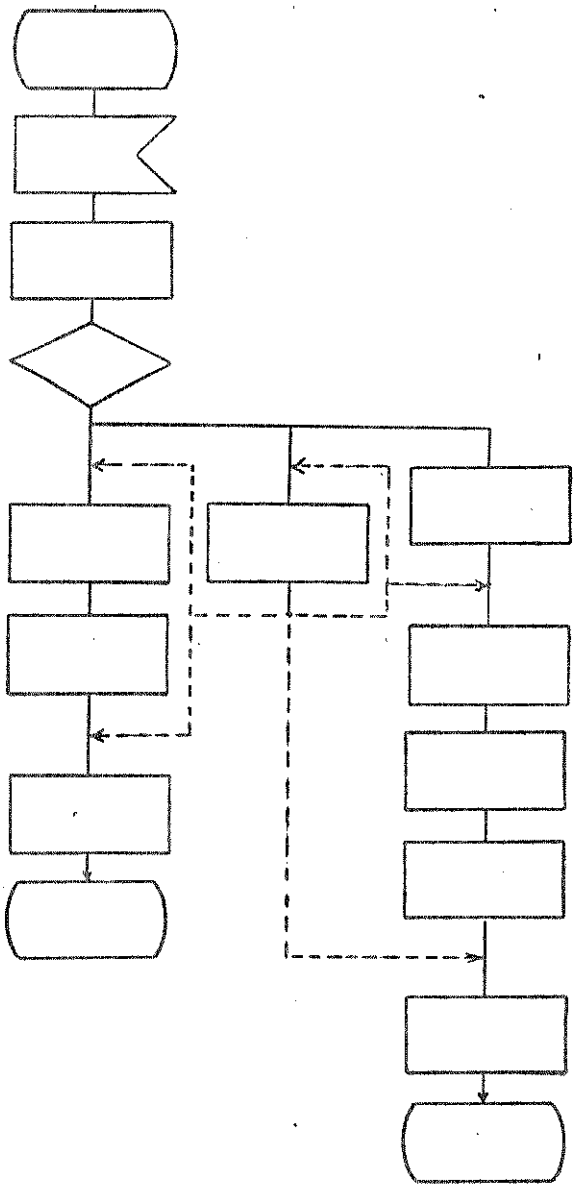


Figura 5.23 - Conexões geradas por um Join fim de ramo

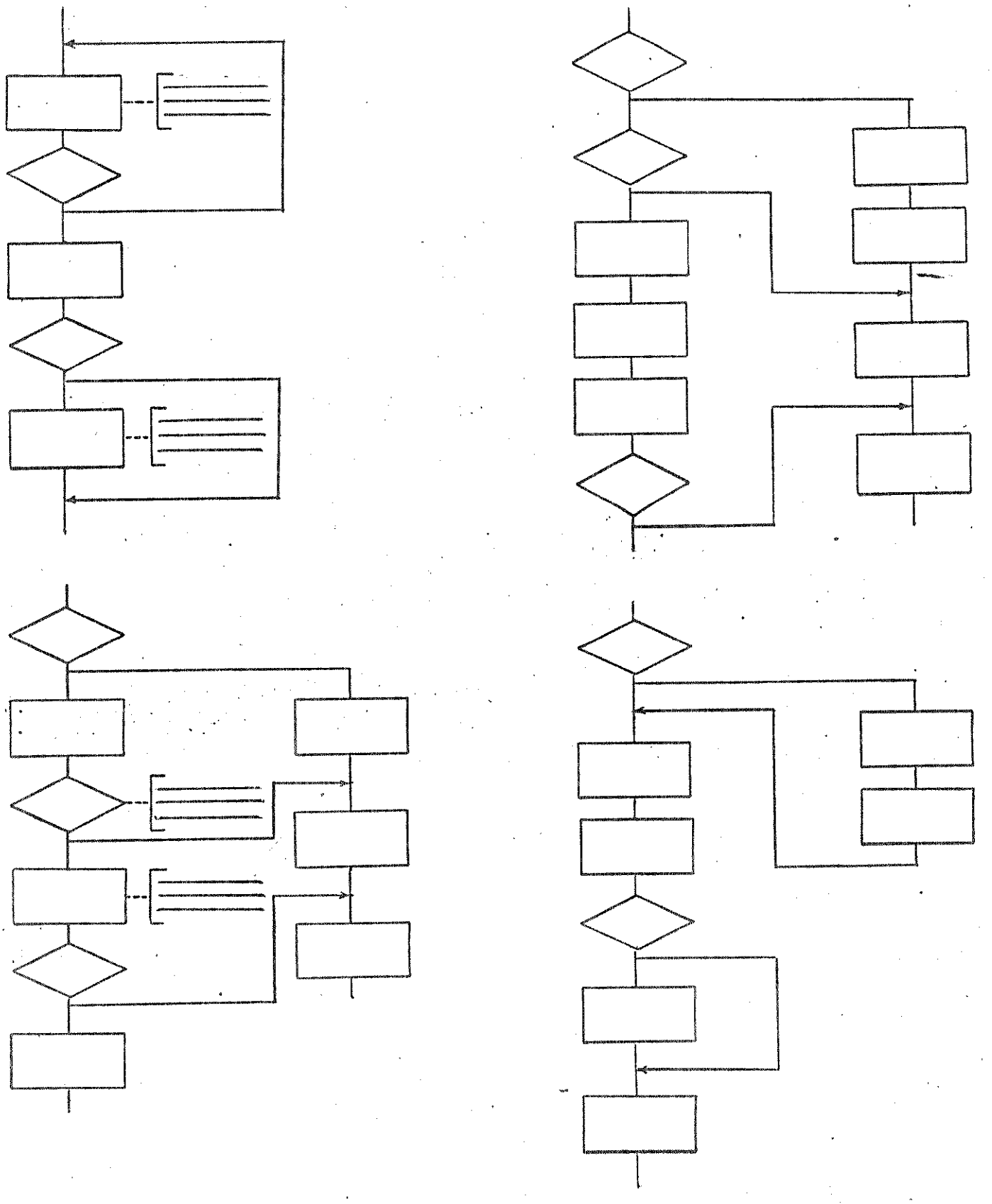


Figura 5.24 - Conexões simples que podem ocupar o mesmo caminho na via

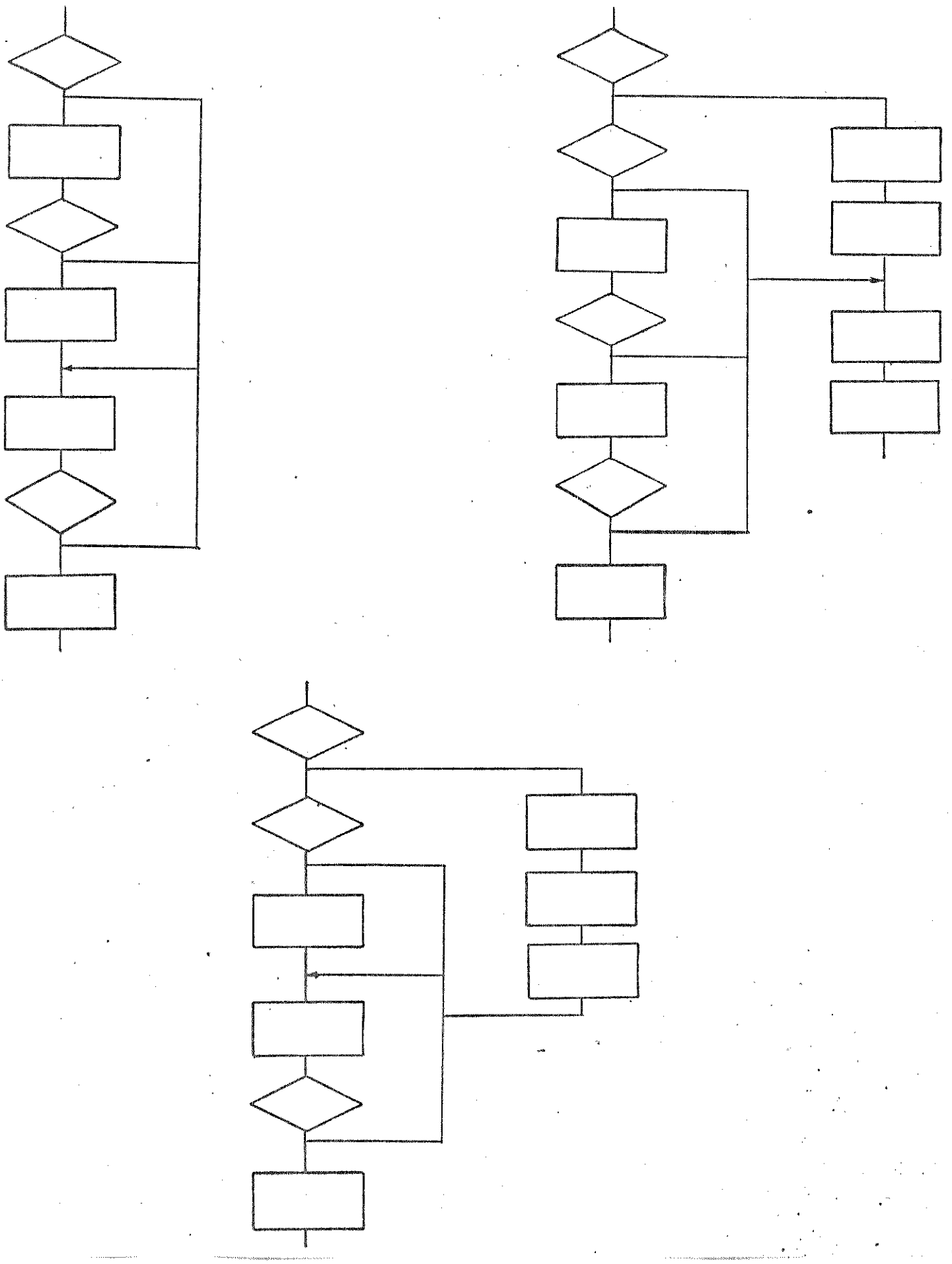


Figura 5.25 - Conexões múltiplas que podem ocupar o mesmo caminho na via

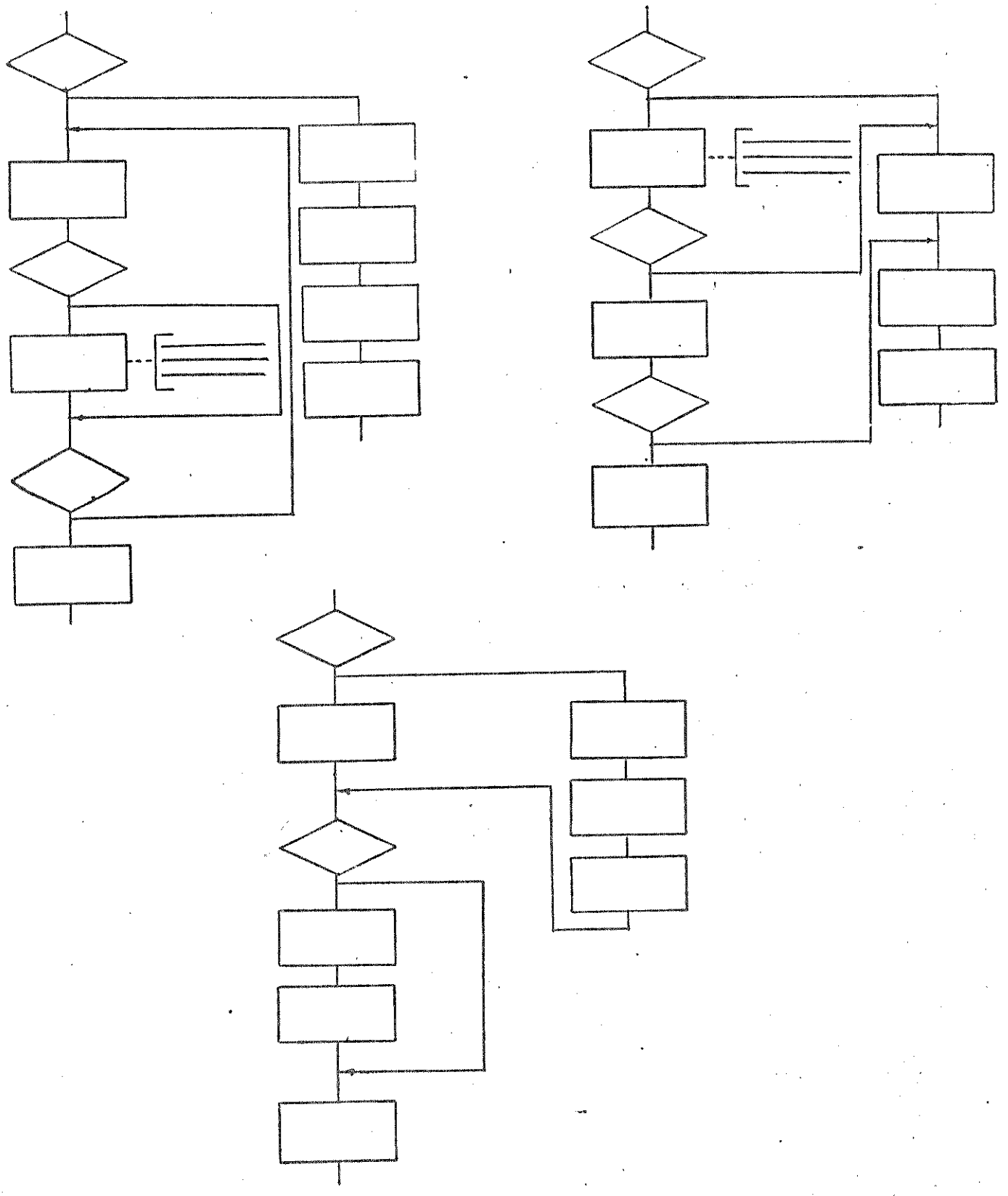


Figura 5.26 - Conexões que ocupam caminhos paralelos na via

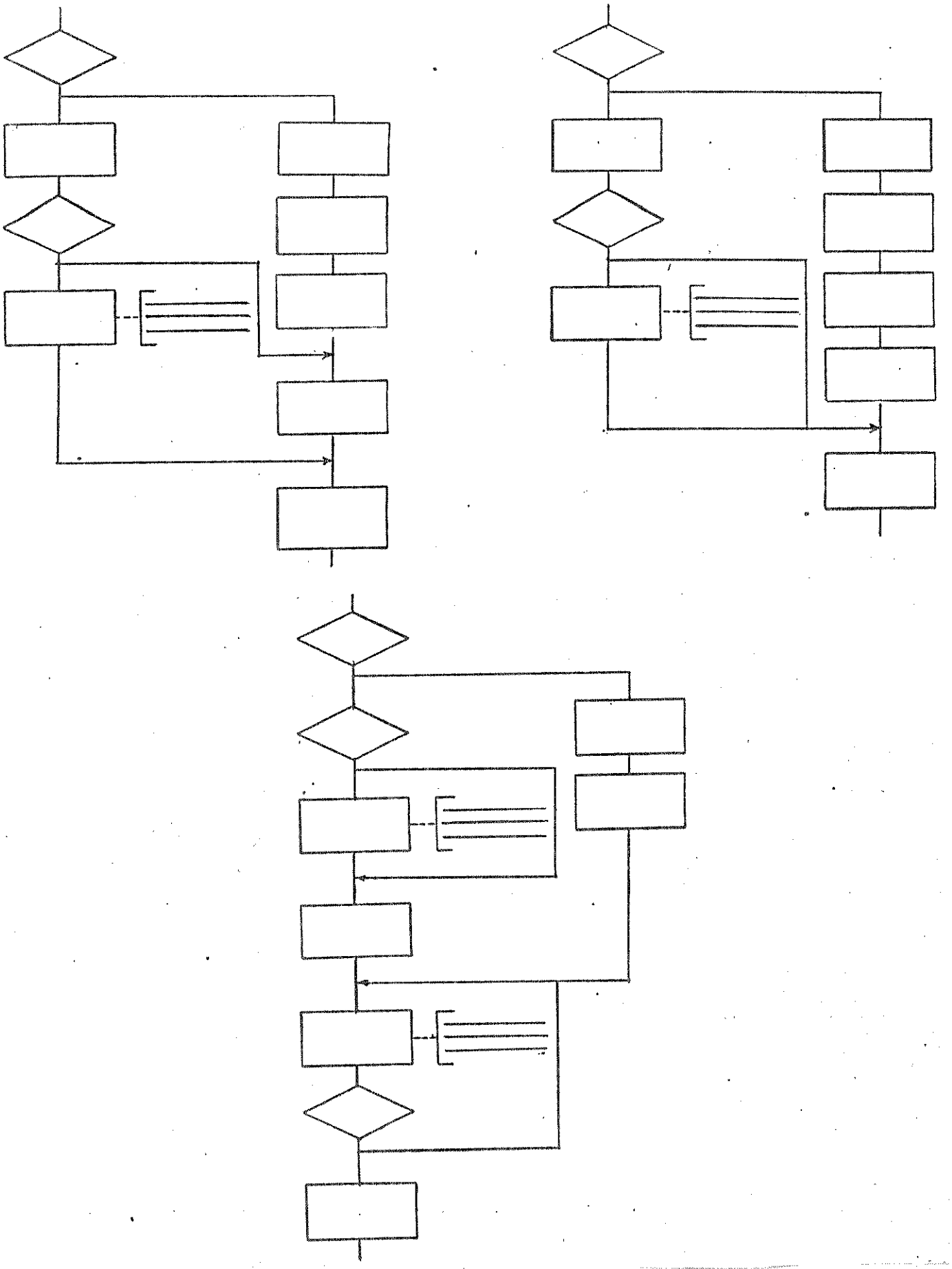


Figura 5.27 - Conexões nas quais uma das vias ocupadas é a própria coluna

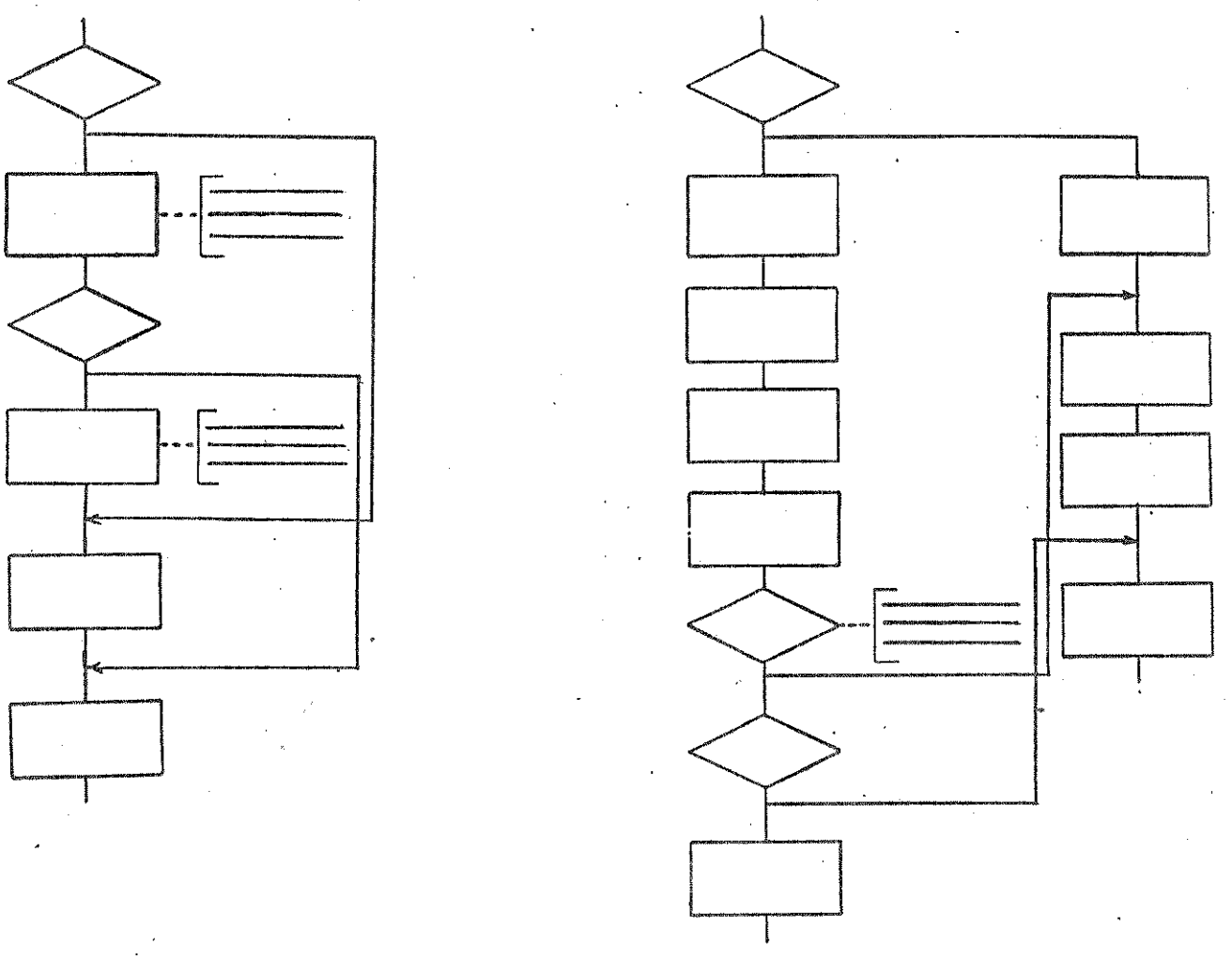


Figura 5.28 - Conflitos nos quais os Labels estão numa dada extremidade

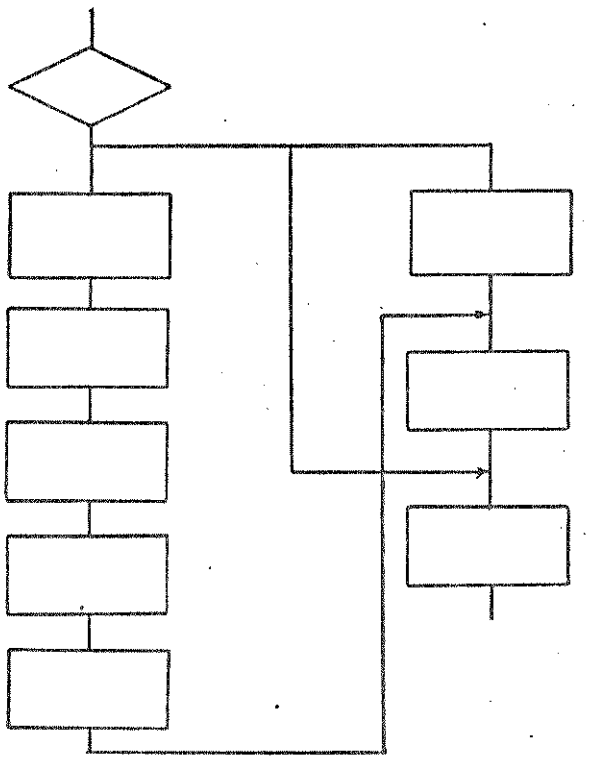


Figura 5.29 - Conflito no qual os Labels estão no centro

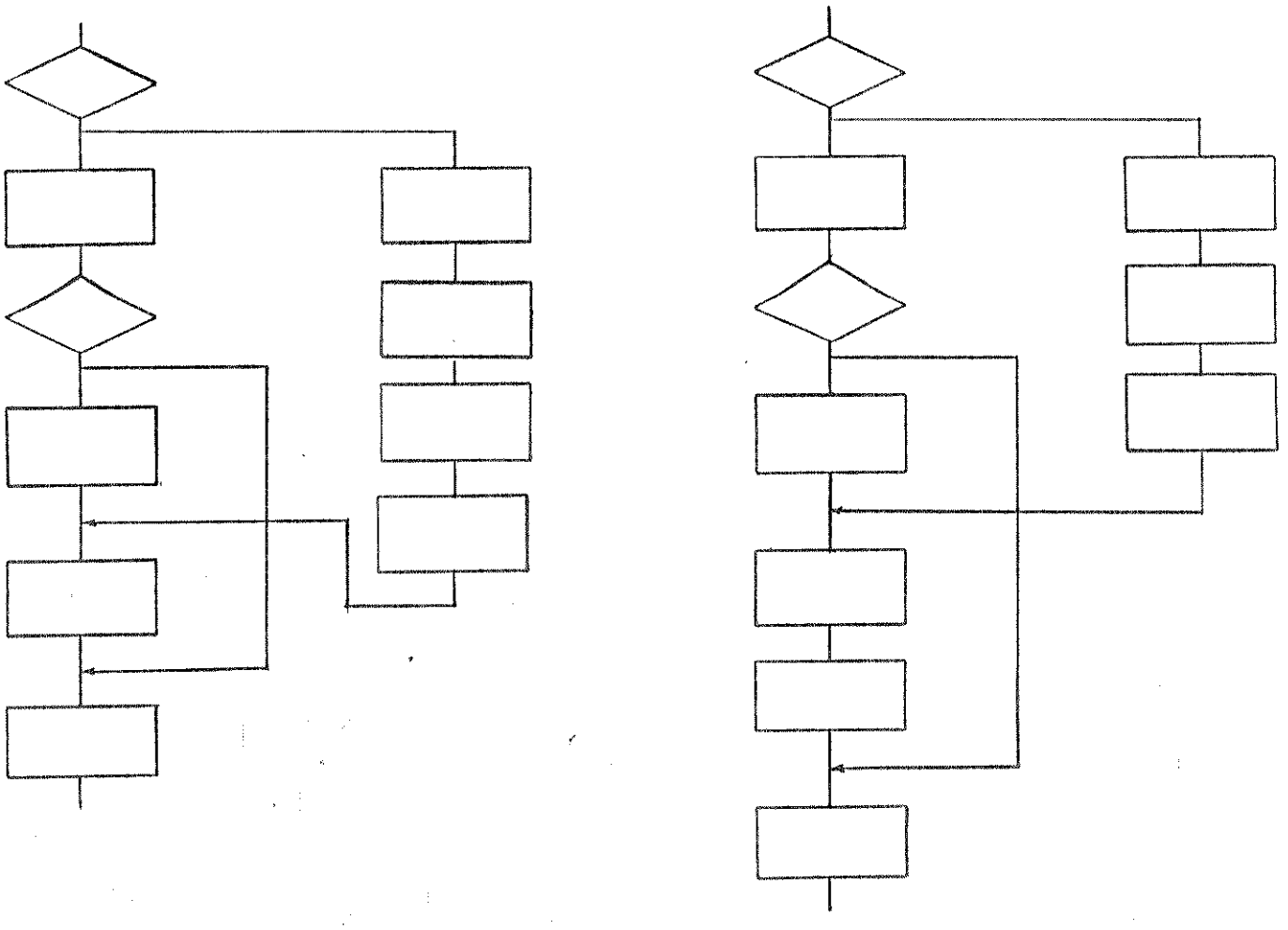


Figura 5.30 - Conflitos nos quais existe um segmento vertical interno a outro

as linhas de conexões possíveis para simplificar e uniformizar o procedimento para tratá-las.

5.3.2.2 - Estrutura de representação de segmentos

Inicialmente é necessário catalogar os pares label-join candidatos a serem substituídos por linhas de conexões numa dada página. Após a estabilização das sub-árvores que compõem a página é feito o levantamento dos label e joins da página e, do seu exame, resultam os pares label-join candidatos. É interessante observar dois fatos relacionados à substituição de labels e joins :

- alguns labels podem aparecer em dois ou mais pares candidatos distintos por estarem referenciados por joins distintos;

- a implementação de uma linha de conexão nem sempre acarreta a substituição do label, pois este pode estar envolvido em outras conexões não realizáveis (joins em outras páginas, ou conflito com os símbolos ou conexões existentes na própria página); mas a linha de conexão é sempre vantajosa porque, na pior das hipóteses, ocasiona a substituição do join (vide figura 5.31, mostrando os dois tipos de conexões : com e sem substituição de label).

Após a determinação dos pares candidatos é necessário verificar a viabilidade de cada par, restringindo-se à utilização de, no máximo, um segmento vertical e dois horizontais, e posicionando-se seus segmentos na página de forma a não entrar em conflito com árvores de símbolos e as conexões já implementadas. A avaliação da viabilidade de uma dada linha de conexão através de acessos às SAs de símbolos e à estrutura de dados das conexões colocadas anteriormente, precedentes à candidata em análise, envolveria um custo muito oneroso em termos de processamento. É

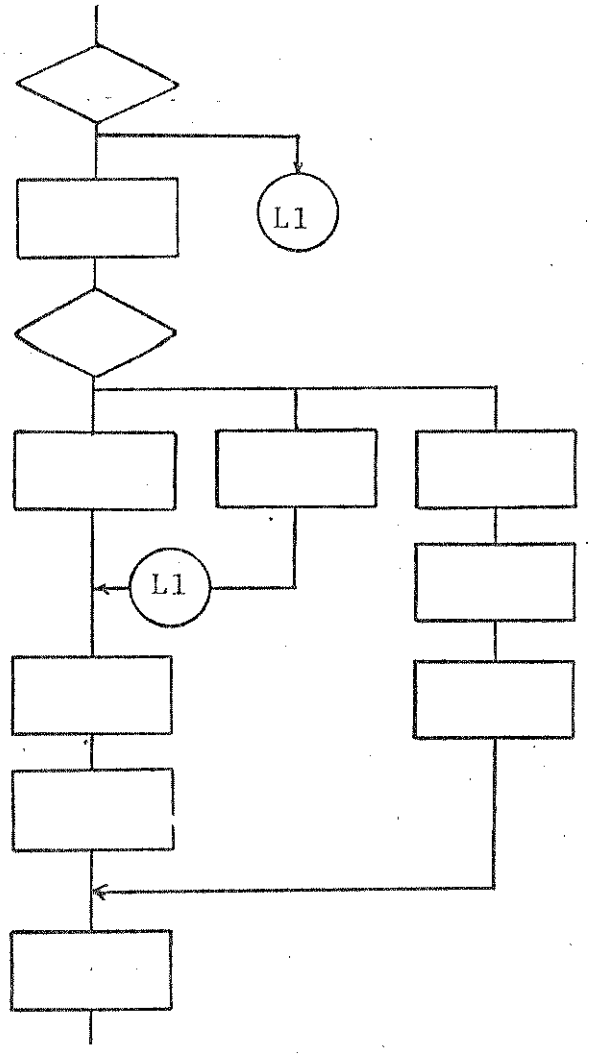


Figura 5.31 - Conexão com ou sem substituição de Label e Join

necessário encontrar uma estrutura de dados própria, capaz de refletir de maneira sintética a ocupação da página através de símbolos, traços de união (verticais e horizontais), comentários associados a símbolos e conexões anteriormente colocadas na página.

A estrutura de dados adequada deve representar a ocupação das vias de conexões associadas a todas as colunas da página. Pode-se observar que todos os elementos dos quais se deseja inferir a ocupação nas vias de conexão podem ser representados por segmentos verticais, ou seja, pelas extremidades de seus segmentos (nesta representação os segmentos horizontais teriam suas extremidades coincidentes). A estrutura de dados escolhida compõe-se de vetores com as seguintes características :

- tem como conteúdo os pares de pontos associados às extremidades dos segmentos que ocupam uma determinada via de conexão, organizados em ordem crescente das cotas das suas extremidades superiores (pode-se tomar as cotas superiores ou inferiores como referência);

- são indexados por três parâmetros : número da coluna, número da via de conexão e número do segmento no vetor;

- a cada vetor é associado um contador que totaliza o número de segmentos que o vetor contém.

O processo de introdução de um segmento no vetor que representa uma dada via de conexão deve posicioná-lo em função das cotas de suas extremidades em relação às dos segmentos já existentes na via e incrementar o número de segmentos da via.

5.3.2.3 - Prioridade entre linhas de conexão

Antes de definir como será a verificação de viabilidade de uma dada conexão em relação aos segmentos pré-existentes, deve-se

considerar em que ordem as candidatas deverão ser testadas. A permanência do conflito após o exame de todas as possibilidades de ocupação de espaços vazios pelos segmentos que compõem a candidata à conexão inviabiliza a implementação da conexão. No entanto, o conflito entre linhas de conexão requer que se estabeleça uma prioridade entre elas, mesmo porque a estrutura de dados proposta implica a estabilidade dos segmentos das conexões colocadas anteriormente à candidata em questão.

Para poder ordenar por prioridade os pares label-join candidatos a linhas de conexão, é necessário associar-lhes parâmetros que permitam mensurar o grau de preferência de uns em relação aos outros. A definição de tais parâmetros é motivada pela natureza do problema, ao dar prioridade à candidata cujos símbolos label-join a serem substituídos estejam mais próximos. Tal evidência apoia-se nos seguintes fatos :

- A substituição de um par label-join por uma linha de conexão concorre para clareza e legibilidade, tanto mais substanciais e desejáveis quanto maior a proximidade dos símbolos a serem substituídos.

- A probabilidade de existirem espaços vazios para viabilizar a inserção dos segmentos da candidata à linha de conexão é maior quando os símbolos de label e join estão mais próximos. Na figura 5.32, se a conexão de maior segmento vertical e de label e join em colunas mais distantes tivesse sido considerada prioritária e ocupado a primeira via disponível (ou seja, a da extrema esquerda, pois a alocação de coluna é sempre de esquerda para a direita), as demais linhas de conexão não teriam sido implementadas.

Pode-se associar a cada candidata à conexão os parâmetros delta-coluna (diferença entre as colunas do label e do join) e delta-cota (diferença entre a cota do label e do join). Outro fator a ser considerado é a direção da linha de conexão, que pode

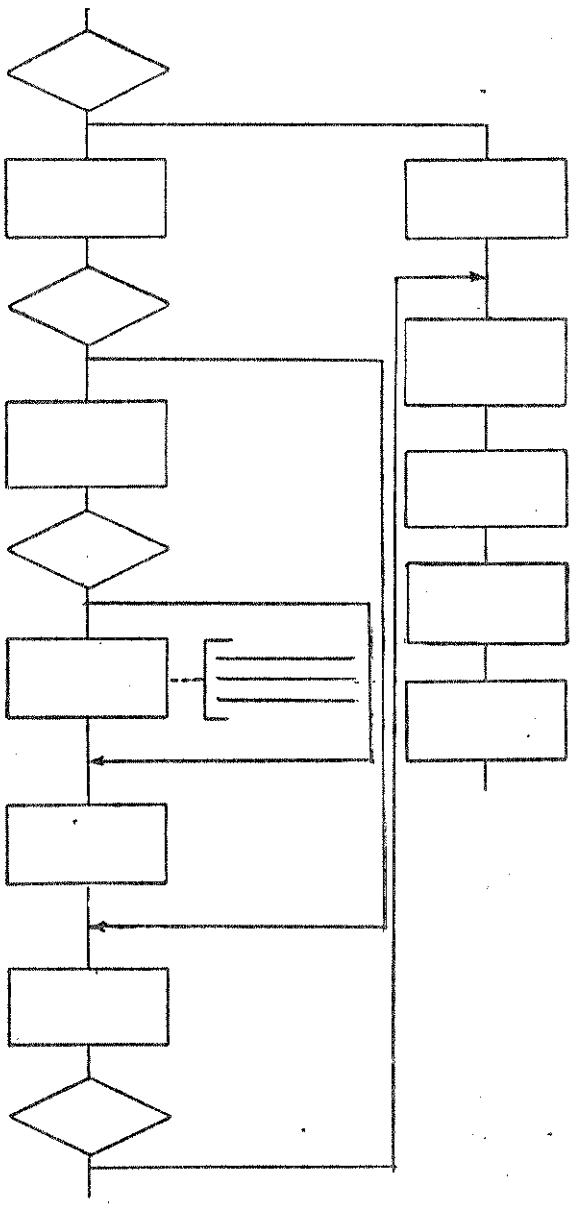


Figura 5.32 - A conveniência de se atribuir maior prioridade às conexões mais próximas

ser : da esquerda para a direita ou vice-versa (em função da diferença de colunas),e de cima para baixo ou vice-versa (em função da diferença de cotas). Como o sentido da expansão dos ramos das SAs é da esquerda para a direita e a representação de saltos condicionais (join acima do label) tende a ser mais frequente do que a de loops (join abaixo do label), convencionou-se considerar mais prioritários (em relação aos seus opostos) os sentidos : da esquerda para a direita e de cima para baixo. Pode-se observar que :

- coluna (label) - coluna (join) = delta_coluna >= 0;
- cota (label) - cota (join) = delta_cota >= 0.

Assim sendo, as candidatas serão ordenadas por prioridade, observando as seguintes condições em ordem decrescente de grau de importância :

- menor delta-coluna;
- delta-coluna positivo (entre candidatas de mesmo valor absoluto de delta-coluna);
- menor delta-cota;
- delta-cota positivo (entre canditadas de mesmo valor absoluto de delta-cota);
- ordem de ocorrência (para efeito de desempate).

5.3.2.4 - Procedimentos adotados

São analisados abaixo os procedimentos utilizados para :

- iniciação dos segmentos referentes às SAs de página;
- definição dos pares label-join;
- ordenação das candidatas;
- exame das alternativas existentes para os segmentos de uma candidata;

- inclusão de tais segmentos nos vetores de segmentos das vias de conexão.

O preenchimento dos vetores das vias de conexão (iniciação dos segmentos referentes aos símbolos colocados na página) bem como o levantamento dos labels e joins existentes na página exigem que os símbolos das SAs da página sejam percorridos após a sua estabilização (pois só então a informação requerida será definitiva). As SAs de símbolos são percorridas de forma recursiva, como foi descrito anteriormente, e o preenchimento dos vetores ocorre naturalmente (devido à forma de percorrer as SAs de símbolos) na ordem crescente das cotas.

Após o levantamento dos labels e joins da página, são formados os pares label-join para cada join da página que referencia um label na mesma. Estes pares são os candidatos a serem substituídos por linhas de conexão (lembrando que um label pode constar de mais de um par). São calculados os parâmetros delta-coluna e delta-cota para cada par e os pares são ordenados segundo a prioridade descrita acima para candidatos a linhas de conexão.

