解　説

3. プログラム設計環境のツール

3.1 形式的仕様記述言語とその支援ツール†

深　澤　良　彰††

1. はじめに

ソフトウェアの生産性・信頼性を向上させるために，ウォータフォール・モデル，プロトタイピング・モデルなど各種のソフトウェア開発モデルが提案されている[1]．しかし，いずれのモデルにおいても，ソフトウェア開発過程のできるだけ早期に，実現したいシステムのイメージを明確化したいということにおいては共通している．そのために，種々の内容・表現形式をした仕様書が作成される．ソフトウェアの要求分析の結果をまとめた要求仕様書，実現すべきソフトウェアシステムの外部仕様を定義する機能仕様書または外部仕様書，達成すべき性能や信頼性の目標を記述した性能仕様書などがウォータフォール・モデルにおける例である．しかし，良い仕様とはなにか，どのようにしたら良い仕様が得られるか，それはどのように表現されるべきかなど多くの問題点を抱えている[2]．

これに対して数多くの研究・実用化が行われ，多岐にわたる方法論，記述形式などが提案されている．そこで用いられている『仕様』の定義もさまざまであるが，本稿においては，できるだけ幅広く研究・開発にふれるために，『仕様記述とは，ソフトウェアのなすべきことすなわち機能を正確に定義すること』という広い定義をすることにする．

仕様記述言語はその形式性から以下の3種類に分けることができる[3]．

◎ 日本語，英語などの自然言語や図などを用いた仕様の非形式的記述．日常使用している言語による表現であるので，理解しやすく，かつ効率的な記述が可能である．一方，振舞いの詳細の正確な記述は困難である．現状のソフトウェアの生産現場では，広く使用されている．

◎ 仕様記述の一部に形式を規定するが，それ以外は非形式的な記述を行う半形式的記述．

◎ 数学的に厳密な議論に耐え得るように，構文と意味が明確に定まった言語による形式的記述．このために，述語論理や代数系を利用する．

形式的仕様は，非形式的仕様に対して，次のような利点をもっている[2)-4)]．

◎ 形式的仕様言語はなんらかの数学的基盤をもっている．これを利用して，プログラムがその仕様を満たしていることを証明できるし，二つの仕様の同等性も証明できる．仕様の矛盾や不完全性も自動的に検出できる．このような検証をソフトウェア開発のできるだけ早い段階から行うことにより，エラーの修復費用を減らすことができる．

◎ 形式的仕様言語は十分に定義された，曖昧さのない意味をもつので，情報伝達の手段として有用である．

◎ 形式的な仕様記述から，実行可能なコードを自動的に生成することが可能である．

形式的仕様記述はこのような長所をもつ．しかし，一方では，記述対象のモデル化が難しい，形式的な記述に習熟するにはかなりの訓練と時間を必要とするなどの短所をもつ[5]．そこで，仕様の作成を容易にし，各種のエラーに対処するための，環境・ツールが必要になる．

各ツールが支援する局面としては，以下のようなものが考えられる．

① 仕様を作成するまでの過程の支援（**3.4**，**3.5** 参照）

② 仕様の吟味：仕様の無矛盾性，完全性などを調べることを支援する（**3.1**，**3.2** 参照）

③ 仕様のユーザへの提示：仕様がユーザの要求を満たしていることを確認することを支援する（**3.1** 参照）

④ プログラムが仕様を満たすことの検証：仕様に

† Formal Specification Languages and Their Support Tools by Yoshiaki FUKAZAWA (Department of Electrical Engineering, School of Science and Engineering, Waseda University).

