知識工学と人工知能 34－5
（1984. 5. 8）

知識ベースを用いたSDL支援システム

加藤英樹　吉田裕之　広瀬貞樹*　鈴木忠道
富士通研究所　（*現神奈川大学）　富士通

1. はじめに

通信ソフトウェア(電子交換機用ソフトウェア)の開発・保守のコストは，実時間性，規模の大きさ，社会的責任の重さといった条件の厳しさもあって莫大なものになってきており，抜本的な対策が必要であるといわれている．我々は，通信ソフトウェアの開発の生産性，信頼性，保守性の向上を目的として，機能仕様記述言語SDL[1]に基づいた通信ソフトウェア開発支援システム(SDL支援システム[2],[3])の研究・開発を行っている．本論文ではこのSDL支援システムについて述べる．

1.1 システムの概要

上記の目的を達成するためには，

(1) ソフトウェアの記述レベルを高くし，人間にわかりやすいものにする，
(2) 設計の初期の記述レベルが高い段階で誤り(バグ)を発見する，
(3) 支援環境の知的レベルを高め，人間が誤りを犯すのを未然に防ぐ，

といったことが必要であると考え，本システムに知識工学的手法を導入した．

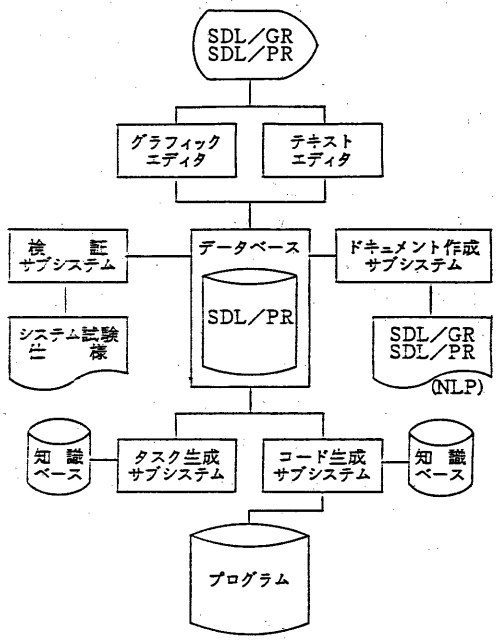

図1.1 システム構成図

1.2 SDL

SDL(Functional Specification and Description Language) は， C.C.I.T.T. (International Telegraph and Telephone Consultative Committee, 国際電信電話諮問委員会)の勧告による，状態遷移図をベースとした電子交換機の機能仕様を記述するための言語で，図形による表現(Graphical form, SDL／GR)と，通常のプログラミング言語と似た表現(Programme-like form, SDL／PR)の二種類の表現がある．

1.3 概要状態遷移図と詳細状態遷移図

SDLは，電子交換機の内部状態間の遷移をフローチャート風に記述する言語であり，内部状態(State)，信号の着信(Input)，発信(Output)，判断(Decision)，処理(Task)など各種のシンボルが用意されている．各シンボル内の記述には特に規定が無く，任意の抽象度で機能仕様を記述できる．我々は，概要状態遷移図(General Level State Transition Diagram, G-STD)および詳細状態遷移図(Detailed State Transition Diagram, D-STD)と呼ぶ，抽象度が異なる二種類の機能仕様記述を用いている．

G-STDは，主に状態間の遷移関係に注目して処理の概要を表現するものであり，遷移のトリガと分岐を，比較的抽象的な名前のInputとDecision(以下ではこれらをG-STD要素と呼ぶ)を用いて記述する．他方，D-STDはより抽象度の低いInput, Decision, Output, およびTask(以下ではこれらをD-STD要素と呼ぶ)を用いて記述され，実際のプログラムとほぼ一対一に対応している．

1.4 システム構成

図1.1は本システムの構成図である．

利用者は「グラフィック・エディタ」を用いて対話的にSDL／GRによる仕様記述(状態遷移図)を作成・更新する．また，通常のテキスト・エディタで直接SDL／PRを取り扱うこともできる．

作成された仕様記述は，中央の「データベース」に更新履歴・バージョンなどの管理情報とともに格納される．

「タスク自動生成サブシステム」は，知識ベース中に格納された電子交換機のプログラマから得た知識を用いて，処理の概要だけを記した抽象度の高い仕様記述から，詳細な仕様記述を生成する．これにより，利用者のコーディングの負担が軽減され，誤りが生じる機会も減少する．