No exame das alternativas possíveis para a implementação de uma linha de conexão associada a uma candidata, é necessário verificar a viabilidade (ou seja, garantir a ausência de conflitos com os segmentos presentes) dos segmentos que a compõem.

A viabilidade de uma linha de conexão apoia-se nas seguintes condições :

- existência de uma via de conexão que possa ser alocada ao segmento vertical, cujo espaço ocupado é equivalente ao delta-cota, se este for maior que zero (o join e label podem estar em colunas distintas, mas alinhados na cota, de tal forma que delta cota igual a zero, não havendo necessidade de segmento vertical para a linha de conexão);

- os dois segmentos horizontais (ou o único, se o segmento vertical for alocado na mesma via de conexão que ocupa o join originador da linha de conexão) que unem o join e o label associados à conexão às respectivas extremidades do segmento vertical, são representáveis (como segmentos com extremidades coincidentes) nas vias de conexão que se situarem entre as suas extremidades (de cada um dos segmentos horizontais).

O segmento vertical poderá ser alocado dentro de um determinado intervalo de vias de conexão. Se delta-coluna é maior que zero, as vias devem pertencer às colunas cuja extremidade esquerda é a coluna do join e cuja extremidade direita é a coluna do label menos 1. Se delta-coluna for igual a zero, o intervalo deverá ser limitado à esquerda pela coluna do join e à direita pela última coluna da página. Se delta-coluna é menor que zero, o intervalo deveria estar, por analogia, compreendido entre a coluna do label (à esquerda) e a coluna do join menos 1 (à direita). No entanto, outros fatores devem ser considerados : delta-cota menor que zero e ramo constituído apenas por um join. No caso de delta-cota menor que zero, o segmento vertical poderá necessitar sua alocação numa via de conexão associada a uma coluna mais à direita da coluna do join menos 1. No caso do ramo join, a limitante da esquerda deve ser a própria coluna do join devido à necessidade de se associar graficamente o segmento vertical com o texto do nome do ramo. Ainda no caso do ramo join, haverá necessidade de uma limitante à direita se tal ramo tiver um irmão lateral direito, pelo mesmo motivo de associação gráfica entre o texto do nome do ramo e o segmento vertical da conexão.

Inicialmente deve-se tentar alocar o segmento vertical na via de conexões do próprio join, o que nem sempre é possível. Se for possível, haverá apenas um segmento horizontal (da extremidade inferior do segmento vertical ao label). Quando for necessário

buscar uma via no intervalo de colunas descrito acima, haverá dois segmentos horizontais. Neste caso, a alocação do segmento vertical numa dada via de conexão (limitada pelo intervalo de colunas) será experimentada da esquerda para a direita. Desta forma, se há conflito numa dada via deve-se tentar a via à direita, se esta estiver no intervalo permitido. Neste ponto, pode-se realizar uma otimização relacionada com o exame dos segmentos horizontais que estiverem à esquerda da via verificada para alocação do segmento vertical (pode ser apenas um deles ou ambos). A busca nas vias à direita da via verificada só deve prosseguir (em caso de conflito com o segmento vertical) quando os segmentos com cota superior e inferior coincidentes associados aos segmentos horizontais que virão a ocupar a via verificada (ou seja, cujo join e/ou label associados estiverem à esquerda desta via) não entrarem em conflito com os segmentos da mesma. Assim sendo, quando ocorre a alocação da via para o segmento vertical, há no máximo um segmento horizontal a ser examinado devido aos seguintes motivos :

- segmento vertical na via de conexão do próprio join, ou seja, a linha de conexão só tem um segmento horizontal;

- pelo menos um dos segmentos horizontais é examinado durante a busca da via de conexão para o segmento vertical.

Resta, finalmente, analisar as situações de conexões múltiplas, ou seja, cujo terminador é o mesmo label. Quando os segmentos da segunda conexão (múltipla em relação a uma linha de conexão já alocada na página) são testados, haverá conflito de um dado trecho (parte de um dos segmentos e/ou segmento inteiro) da linha em teste com os segmentos da conexão múltipla que a antecedeu. Diversas situações são possíveis e tentar analisar caso a caso seria trabalhoso. A solução mais simples é a de associar um terceiro parâmetro a cada segmento que compõe os

vetores que representam as vias de conexão, que seria o label associado à linha de conexão do qual o segmento faz parte (se o segmento não for componente de linha de conexão, ou seja, correspondente a símbolos ou ligação entre símbolos, tal parâmetro será zerado). Com a introdução deste novo parâmetro, pode-se voltar ao mesmo procedimento utilizado para conexões não múltiplas, com a seguinte adaptação :

- buscar alocar os segmentos sem se preocupar com possíveis linhas de conexões múltiplas já alocadas;

- a caracterização de um conflito só ocorre se os labels associados aos segmentos em conflito forem distintos;

- a inserção de um segmento num dado vetor associado a uma via de conexão pode, quando existem conexões múltiplas, ocasionar apenas a ampliação do espaço ocupado por um dado segmento, através da atualização de uma ou de ambas as extremidades (cota superior e inferior) que o compõem.

Conclui-se que a substituição de pares label-join numa dada página, através da solução proposta que pode ser denominada alocação de linhas de conexão por segmentação, traduzem-se em um algorítmo simples e genérico que permite verificar todas as possibilidades existentes numa dada página.

5.4 - Comparação com técnicas existentes

Neste item, as soluções adotadas para o SADG são comparadas com técnicas utilizadas em geradores autoráticos de fluxogramas e em roteamento de circuito integrado.

5.4.1 - Geradores Automáticos de Fluxogramas

A publicação de trabalhos sobre geração automática de fluxogramas despertou o interesse da comunidade científica no período desde o final da década de 1950 até a metade da década de 1970. Sua utilização para geração de documentação a posteriori buscando representar a implementação mostrou-se pouco útil, devido ao nível de detalhe que incorpora e ao fato de não fazer parte do processo que leva à concepção da implementação.

À medida que o custo de software nos projetos de grande porte passou a representar parcela significativa no custo do desenvolvimento, com a necessidade da padronização da especificação e do particionamento de sistemas complexos em camadas com níveis distintos de detalhamento, as linguagens de especificação emergem como solução natural para o projeto e a documentação do software.

A representação de uma linguagem de especificação em forma textual e gráfica (ambas biunivocamente correspondentes), bem como o advento das impressoras e terminais de vídeo gráficos, reacende o interesse pela geração automática de fluxogramas (utilizada na conversão da forma textual da linguagem de especificação para a sua forma gráfica correspondente).

A caracterização da geração de fluxogramas para as linguagens de especificação, se distingue especialmente por dois aspectos :

- a especificação antecede a implementação e está num nível de detalhe mais alto, ou seja, a implementação é a realização do fluxograma prescrito na especificação;

- a representação de níveis de detalhe emerge naturalmente dos diferentes níveis de documentação da especificação e no nível mais baixo é possível evidenciar pontos comuns ou repetitivos através do uso de procedimentos e macros.

Os geradores automáticos de fluxogramas podem ser divididos em unidimensionais e bidimensionais (alguns deles ditos

bidimendionais são na realidade bons geradores unidimensionais) [WATK73].

O primeiro gerador automático de fluxograma unidimensional foi proposto por Scott [SCOT58] e apesar de muito simples, trouxe a base para os sistemas posteriores. O programa fonte é listado na sua sequência, sem manipulação na ordem das declarações, que são representados cercados por blocos retangulares. Linhas à direita dos blocos representam transferência de controle por saltos condicionais e, à esquerda, transferência de controle por loops. Tais linhas se interceptam e se distinguem pelo par (ponto de saída e ponto de entrada nos blocos). Dois pontos de crítica são que a abrangência dos blocos em relação às declarações que contêm não tem nenhuma razão racional e sua descrição se baseia na listagem de programa ao invés de descrever os passos lógicos envolvidos (o que ocorre naturalmente quando se utiliza uma linguagem de especificação como SDL).