†† 早稲田大学理工学部電気工学科

基づいて設計，実現されたプログラムが仕様を満たすことを示すことの検証の支援を行う（**3.2**，**3.3** 参照）

2. 形式的仕様記述言語

仕様記述言語は種々の見地から分類が試みられている．前述の形式性による分類もその一つである．形式的仕様記述言語について詳述することは，本稿の主眼でなく，かつ，仕様記述言語の発展にともない，明確な分類が行いにくくもなってきている．そこで，以下のように分類し，その概要について述べるにとどめる．

2.1 操作的仕様記述言語

意味の明らかなプログラミング言語や関数や集合などの数学的概念やオートマトンなどの抽象機械によって，プログラムや抽象データ型の作成法を与える方法である．このようなアプローチはモデル指向と呼ばれることもある．このアプローチの強みは，モデルの存在により仕様の一貫性が保証され，しかも，そのモデルはインプリメンタに対する有用なヒントとなる．

提唱されているモデルには，状態機械モデル[6)~8)]，抽象モデル[9)~11)]，ペトリネット・モデル[12)]などがある．また，このような概念にたって設計された仕様記述言語には，Special[13)]，Z[14)]，Ina Jo[7)]，Gypsy[15)]，PAISLey[16)]などがある．

2.2 定義的仕様記述言語

プログラムや抽象データ型に要求される性質を列挙することにより，その記述を行おうとする考え方である．このアプローチでは，等式論理などでその記述対象の性質を定義し，プログラムや抽象データ型をどのように作成するかについては記述しない．定義的なアプローチは，大きく，代数的アプローチと，それ以外のアプローチに分類できる．

（1） 代数的アプローチ

Ada，Modula-2 などの比較的新しいプログラミング言語にみられる抽象データ型は，モジュール化，プログラムの再利用などの面から，信頼性の高いプログラムを効率的に作成するための有力な手段であると考えられている．ここで，データ型を代数であると考えて，抽象データ型を等式の集合として表現することができる．このようにして表現されたものを代数的仕様記述法[17)~19)]と呼ぶ．具体的には，データ型を多ソート代数であると考えて，その表現として，ソート記号，演算記号，等式を列挙することになる．これに，シンタックス・シュガを加えるとともに，仕様のパラメータ化などの仕組みを導入することにより，より強力で，書きやすく，かつ，読みやすい仕様記述言語を実現する．

代数的仕様記述のための意味論としては，始代数[20)]，終代数[21)]などを用いるものが提案されている．前者は，仕様として記述されている公理から等しいことが導出できない二つの項は異なるという立場である．一方，後者は，二つの項が等しくないと結論できないときに同一であるとする立場である．

代数型仕様記述言語の計算機構のモデルとしては，代数系を構成する関数の集合に対して，操作的な意味を与える項書換えシステム[22),23)]を通常採用する．

代数的仕様記述言語には，OBJ[5)]，CLEAR[24)]，PLUSS[25)]，ACT-ONE[19)]，CIP-L[26)]などがある．また，我が国でも，ASL[27)]，Dimple システム[28)]など盛んな研究・開発が行われている．

（2） 非代数的アプローチ

代数によらない定義的な仕様の起源は，Hoare のプログラムの正当性の証明[29)]とデータ型の実現の正当性の証明[9)]に求めることができる．そこでは，前提条件（pre-condition）と終了条件（post-condition）を示す述語論理式を，プログラムの入出力仕様や抽象データ型の演算仕様として使用している．

述語論理は形式的であり，記述性にも優れている．しかし，論理学においてもっとも基本的な一階の論理式でさえも，一般にはそのままでは実行できない．そこで，なんらかの変換（たとえば，unfold/fold 変換）を行ったり，手続き的な解釈を与えたりして，実行のための機構としている[22),23)]．

3. 支援環境・ツール

3.1 直接実行系

形式的仕様を直接的に実行し，その結果をユーザにみせることにより，作成された仕様がユーザの要求を満たしていることを確認する．また，この提示内容は，仕様に基づいて，設計，プログラミングなどを行うものにとっても，有用である．しかし，実システムの処理速度などについては，提示できないことがほとんどである．