詳細な仕様記述は，「コード生成サブシステム」により実際のプログラムに変換される．ここではターゲット言語に関する知識ベースが使われている．

「検証サブシステム」は入力された仕様記述中の誤りを発見すべく，デッドロックの検出，処理の抜けのチェック，仕様記述の記号実行などの機能を持っている．また，実機試験用の試験仕様を生成する．

「ドキュメント作成サブシステム」は，データベース中の仕様記述から，状態遷移図の清書図面などの種々のドキュメントを作成する．

2. グラフィックエディタ

通信ソフトウェアの機能仕様はSDLで記述される．SDLには，前述のように，図形による表現(SDL／GR)と，通常のプログラミング言語と似た表現(SDL／PR)とがあるが，本グラフィックエディタはSDL／GRによる編集をサポートする．編集後の仕様記述はSDL／PRに変換され，GEM(Generalized Editing and Management Facilities)[4]と呼ばれるソフトウェア開発管理システム中に保存される．直接SDL／PRを編集するには，GEMのテキストエディタを用いる．

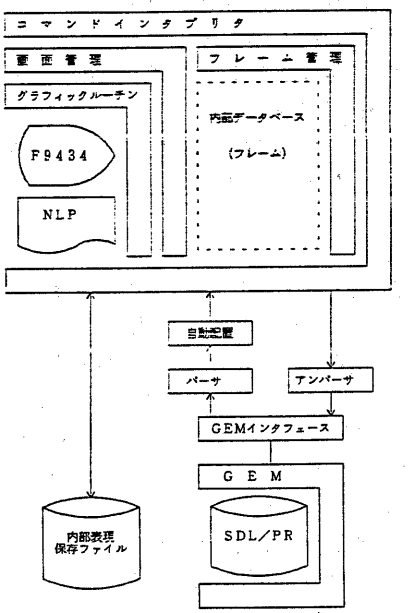

図2.1 グラフィックエディタの構成図

2.1 グラフィックエディタの特徴

本エディタは，次のような特徴をもっている．

(1) 対話型編集

編集は，グラフィック画面を見ながら対話的に行われる．

(2) メニュー方式の編集コマンド

コマンドには，大きく分けてメニューコマンドとキーボードコマンドの二種類がある．メニューコマンドは，ディスプレイ画面上に用意されたメニューをタブレット，あるいはジョイスティックなどのポインティング・デバイスで指すことで実行される．ほとんどの編集コマンドは，このメニューコマンドであり，キーボードを使わないので，ユーザの負担は軽い．キーボードを用いるコマンドには，補助的なコマンド，あるいは通常のTSSコマンドなどがある．

(3) フレームの利用

内部データ構造としてフレームを用い，フレームのインヘリタンス機構，デモン機能を活用してコンパクトでかつ柔軟な構造を実現している．

(4) SDL／PRとSDL／GRとの相互変換

GEM中に保存されている仕様記述は，SDL／PRパーサによって内部データ構造(フレーム)に変換され，画面上にはSDL／GRで表示される．また，編集後の機能仕様はSDL／PRに変換されGEM中に保存される．GEMのテキストエディタなどで直接入力されたSDL／PRには，座標などの図形情報が含まれていないので，内部表現に変換後，自動配置ルーチンが自動的にこれを付加する．

(5) 版管理，更新履歴管理

編集されたデータはSDL／PRに変換され，GEM中に保存される．このため，GEMが備えている版管理，更新履歴管理機能がそのまま利用でき，過去の更新履歴，あるいは昔の版を取り出すことなどができる．

2.2 グラフィックエディタの構成

図2.1は，エディタの構成図である．エディタは，

(1) ユーザからのコマンドを解釈・実行するコマンドインタプリタを中心に，
(2) SDL／PRを内部表現に変換するパーサ，
(3) 図形情報を付加しSDL／GRに変換する自動配置ルーチン，
(4) 内部データベースを管理するフレーム管理，
(5) ディスプレイの画面を管理する画面管理，
(6) SDL／GRをSDL／PRに変換するアンパーサ，
(7) GEMとのインタフェースルーチン，
(8) グラフィックディスプレイやNLPを制御するグラフィックルーチン，