Knuth [KNUT63] descreve um gerador similar, que visa fornecer documentação para aprendizado sobre um dado programa. Preconiza três níveis de documentação : passos do algoritmo em forma textual, fluxograma e listagem de programa (que na realidade são três formas distintas de representar o mesmo nível de detalhe). As secções do programa encapsuladas por blocos são fornecidas pelo usuário através de labels e comentários acrescentados à implementação. Linhas de tranferência de controle são semelhantes às descritas por Scott (aliás, as duas abordagens são semelhantes, a menos dos labels fornecidos pelo usuário para distinguir os vários blocos).

Outros geradores unidimensionais usam técnicas semelhantes para produzir a forma gráfica mas utilizam diferentes notações :

- Saalbach e Sapovchak [SAAL65] utilizam um algoritmo mais simples de geração e adotam o esquema de Knuth em relação aos comentários de usuários;

- FLOWGEN/F [CALC68] foi implementado seguindo o método de Scott;

- O'Brien e Beckwith [BECK68] descartaram notação, gerando apenas os blocos para serem preenchidos pelos usuários;

- Sherman [SHER66] com FLOWTRACE emprega uma tabela dirigida por sintaxe para reconhecimento da linguagem, que permite selecionar o tipo de notação a ser utilizada.

A primeira tentativa para produzir geradores automáticos de fluxogramas bidimensionais é de Haibt [HAIB59] que propõe níveis de detalhe dos fluxogramas onde cada nível é composto de seis ou sete blocos representados na mesma página, e cada bloco é então subdividido até chegar às declarações do programa. Este método de subdivisão envolve o agrupamento de declarações num sub-bloco que tenha um entrada e uma saída, e o posterior agrupamento destes sub-blocos no número de blocos da página (a combinação dos sub-blocos é evitada e só ocorre quando for necessária para manter o limite estabelecido para o número de blocos da página). A colocação dos blocos na página é bastante simples (devido ao pequeno número de blocos na página), tendo blocos no centro (maioria dos blocos alinhados) e blocos à direita e à esquerda, com linhas de conexão entre eles. A desvantagem deste método é a divisão em níveis de detalhe limitados a apenas seis ou sete blocos por página, que leva à combinação de vários sub-blocos num só.

AUTOFLOW [GOET65] apresenta quatro colunas por página, tendo cada uma delas dois caminhos para linhas de conexão (uma à direita e outra à esquerda). Quando não há caminho disponível, é inserido um conector para simbolizar a transferência de controle e quando

uma coluna é preenchida, os blocos passam a ser colocados na coluna seguinte até preencher a página. Apesar da saída gráfica do AUTOFLOW ser a melhor dentre os geradores automáticos de fluxograma descritos acima, esta não sugere um layout bidimensional, e sim colunas colocadas lado a lado numa página e conectadas entre si.

Hain [HAIN65] apresentou um sistema radicalmente diferente, que utiliza um representação em árvore, que não é adequada para saída gráfica para fluxogramas.

FLOG [WATK74] é o gerador automático de fluxogramas que obtém melhor desempenho na saída gráfica. Utiliza a notação de Krider [KRID64], que busca obter digrafos com uma entrada e uma saída (com requisitos semelhantes aos descritos por Haibt). Este processo extrai digrafos com uma entrada e uma saída substituindo-os no digrafo principal por um único bloco com a referência da sub-página onde são posteriormente representados (conceito semelhante à utilização de macros). O fluxograma é inicialmente representado sob a forma de notação de Krider e processado de forma a extrair digrafos e permitir a colocação de até 16 blocos por página. Muitas vezes, no entanto, o sistema falha em obter um sub-bloco com apenas uma entrada e uma saída, além de representar poucos símbolos por página (16). A colocação dos símbolos nas páginas é realizada através de uma matriz bidimensional de ocupação da página e linhas de conexão distintas podem se cruzar.

A distribuição de símbolos na página, realizada pelo SADG, tem condições de contorno bem distintas das que nortearam o projeto dos geradores automáticos de fluxogramas apresentados acima. Não representa implementação em alto nível de detalhe, mas uma linguagem de especificação onde os níveis de documentação são representados em diferentes níveis de documentos, podendo o

usuário (e não o gerador automático) se utilizar de chamadas de procedimentos ou macros para esconder detalhe num dado contexto. O número de símbolos admissível é superior ao dos geradores citados e cada símbolo carrega textos internos (com grande número de caracteres) e comentários fornecidos pelo usuário, e pode ser representado em dois tamanhos opcionais. Linhas de conexão devem ser implementadas sempre que possível e não podem se cruzar.

Desta forma, não podendo se utilizar do artifício de substituir automaticamente partes do fluxograma por blocos a serem representados em sub-páginas, a implementação do SADG concilia critérios próprios para uma distribuição de símbolos eficiente nas páginas, buscando ocupar o espaço disponível e diminuir a fragmentação através da diminuição do número de conectores automáticos.

As técnicas aplicadas por Watkins [WATK74], poderiam ser aproveitadas para sugerir um critério complementar aos critérios estabelecidos para a implementação do SADG. Esse critério levaria à representação em uma nova página de uma decisão, juntamente com seus ramos que não coubessem inteiramente na página atual (devido à sua coluna e cota iniciais), mas que ao ser transportada para uma nova página pudesse ser representada contiguamente a partir do início da página. Para que esse procedimento não violasse o critério de representação completa de uma SA, a decisão pendente aguardaria sua vez de ser tratada, e o seu tratamento poderia conduzir a uma grande perda de espaço na página na qual foi tratada a última pendência anterior a ela, devido à obrigatoriedade de sua representação começar no início de uma nova página. O adiamento de sua representação na página, da qual foi retirada pelo critério complementar, também pode levar à perda de espaço, sobretudo nos casos em que a cota inicial da decisão não coincide com a inicial do ramo que a contém e leva a uma

fragmentação deste ramo. O provável aumento de legibilidade da representação contígua de decisões e seus ramos não foi julgado compensador em confrontação com o critério de ocupação de espaço na página; de forma que nenhum critério inspirado nas técnicas de geração automática de fluxogramas foi acrescentado aos critérios originais de distribuição de símbolos no SADG.

5.4.2 - Roteamento de Circuito Integrado

A estrutura de representação da ocupação da página, utilizada no SADG, permite a implementação de todas as linhas de conexão possíveis na página sem que estas se cruzem entre si (ou atravessem os símbolos distribuídos a priori na página).

Há alguma semelhança entre este tipo de problema e aqueles resolvidos por algoritmos de roteamento para circuito integrado, onde em analogia à tentativa de substituir pares label-join por linhas de conexão após a distribuição dos símbolos nas páginas, busca-se interconectar células do circuito integrado após o posicionamento das mesmas. Existem três estratégias básicas de roteamento [LIES86] : roteamento de labirinto, propagação de retas e roteamento de canais. Destas três, as técnicas utilizadas para propagação de retas são as que mais se aproximam do problema de linhas de conexão resolvido no SADG. Propagadores de retas podem ser divididos em dois grupos : propagação de retas por busca e por expansão.

Um algoritmo de busca de reta é descrito em [HIGH69]; vide figura 5.33. Os segmentos ortogonais mais longos que não cruzem nenhum obstáculo são traçados pelos dois pontos a ser interligados. Se estes segmentos se interceptam, uma solução foi encontrada; caso contrário, são traçados os segmentos perpendiculares mais longos que interceptam os segmentos

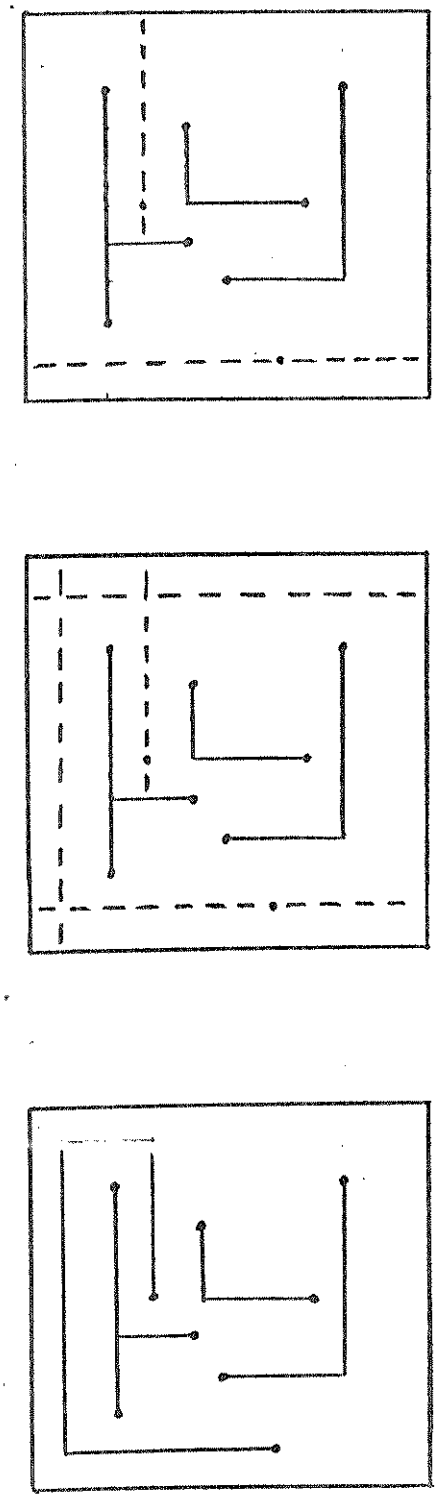


Figura 5.33 - Propagação de retas por busca

anteriormente traçados. Este processo é repetido até que sejam traçados dois segmentos que se cruzem ou até que nenhum outro segmento possa ser traçado.

Um algoritmo de expansão de retas é descrito em [HIGH80]; vide figura 5.34. Um segmento é inicialmente traçado através do ponto geométrico que representa a localização de um dos pontos a serem interligados. A seguir o contorno da região que pode ser coberta com todos os segmentos perpendiculares ao primeiro (não apenas um como no caso anterior) é determinado. Esta operação é seguida pela demarcação das regiões que podem ser cobertas por todos os segmentos de reta perpendiculares aos segmentos determinados no estágio anterior. Este processo é repetido até que o ponto destino seja englobado numa demarcação ou até que nenhum contorno possa ser encontrado.

As linhas de conexão são implementadas em vias de conexão fixas e, através do posicionamento relativo do label e do conector, já são estabelecidas restrições ao domínio de pesquisa. As condições de contorno são distintas e a solução adotada no SADG de representação da ocupação da página (que fornece subsídios para a tentativa de estabelecer linhas de conexão) é particularizada para a resolução do problema de linhas de conexão e difere das técnicas utilizadas pelos roteadores por propagação de retas.

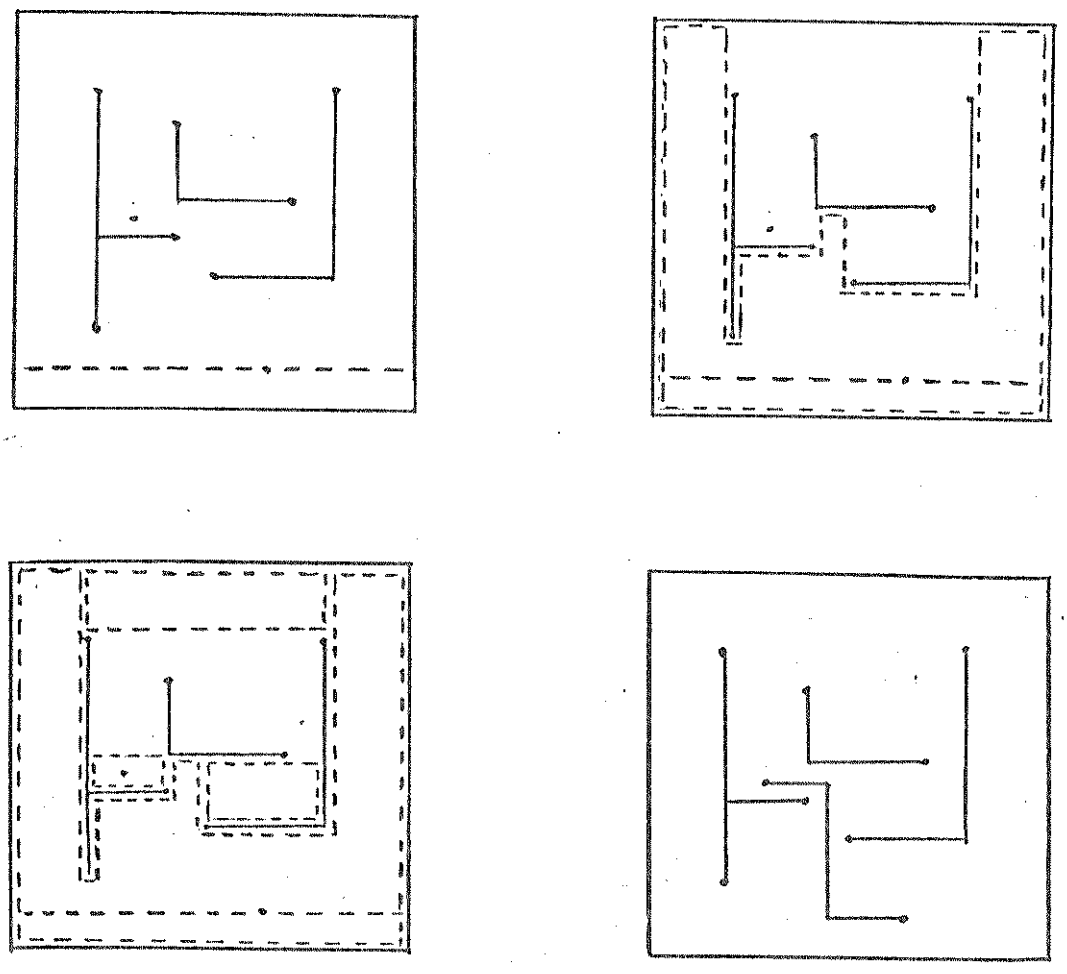


Figura 5.34 - Propagação de retas por expansão

6 - MÓDULO FORMATADOR

Este é o mais simples dos módulos do SADG e suas funções resumem-se ao processamento dos textos, ao pré-processamento das páginas para a geração de um arquivo para saída em impressora gráfica ou ao controle das rotinas gráficas para saída em terminal gráfico.

O processamento dos textos associados aos símbolos, aos comentários, aos nomes de ramo de decisão, etc, consiste na hifenação e centralização do texto no espaço que lhe é destinado.

A hifenação é necessária porque, no caso de uma palavra não caber em uma linha, a sua simples remoção para a próxima linha pode ser (em função do número de caracteres da palavra) bastante indesejável para o melhor aproveitamento do espaço disponível para um dado texto e a quebra da palavra em qualquer ponto (sem respeitar as regras de separação silábica) é prejudicial para a inteligibilidade do texto.

Os casos de separação silábica que envolvem distinção entre hiatos e tritongos não apresentam regras bem definidas e, portanto, o algorítmo aqui apresentado não separa vogais. O algorítmo parte do princípio que não existe sílaba sem vogal; a hifenação só será realizada na ocorrência de consoantes entre vogais. Pode-se detalhar o algorítmo de hifenação da seguinte forma :

- inicializa-se o ponteiro da vogal esquerda em função da posição da primeira vogal da palavra e o ponteiro da vogal direita em função da posição da próxima vogal (se houver);

- se o número de consoantes entre vogais for um : hifena segundo a regra V/CV (onde V corresponde a vogal, e C a consoante);

- se o número de consoantes for dois : hifena segundo a regra VC/CV se não for exceção ou segundo a regra V/CCV se for exceção (as exceções são consoantes duplas que não se separam, a saber : R precedido de B,C,D,F,G,P,T,V; L precedido de B,C,F,G,P,V; H precedido de L,N);

- se o número de consoantes for três : hifena segundo a regra VCC/CV (ex : constituir), ou VC/CCV se for exceção (ex : contratar);

- se o número de consoantes for quatro : hifena segundo a regra VCC/CCV (ex : reconstrução, obstruir);

- a seguir : ponteiro da vogal esquerda = ponteiro da vogal direita e ponteiro da vogal direita passa apontar a próxima vogal, se houver (se o número de consoantes entre vogais for zero, só este passo será executado, ocasionando a não hifenação, como se deseja).

Para simplificar o procedimento de centralização, considerou-se sempre como retangular o espaço disponível para a colocação de um determinado texto. No caso de símbolos que não têm formato retangular, como uma decisão por exemplo, considera-se o máximo retângulo interno ao símbolo. Calcula-se então o número máximo de linhas que podem ser alocadas ao espaço retangular no qual o texto deverá ser colocado. O texto é então distribuido e hifenado nas linhas, tendo como limite o número máximo de linhas permitidas em questão. Se o número de linhas utilizadas é menor que o máximo, tais linhas são igualmente espaçadas em função da largura do retângulo. Se o texto ocupa apenas uma linha, esta ficará no centro do retângulo e o texto no centro da linha.

O pacote gráfico PLXY11 da printer plotter LXY11 apresenta alguns inconvenientes :

- só representa caracteres maiúsculos no modo gráfico;

- processamento lento devido à sua generalidade (o eixo vertical de uma página lógica pode comportar até 41 páginas físicas);

- resolução dos caracteres no modo gráfico apresenta padrão bem inferior ao modo texto pois, devido à possibilidade de imprimir uma dada cadeia de caracteres em qualquer ângulo, os pontos que compõem um dado caracter são transportados um a um, e o arredondamento das coordenadas dos pontos para coincidir com as posições discretas das agulhas da impressora (impressora de pontos) leva a distorções nos símbolos, fazendo variar seu formato e legibilidade em função da posição na página.

A printer plotter LXY11 é comandada por caracteres especiais do arquivo texto que lhe indicam :

- mudança de página;

- mudança de linha;

- linha gráfica : os caracteres da linha devem ser considerados como portadores de informação dos pontos a serem impressos, de tal forma que parte dos bits da representação ASCII de cada caracter terá associação com os estados das agulhas (levantadas ou abaixadas), cuja sequência está associada às agulhas numa determinada posição na linha.

Objetivando através da particularização do pacote gráfico PLXY11 ganhar em legibilidade dos caracteres e tempo de processamento, resolveu-se implementar um pacote gráfico próprio. As rotinas deste pacote gráfico permitem :

- mudar de página;

- posicionar um ponto de referência na página;

- traçar uma reta do ponto de referência a um outro ponto;

- imprimir uma cadeia de caracteres (cujos moldes são fixos para a representação no formato A3 e A4, de tal forma que não ocorra rotação da cadeia de caracteres).

O Formatador percorre os símbolos das SAs da página, associando cada elemento do desenho (símbolo, comentário, texto interno aos símbolos, texto associado a nomes de ramo de decisão ou referência de labels e conectores, traços horizontais, etc) a uma rotina gráfica específica, como por exemplo : tarefa, decisão, comentário, conexão, conector automático, etc. Estas rotinas utilizam-se das rotinas do pacote gráfico, de forma que, a cada símbolo, será associada uma sequência de chamadas de rotinas gráficas.

As rotinas gráficas (com exceção daquela que provoca mudança de página) preenchem uma matriz, onde cada elemento está associado às coordenadas relativas (a um ponto de origem) de um ponto a ser impresso na página. A rotina de mudança de página atua da seguinte forma :

- Chamada Inicial : Zera a matriz que representará os pontos na página e envia para o arquivo de impressão o caracter de controle que provoca alteração de página bem como os caracteres que provocam saltos nas linhas até o posicionamento desejado como origem da página lógica (representação do desenho em formato A3 na página física da printer plotter).

- Chamadas Seguintes : Converte cada linha da matriz em um trem de caracteres com informação de posicionamento de agulhas, finalizando com os caracteres indicativos de linha em modo gráfico e posicionamento da nova linha, e os transfere para o arquivo de impressão. Após converter e transferir todas as linhas da matriz, envia os caracteres de mudança de página e posicionamento da linha desejada (deixando em branco as linhas anteriores) e zera a matriz de pontos.

É importante observar que o Formatador se enquadra em dois níveis distintos :

- o primeiro nível está relacionado com o sequenciamento das rotinas associadas aos elementos do desenho a ser impresso, bem como com as rotinas de hifenação e centralização do texto, que independem do pacote gráfico e do terminal utilizado;

- o segundo nível é o de geração do arquivo de impressão que é função estrita do terminal e do pacote gráfico utilizado.

O interfaceamento entre esses dois níveis é feito pelas rotinas :

- IniciaArquivoPrintavel;
- ProximaPagina;
- PosionaPontoReferencia;
- TracaReta;
- PrintaString;
- FinalizaArquivoPrintavel;
- TracaSegmentoCircular.

Estas rotinas intermediárias são chamadas pelas rotinas do nível 1 (associadas aos elementos do desenho) do Formatador e são constituídas de chamadas das rotinas do nível 2 (pertencentes ao pacote gráfico). A rotina TraçaSegmentoCircular é implementada a partir das rotinas do pacote gráfico que causam posicionamento do ponto de referência e traçado de retas a um dado ponto. As rotinas intermediárias têm como objetivo absorver as alterações pertinentes ao pacote gráfico e/ou ao terminal de saída, sem que seja necessário alterar o corpo principal do Formatador (constituído pelas rotinas de nível 1).

Assim sendo, o Formatador realiza suas funções de hifenação e centralização de textos com procedimentos simples e eficientes e mantém independência entre o SDL-GR e o terminal de saída escolhido. O pacote gráfico descrito acima perde em generalidade mas ganha em eficiência traduzida pela melhoria de legibilidade e diminuição do tempo de processamento. Sua parametrização para

admitir os formatos A3 e A4 torna-o genérico para a geração automatizada de SDL-GR a partir de SDL-PR.

A visualização das páginas do documento em terminal gráfico é bastante conveniente ao usuário, principalmente porque o hardware de tais terminais trabalha com notação vetorial, dispensando a formatação matricial dos pontos da página, constituindo-se portanto num rápido meio de consulta dos documentos SDL-GR. O terminal gráfico utilizado foi o VT125, que dispõe de notação vetorial para posicionar referência, traçar linha e segmento de círculo. O único problema que ocorre é que os formatos necessitam ser diminuídos proporcionalmente para caber no terminal (o ideal é usar um terminal com área útil de tela maior), o que conflita com os caracteres que têm tamanhos padrões. Assim sendo, apenas algumas linhas do texto de cada símbolo são mostradas, mas o usuário pode solicitar o texto completo através da referência da coluna e do símbolo desejado (ex : C3S5, que indica símbolo 5 da coluna 3).

7 - PERSPECTIVAS DE EVOLUÇÃO DO SADG

A automação do ciclo de vida de software dos projetos das administrações telefônicas que utilizam SDL e CHILL, mostra uma tendência acentuada da implementação das seguintes funções :

- tradução PR/GR e vice-versa;
- editor gráfico;
- gerador do fonte da linguagem de implementação através de SDL;
- simulador.