形式的仕様記述言語の中でも，実行可能仕様記述言語と呼ばれるクラスの言語で書かれた仕様に対して，変換を繰り返し，最終的に実行可能なプログラムを生成するシステムを直接実行系と呼ぶことにする．このようなシステムは，プロトタイピングの実現手段の一

つとして注目を浴びてきている．プロトタイピングの意義，実現法などはそれ自身大きな問題であり，詳細な解説は他[30]に譲る．

3.2 検証系

検証可能性は，形式的使用記述の利点の一つであり，これを支援するツールは非常に重要である．このようなツールは検証条件生成系と定理証明系に大別できる．

プログラムの検証において，プログラミング言語の公理的定義と，プログラムの入力と出力との関係の記述を利用し，検証条件と呼ばれる表明（assertion）の列を生成するものを検証条件生成系と呼んでいる．これを仕様に適用することができる．このようなツールには，Stanford Program Verifier[31]，Ford Aerospace Pascal Verifier[32]，Gypsy[15] などがある．この検証条件生成系を分析用ツールと考えることもできる．

書き換え規則を用いている検証条件生成系は，代数的仕様に付け加えられている公理の一貫性を対話的に検証するのに用いられる．この形態をとっているものには，AFFIRM[33]，CIP[26] などがある．

定理証明系は，次の2種類を目的として使用される．

- 仕様から，定理として表現可能なある性質を生成したり，それをチェックしたりする．
- 設計と実現が仕様によって要求されている性質を保っていること，すなわち正当であることを検証する．このとき，仕様の一貫性や完全性をチェックすることもできる．

定理証明については，数多くのアプローチが存在する．そして，それらは，そこに含まれる対話の量と質により変わる．十分に自動化された決定手続きは，もっとも効果的である．しかし，開発プロセス全体の全機械化は証明に要する計算量の点から難しい．他方の極端が，知識工学的手法に基づいた定理証明系である．これらは，発見的手法と複雑な証明を構成するための補助定理の知識ベースを利用する．このようなアプローチには，Oppen の決定手続きと Boyer-Moore の発見的定理証明法を組み合わせて用いている Ford Aerospace Pascal Verifier[32] がある．

ユーザが，それ以後の証明において，利用できるような証明戦略を与えるという，より抽象的なレベルで定理証明系と対話するような研究もある．このアプローチは，たとえば，LCF によって採用されている[34]．

3.3 テスト支援系

形式的仕様とそれに基づいて作成されたプログラムが与えられたとする．このプログラムが仕様を満たすことを数学的に検証できれば，テストは不用となる．この検証を検証プログラムによって行ったとすれば，その検証プログラムの信頼性を考慮する必要がある．また，このような方法は，現状では，小規模のプログラムに対してさえ，多くの資源を必要とする．もし，人手でこの検証を行ったとすれば，その過程でエラーをおかす可能性があるので，テストを必要とする．

テストには，モジュールテスト，統合テスト，機能テストなど種々のレベルがあり，これらは相補ってシステム全体が正しく実現されていることを確かめる[35]．形式的仕様に基づいたテストにおいては，その性質から機能テストを支援する．

テスト支援系としては，直接実行の過程を監視するために，ブレーク・ポイントを設定したり，トレースをしたりする機能を提供するもの[36]，プログラムは仕様から人手で作成するものとし，仕様記述からテストデータなどのテスト環境を提供するもの[37]などが報告されている．

Gannon らは，代数的公理の集合として，システムを記述するシステムについて述べている[38]．そこでは，テストデータを生成し，すべての文と公理のすべての分岐が少なくとも1回は実行されるまで実行を繰り返す．

3.4 仕様ライブラリ

優先順位付きのキューやビットマップの仕様を作成するたびごとに，整数や集合の公理の定義から始めるなど，仕様を記述する際に，常にゼロから記述することは，非常に非効率的である．そこで，一般的な場合には，うまく応用して使用することができ，一般的でない場合にも，モデルとして役に立つような再利用可能な仕様の断片のライブラリが必要となる．

