解　説

Adaの適用例（その1）

DIPS プロジェクトにおける Ada の開発適用状況†

福山峻一††　吉岡　堯††
奥瀬尚位††　藤丸政人††

1. はじめに

NTTでは，DIPS* 上に Ada システム（以下 DIPS Ada と呼ぶ）の開発を進めている．59年春にサブセットシステムの開発を完了し，さらにフルセットシステムを開発中である．サブセットシステムは，順次 DIPS の各種基本ソフトウェアの開発に適用を進めてきている．国内はもとより，国際的にみても前例の少ない中での実用レベルの開発及び適用であり試行錯誤の繰返しであるが，各種の新しい経験を積みつつある．

本稿では，まず DIPS Ada の開発内容について概要[3),4)]を紹介する．次に，DIPS プロジェクトの中で試みられた Ada を有効利用するための手法とその反省点を次の観点から紹介する．

① Ada 言語機能の有効利用
② Ada 支援ツール群の有効利用
③ Ada システムを効果的にするための環境条件

また，途中経過であるが，Ada 支援ツールの開発における Ada の導入効果を，従来言語（SYSL**）による場合と比較して紹介する．

2. DIPS Ada の概要

2.1 言語機能

DIPS Ada で実現されている言語機能セットは，DIPS 基本ソフトウェアへの早期適用を図るため，ANSI 標準仕様[5)]を表-1 に示すようにサブセット化した．

† Outline of DIPS Ada System and its Usage by Shunichi FUKUYAMA, Takashi YOSHIOKA, Naoi OKUSE and Masato FUJIMARU (NTT Software Production Technology Laboratories).
†† NTT ソフトウェア生産技術研究所
* DIPS: Denden Information Processing System の略で NTT の標準計算機システムの総称[1)].
** SYSL: SYStem description Language の略で，PL/I をベースにした DIPS 用のシステム記述言語[2)].
注）Ada は米国政府，AJPO の登録商標である．

2.2 コンパイラ

2.2.1 コンパイラの構成

DIPS Adaコンパイラのフェーズ構成を図-1 に示す．すなわち，フェーズを機種独立で他機種へ移植可能なフロントエンドⅠ・Ⅱと機種依存のバックエンドに分割している．バックエンドを機種ごとに置換えることにより，各機種に合った目的プログラムの生成を可能としている．

コンパイラと支援ツールが共通に参照する中間言語として，標準化への検討が進められている木構造形式の高水準中間言語 DIANA[6)] (Descriptive Intermediate Attributed Notation for Ada) を採用した．また，DIANAの下に機種独立で機械語生成が容易な4つ組形式[7)]の低水準中間言語を設定した．

2.2.2 最適化技法

基本ソフトから応用ソフトまでの広範囲なソフトウェアの記述を可能とするためには，コンパイラが高性能な目的プログラムを生成することが必須となる．このため DIPS Ada ではフロー解析[8)]による最適化のほか，外部手続きに関するレジスタ割付け，プロロー

表1 DIPS Ada 言語仕様

言語機能		DIPS Ada
基本機能	データ型	○
	式・文	○
	サブプログラム	○
	パッケージ	○
	タスク	×
	分離コンパイル	○
	例外処理	○
	ジェネリック	×
	内部表現	○
	入出力	○
機械語		○
レジスタコンベンション記述など		○(充実)