No último quadriênio, SDL sofreu incrementos substanciais que encurtaram a ponte que o separa de CHILL. Por outro lado, a entrada em forma gráfica tem se tornado bastante atraente devido à evolução dos terminais gráficos e a tendência de utilização de workstation.

Neste capítulo, são analisadas as conveniências de uma evolução do SADG para uma ferramenta mais geral que possa automatizar o ciclo de vida de software, tendo SDL como linguagem de especificação e documentação e CHILL como linguagem de implementação. Tal ferramenta, que acrescentaria alguns módulos à estrutura do SADG e algumas funções aos módulos atuais, poderia ser denominada SAS (Sistema de Automação de Software). A oportunidade tecnológica de implementação é bastante concreta, segundo as necessidades do projeto TRÓPICO-RA (CPA Temporal de Grande Porte) no CPqD.

A seguir são analisadas as conveniências e a viabilidade da evolução do SADG para esta ferramenta mais abrangente.

7.1 - Geração Automática do Fonte CHILL através de SDL

7.1.1 - Considerações sobre a correspondência entre SDL e CHILL

CHILL é uma linguagem de especificação que sofreu influência do ALGOL-60 (na consistência de tipos e definição do escopo de visibilidade de dados) e de SIMULA (conceito de ativação e desativação de instância de processos). Em CHILL, são implementados mecanismos de concorrência, sincronização (através de sinais, buffers e eventos) e regiões críticas. Existe ainda a possibilidade de discriminar a nível de comando, procedimento ou módulo (região que encapsula a visibilidade dos dados internos a ela), ações que compõem o que se chamou de exception handler, ou seja, ações que deverão ser executadas se algumas das exceções ocorrerem. É possível exportar ou importar dados entre módulos distintos de forma a explicitar a visibilidade dos dados.

No início desta década ocorreram alguns esforços de algumas organizações para implementar uma ferramenta para gerar CHILL a partir de SDL [GUTF82] [GAIS83] [CAIN81]. Estas iniciativas encontraram muitas lacunas em SDL que motivaram o envio de contribuições ao SDL anterior, que acabaram sendo incorporadas ao SDL atual, tais como : concorrência, sincronização, definição de dados.

Houve um estreitamento muito grande entre ambas as linguagens, simplificando consideravelmente a tarefa de converter a linguagem de especificação SDL (da qual se pode extrair automaticamente a forma gráfica) para a linguagem de implementação CHILL. No entanto, restam ainda alguns casos específicos para os quais a correspondência precisa ser definida por ocasião da implementação de uma ferramenta para automatizar a conversão SDL para CHILL.

O problema de estruturação (uma vez que as transições SDL não são estruturadas) fica resolvido com a extensão feita ao SDL-PR no SADG incorporando a estrutura loop que permite a representação de

qualquer loop condicional sem especificar labels e GOTOs; o mesmo ocorre com a representação de IFs e CASEs através de decisão cujos ramos abertos (não terminados com GOTO ou nextstate) são acrescidos automaticamente de um símbolo terminador GOTO para um ponto comum caracterizado como final da decisão.

Uma característica importante associada à geração automática da implementação é a existência de dois estágios de detecção de falhas. Na maioria dos compiladores somente a primeira mensagem de erro pode ser garantida como válida, uma vez que o sincronismo é perdido quando erros estruturais são detectados. A automação da geração do programa fonte garante que o programa a ser submetido ao compilador CHILL é estruturalmente correto de tal forma que, em princípio, todo erro detectado na compilação será real e não espúrio.

7.1.2 - Sistematização e Automação do Ciclo de Vida de Software

Na concepção da arquitetura de software dos projetos de desenvolvimento, a implementação não antecede o particionamento do sistema em módulos e a definição da interface entre os mesmos. Um conceito análogo pode ser estendido à definição da estrutura de programa e dos procedimentos que o compõem. Ou seja, para obter a definição da ambientação completa do procedimento em relação à estrutura externa na qual está inserida (conservando a analogia : à sua interface com o corpo do programa, ou com o procedimento que a contém), basta definir os seus parâmetros e as suas variáveis globais. A partir desta amarração, a implementação tem o seu escopo totalmente definido.

Na busca de estabelecer uma interface bem definida entre a especificação e a implementação e ao mesmo tempo assegurar através de mecanismos automatizados a total coerência entre ambas, é

conveniente utilizar o conceito suportado por CHILL para módulo de definição e implementação.

Na fase de especificação, será feita a decomposição de um dado BI (bloco de implementação software) buscando caracterizar a sua estrutura e a sua interface com os procedimentos que o compõem, sem se importar com o corpo das mesmas. A definição de tais procedimentos integra um módulo de definição, separado do módulo associado a um dado BI. O módulo do BI faz referência somente ao modúlo de definição dos procedimentos, importa o nome dos procedimentos (contidas no módulo de definição) e exporta os dados que correspondem às variáveis globais a serem utilizadas pelos procedimentos. No módulo de definição dos procedimentos, são declarados os parâmetros e a importação dos dados globais (definidos no módulo principal) para cada um dos procedimentos, omitindo-se a implementação do corpo dos mesmos.

Quando a fase de especificação atinge este estágio, a fase de implementação restringe-se à implementação de procedimentos disjuntas cujo escopo está amarrado pela especificação. Na realidade o congelamento das especificações poderá ser levado a cabo em várias etapas, onde cada uma delas caracteriza um dado nível de congelamento como :

- a definição das procedimentos que compõem um dado módulo;
- a definição dos parâmetros dos mesmos;
- a definição das variáveis globais dos mesmos;
- a definição dos procedimentos em procedimentos mais internas, retomando os passos acima.

Com a automação da geração do programa fonte CHILL através da especificação em SDL, pode-se atingir características altamente eficientes para geração e manutenção do software gerado e para a gerência dos recursos alocados. A sistematização da gerência dos módulos de definição e de implementação poderá ser conseguida

através da implementação de uma estrutura de biblioteca de tais módulos que permita :

- manter independência entre o módulo que contêm a especificação do BI, o módulo que contém a definição dos procedimentos que foram especificados a partir da decomposição do BI e os módulos que contém a implementação do corpo dos procedimento;

- consistência automática entre os módulos, de forma a assegurar que a implementação corresponde à especificação, à medida que a implementação for sendo realizada;

- vedar alteração dos módulos de definição aos implementadores, de forma que as realimentações que ocorrerem durante a implementação conduzam a uma alteração formal do módulo de definição acessível somente às pessoas autorizadas para tal;

- associação de correspondência de versões entre módulos de definição e de implementação, de forma que a alteração de um dado módulo de definição desqualifique todas as implementações que possam fazer uso das especificações alteradas;

- alta versatilidade na alocação de recursos para a implementação dos procedimentos, uma vez que o processo de decomposição dos procedimentos e definição do escopo associado permite a obtenção de módulos pequenos, bem definidos e independentes;

- teste de alguns módulos (já implementados) ambientados pela simulação daqueles com os quais mantém interface (ainda não implementados), uma vez que esta interface já é conhecida;

concluindo-se que a estrutura permite a implementação e integração de módulos independentemente.

É importante salientar que o SDL atual (Red Book) permite a especificação detalhada da implementação, uma vez que abrange os seguintes conceitos :

- procedimentos em todos os níveis (sistema, bloco, processo, procedimento);

- macro em todos os níveis (será usada apenas evitar repetição de trechos na especificação, serdo incluída no corpo do programa durante a geração do fonte CHILL se o compilador não suportar macros);

- definição de dados abstratos e com tipos pré-definidos;

- concorrência (start e stop de instância de processos);

- substituição de texto livre por expressões, comandos de atribuição e comandos condicionais.

A ferramenta que automatiza a geração do fonte CHILL deve suportar todas estas características, deixando para cada projeto a definição do nível de detalhe a que deve chegar a especificação bem como a partição de tal fase em níveis distintos de detalhamento.

Um aspecto importante a ser salientado é que a consistência de tipos de dados (referente à análise semântica dos dados) é uma tarefa bastante árdua e não convém duplicar as funções do compilador CHILL neste sentido. Notação dupla para representação de dados seria também bastante inconveniente para o usuário, de forma que usar em SDL a mesma notação para definição de dados utilizada pelo CHILL constitui-se num fator de clareza e uniformidade para os usuários de ambas as linguagens. Desta forma, a automação da geração de CHILL através de SDL não irá além da análise sintática da estrutura de dados, deixando a detecção de erros semânticos para o compilador CHILL.

7.2 - Análise da Forma Gráfica como entrada

O SADG, que converte a forma textual de SDL para a forma gráfica, é bastante útil especificamente na fase de projeto na

qual os blocos de implementação de software estão projetados e requer-se a entrada de uma quantidade maciça de informação no sistema. Como pode-se constatar, o editor gráfico é em geral implementado nas ferramentas que dão suporte a SDL para permitir a entrada no sistema diretamente na forma gráfica. Isto se deve ao fato de que a conveniência da utilização da forma gráfica está associada à possibilidade de utilizá-la no início do projeto, quando a especificação está sendo concebida.

Atualmente dois fatores impulsionam a utilização da forma gráfica como forma de entrada : o barateamento dos terminais gráficos e a popularização do uso de estações de trabalho, uma vez que o seu maior inconveniente seria o de vincular um processamento interativo ao mainframe quando um número considerável de pessoas estivessem utilizando um editor gráfico. A possibilidade de terminais gráficos versáteis e eficientes funcionando como estações de trabalho associados a um computador principal viabiliza a utilização da forma gráfica de forma generalizada.

7.2.1 - Requisitos de Qualidade do Interpretador Gráfico

Algumas implementações de editor gráfico trazem consigo uma série de características que frustram as necessidades e as expectativas dos usuários, tais como :

- eficiência do diálogo com o usuário : a localização das coordenadas de pontos ou símbolos na tela pode ser lenta e ineficiente;

- alto grau de interação : funções que deveriam ser realizadas automaticamente (como a atualização da referência de páginas de conectores após remoção ou inserção de página) são deixadas a cargo do usuário podendo acarretar uma quantidade de trabalho extra considerável;

- ausência de funções de um interpretador : a remoção ou inserção de símbolos de decisão, labels e conectores, sem levar em conta a estrutura (decisão ou loop condicionais) da qual fazem parte, pode levar a inconsistências e a uma desorganização na estrutura do programa, inadmissível para o caso de geração automática de CHILL através de SDL;

- sobreposição espacial de símbolos : as funções que permitem mover e recuperar (após ter salvo em arquivo) partes do diagrama em uma dada página ou realizar conexões devem ser suficientemente inteligentes para impedir sobreposição de elementos do desenho na tela;

- facilidades para inserção ou remoção : movimentação automática de partes do diagrama para abrir espaço para inserção, ou compactação após remoção.

Um dos problemas mais críticos associados com a entrada gráfica é provavelmente o fato de que, quando a forma gráfica é convertida para forma textual e posteriormente reconvertida para a forma gráfica, a distribuição dos símbolos se altera radicalmente, o que pode levar à perda de legibilidade e clareza originais em função da conversão automática da forma textual para a forma gráfica. Esse problema pode ser evitado se forem observadas as seguintes condições :

- a conversão automática da forma textual para a gráfica deve ter boa performance em relação à distribuição espacial dos símbolos nas páginas, à substituição de conectores e labels por linhas de conexão, e à ordenação e referência de página de origem e destino para labels e conectores;

- a entrada em forma gráfica deve permitir ao usuário determinar opções correspondentes a diretivas na forma textual forçando mudança de página a partir de um dado símbolo ou em

situações bem específicas (como para ramo de decisão a partir de uma dada coluna);

- possibilidade de reformatar as páginas de uma dada estrutura (processo, procedimento ou macro) com a mesma lógica utilizada pela conversão automática da forma textual para a forma gráfica, como forma de obter a mesma saída que seria obtida passando pela cadeia de conversão GR-PR-GR.

Finalmente, seria desejável a possibilidade de visualizar o maior contexto possível de uma dada estrutura, o qual seria obtido através da diminuição dos símbolos e da manipulação de toda a árvore que compõe uma dada estrutura sem a quebra em sub-árvores ocasionada pela paginação. Uma vez que a diminuição dos símbolos não será em geral suficiente para visualizar toda a estrutura no vídeo, deveria ser possível selecionar um determinado símbolo a partir do qual ocorra a expansão dos símbolos (a ele filiados) no vídeo. Se o terminal gráfico utilizado permitir o armazenamento em sua memória interna de mais informação do que a mostrada no vídeo, poderá também dispor da função de scrolling, que permite que a janela fornecida pelo vídeo percorra nos quatro sentidos a informação mais abrangente contida na memória interna. O chaveamento de contexto, ou seja, selecionar um símbolo e transitar da página lógica (mais abrangente) para a página efetiva (de dimensões reais), pode ser bastante interessante para o usuário.

A solução proposta para todos estes problemas e para a viabilização das possibilidades listadas está fundamentada nos seguintes pontos :

- a eficiência do diálogo com o operador pode ser bastante aumentada através da possibilidade de uso de "mouse" para selecionar um ponto ou um símbolo na tela e/ou de mnemônicos para

selecionar coluna, símbolo, etc (ex : C5S3 seleciona o símbolo 3 da coluna 5);

- funções de interpretador gráfico para evitar inconsistência ou desestruturação (de loops ou decisões com múltiplos ramos) durante a inserção ou remoção;

- reformatação de uma dada estrutura (processo, procedimento ou macro) utilizando o próprio módulo paginador (responsável pela distribuição dos símbolos nas páginas);

- marcação de pontos de quebra condicionais no desenho de uma dada página, que ocasionam mudança de página em determinadas circunstâncias e cuja informação corresponde ao uso de diretivas na forma textual;

- possibilidade de visualizar páginas mais abrangentes, selecionar símbolo para "zoom" ou para o chaveamento para a página de dimensões reais;

- funções usuais de remoção, inserção, deslocamento, salvamento ou recuperação de parte do desenho, leitura ou alteração de texto;

- funções de auxílio à inserção e remoção para evitar ao usuário a necessidade de detalhar deslocamento para criar espaço (para inserção) ou compactar espaço liberado (após remoção).

7.3 - Módulos Componentes do SAS

O SAS (Sistema de Automação de Software) é uma ferramenta de CAD, cuja proposta inicial, é automatizar parte do ciclo de vida de software no que tange à geração da documentação em SDL (em ambas as formas PR e GR) e conversão de SDL para CHILL.

O SAS integra as seguintes funções :

- Interpretador Gráfico;

- Tradução de SDL-GR para SDL-PR;

- Tradução de SDL para CHILL;

- funções disponíveis no SADG (que será incorporado como subconjunto do SAS).

Os módulos de definição e implementação de procedimentos, ficarão residentes numa biblioteca, cujo gerenciamento garantirá consistência (levando em conta a correspondência de datas de criação entre versões de ambos os tipos de módulos) entre tais módulos.

Na figura 7.1 é mostrado o diagrama de blocos dos módulos componentes do SAS; a descrição sucinta das funções é dada a seguir :

- Compilador : recebe como entrada o arquivo na forma SDL-PR e o traduz para a forma GR global (forma intermediária que contém a representação hierárquica de todo o sistema descrito em SDL), bem como uma lista indentada do fonte PR com os erros detectados na compilação. É o mesmo módulo componente do SADG com o acréscimo da análise sintática da estrutura de dados (uma vez que análise semântica dos dados será realizada pelo compilador CHILL) e da possibilidade de aceitar diretivas do usuário.

- Gerador : geração do programa fonte CHILL através da forma GR global, resolvendo as referências dos procedimentos residentes na Biblioteca de Módulos.

- Paginador : recebe como entrada a forma GR global e realiza a distribuição de símbolos nas páginas gerando como saída a forma GR paginada. É o mesmo módulo componente do SADG alterado para dispor da possibilidade de reformatar apenas uma estrutura (processo, procedimento ou macro) a partir da forma GR paginada (atualizando as referências a páginas inseridas ou removidas) e da possibilidade de adaptar a paginação em função de diretivas definidas pelo usuário.

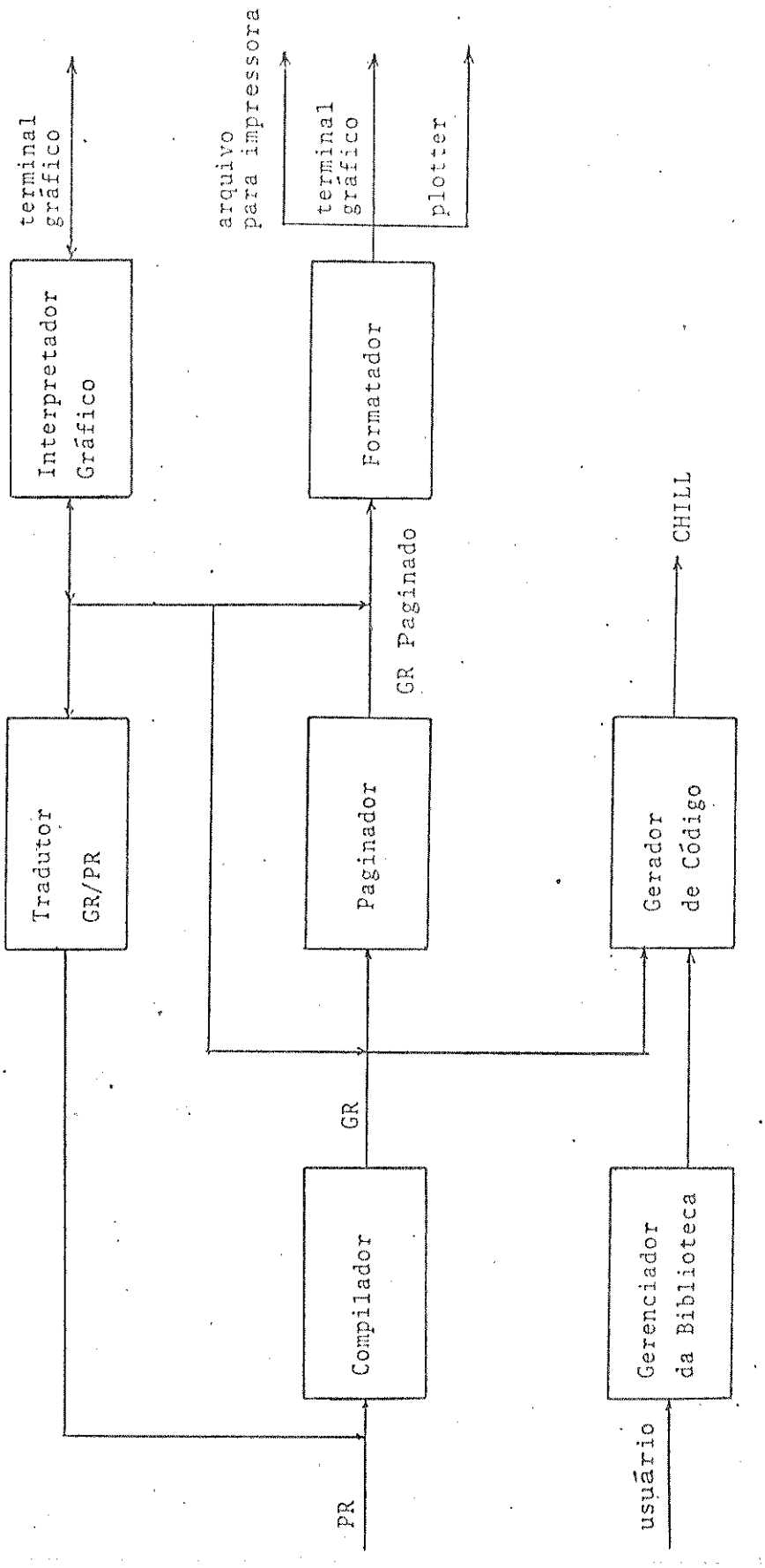


Figura 7.1 - Módulos do SAS

- Formatador : converte a forma GR paginada em arquivo de saída para impressora gráfica ou possibilita saída por terminal gráfico. É o mesmo módulo componente do SADG, podendo ser acrescido de saída para plotter.

- Tradutor GR/PR : realiza a conversão da forma GR paginada para a forma PR.

- Interpretador Gráfico : permite a criação, alteração ou reformatação (através de interface com o módulo Paginador) da forma GR paginada. Prevê utilização do VT125 como terminal para desenvolvimento, podendo depois ser convertido para operar no terminal gráfico a ser usado em microcomputador (utilizando "mouse").

- Gerenciador da Biblioteca de Módulos : permite criação e remoção e realiza a consistência de versões dos módulos de definição e implementação de procedimentos da biblioteca.

8 - CONSIDERAÇÕES FINAIS

O SADG foi apresentado na seção de ferramentas do II SDL Forum em Helsinki (Filândia) em março de 1985 [BATI85], e figura em publicação do CCITT de agôsto de 1985 [CCIT85], com a sigla em inglês GDAS (Graphic Documentation Automated System), como uma das ferramentas implementadas pela comunidade internacional para dar suporte a SDL.

Infelizmente os artigos disponíveis sobre as demais ferramentas que executam funções semelhantes ao SADG são, em geral, vagos acerca da implementação em relação à tradução SDL PR/GR e a fonte de comparação se restringe à ferramenta EGS, cujas versões 3 e 4 do código executável são disponíveis no CPqD, onde vinha sendo utilizado para a geração da documentação do projeto TRÓPICO, até ser substituído pelo SADG.

As técnicas utilizadas para a geração automática de fluxogramas e para roteamento de circuito integrado têm requisitos distintos em relação à geração de fluxogramas a partir de uma linguagem de especificação, como foi analisado no item 5.4.

A motivação inicial mais significativa para o desenvolvimento do projeto SADG, foi o grande espalhamento na distribuição de símbolos nas páginas de documentos gerados através do EGS. Como teste comparativo foi escolhido o bloco de implementação software AAS (Supervisor de Assinantes), pertencente ao software de Operação, Manutenção e Supervisão da Central de Comutação Telefônica TRÓPICO-R (com 3675 símbolos gráficos e 156 páginas em SDL-GR no formato A3 horizontal).