仕様ライブラリは，閉じたライブラリであってはならない．また，数学的な仕様や実現のための仕様のみでは十分ではなく，適用分野を指向したものも必要である[39]．

3.5 構文依存ツール

形式的仕様記述言語の構文情報を基に，各種のチェックを行ったり，仕様を人手でチェックするために入力仕様の変形を行うツールである．たとえば，エディタ（または，構文エディタ），構文チェッカ，清書プログラム，整形プログラム，図的表現プログラム，クロ

スレファレンス作成プログラムなどがこれに属する.

これらのツールは，プログラミング環境における技法をそのまま流用することにより実現される．しかし，実際の大規模プロジェクトで使用される場合には，非常に重要である.

3.6 そ の 他

ヘルプ機能やウィンドウの管理などは使い勝手を良くするために必要である．特に高機能ワークステーションの普及にともなって，これらへの要求は増大してきている．また，OS のモジュールの呼び出しなど OS とのインタフェースをとる機構や，プログラミング言語とのリンクをとる機構などが実現されることもある.

プロジェクトの管理や仕様のバージョン管理のため，システムのロギングをとることができるようになっているツールもある[40].

4. 形式的仕様記述言語とその支援ツール例

仕様記述言語において支援されているツールの一部を付録-1 に示す.

本章では，形式的仕様記述言語とその支援環境・ツールの例として，GTE で開発された形式的仕様記述言語 RTRL (Real Time Requirement Language) とその支援環境[8] について述べる．この言語は，操作的仕様の考え方に従って定義され，拡張された有限状態機械 (Finite State Machine：以下 FMS と略す) に基づいている.

この環境の概略を図-1 に示す．RTRL に従って記述された仕様は，RLP (Requirement Language Processor) と呼ばれるシステムによって，一貫性はあるか，冗長性はないか，完全かなどをチェックされる．システムの動作の形式的モデルが作成されると，テストプラン生成系 (TPG) により機能要求テストシナリオを作成する．そのために，テスト担当者は，TPG と対話的に，状態・刺激・反応・状態間の遷移を含む領域の中から制約を選ぶ．この例を以下に示す.

- システムのパフォーマンス
- ユーザの作業についての最大，最小，継続時間
- トレース対象とするパス
- 特定のループを回る回数

テスト制約が入力されると，TPG は，この制約を満たすようなテストパスを決定する．TPG は，その後，これらのテストパス上でのタイミングの制約や内部のシステム状態のもとで，適用される刺激の集合と検証される反応からなる列であるシナリオを生成する．これが自動テスト実行系 (ATE) への入力となる．シナリオは，テスト記述のための形式的記法であるテストシナリオ言語 (TSL) で記述される．TSL による記述は，ATE により翻訳され，テスト対象システム (SUT) 依存のハーネスによって実行される．SUT は，その実現が機能仕様を満たしていれば，検証されたことになる.

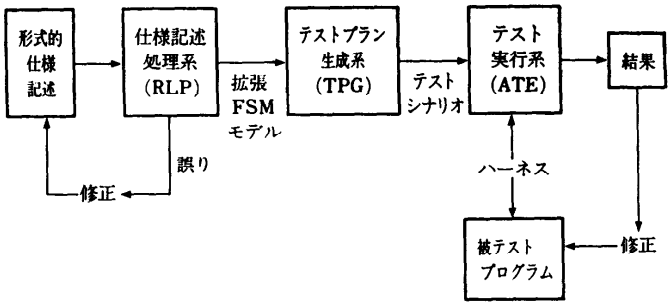

図-1 システム構造図

このシステムは，非常に高い信頼性を要求される実時間適用業務システム（例：GTD-5 EAX システム：Pascal で約 33 万行のシステムで，20 年間に 1 時間以内の停止を目標）において使用された．その結果として，3000 個の遷移をもつシステムにおいて，数十秒のオーダで，全ての遷移をカバーすることができ，十分実用的であることが確認されている.