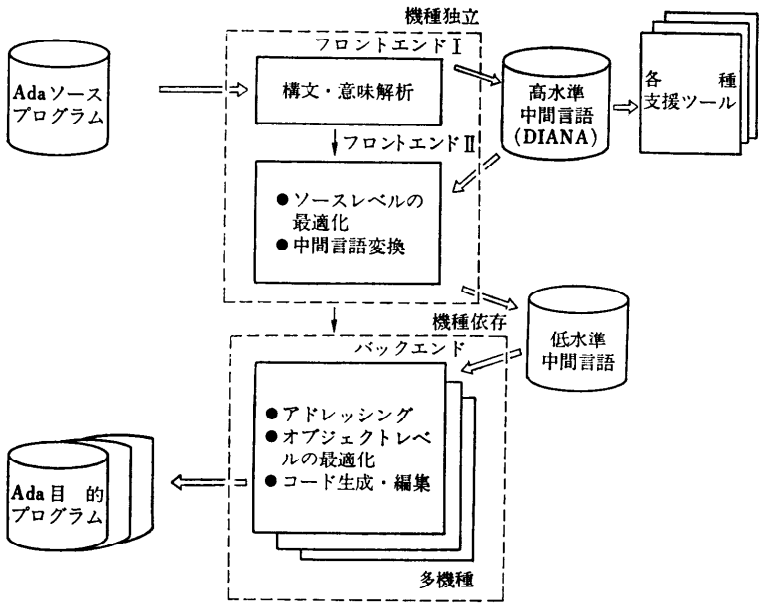


図-1 コンパイラのフェーズ構成

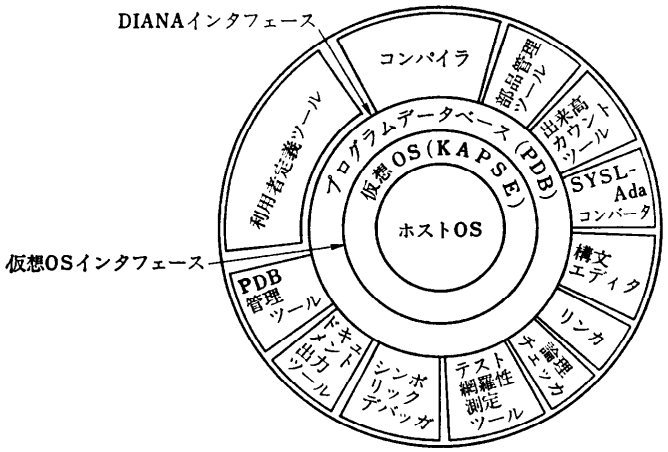


図-2 DIPS Ada システム構成

グ・エピローグ処理の効率化等の最適化処理を実現している．その結果，目的プログラムの性能は，SYSL による場合と比較して同等以上になっている．

2.3 支援システム

2.3.1 支援システムの構成

支援システムの構成を図-2 に示す．構成上の特徴を以下に述べる．

（1） 仮想 OS インタフェースの実現

コンパイラや支援ツールを各種 OS 環境へ移植容易とするため，Ada 支援環境に対する要求仕様 (STONEMAN[9]) では仮想 OS インタフェース (KAPSE: Kernel Ada Programming Support Environment) の概念が導入されている．DIPS Ada システムでは，この概念を Ada パッケージ機能を用いて OS マクロの機種依存パラメータやコンパイラ，支援ツールでは使用しないパラメータの隠蔽を行うことにより実現している．

（2） プログラムデータベースの実現

コンパイラがソースプログラムを解析して生成した中間情報 (DIANA) をプログラムデータベース (PDB: Program Data Base) として保存している[10]．この PDB の中間情報を各支援ツールでも共通的に使用することにより，一貫した開発支援及び効率的なシステム構成を可能としている．

2.3.2 支援機能

支援機能としては，プログラムバグの早期検出，机上レビュ，テスト作業等の確実化・効率化をねらいとして，表-2 に示すように設計から保守までのソフトウェアライフサイクル全体を支援できる機能[11),12)]を提供している．

これら支援機能の実現において，プログラム誤りの早期検出のためデータフロー解析，データ値域解析などの解析技術[13)]を有効利用するとともに，ソースプログラム情報の解析時間の短縮のため PDB の中間情報を有効利用している．

3. Ada 適用上の課題 と提案

DIPS プロジェクトでは従来の SYSL による開発から Ada による開発に切替えつつある．DIPS の基本ソフトウェアのうち，Ada を始めとする言語処理系，OS, データベース管理システム，各種通信処理パッケージ，知識処理言語等約 300 K ステップに Ada を適用中である．本章では，これらの適用過程で当面し，解決の試みられている Ada システム有効利用の

表-2 DIPS Ada 支援機能の概要

支援ツール	主な機能
部品管理ツール	プログラム部品の登録検索，部品仕様書作成
構文エディタ	構文テンプレート機能（例えば if A then B else C end if では A, B, C のみ入力）および構文チェック機能をもつスクリーンエディタ
論理チェッカ	変数の設定・参照の誤り（初期値設定抜け等），制御移行の誤り（実行不能文等）および変数の値域の範囲誤り（添字範囲オーバ等）の検出
テスト網羅性測定ツール	テストケース（プログラム・パス）の抽出，テスト網羅率測定
シンボリックデバッガ	変数，プログラムの流れのトレース，データの設定 処理の流れの画面への表示，バックトレース機能
ドキュメント出力ツール	モジュール／テーブル関連図，コンパイル順序関係，HCP チャートの出力
出来高カウントツール	プログラム出来高，特性の測定
SYSL-Ada コンバータ	SYSL-Ada ソースプログラム変換
PDB 管理ツール	PDB の内容の編集，表示機能