As figuras 8.1.a a 8.1.d mostram algumas páginas do AAS, geradas através da versão 3 do EGS. A legibilidade é boa, mas a distribuição de símbolos é extremamente dispersiva, totalizando 406 páginas para todo o AAS (no formato A3 horizontal, que foi o

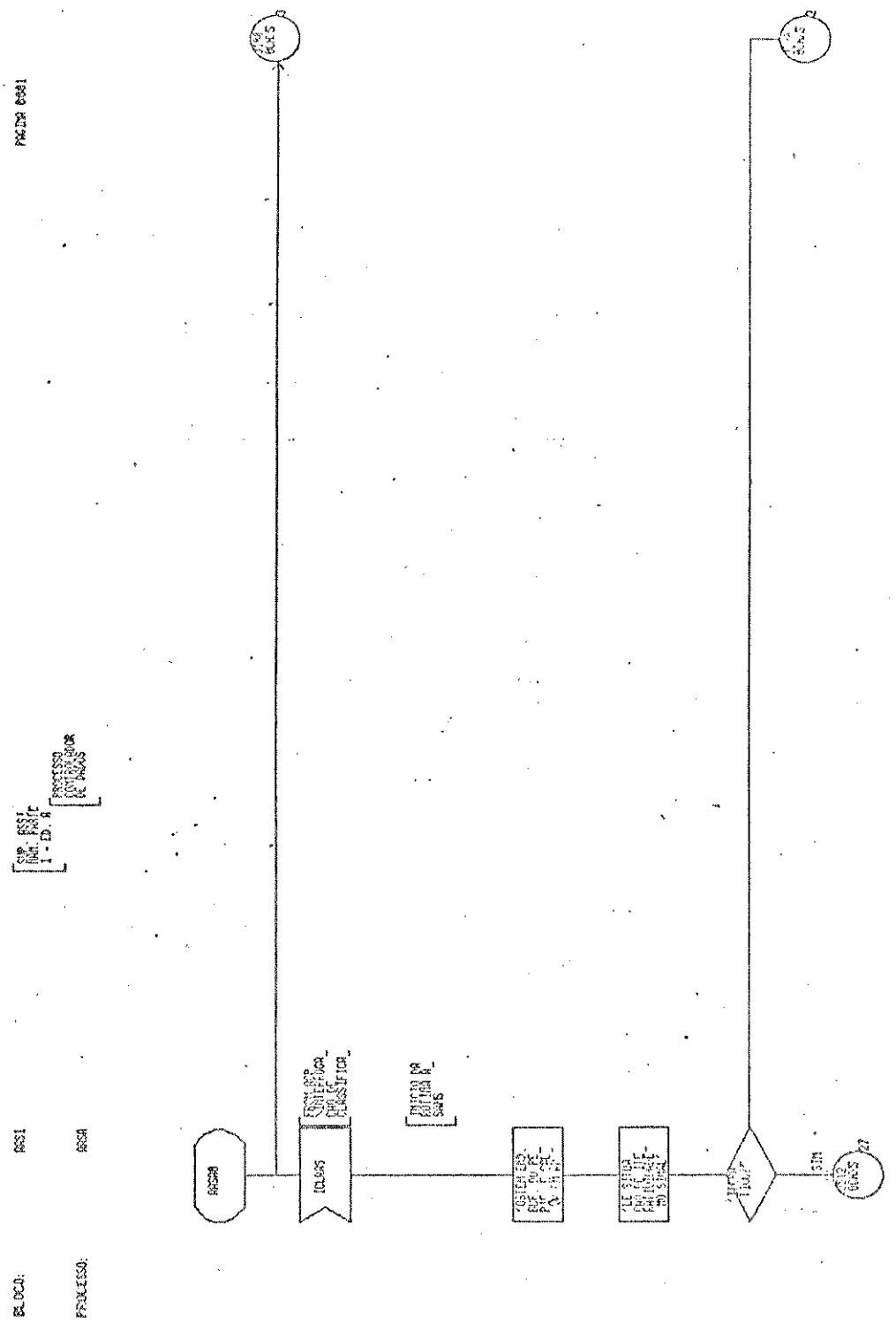


Figura 8.1.a – Página de saída do EGS v.3

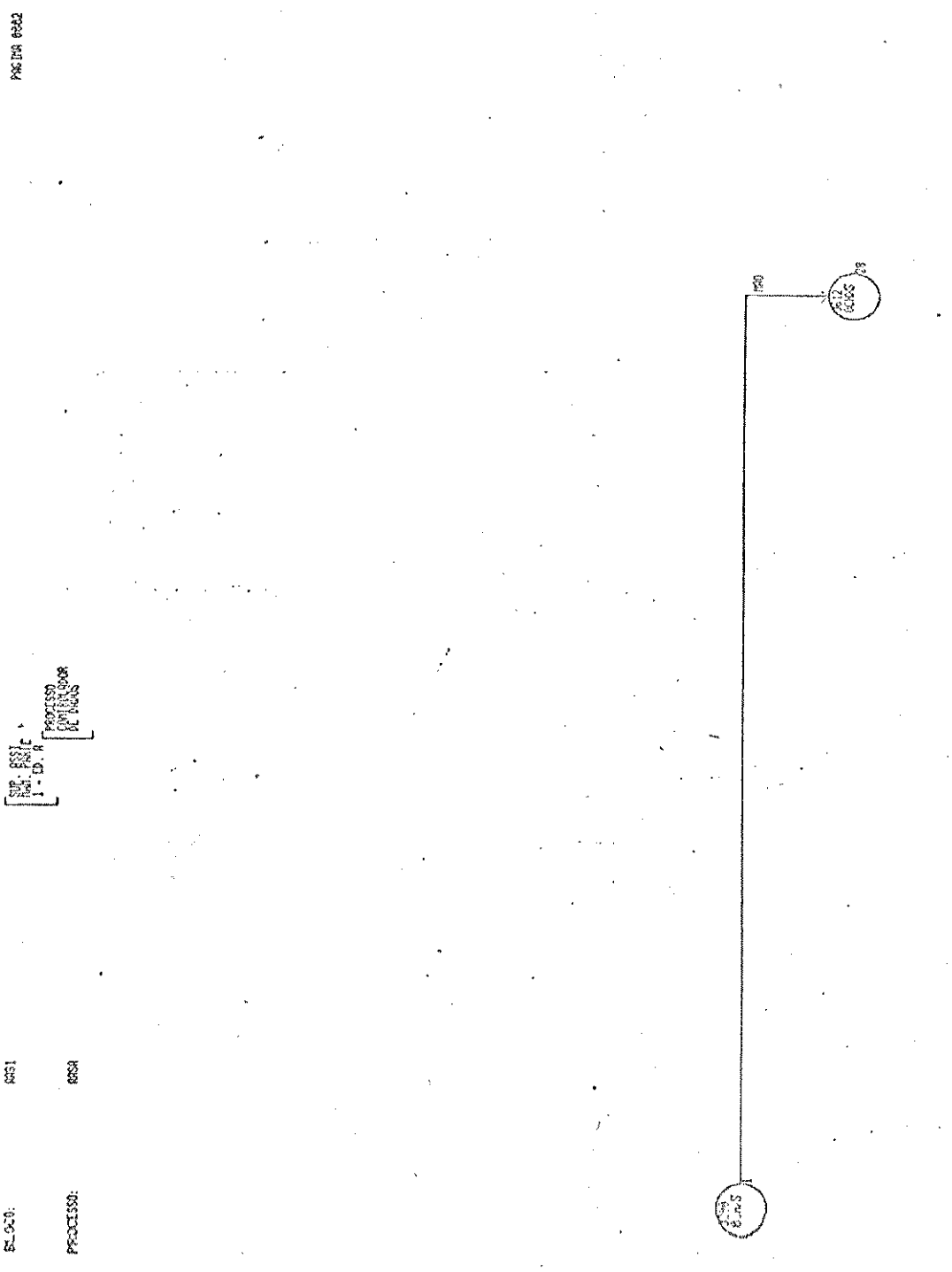


Figura 8.1.b - Página de saída do EGS v.3

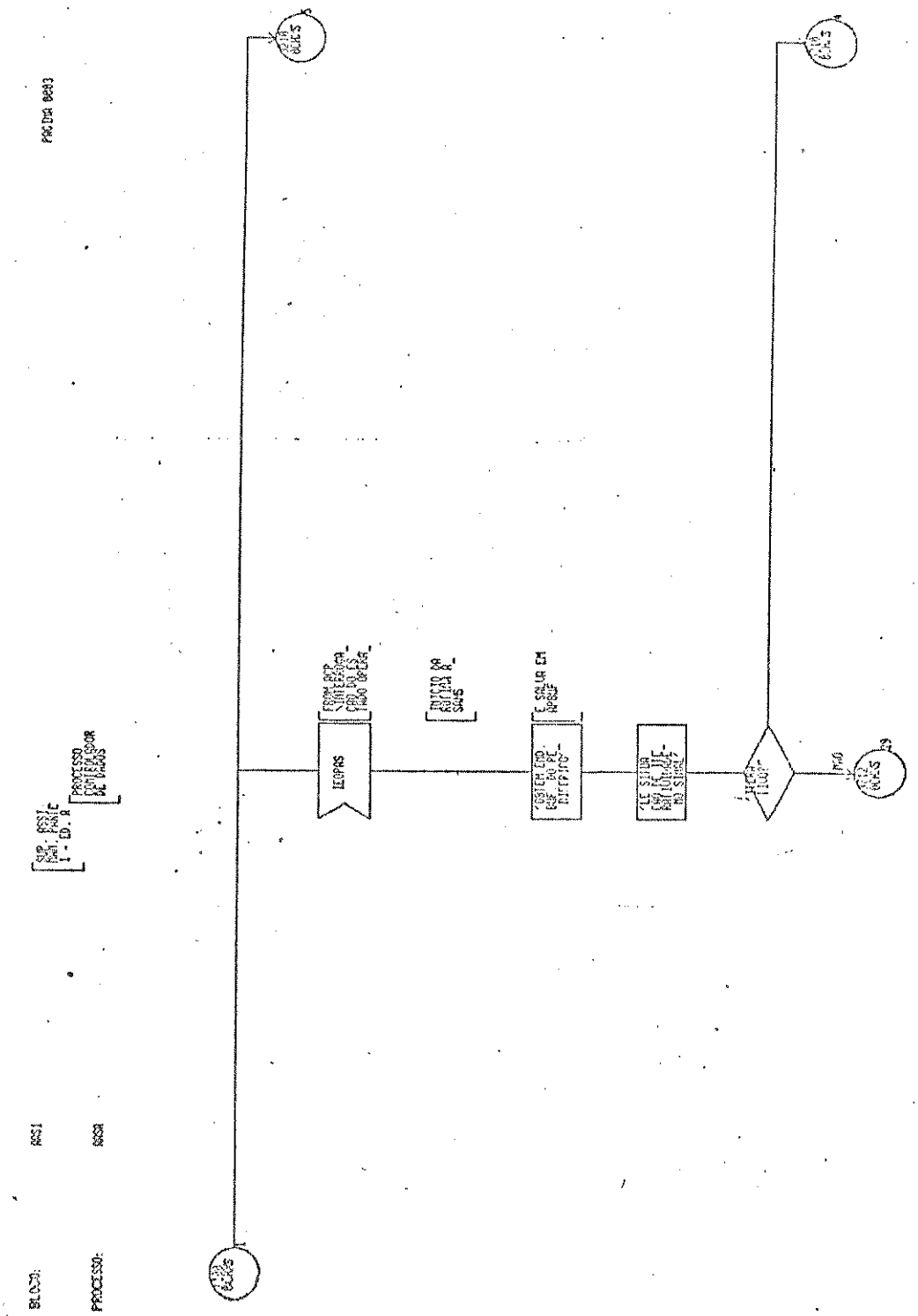


Figura 8.1.c – Página de saída do EGS v.3

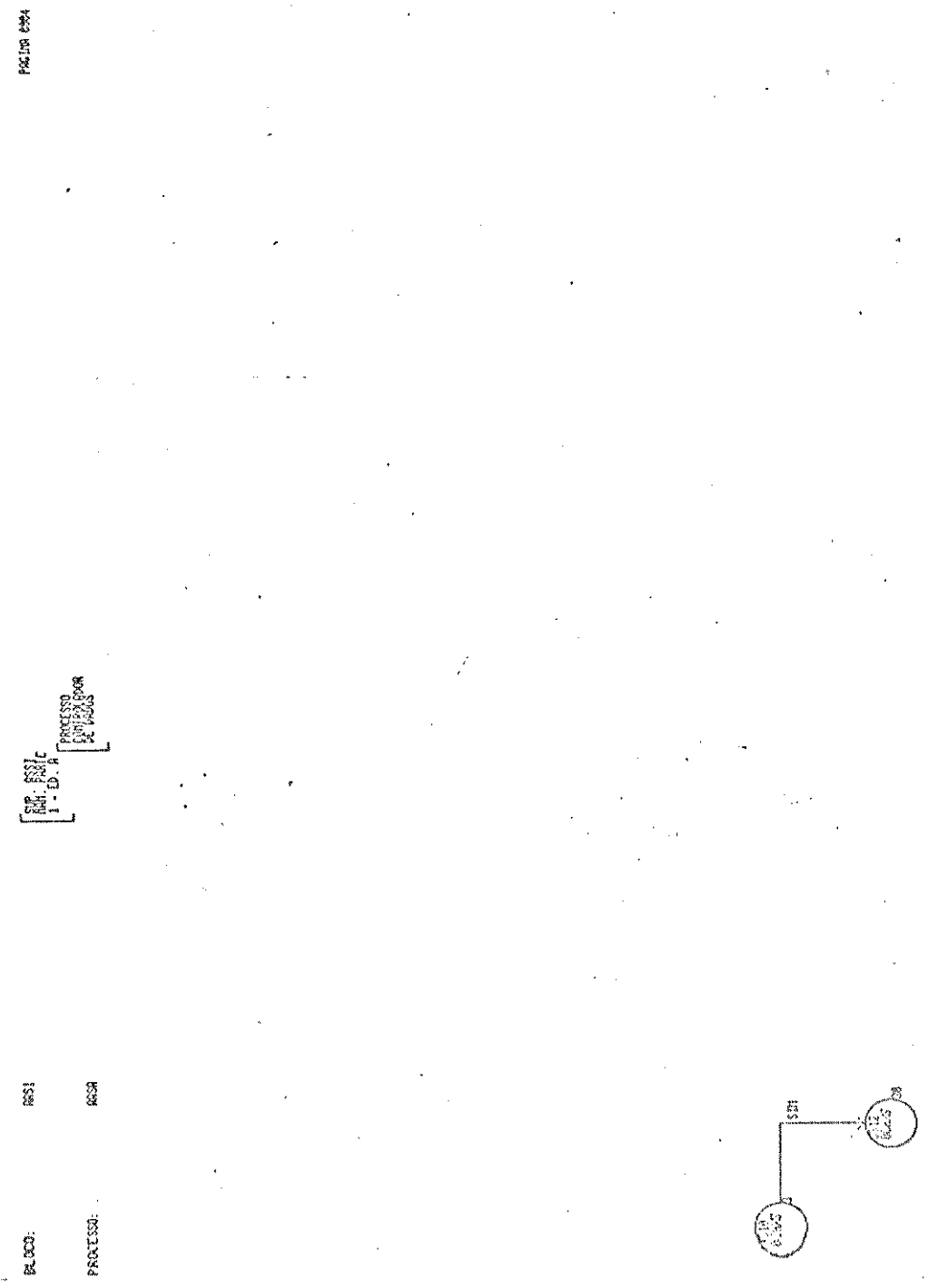


Figura 8.1.d - Página de saída do EGS v.3

usado como referência). Além do mais, o AAS teve que ser dividido em três partes para ser submetido ao EGS.

Na versão 4 do EGS, foram implementados os novos símbolos referentes ao SDL do último quadriênio (cuja implementação o SADG também dispõe). Supostamente o software correspondente à função de tradução foi reescrito para otimizar a distribuição de símbolos [GASS85]. O número de páginas (para o AAS) caiu de 407 para 156, mas não devido a utilização de melhores algoritmos para a distribuição de símbolos, como se pode constatar das figuras 8.2.a a 8.2.f, nas quais são mostradas algumas páginas do AAS geradas pela versão 4 do EGS. Na realidade, foram diminuídos os espaços entre os símbolos na vertical e entre as colunas, com o objetivo de forçar a probabilidade de maior densidade de símbolos nas páginas; isto porque o EGS trabalha com o esquema do método de fracionamento simples (vide ítem 5.2.3.1) [MACA82], onde inicialmente um dado processo tem seus símbolos distribuídos numa página infinita que posteriormente é dividida em páginas de dimensões reais. A restrição da necessidade de dividir o AAS em três partes permaneceu, e apenas os símbolos médios são utilizados, o que na prática é muito mal aceito pelo usuário, pois o espaço para o texto interno aos símbolos é muito reduzido de forma geral. O desperdício de grandes espaços nas páginas e a má distribuição de símbolos, adicionam-se à grande perda de legibilidade , como se pode constatar, pois das 156 páginas :

- 3 só têm conectores;
- 32 utilizam menos da metade da página;
- 49 utilizam menos de um quarto da página.

O SADG gera forma gráfica do AAS com 156 páginas, e nas figuras 8.3.a a 8.3.d pode-se ver casos de : página normal, efeito de borda e escolha entre as sub-árvores de símbolos nas páginas. A legibilidade é bastante alta, e o espaço reservado aos