5. お わ り に

これまで述べてきたように，形式的仕様記述の作成を支援するための種々のツールが開発されてきている．しかし，現状をみる限り以下のような問題点が残されている.

- 環境としての機能強化・統合化：各種の支援ツールを有機的に組み合わせ，ソフトウェア開発の各段階を支援する.
- 大規模システムへの適用：問題の分割方式としては，現在は，段階的詳細化以外の積極的手段が存在しない.
- プログラム・モジュール，仕様，検証結果などの再利用：実際のソフトウェア生産における形式的技法は高価につくので，モジュールを蓄え，仕様にあったものを取り出す知識データベースが要求される.

• 並行処理：並行プログラムの仕様記述方法としては，状態変数アプローチ，イベントアプローチなどが提案されているが，さらに洗練された記述法・ツールが必要であると思われる．

• 性能評価：システムの関数的，定性的行動のみでなく，性能に関する定量的な仕様やその検証も重要である．

• 実用性：対象領域を限定し，その分野の技術者との協力により，効果的な方法，言語，システムを開発する必要がある．

• 教育：既存のソフトウェア技術者にとって，形式的な記述は決して親しみのわくものではない．このような技術者に対して，どのようにして教育していくかは非常に重要な問題である．部分的には，自然言語の積極的活用も採り入れていく必要があろう．

形式的仕様言語とその支援ツールについて，主なものを紹介してきた．しかし，次の分野については，紙面の都合上とりあげることができなかった．興味のある方は参考文献を参照されたい．

• プログラミング言語の構文，意味などを形式的に記述し，そこからエディタ，コンパイラなどを含むプログラミング環境を生成する研究[47]

• 通信のプロトコルなどの記述を対象とする仕様記述言語に関する研究[48]

• 既存の高水準言語の機能をさらに上げることにより，仕様の記述を可能にする研究[49),50]

謝辞 日頃より，形式的仕様記述について数々のご意見を戴いている，早稲田大学情報科学研究教育センター『ソフトウェア開発における仕様記述あるいは検証の実用性に関する研究部会』のメンバの方々に心より感謝致します．