ための課題について，言語機能，支援機能，及びシステム利用環境の3つの観点から紹介する．

3.1 言語機能の適用上の課題と提案

3.1.1 設計法

Ada 記述を前提に設計段階で実施した事項を下記する．

（1） Ada 機能へのマッピング方法

Ada の特徴的プログラム構造であるパッケージ機能を用いることにより詳細情報の隠蔽，あるいは共通インタフェース情報の参照と管理の容易化等が期待できる．このため，詳細設計に先立ち，機能設計工程の後段で次の3種類の情報をパッケージ化対象として抽出することを行った．すなわち，

① 翻訳単位間で共用されるデータあるいはデータ型の集合

② 共通サブプログラム群，すなわち，手続き及び関数にする対象

③ 抽象データ型

ただし，抽出したこれらパッケージ化対象を，機能階層や使用契機の違い等に着目して再分類しパッケージ化しないと翻訳単位の規模が巨大となり，翻訳時間が増大するとか翻訳不能になる恐れがあり注意を要する．

分類	用途	記号	意味	対応する Ada 文
処理	一般の処理	×××（○）	通常の処理	代入文，コード文，空文等
		×××（◎）	繰返し処理	loop 文（loop / for……loop / while…loop）
		××× ( )( )	振分け処理 “( )”内はケース条件の記述	case 文 if 文
		( )	上側の図における右向き矢印と下側の図における下向き矢印は「その他」の場合を表わす	
		×××（⊗）	エラーチェック	コンパイラによりオブジェクト中に自動生成される範囲制約の検査等を除き，処理アルゴリズムとして陽に規定される誤り判定
	モジュール化	×××（◎）	サブプログラム呼出し	手続き，関数の呼出し
		×××（⊛）	マクロ呼出し	パッケージ内手続きとして宣言されたOSマクロの呼出し
		×××（⊖）	その他のモジュール化	ブロック文，あるいは別ページに詳細化される処理

分類	用途	記号	意味	対応する Ada 文
制御	条件付け	◇ ◇	判断（矢印で yes の向きを示す）．記号の右側に判断方法を記述する	if 文
		◇ ( )	例外条件（データ並びの終りなど）．“( )”内に条件記述	if 文
	開始	┬	ページ内最上位レベルの開始	パッケージ，手続き，関数およびブロック文の処理開始（begin）に対応
		○	実現方法の開始	階層分けされた一連の処理のまとまり．
	終了	┴	ページ内最上位レベルまたは実現方法の終了	exit 文，end，end loop，goto
		▽ ▽ ▽	三角形の個数分または数字で示された分だけ上のレベルの処理を終える	exit 文（脱出先ループ名指定）
		┴	ページ内最上位レベルの処理を終える	return 文
		(×××)	エラー出口（例外的な制御が行われることを示す）	raise 文

（凡例）：“×××”は処理説明　　図-3 HCP チャート記法と Ada 文との対応付け

	Ada言語機能	記 法	記述上の留意事項
1	パッケージ仕様部	×××(パッケージ名) ◎×××(サブプログラム名) ○×××(パッケージ名) ◎×××(サブプログラム名)	1つのパッケージに包含される手続き，関数，パッケージを列挙する． ◎：手続き，関数とする処理 ○：パッケージ化する処理 ×××：処理内容
2	パッケージ本体部	×××(サブプログラム，パッケージ名) (実行部)	パッケージ内に含まれる手続き，関数1パッケージの実現詳細 ×××：処理内容
3	例外ハンドラ(縦型も用いる)	実行部終了 EXCEPTION ( )×××( )×××( )×××	( )内は，例外条件名を ×××：例外処理内容

図-4 Ada 固有のものとして規定した HCP チャート記法

(2) HCP チャートの適用

NTT では，近年設計図法としては HCP チャート[14]を用いることが一般的となっており，Ada の適用を前提とした場合も HCP チャートを用いることとしている．HCP チャート記法と Ada 言語機能との対応付けは**図-3** に示す形で一般的に行っている．この対応付けを実施する上で，次の2点は**図-4** に示す Ada 固有の記法を新たに規定した．