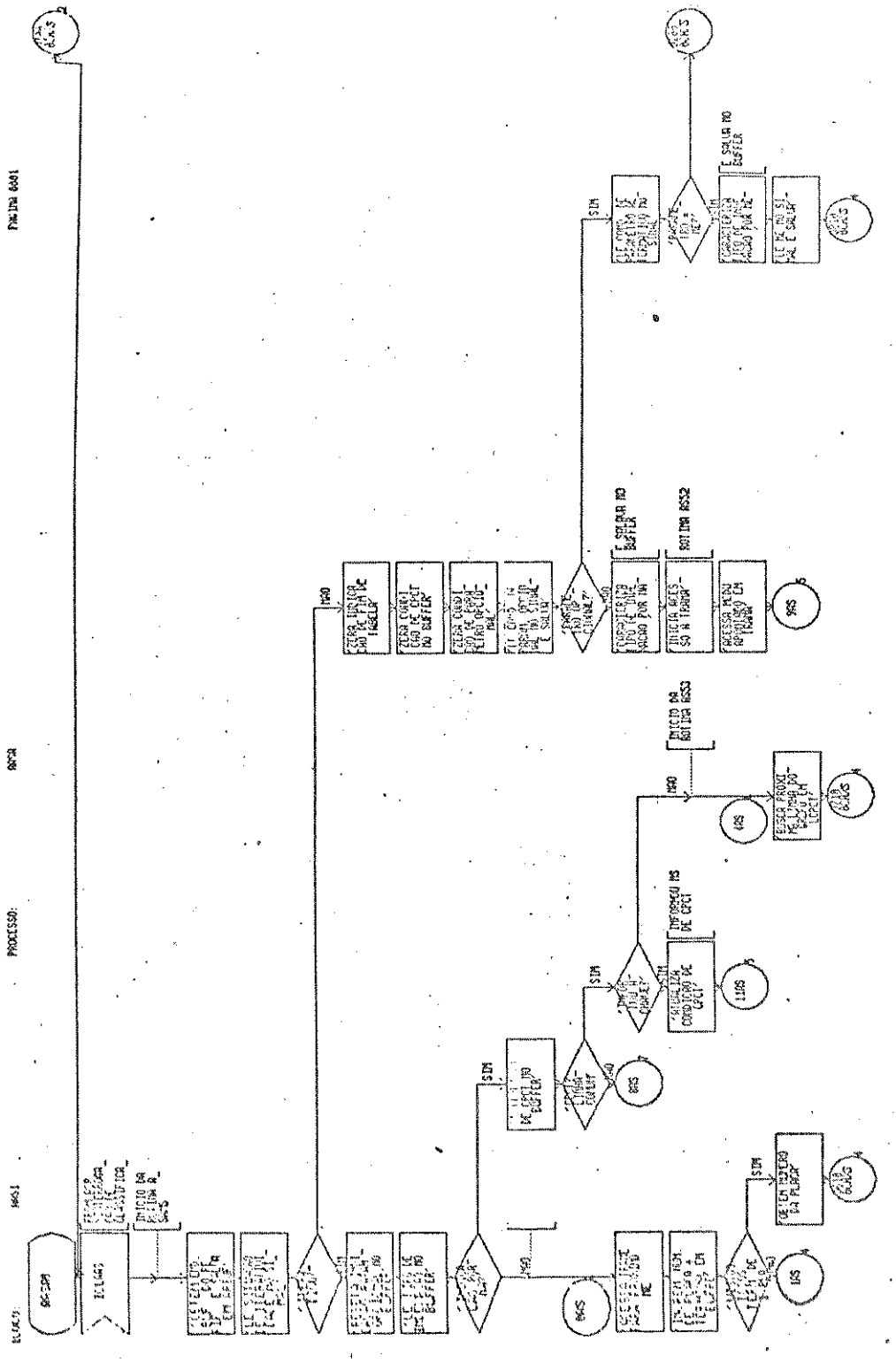


Figura 8.2.a - Página de saída do EGS v.4

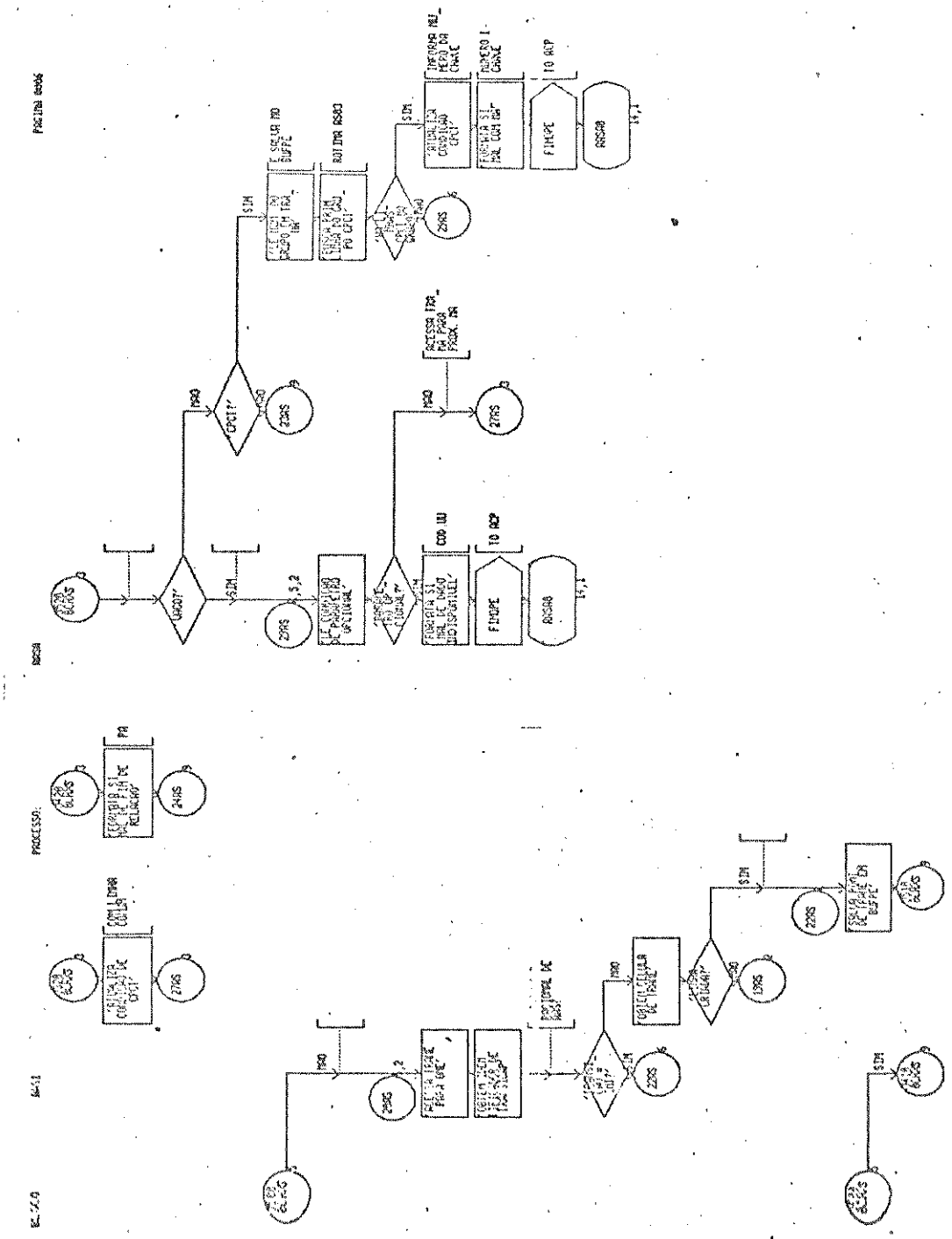


Figura 8.2.b – Página de saída do EGS v.4

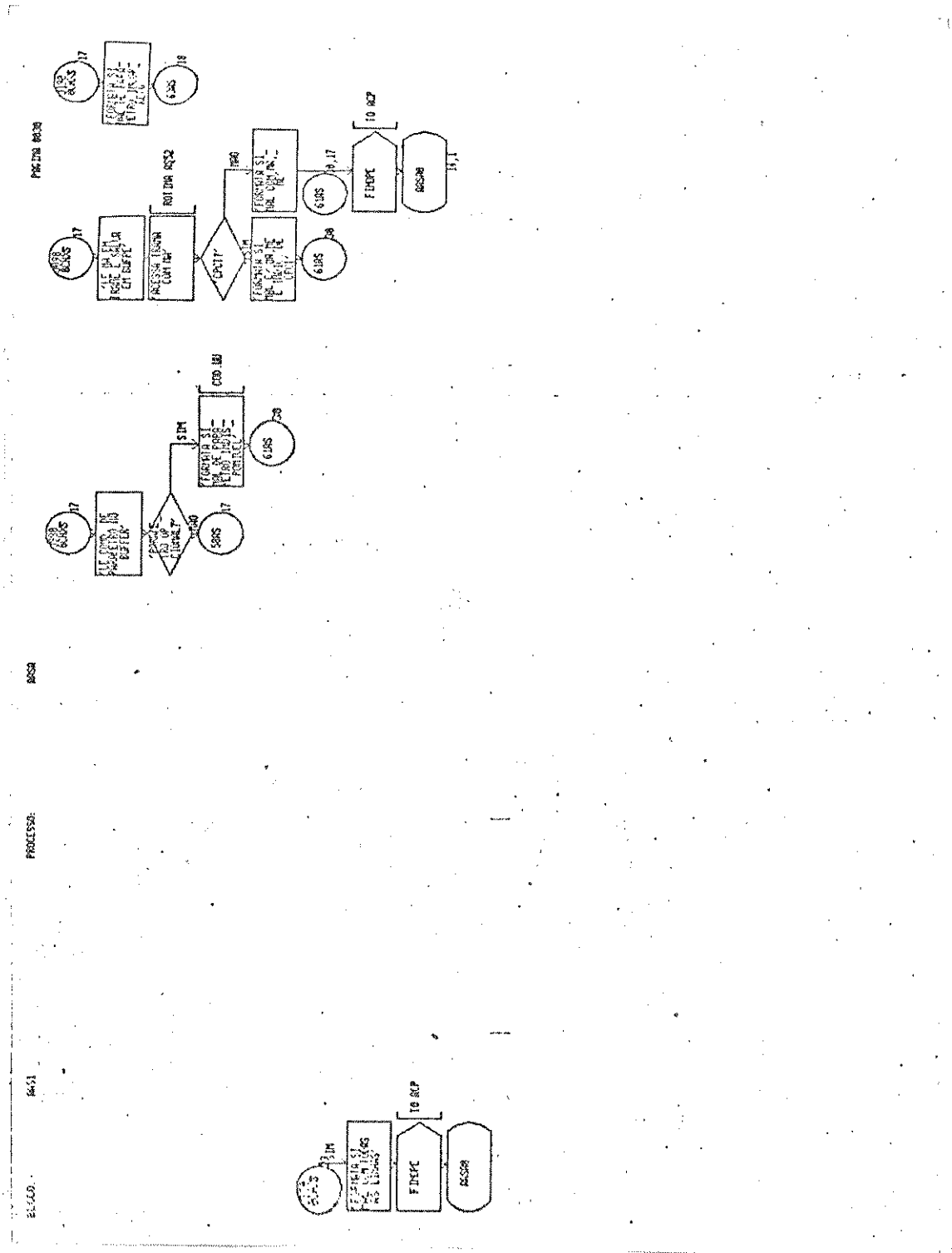


Figura 8.2.c - Página de saída do EGS v.4

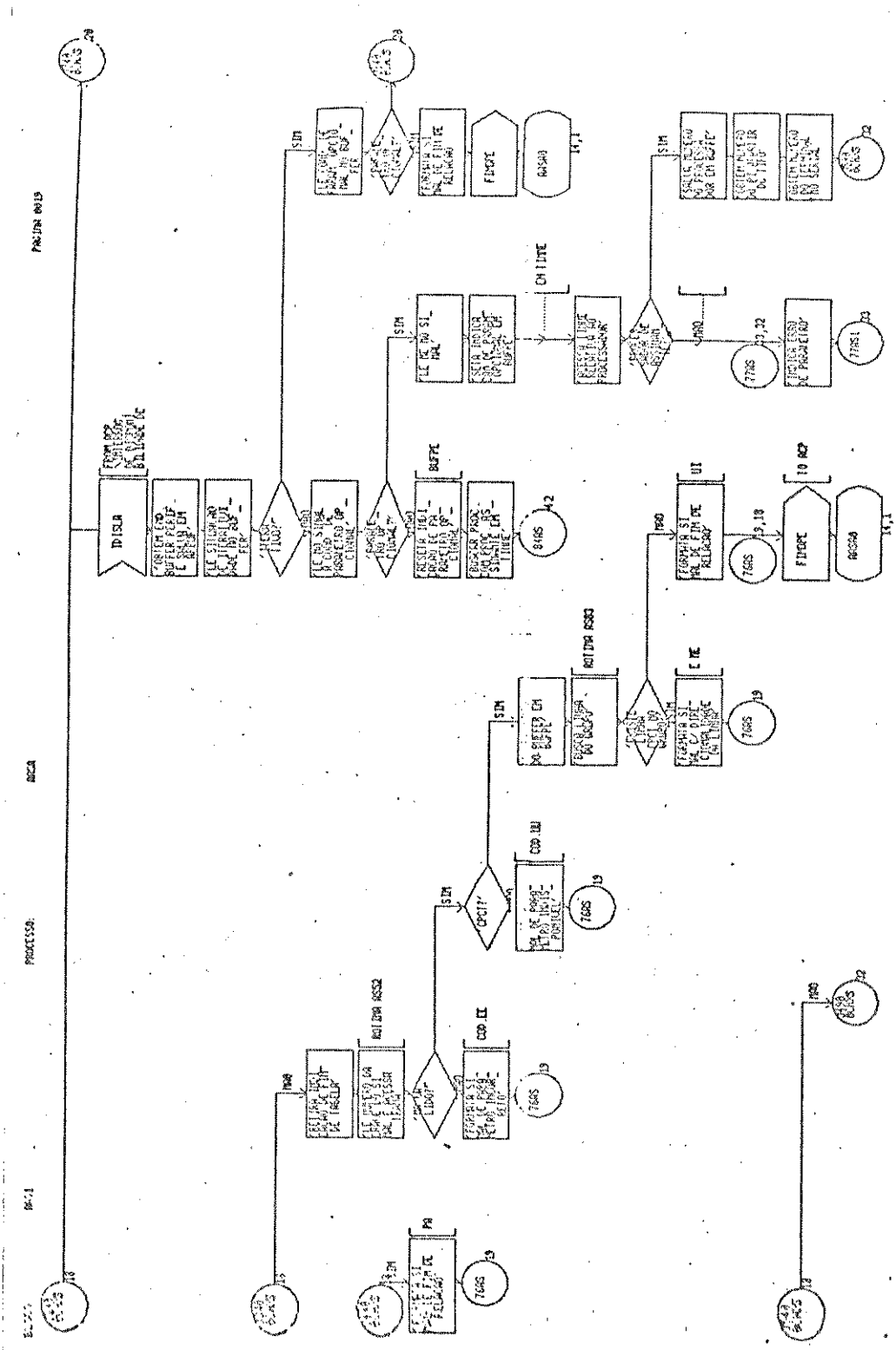


Figura 8.2.d - Página de saída do EGS v.4

Figura 8.2.e - Página de saída do EGS v.4

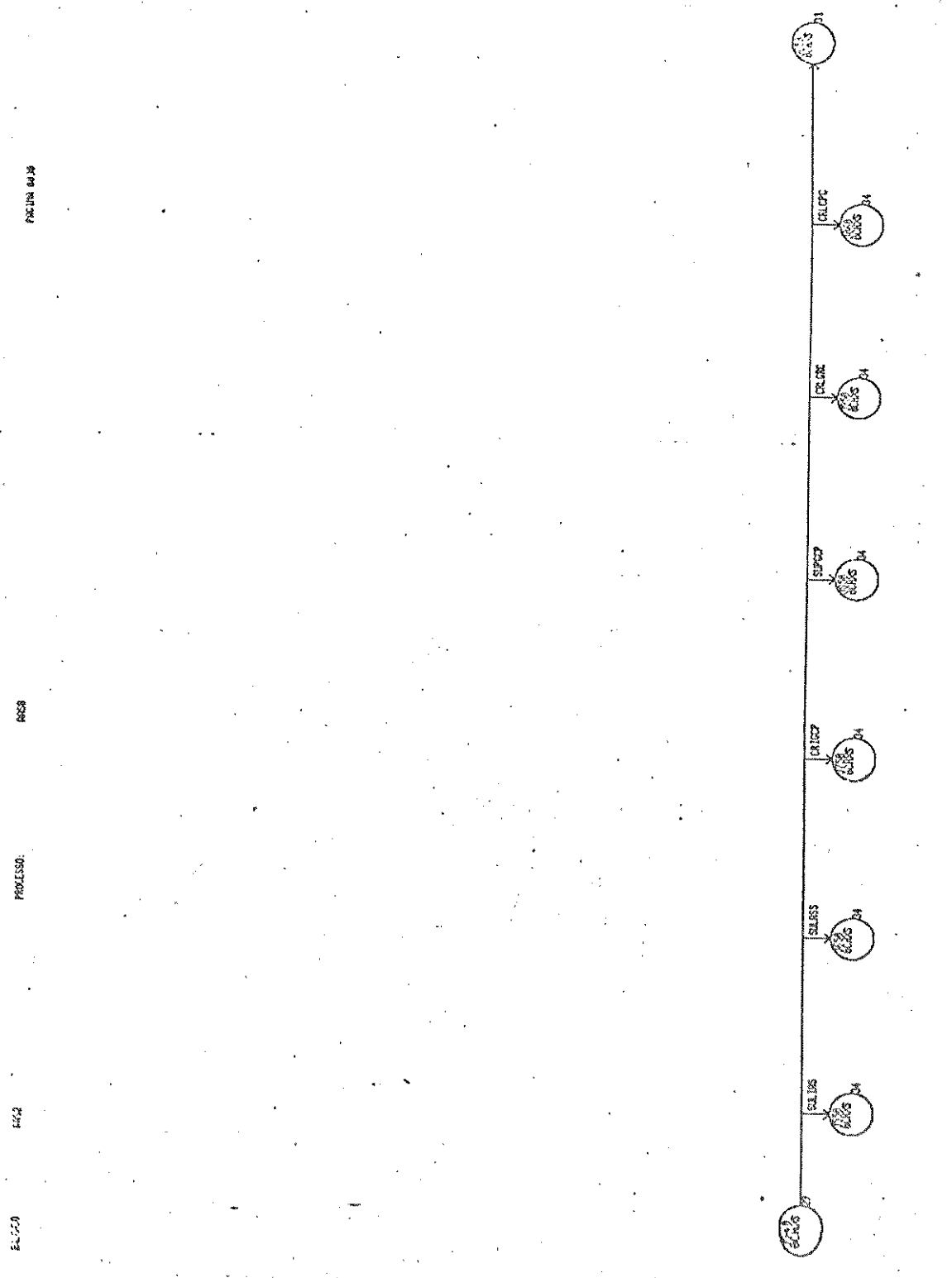


Figura 8.2.f - Página de saída do EGS v.4

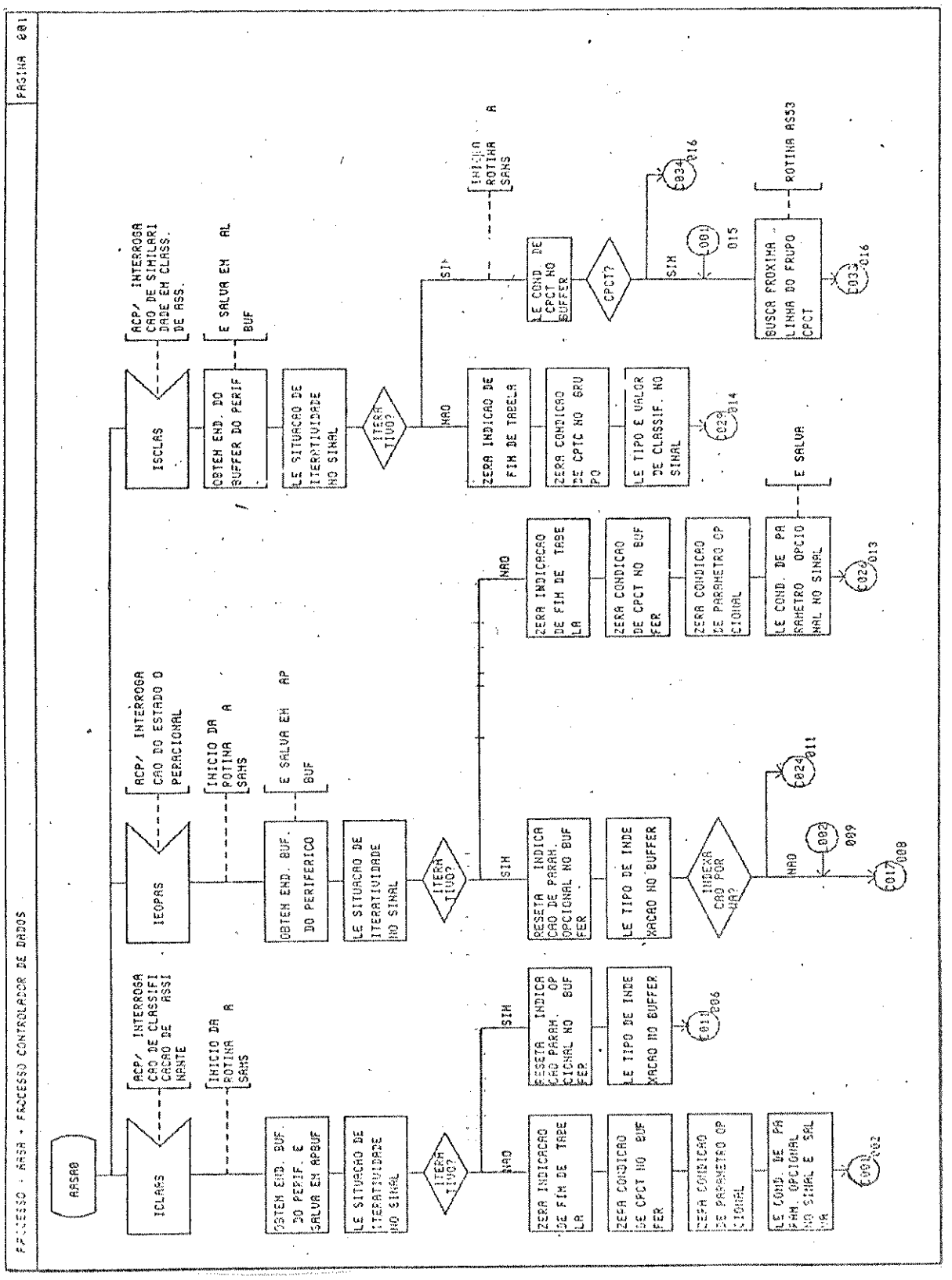


Figura 8.3.a – Página de saída do SADG

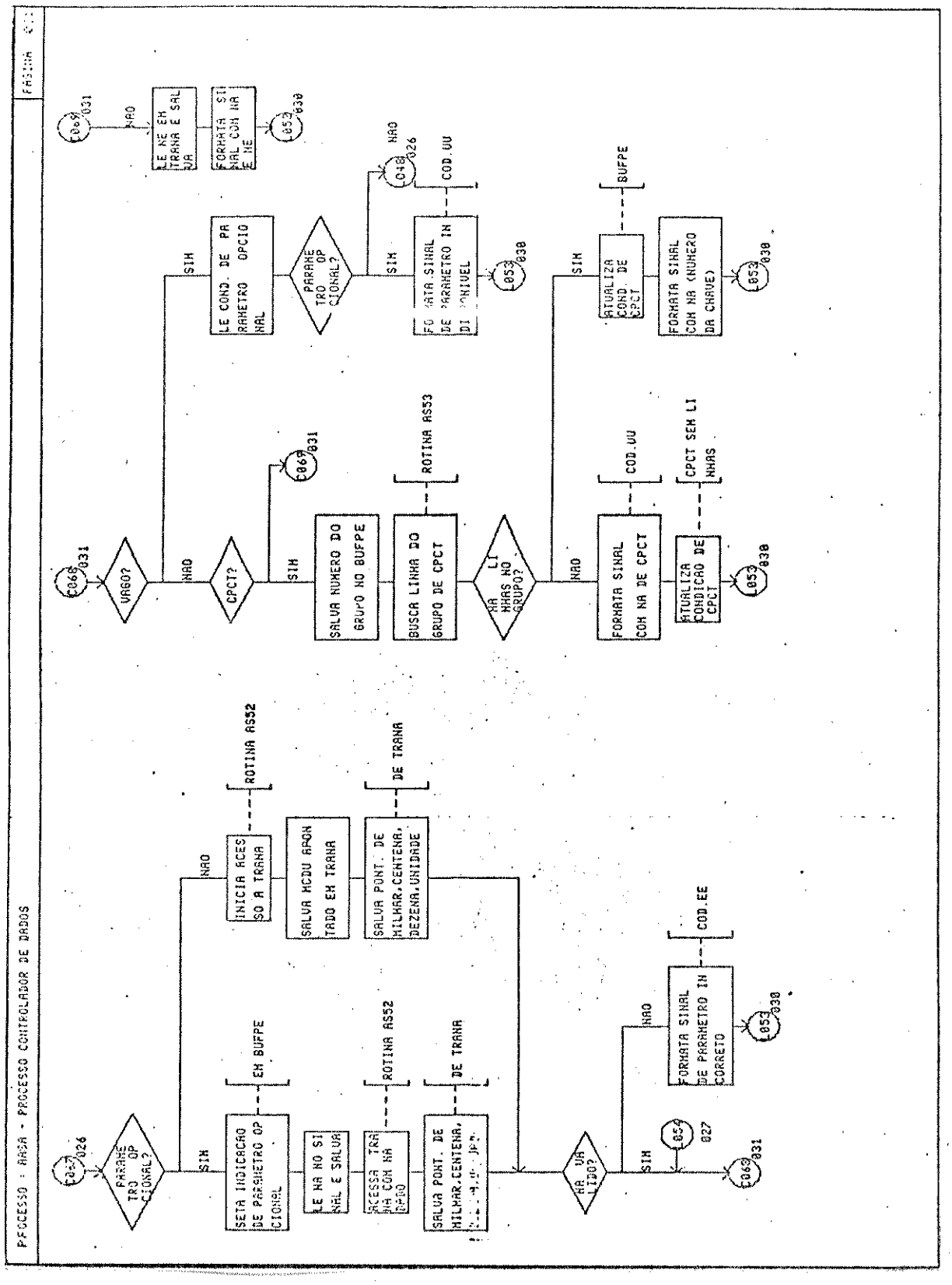


Figura 8.3.b – Página de saída do SADG

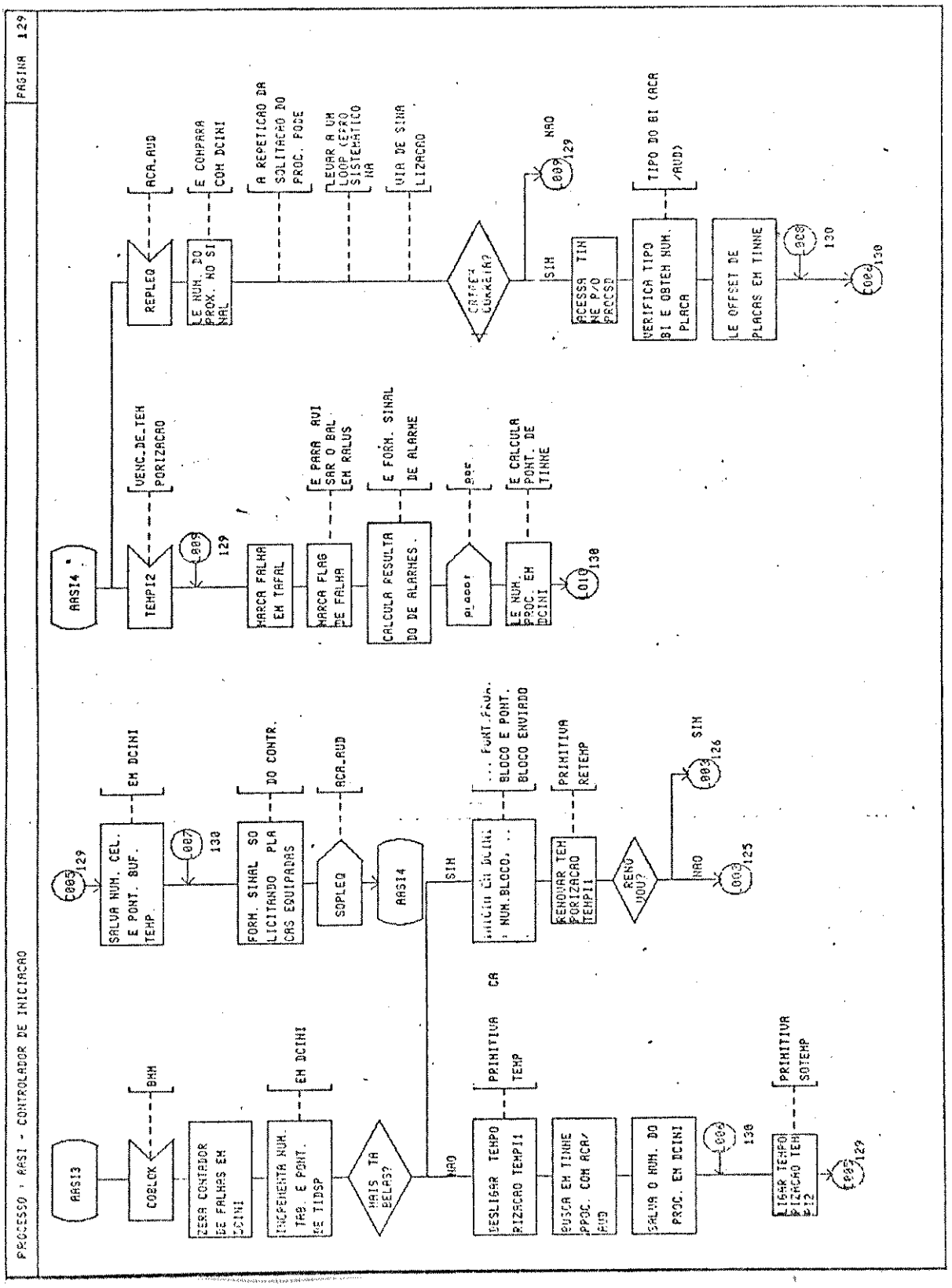


Figura 8.3.c - Página de saída do SADG

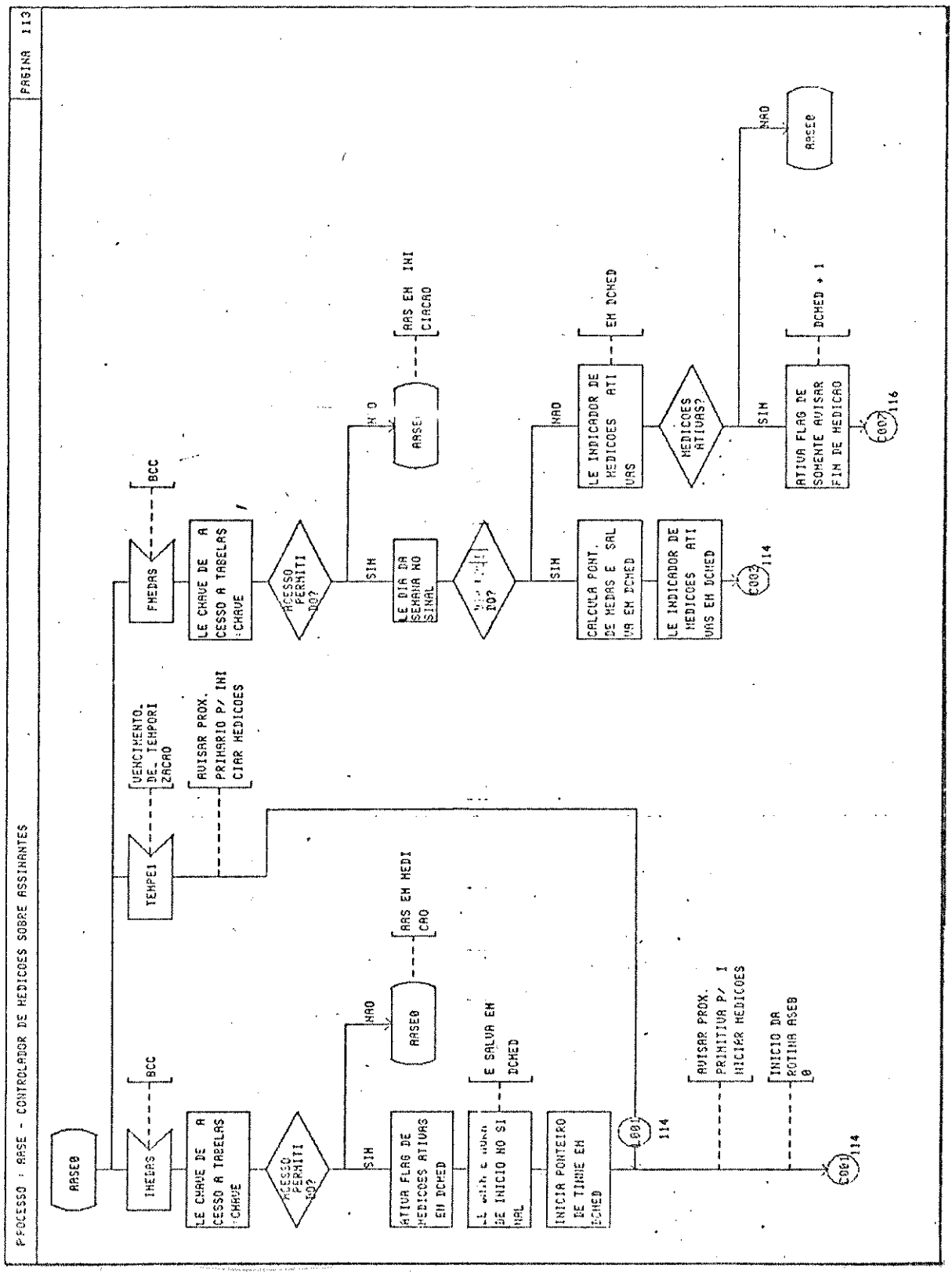


Figura 8.3.d – Página de saída do SADG

textos é muito maior, viabilizando a efetiva utilização do documento.

Outro fator de comparação é a substituição de labels e conectores por linhas de conexão; o EGS não dispõe desta função que é muito apreciada pelos usuários (uma vez que é sempre feita manualmente). Algumas páginas com conexões podem ser vistas nas figuras 8.4.a a 8.4.g onde aparecem exemplos de conexões simples, paralelas e múltiplas.

Um fator preponderante de comparação é o tempo de processamento, essencial para otimizar o esforço computacional, uma vez que a demanda de máquina tende a crescer vertiginosamente com o crescimento da utilização de ferramentas. O gargalo do sistema em termos de processamento é a geração do arquivo de impressão (função implementada no SADG no módulo Formatador) que no caso do EGS tem duas agravantes : só dispõe de processamento interativo e tem um processamento bastante lento.

No SADG a geração do arquivo de impressão pode ser feita totalmente em batch e o tempo de CPU de geração da forma de impressão foi diminuido cerca de 13 vezes em relação ao tempo do EGS, segundo testes comparativos realizados no VAX-750 entre ambas as ferramentas.

Na tabela 8.1 são apresentados os tempos de CPU de execução de cada um dos módulos do SADG no VAX-785 para a maioria dos módulos de implementação da central TRÓPICO-R. Num período de 10 horas durante a noite, no qual se possa considerar que o computador só estará executando o SADG, é possível gerar a forma gráfica de todos os blocos de implementação software da central TRÓPICO-R (que constam da tabela 8.1) aproximadamente 6 vezes.

A aceitação dos usuários foi amplamente satisfatória desde o início da utilização operacional do SADG (em dezembro de 1985) e não houve realimentações que demonstrassem insatisfação quanto à