から構成されている．

図2.2にグラフィックエディタの画面を示す．

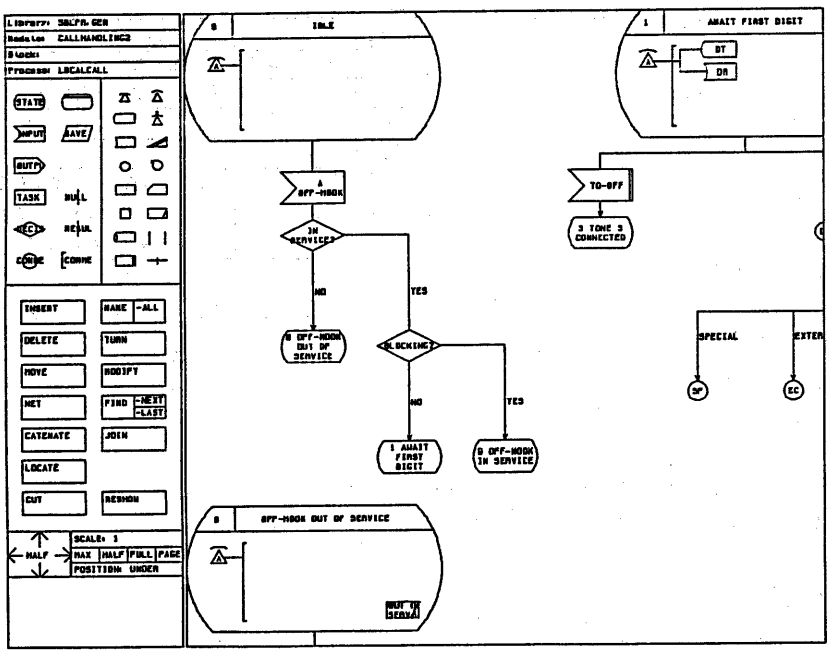

図2.2 グラフィックエディタの画面

3. タスク生成サブシステム

3.1 タスク生成サブシステムの概要

タスク生成サブシステムは，G-STDを入力し，知識を利用してこれを具体化し，D-STDを出力する．G-STDをspecificationとし，D-STDを対応するimplementationとすれば，これを一種の自動プログラミングシステムとみなすことができる．

図3.1はタスク生成サブシステムの構成図である．

データベース設定部は，G-STD要素定義，D-STD要素定義を各々の内部データベースに格納する．

内部表現生成部は，入力した各G-STD要素を，G-STD内部データベースからパターンマッチングを用いて検索し，内部表現に変換する．ここで内部表現は各G-STD要素が期待する効果，すなわち「何をすべきか」を表わしている．

推論部は，その「効果」をD-STD内部データベースからパターンマッチングで検索して，それを達成できるD-STD要素(の組合わせ)を推論し，D-STDによる記述を生成する．

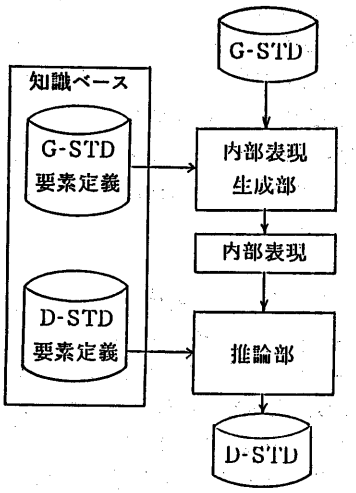

図3.1 タスク生成サブシステムの構成

3.2 G-STD, D-STD要素定義の例

図3.2，図3.3は各々G-STD要素，D-STD要素の定義例である．ここで各属性は次のような意味を表わしている．

- name
  この要素の名前．
- condition
  パターンマッチング時の条件．
- required-effect
  このG-STD要素が期待する効果．
- effect
  このD-STD要素が達成できる効果．
- pre-required-effect
  このD-STD要素を用いる際に，あらかじめ達成されているべき効果．
- post-required-effect
  このD-STD要素を用いる際に，それより後で達成されるべき効果．

図中でXはパターンマッチングの際にinstantiateされる変数を表わしている．

種別	属性	記述
入力	名前	X off-hook
	マッチング条件	Xは加入者名
	意味	加入者Xが受話器を上げた

判断	名前	busy
	意味	(OR (NOT 加入者用作業領域を確保できた) (NOT トランクを確保できた))