① パッケージとする範囲の表記方法

② 複数の例外発生に対応する後処理の表記方法

また，次の点については新しい表記法を HCP チャートに導入することを検討している．

③ Ada の厳密にして豊富なデータ型の表記方法

④ タスクとする範囲の並列処理の表記方法

3.1.2 コーディング要領

Ada の言語機能を有効に使いこなすためと，多人数で分担開発する場合のプログラムの仕上り品質の均質化及び読解性の向上をねらいとして，次のような点はコーディング作業要領として定め，プログラマに周知している．

① データと型などの区別を容易にする名前の付与方法．

② パッケージ，手続き，関数をコーディングする場合のソースプログラムのレイアウト方法(**図-5**)．

③ 性能のよい目的プログラムを得るための効果的なコーディング方法．

これらの規定内容は，今後も Ada 使用経験を反映し充実していく．

3.1.3 効率的な Ada 翻訳について

Ada の翻訳では，次のような場合に再翻訳回数の増加が問題となった．

(1) 仕様が確定した後は，修正は本体部に集中する．このとき，仕様部と本体部とを分離せずに同一翻訳単位にしていると，仕様部も同時に再翻訳されるため，これを with で参照している他の翻訳単位の再翻訳が必要になる．

(2) **図-6** のケースで，プログラム A の仕様部が修正となったとき，これを参照しているプログラム B の仕様部に with A 記述をしているプログラム C の再翻訳も必要となり，余分な再翻訳作業が必要となる．

(3) 翻訳順序関係を正しく把握して翻訳作業を進めないと，順序誤りによる翻訳作業のやり直しが必要となることがある．

このような翻訳回数の増加を防止する方法を以下に紹介する．

(1) 仕様部と本体部を分離し，別翻訳単位とする．このように分離すれば当該本体部の再翻訳のみですむため，翻訳時間並びに再翻訳回数が削減できる．

(2) with 記述は可能なかぎり本体部で実施する．すなわち，図-6 において，プログラム B の with 記述をその本体部で行えば，プログラム A の仕様部修正に伴なう再翻訳は，プログラム B の本体部だけでよいことになる．

(3) 翻訳順序出力ツール (DIPS Ada の場合では **2.3** 節で述べたドキュメント出力ツール) を用いて翻訳順序を正しく把握し，無駄な再翻訳作業が入らないようにする．

(1) パッケージ仕様部レイアウト

package パッケージ名 is

— — — —	パッケージ全体を説明するコメント
宣言部	型，部分型の宣言 * オブジェクトの宣言
	例外条件の宣言 *
	パッケージの宣言 *
	サブ プログラム宣言 * 手続き 関 数
	プライベート部 *

end パッケージ名;

(2) パッケージ本体部レイアウト

package body パッケージ名 is

— — — —	パッケージ全体を説明するコメント
宣言部	型，部分型の宣言 * オブジェクトの宣言
本体定義部	パッケージ本体 *
	サブ プログラム本体 * 手続き 関 数
	〃
	〃 以下，続く

begin

文の列 *

exception

例外条件 実行文 *

end パッケージ名;

注) * 印は，必須部分ではなく必要なもののみを記述のこと．

図-5 ソースプログラムのレイアウト規定 (パッケージの場合)

3.2 支援系の有効利用法

生産性及び信頼性の向上を図るためには各支援機能を有効に利用していくことが必要である．DIPS Ada では図-7 に示すように各工程で有効な支援機能を用意している．プログラム開発の各工程において以下のような形で支援機能を利用することとしている．

(1) 設 計 段 階

部品管理機能により，既存のプログラム（部品）の中から再利用可能なものを取り込み，ソフトウェア生産を効率化する．

(2) 製 造 段 階

構文エディット機能により，入力の省力化を図るとともに構文の誤りを摘出する．また，論理チェック機能により，コンパイラでは検出困難な論理的な誤りの一部を早期摘出する．これら誤りの早期検出により，生産性向上を図る．