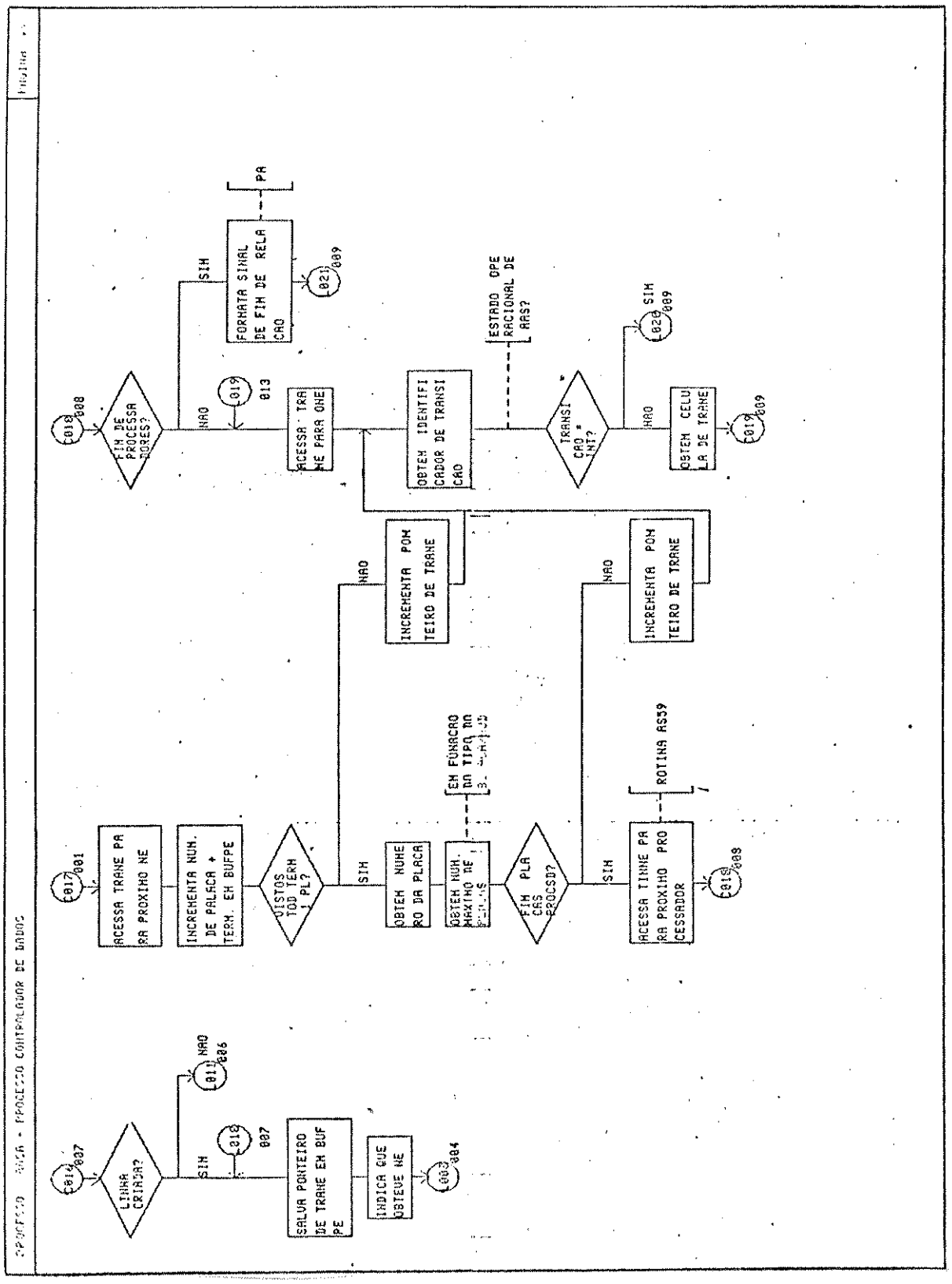


Figura 8.4.a - Página de saída do SADG com alguns tipos de conexões

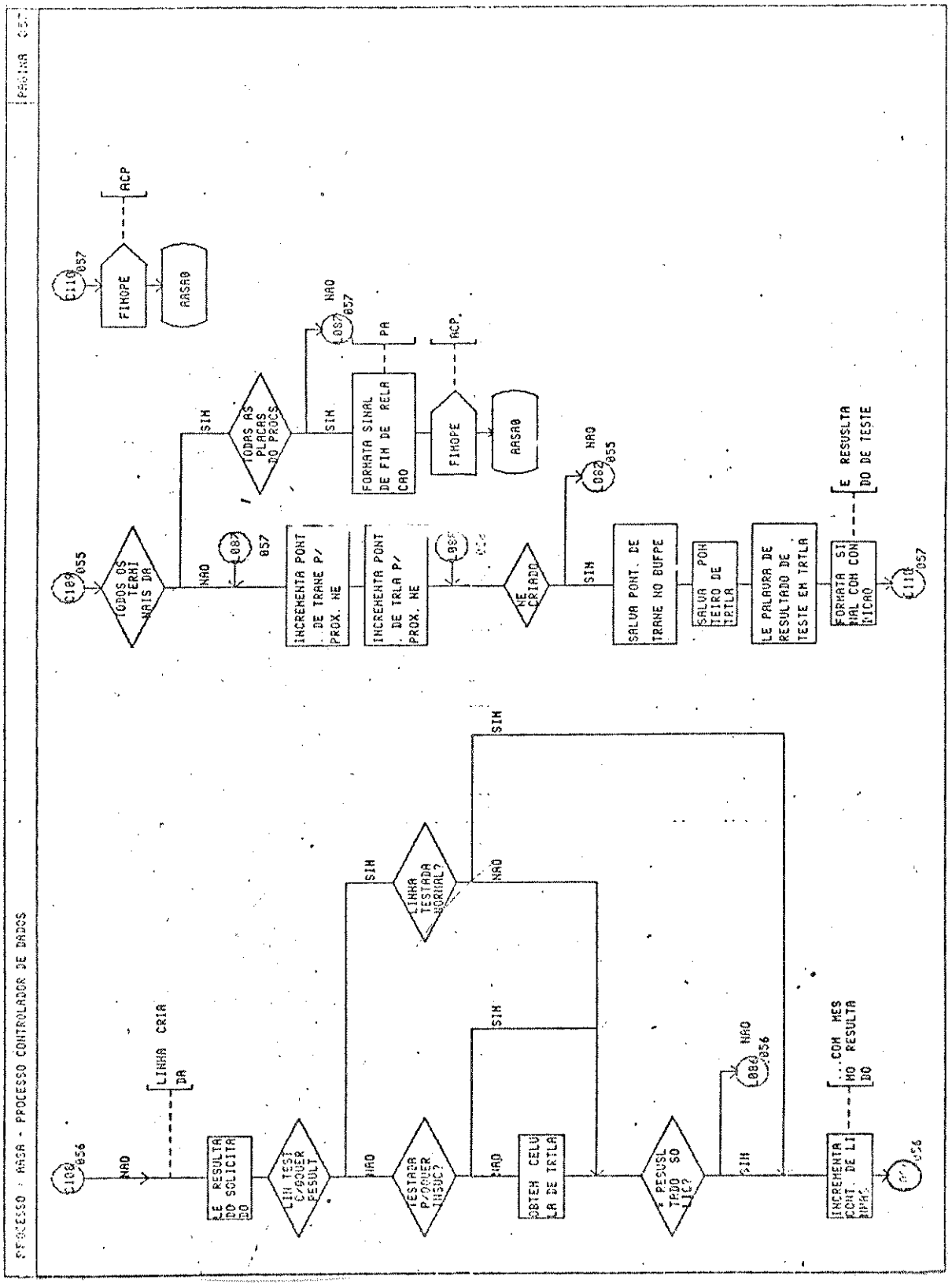


Figura 8.4.b - Página de saída do SADG com alguns tipos de conexões

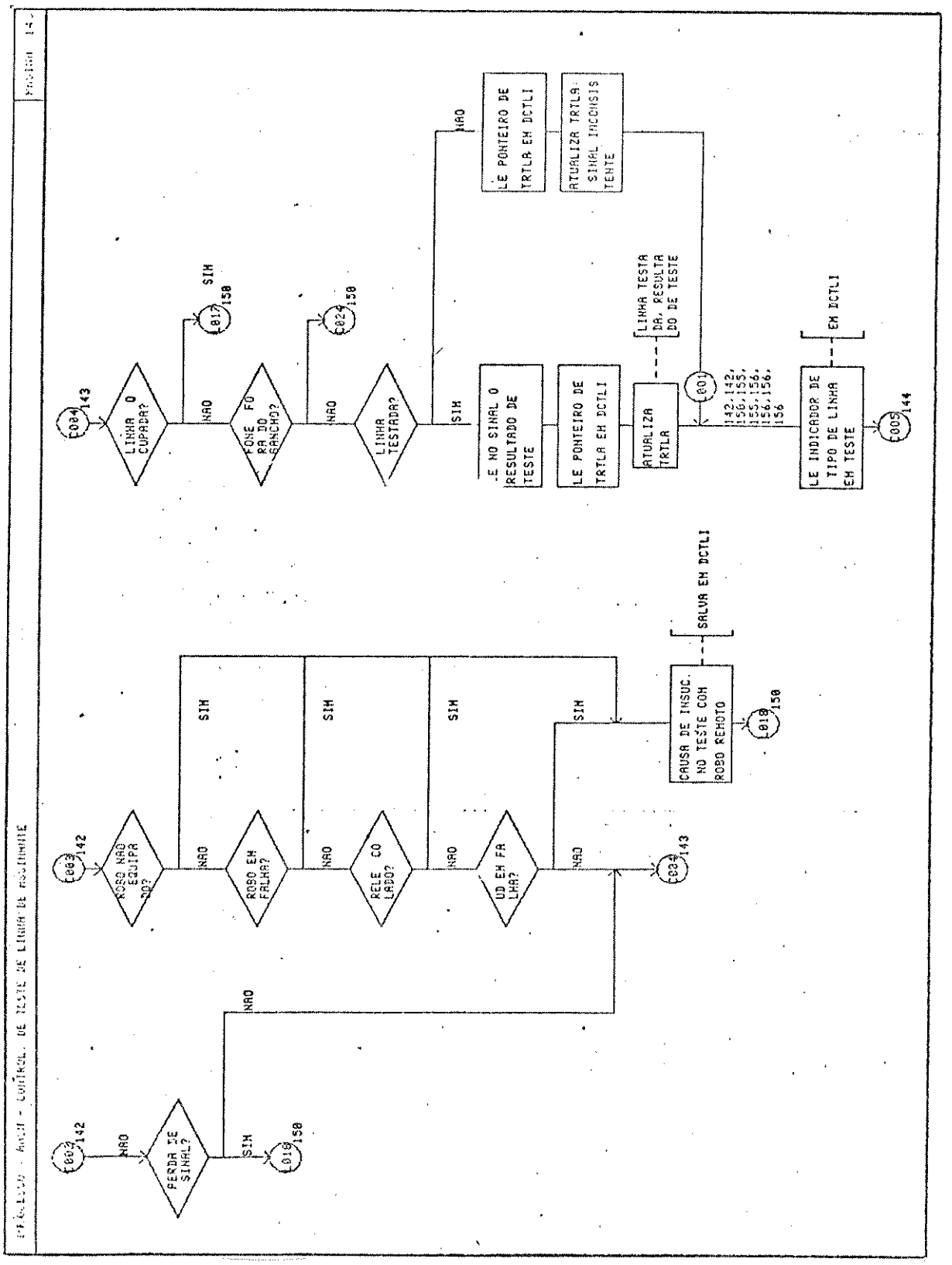


Figura 8.4.c - Página de saída do SADG com alguns tipos de conexões

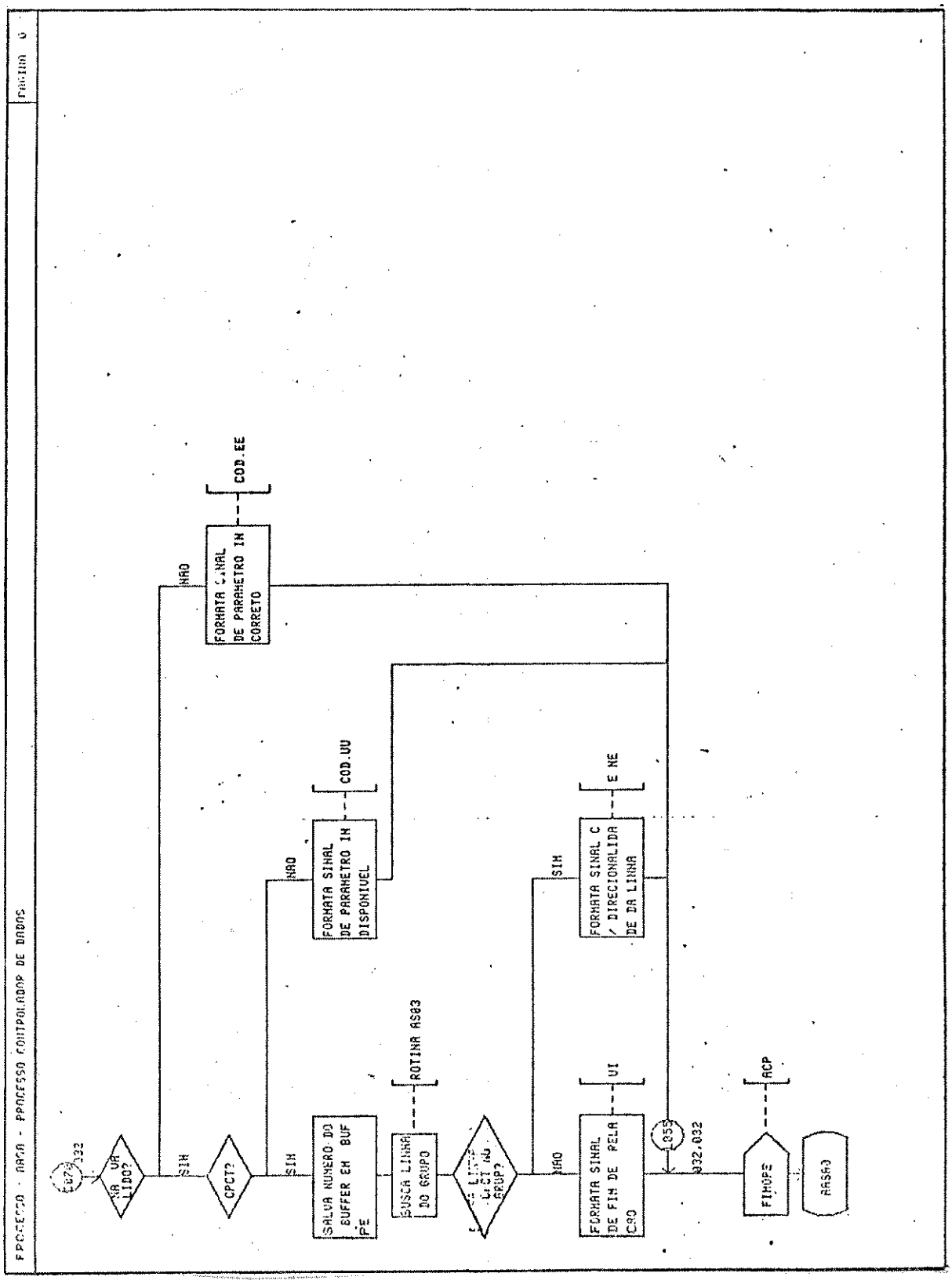


Figura 8.4.d – Página de saída do SADG com alguns tipos de conexões

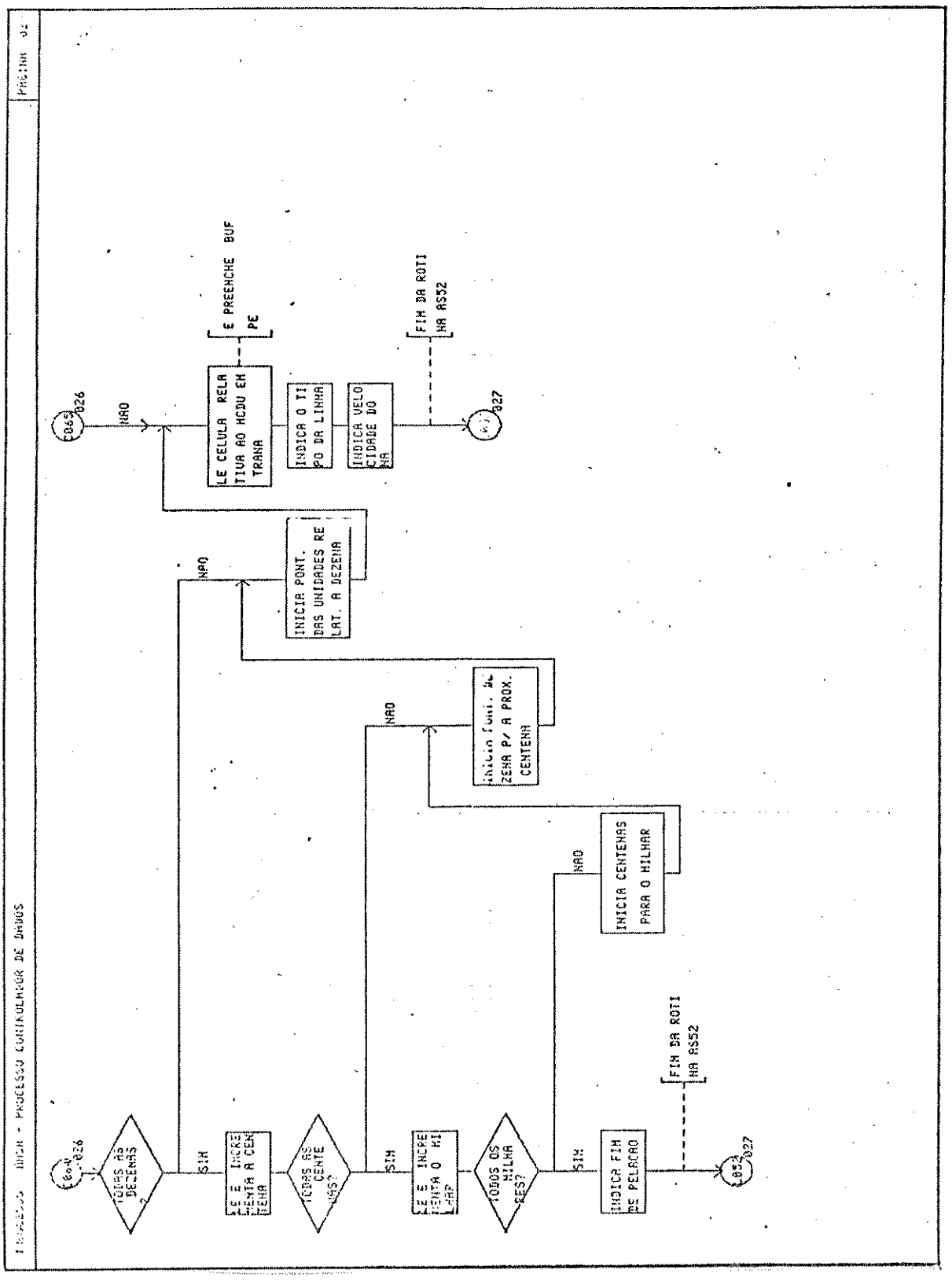


Figura 8.4.e - Página de saída do SADG com alguns tipos de conexões

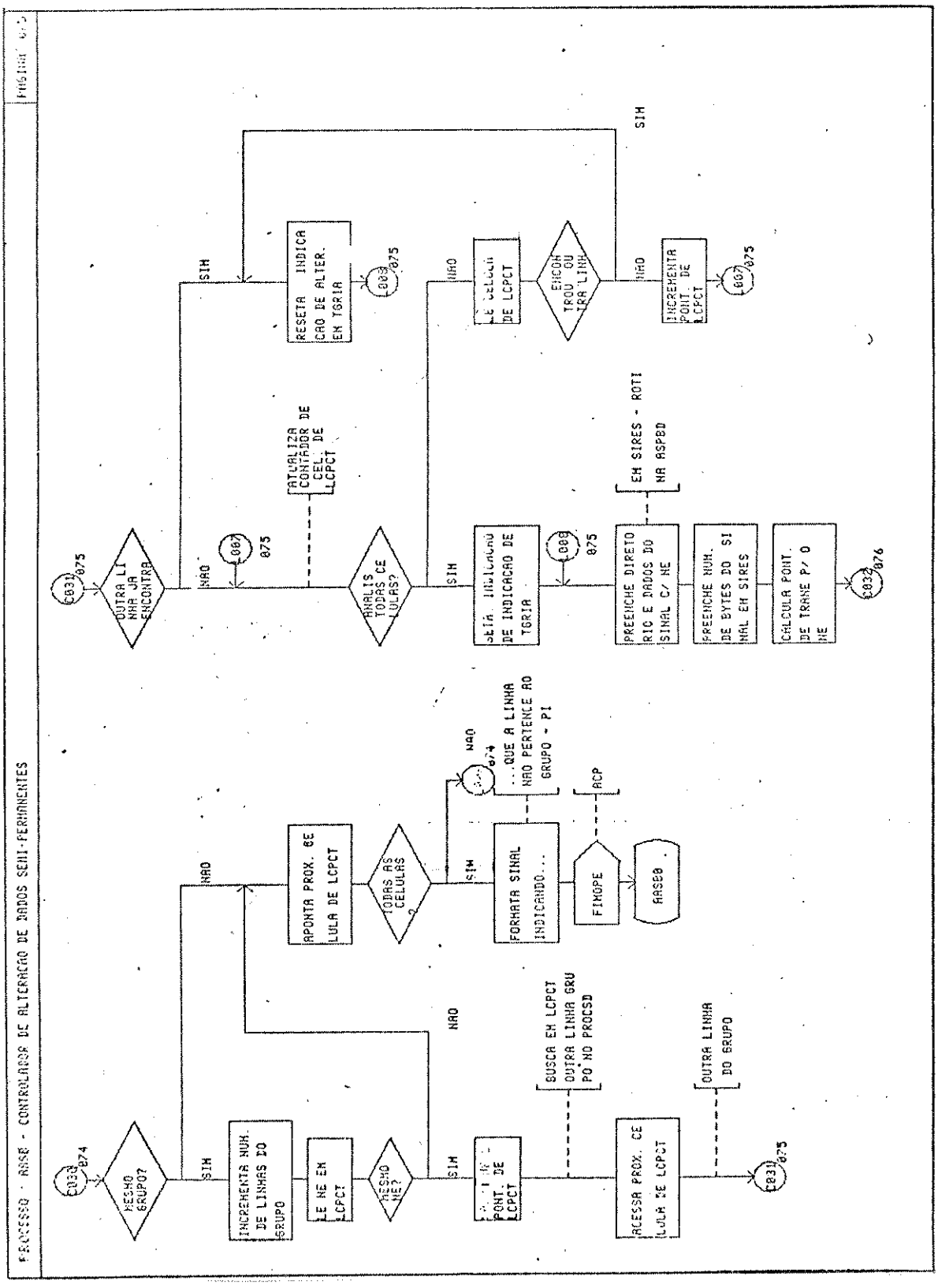


Figura 8.4.f - Página de saída do SADG com alguns tipos de conexões

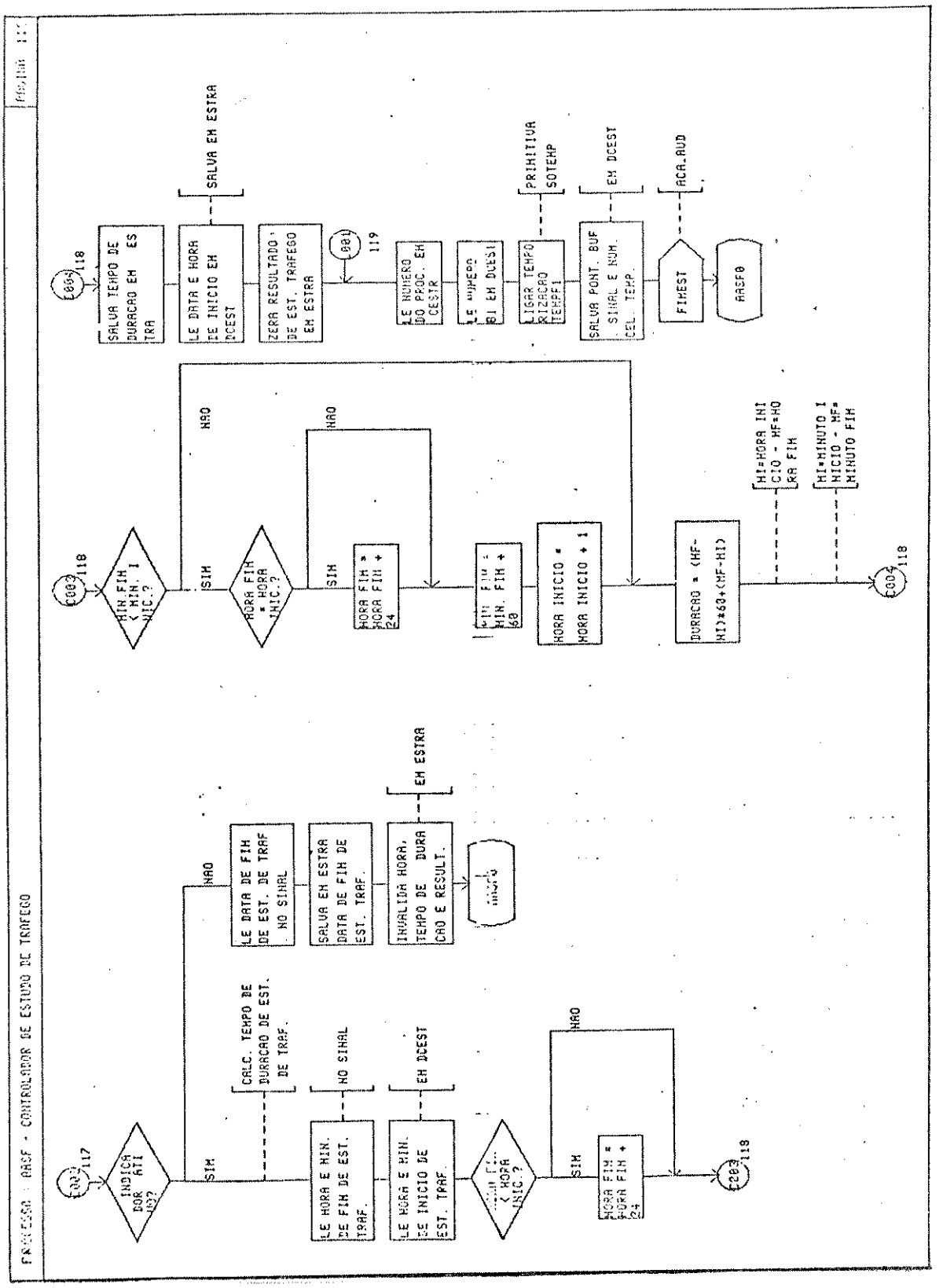


Figura 8.4.g - Página de saída do SADG com alguns tipos de conexões

BI	NP	NL	NS	TC	TP	TF	TT	RT	RP
BRF	10	462	252	13	9	28	50	198	36
ART	14	543	315	14	9	34	57	181	29
BCI	16	604	351	15	9	39	63	179	26
ATN	16	624	355	15	9	39	63	177	25
BCL	17	637	358	14	8	37	59	165	22
ACP	21	876	464	17	10	48	75	162	21
BPR	21	980	558	18	10	48	76	136	18
BCO	24	885	479	18	9	52	79	165	19
ARG	36	1603	863	27	13	81	121	140	15
AEM	39	1674	892	25	13	82	120	134	15
BAL	39	1721	924	26	12	86	124	134	13
ARM	40	1552	882	24	12	81	117	133	14
BCC	40	1739	962	27	14	86	127	132	14
BCM	44	1823	959	27	12	92	131	137	12
ATX	49	2112	1176	29	13	99	141	120	11
ACE	51	1892	1085	29	12	100	141	130	11
NUC	54	2489	1335	40	19	123	182	136	14
AEA	56	2069	1215	32	14	113	159	131	11
NUI	61	2739	1471	44	17	137	198	135	11
AJU	73	3040	1676	41	18	149	208	124	11
ARE	79	3260	1653	[illegible]	[illegible]	[illegible]	229	138	10
AJB	85	4082	2140	57	22	186	265	124	10
ACA	97	4437	2413	58	20	196	274	113	8
AJA	104	4393	2427	56	20	204	280	115	8
BMM	115	4302	2780	66	23	234	323	116	8
AUD	144	6089	3242	82	32	297	411	127	10

Tabela 8.1.a - Tempos de CPU dos módulos do SADG, com os blocos de implementação software da central TRÓPICO-R

BI	NP	NL	NS	TC	TP	TF	TT	RT	RP
ADG	153	6828	3510	90	27	311	428	122	8
AAS	156	6534	3675	83	31	340	454	123	8
ATA	365	11077	7160	195	71	695	961	134	10

Total de 29 BIs com :

NP	NL	NS	TC	TP	TF	TT
2019	81066	45572	1229	505	4182	5916

Média (aproxima-se mais do BI AJU) com :

NP	NL	NS	TC	TP	TF	TT
70	2765	1571	42	17	144	204

BI = Bloco de Implementação Software

NP = Número de Páginas de Saída no Formato A3H

NL = Número de Linhas do BI em SDL-PR

NS = Número de Símbolos do BI em SDL-GR

TC = Tempo de CPU do Compilador (segundos)

TP = Tempo de CPU do Paginador (segundos)

TF = Tempo de CPU do Formatador (segundos)

TT = Tempo de CPU Total (segundos)

RT = TT / NS (milisegundos)

RP = TP / NS (milisegundos)

Tabela 8.1.b - Tempos de CPU dos módulos do SADG, com os blocos de implementação software da central TRÓPICO-R

performance do SADG em relação à saída gráfica ou ao tempo de processamento. Ao contrário, o desempenho do SADG motivou a realização do projeto SAS, que engloba as funções do SADG como foi explicado anteriormente.

Concluindo, pode-se afirmar que o objetivo deste trabalho, o de desenvolver um ferramenta prática para tradução da forma PR para a forma GR do SDL, foi plenamente atingido. Ressalve-se que sofisticações nos algoritmos utilizados, que pudessem levar a melhores resultados, foram descartadas por implicarem num tempo de processamento inaceitável para uma melhoria apenas marginal na distribuição de símbolos.

REFERÊNCIAS BIBLIOGRÁFICAS

[BAGN81a] Bagnoli P., Giorcelli S., Saracco R.; "Applicattion of CCITT SDL for Software Development and its Maintainability"; ISS 81 CIC, Montreal (Canadá), setembro, 1981.

[BAGN81b] Bagnoli P., Cattoni E., Saracco R.; "Software Tools for SDL Documentation Handling and Software Development Methodology for Automatic Generation of SDL Documentation"; 2nd SETSS Conference, Warwick (UK), 1981.

[BARR79] Barret W. A., Couch J. D.; Compiler Construction : Theory and Practice; Computer Science Series, 1979.

[BATI85] Batista J. J., Jino M.; "Graphic Documentation Automated System"; II SDL Forum, Helsinki (Finlândia), março, 1985.

[BAUE76] Bauer F. L., Remer F. L. et all; Compiler Construction; Springer Verlag, segunda edição, 1976.

[BECK68] Beckwith R. C., O'Brien F.; "A Technique for Computer Flowchart Generation"; The Computer Journal, vol. 11, pg. 138-140, 1968.

[BELI85] Belina F., Nilsson G.; "SDT, A System Specification and Design Tool"; II SDL Forum, Helsinki (Filândia), março, 1985.

[BORG82] Borgnaes M., Waenmann A.; "Use of SDL at Ericsson and Tools for Drawing SDL Diagrams"; I SDL Forum, Florença (Itália), setembro, 1982.

[BRAE79a] Braek R.; "Unified System Modelling and Implementation"; ISS, Paris (França), 1979.

[BRAE79b] Braek R.; "Functional Specification and Description Languages as a Practical Tool for Improved Systems"; 3th Word Telecommunication Forum, Genebra (Suíça), 1979.

[BRAE82] Braek R.; "SDL, SOM and DASOM"; I SDL Forum, Florença (Itália), 6-9 setembro, 1982.