図3.2 G-STD要素定義の例

3.3 推論アルゴリズム

推論部の基本アルゴリズムは比較的単純である．

(1) required-effectとマッチするeffectを持つD-STD要素をD-STD内部データベースから検索する．

(2) 発見したD-STD要素のconditionを評価し，その結果が真であればこれを選び，挿入する．

(3) 選んだD-STD要素にpre-required-effectがあれば，挿入した位置より上方で既にこれが達成されているかチェックする．達成されていなければ，それを新たなrequired-effectとして再帰的に動作する．

(4) 選んだD-STD要素にpost-required-effectがあれば，挿入した位置より下方で既にこれが達成されているかチェックする．達成されていなければ，それを新たなrequired-effectとして再帰的に動作する．

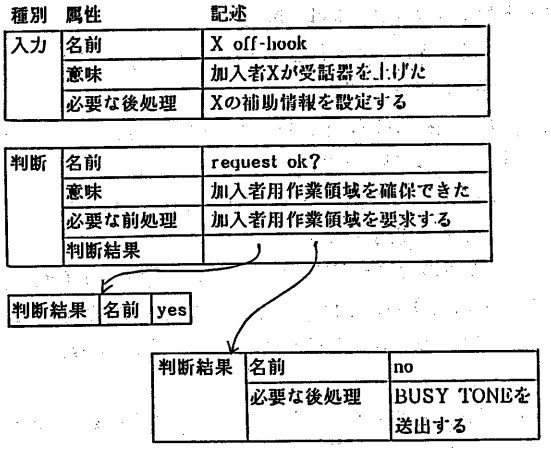

図3.3 D-STD要素定義の例

(5) 適当なD-STD要素が選べなければfailし，バックトラッキングを起こす．
ただし，Decisionの場合は，適当なD-STD要素を選べなかった時に，

(6) required-effect が(OR . . .), (AND . . .), (NOT . . .)のいずれかであれば，それに応じて条件を分解・反転し，それらを新たなrequired-effectとして再帰的に動作する．

図3.4は本サブシステムによる変換の例である．

3.5 タスク生成サブシステムのまとめ

タスク生成サブシステムは一種の自動プログラミングシステムである．一般の自動プログラミングにおける最大の問題点は「制御」を作り出さねばならないことであろう．本サブシステムでは，一つには扱う対象が状態遷移図という限られた制御しか持たないものであるために，もう一つには入力であるG-STDが十分な制御情報を含んでいるために，この点をさほど問題にせずに済んでいる．

現在，本サブシステムを実用化するにあたり，知識の収集を進めている．明らかに，このシステムの能力は格納されている知識の量と質に大きく依存する．したがって，知識ベースの作成，更新，一貫性のチェックなどの支援機能を持った「知識エディタ」を開発することが重要であろう．

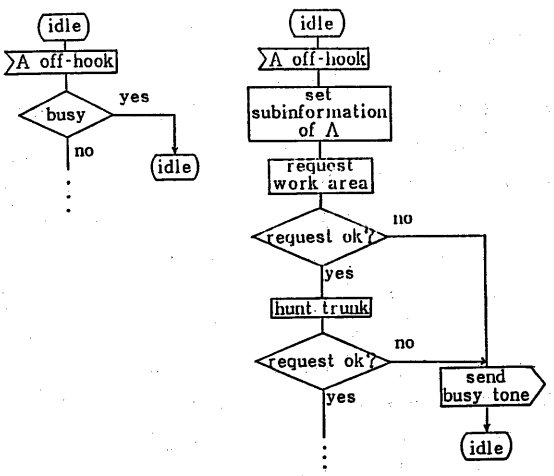

図3.4 G-STD→D-STD変換の例
(発呼処理の一部)