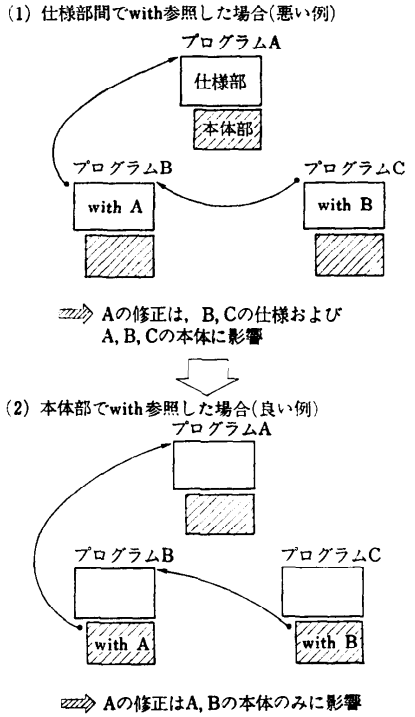


図-6 WITH 参照の良い例・悪い例

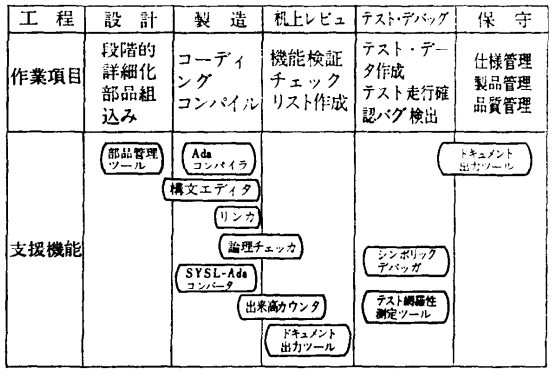


図-7 DIPS Ada 支援環境と適用工程

（3） 机上レビュ段階

ドキュメント出力機能を利用し，手書き設計書と自動生成ドキュメントとの突合わせチェックを行うことにより，仕様が正しくプログラム（機能）として実現されているかを確認する．

（4） テスト段階

プログラムパスの出力機能により，テストデータ作成のための内部仕様に基づくチェック項目抽出時のチェック項目洩れを防止する．また，テスト網羅性測定機能により，テストの進捗状況の把握及び機能確認が完了した段階でテスト洩れを防止する．

（5） デバッグ段階

シンボリックデバッグ機能により，Ada ソースプログラムレベルでのデバッグを行い，デバッグ作業を効率化する．

（6） 保 守 段 階

ドキュメント出力機能により，プログラムの最新内容に合致したプログラム設計書を得て，保守を効率化する．

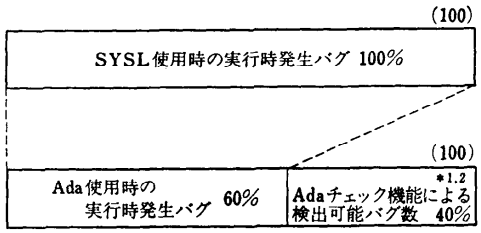
3.3 使用環境の整備

（1） 各種標準パッケージの提供

ソフトウェアの生産性向上を図るためには，当該適用分野で必要とする共通処理を標準パッケージとして提供する必要がある．DIPS Ada では次のようなパッケージを用意している．

① 基本ソフトウェア用として：システムコール用パッケージ

② アプリケーションソフトウェア用として：事務処理編集用，図形処理用，日本語データ処理用などのパッケージ，及び数学関数ライブラリ



*1 コンパイル時検出バグ
① 制御表指示誤り
② 変数使用誤り
③ パラメータ，アーギュメント対応誤り等

*2 論理チェッカ検出バグ
① 初期値設定抜け
② 値域範囲逸脱
③ 無限ループ
④ ポインタ値誤り等

図-8 信頼性向上効果

（2） 他言語記述プログラムとの結合用支援ツール

一般的な Ada の適用ケースとして，すでにかなりの量の他言語で書かれたプログラムが存在している環境に Ada を適用し，でき上がったプログラムを相互に結合して利用するというケースが考えられる．このため DIPS Ada の場合，次のような支援ツールを開発中である．