[CAIN81] Cain G. J., Jackson L. N., Vesetas R., Walter A., Yong W. B.; "Computer Aided Software Generation (the MELBA system for generating CHILL code); IEE SETSS Conference, University of Warwick (UK), julho, 1981.

[CALC68] CALCOMP; "FLOWGEN/F Flowchart Software Package"; Digital Plotting Newsletter, maio/julho, 1968.

[CCIT80] CCITT; Recommendations Z.100 a Z.103, Orange Book, 1980.

[CCIT84a] CCITT; Recommendations Z.100 a Z.104, Red Book, 1984.

[CCIT84b] CCITT; SDL User Guidelines, Red Book, 1984.

[CCIT84c] CCITT; Recommendation Z.200, Red Book, 1984.

[CCIT85] CCITT; CCITT SDL Newsletter, agosto, 1985.

[COAK78] Coakley F., Weaver R., Hill A.; "The SX1 Program Construction Tool"; IEE, junho, 1978.

[CORK76] Corker M., Coakley F.; "Automatic Code Generation for SPC call processing"; IEE SETSS Conference, Saltzburg (Áustria), 1976.

[DELL81] Dell P. W., Jackson L. A., Brien R.; "Computer Aided Design for Software"; Software and Microsystems, outubro, 1981.

[FERR82] Ferrero C.; "EGS : Software Package for SDL"; I SDL Forum, Florença (Itália), setembro, 1982.

[GAIS83] Gaismaier B., Schirmeier H.; "An integrated Software Development Method for Switching System Based on CCITT SDL"; 4th SETSS Conference, WarwicK (UK), 1983.

[GASS85] Gassino E., Bagnoli P.; "EGS Evolution to Support Full SDL"; "II SDL Forum, Helsinki (Finlândia), março, 1985.

[GERR81] Gerrand P., Nguyen K. D.; "CADDIE : A Computer Graphics Aid for the Behavioral Specification of New Communications Facilities for the Telecommunications Network"; A.T.R., Vol. 15, N. 1, 1981.

[GERR82] Gerrand P.; "CADDIE : A Computer Aided Design

Developed in Australia"; I SDL Forum, Florença (Itália), setembro, 1982.

[GHIS85] Ghisio O., Barra S.; "The Use of Artificial Intelligence in the Transformation from SDL to CHILL"; II SDL Forum, Helsinki (Filândia), março, 1985.

[GIAR83] Giarratana V., Modesti M.; "An SDL into CHILL Skeleton Translation System"; 4th SETSS Conference, WarwicK (USA), 1983.

[GOET65] Goetz M. A.; "Recent Developments in Automated Program Documentation"; Proc. of the 4th Annual Meeting of the U.A.I.D.E., 1965.

[GUTF82] Gutfeldt H.; "SDL and CHILL Structured Programming"; I SDL Forum, Florença (Itália), setembro, 1982.

[HAIB59] Haibt L. M.; "A Program to Draw Multilevel Flowcharts"; Proc. Western Joint Computer Conference, pg. 131-137, 1959.

[HAIN65] Hain G., Hain K.; "Automatic Flowchart Design"; Proc. 20th National Conf. of the ACM, pg. 513-523, 1965.

[HIGH69] Hightower D. W.; "A Solution to Line-Routing Problems on the Continuous Plane"; 6th Design Automation Workshop, pg. 1-24, junho, 1969.

[HIGH80] Hightower D. W., Boyd R. L.; "A generalized Channel Router"; ACM IEEE 17th Design Automation Conference,

Minneapolis (EUA), pg. 12-21, junho, 1980.

[KNUT63] Knuth D. E.; "Computer Drawn Flowcharts"; Communications of the ACM, vol. 6, pg. 555-563, 1963.

[KONI82] Koning H.; "Interactive generation of SDL diagrams"; I SDL Forum, Florença (Itália), setembro, 1982.

[KOWA83] Kowaltowski T.; Implementação de Linguagens de Programação; Guanabara Dois, 1983.

[KRID64] Krider L.; "A Flow Analysis Algorithm"; Journal of the ACM, vol. 11, pg. 429-436, 1964;

[LIES86] Liesenberg H. K. E.; Projetos VSLI e seu Suporte Computacional; V Escola de Computação, Belo Horizonte (MG), julho, 1986.

[MACA82] Macaleno G.; "Drawing SDL diagrams is not so easy"; I SDL Forum, Florença (Itália), setembro, 1982.

[RUST85] Rustad H.; "SILL, a Tool for Translation from SOM/SDL to CHILL with Extensive High Level Test and Debug Facilities"; II SDL Forum, Helsinki (Finlândia), março, 1985.

[SAAL65] Saalbach C. P., Sapovchak B. J.; "The Flowchart Diagram"; Proc. of the 4th Annual Meeting of the U.A.I.D.E., 1965.

[SCOT58] Scott A. E.; "Automatic Preparation of Flowchart

Listings"; Journal of the ACM, vol. 5, pg. 57-66, 1958.

[SHER66] Sherman P. M.; "FLOWTRACE, a Computer Program for Flowcharting Programs"; Communications of the ACM, vol. 9, pg. 845-854, 1966.

[SILV81] Silva H. V. R. C.; Recuperação de Erros em Analisadores Sintáticos Descendentes; Tese de Mestrado, UNICAMP (IMECC), 1981.

[TAIP82] Taipale O.; "A tool for Drawing SDL Diagram"; I SDL Forum, Florença (Itália), setembro, 1982.

[TAIP85] Taipale O.; "A Compiler for Translating SDL/PR Documents to SDL/GR"; II SDL Forum, Helsinki (Finlândia), março, 1985.

[TINK85] Tinker R., Howcroft A. R., Lewis R.; "An SDL Support Environment, SX1-CADOS"; II SDL Forum, Helsinki (Finlândia), março, 1985.

[VEFS85] Vefsnmo E.; "DASOM - An Advanced Computer Aided Tool Supporting an SDL Oriented Methodology"; II SDL Forum, Helsinki (Finlândia), março, 1985.

[WATK73] Watkins R. P.; "A Survey of Automatic Flowchart Generators"; The Australian Computer Journal, Vol. 5, pg. 132-140, novembro, 1973.

[WATK74] Watkins R. P.; "The Design of Flog, an Automatic

Flowchart Generator"; The Australian Computer Journal, Vol. 6, pg. 98-123, novembro, 1974.

[WIRT76] Wirth N.; Data Structures + Algorithms = Programs; Prentice-Hall, 1976.