4. コード生成サブシステム

4.1 コード生成サブシステムの構成

コード生成サブシステムは，図4.1に示すように，

(1) SDL／PRにより記述された機能仕様を内部表現に変換するパーサ，

(2) コード生成に用いる規則(以後，生成規則と呼ぶ)を蓄えた知識ベース，

(3) 知識ベース中の生成規則を解釈・実行しながらコードを生成するインタプリタ，

(4) 生成されたコードを最適化するオプティマイザ，

から構成されており，SDLで記述された詳細機能仕様を入力し，プログラムコードを出力する．また，知識ベースには，

(1) SDLと生成するコードとの対応関係(生成規則)，

(2) 入力される仕様記述に関係する付随的な知識(主にデータ構造に関する知識)，

がルールの形で格納されており，対象言語の変更や機能の拡張を容易にしている．

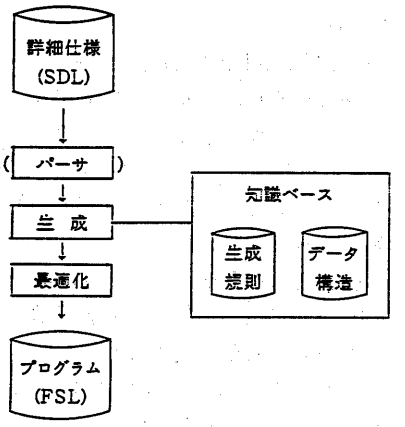

図4.1 コード生成サブシステムの構成

本サブシステムでは，入力のD-STD中の記述を通常のプログラミング言語と同程度に制限している．これは本サブシステムの実用性を重視したためで，より自由な(自然言語に近い)表現の受理，あるいはG-STDからD-STDへの変換のような高度な知識と推論を要する処理は，「タスク生成サブシステム」で行う．

生成される言語は，ほとんど知識ベース中の規則にのみ依存し，CHILLなど種々の言語が可能だが，今回はFSL(Fujitsu System Language)と呼ばれる通信ソフトウェア記述用の言語について述べる．

4.2 生成方法

SDL(G-STD)による仕様記述は，State・Nextstate, Input, Output, Task, Decision などのシンボルから成り，FSLのプログラムはCASE・COND, IF・THEN・ELSE, GOTO, タスクマクロ(アセンブラのマクロと同様のもの)，転送命令などの低レベルの命令から構成される．

表4.1にこれらの対応関係の一例を示す．SDL中に現われる記述のかなりの部分はFSLのコードと直接的に対応しており，通常のコンパイラと同様の手法で翻訳可能であるが，次のような問題点がある．

(1) 文脈依存性

表4.1の例7の"'Event OK?'"という記述は，直前の要求に対する返答がOKだったかを調べるもので，実際に値を調べなければならない変数は直前の処理によって変わる．したがって，文脈に依存した解釈をすることが必要である．

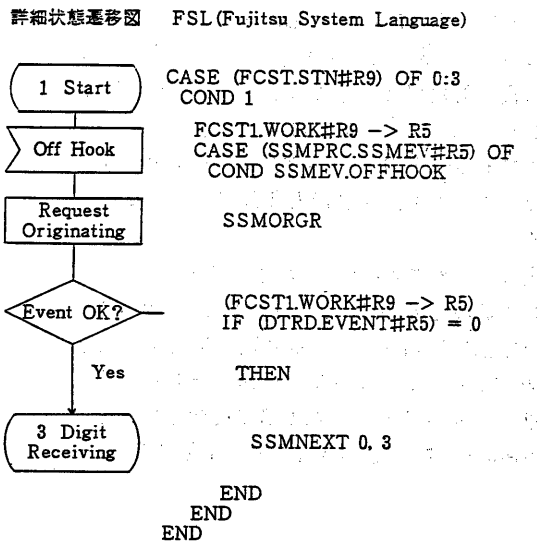

図4.2 D-STDとFSLの対応

(2) アクセス・パスの省略

システムテーブルのような複雑な構造へアクセスする時，SDLによる記述ではそのアクセス経路は明示されない．これを自動的に補わなければならない．

(3) 最適化

電子交換機では記憶領域，処理時間などの制約が厳しいため，生成するコードの最適化が必須である．例えば，以後の処理でよく使われる変数の値をレジスタに常駐させ，冗長な命令を省くような処理が要求される．