① 他言語記述プログラムを Ada 記述へ変換するコンバータ（記述言語の一本化）．

② 他言語記述プログラムを結合した状態でのデバグを支援するシンボリックデバッガ．

4. 信頼性・生産性向上効果

ソフトウェア開発支援ツールに適用した場合の一例であるが，Ada を適用した場合の信頼性，生産性の

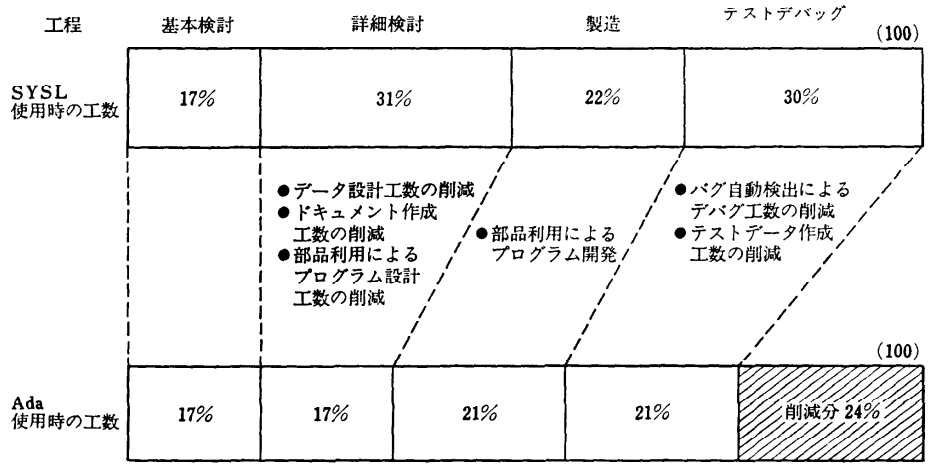


図-9 生産性向上効果

向上効果を SYSL 適用時と比較して紹介する.

信頼性については，Ada コンパイラ及び支援システムにより，プログラム実行テスト以前に約40%のバグが検出可能となることを確認した(図-8).

生産性については，約24%削減できることを確認した（図-9）．この効果は，Ada の言語機能自体による信頼性，生産性向上の効果に加えて，各種支援機能の有効利用により，ドキュメント作成工数の削減，データ設計工数の削減，テスト工数の削減及び早期バグ検出にともなう工数削減などによるものである.

引続き，信頼性，生産性に関するデータを収集し，Ada の適用効果を分析していく.

5. おわりに

DIPS プロジェクトにおける Ada システムの開発概要と，それを実用レベルで使いこなす上での課題について，これまでの使用経験から得られた事項を紹介した．今後 Ada の導入を試みようとしている方々にとって幾分なりとも参考になれば幸いである．経験から効果的と考える事項を以下に要約する.

① 設計の早期段階におけるパッケージ化対象の抽出

② HCP チャート等構造化設計容易な設計図法の利用

③ コンパイル回数削減のためのノウハウの作業要領化

④ 支援ツールの整備とその有効利用方法

⑤ 共用の容易化と破壊防止を考慮した PDB の運用管理方法

今後．Ada を用いて生産性を上げ，記述ソフトの流通性を確保するためには，さらに Ada を使いこなすためのノウハウの蓄積と整理を行い技術を確立していくことが必要である．このための情報交換が学会等の場で一層活発化することを期待したい.

最後に，DIPS Ada の開発を推進され，本稿の執筆に際して適切なご助言をいただいたソフトウェア生産技術研究所プログラム言語研究室長細谷偉一氏に感謝の意を表します.

参考文献

1) 新井，高村：データ通信開発の現状と課題，通研実報，Vol. 33, No. 12, (1984).
2) 細谷他：システム製造用語 SYSL，通研実報，24, No. 1, p. 95 (1975).
3) 細谷，田中，藤丸，山口：Ada コンパイラおよび支援システムの実用化，通研実報，Vol. 33, No. 12 (1984).
4) 細谷，田中，藤丸，山口：Ada 処理系の実現方法，ソフトウェア工学研究会，39-4 (1984).
5) American National Standard Institute: Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983(1983).
6) Tartan Laboratories Incorporated: Draft Revised DIANA Reference Manual, Revision 2.1 (1982).
7) David Gries: Compiler Construction for Digital Computers, 11. 3(1978).
8) Hecht M. S.: Flow Analysis of Computer Programs (1977).
9) U. S. Department of Defence: Requirements for Ada Programming Supports Environments. "STONEMAN" (1980).
10) 田中，藤丸，山口他：Ada 言語システム共通中間言語と効率的実現法，情報処理学会全国大会論文集（57年後期）p. 359 (1982).
11) 田中，藤丸，山口他：中間言語に基づく Ada ドキュメント自動生成法，情報処理学会全国大会論文集（59年前期），p. 541 (1983).
12) 田中，藤丸，山口他：Ada コンパイラと論理検証ツールの最適構成について，信学会部門全大(58年度)，p. 514 (1983).
13) 細谷，堀田他：設定参照グラフを用いたプログラム誤り自動検出法，ソフトウェア工学研究会，23-2 (1982).
14) 花田収悦：プログラム設計図法，企画センター (1983).

(昭和61年1月20日受付)