表4.1 SDLとFSLの対応例

番号	SDL／PRによる記述	対応するFSLコード	備　　考
1	State 1 'Start';	CASE (状態番号) OF ... COND 1	状態番号による分岐
2	Input 'Off-hook from LC';	CASE (入力信号) OF ... COND SSMEV.OFFHOOK	入力信号の値による分岐
3	Output 'Stop receiving digit';	SSMSEVT 1 SPRCVDGT	信号発信用タスクマクロ
4	Task 'Set time of "ORG"';	SSMTCLT ORG	時刻情報収集用タスクマクロ
5	Nextstate 1 'Start';	SSMEND 0	状態遷移・終結用タスクマクロ
6	Join ラベル;	GOTO ラベル	無条件分岐
7	Decision 'Event OK?'; ('Yes (Event=0)'): ... ;	ある変数のあるフィールド -> レジスタ IF (あるフィールド)=0 THEN ...	必要な変数の値をフェッチする命令群 条件判定 分岐

図4.2にD-STDとFSLの対応の実例を示す．FSLの命令について説明すると，最初の2行はCASE文とCASEラベルで，"ベースレジスタR9で指される，FCSTという名前の構造体のSTNフィールドの値が1ならば . . . "という意味である．"->"は転送命令を表わしており，左辺を右辺に転送する．6行目のSSMORGはタスクマクロの一つである．7行目の転送命令が括弧に入れてあるのは，最適化により，この命令は実際には生成されないことを示している．

SDL		FSL	
種類	名前	種類	オペランド
Input	Off-hook from LC	COND	SSMEV.OFFHOOK
Task	Set time of *X	マクロ	SSMTCLT *X
Decision	CTA	CASE	(CTA)
Decision	Event OK?	Decision	Event = 0
Decision	*Var = *Val	IF	(*Var) = *Val
Result	Yes	THEN	
Nextstate	1 Start	マクロ	SSMEND 0
Nextstate	*Ident	マクロ	SSMNEXT 0,*Ident

図4.3 生成規則の例

図4.3は生成規則の例である．図中の*印はルール中の変数を示し，変数はPrologと同様にunifyされる．

コード生成部全体は前向き連鎖で動作している．

5. プロトタイプ

以上述べてきたシステムのプロトタイプをFACOM M-180II上でUtiLisp[6]を用いてインプリメントした．表5.1にその結果を示す．プログラムの行数は，グラフィック・ルーチン等を含んだ総計であり，UtiLispのプリティ・プリンタで一行80文字とした場合の行数である．ルール数は，タスク生成ではG-STD用のルールが5個，D-STD用のルールが36個，コード生成では生成規則が145個，データ構造の規則が23個である．

表5.1 プロトタイプ

◇言語	UtiLisp (OSIV/F4)		
◇規模	プログラム	総計	25,000行
	ルール	タスク生成	5+36個
		コード生成	145+23個
◇実行例	(呼処理の一部)		
	概要状態遷移図	SDL/GR	15シンボル
	詳細状態遷移図	SDL/GR	62シンボル
	プログラム	FSL	200行
	CPU時間	M-180II	60+2秒

SDL/GRで表わして15シンボルのG-STDをタスク生成サブシステムに入力し，生成されたD-STDのシンボル数が62個，生成に要したCPU時間が60秒であった．このD-STDをコード生成サブシステムに入力した場合，生成されたコードがFSLで200行，所要CPU時間は2秒であった．

6. 結論

機能仕様記述言語SDLに基づき，知識ベースを用いた，通信ソフトウェア開発支援システム(SDL支援システム)の概要と試作したプロトタイプについて述べた．知識の導入で，より知的な支援環境が実現できたと考えている．

今後の課題としては，実用性の評価，呼処理以外の処理への適用，知識ベースの充実，知識の(半)自動獲得などがある．

最後に，研究の機会を与えて下さいました通信ソフトウェア管理部の中村部長と藤本課長，日頃御指導頂くソフトウェア研究部の林部長と上原部長代理に感謝致します．

参考文献

[1] CCITT: Recommendations Z.101-Z.104, "Functional Specification and Description Language(SDL)", 1981.

[2] 鈴木他: "SDL(通信ソフトウェア機能仕様記述言語)支援システム", 情報処理学会第28回全国大会, 1984.

[3] 加藤他: "知識ベースを用いたSDL支援システム", 昭和59年度電子通信学会総合全国大会, 1984.

[4] 富士通: "FACOM OSIV/F4 GEM1/GEM2使用手引書", 1979.

[5] 金谷他: "電子交換プログラム記述高度化の一手法について", 信学技法SE78-68, 1978.

[6] TAKASHI CHIKAYAMA: TECHNICAL REPORTS, "UtiLisp Manual", METR 81-6, University of Tokyo, 1981.