参考文献

1) 大場 充：ソフトウェアの開発技術，SE シリーズ3，オーム社 (1988).
2) Meyer, B.: On Formalism in Specifications, IEEE Software, Vol. 2, No. 1 (1985).
3) Liskov, B. H. and Berzins, V.: An Appraisal of Program Specification, in Research Derections in Software Technology, edited by P. Wegner (1979).
4) 鳥居宏次，二木厚吉，真野芳久：プログラミング方法論の展望，情報処理，Vol. 20, No. 1 (1979).
5) Goguen, J. A. and Tardo, J. J.: An Introduction to OBJ: A Language for Writing and Testing Formal Algebraic Program Specifications, Proc. Conf. on Specifications of Reliable Software, IEEE Comput. Soc. (1979).
6) Parnas, D. L.: A Technique for Software Module Specification with Examples, CACM, Vol. 15, No. 5 (1972).
7) Berry, D. M.: Toward a Formal Basis for the Formal Development Method and Ina Jo Specification Language, IEEE Trans. on SE, Vol. SE-13, No. 2 (1987).
8) Chandrasekharan, M., Dasarathy, B. and Kishimoto, Z.: Requirement-Based Testing of Real-Time Systems: Modeling for Testability, IEEE Computer (1985).
9) Hoare, C. A. R.: Proof of Correctness of Data Representation, Acta Inf., Vol. 1, No. 1 (1972).
10) Claybrook, B. G.: A Specification Method for Specifying Data and Procedural Abstractions, IEEE Trans. on SE, Vol. SE-8, No. 5 (1982).
11) Belkhouche, B. and Urban, J. E.: Direct Implementation of Abstract Data Types from Abstract Specifications, IEEE Trans. on SE, Vol. SE-12, No. 5 (1986).
12) Kramer, B.: SEGRAS - A Formal and Semigraphical Language Combining Petri Nets and Abstract Data Types for the Specification of Distributed Systems, Proc. 9th. IECE (1987).
13) Robinson, L. and Roubine, O.: SPECIAL—A Specification and Assertion Language, Technical Report CSL-46, Stanford Research Institute (1977).
14) Spivey, J. M.: Understanding Z: A Specification Language and its Formal Semantics, Cambridge Tracts in Theoretical Computer Science (1988).
15) Ambler, A. L.: Gypsy: A Language for Specification and Implementation of Verified Programs, ACM SIGPLAN Notices, Vol. 12, No. 3 (1977).
16) Zave, P. and Schell, W.: Salient Features of an Executable Specification Language and Its Environment, IEEE Trans. on SE, Vol. SE-12, No. 2 (1986).
17) 稲垣康善，坂部俊樹：抽象データタイプの代数的仕様記述法の基礎，情報処理，Vol. 25, No. 1, 5, 7, 9 (1984).
18) 稲垣康善：抽象データ型の概念と仕様記述法，情報処理，Vol. 27, No. 2 (1986).
19) Ehrig, H. and Mahr. B.: Fundamentals of Algebraic Specification 1, Springer (1985).
20) Goguen, J. A., Thatcher, J. W., Wagner, E. G. and Wright, J. B.: Initial Algebra Semantics and Continuous Algebras, JACM, Vol. 24, No. 1 (1977).
21) Bergstra, J. A. and Tucker, J. V.: Initial and Final Algebra Semantics for Data Type Specification: Two Characterization Theorems,

SIAM J. Comput., Vol. 12, No. 2 (1983).
22) Partsch, H. and SteinBruggen, R.: Program, Transformation Systems, ACM Computing Surveys, Vol. 15, No. 3 (1983).
23) 二木厚吉, 外山芳人: 項書き換え型計算モデルとその応用, 情報処理, Vol. 24, No. 2 (1983).
24) Burstall, R. M. and Goguen, J. A.: An Informal Introduction to Specification Using CLEAR, in the Correctness of Problem in Computer Science (1981).
25) Bidoit, M. and et al.: Exception Handling: Formal Specification and Systematic Program Construction, IEEE Trans. on SE, Vol. SE-11, No. 3 (1985).
26) The CIP Language Group: The Munich Project CIP, Vol. 1: The Wide Spectrum Language CIP-L, Lecture Notes in Computer Science 183 (1985).
27) 嵩 忠雄, 谷口健一, 杉山裕二, 関 浩之: 代数的言語 ASL/*一意味定義を中心に, 電子通信学会論文誌, Vol. J 69-D, No. 7 (1986).
28) 川辺恵久, 坂部俊樹, 稲垣康善, 本多波雄: 抽象データ型の直接実現システム, 信学技報, AL 83-65 (1984).
29) Hoare, C. A. R.: An Axiomatic Basis for Computer Programming, CACM, Vol. 12, No. 10 (1969).
30) 本位田真一, 伊藤 潔: プロトタイピング支援ツール, 情報処理, Vol. 30, No. 4 (1989).
31) Igarashi, S., London, R. L. and Luckham, D. C.: Automatic Program Verification 1: A Logical Basis and its Implementation, Acta Inf., Vol. 4 (1975).
32) Nagel, J. and Johnson, S.: Practical Program Verification: Automatic Program Proving for Real-time Embedded Software, Ford Aerospace and Communications Corp. WDL-TR 9859 (1982).
33) Guttag, J. V. and Horning, J. J.: The Algebraic Specification of Abstract Data Types, Acta Inf., Vol. 10 (1978).
34) Gordon, M. J., Milner, R. and Wadsworth, C.: Edinburgh LCF, Rep. CSR-11-77, Dept. of Computer Science, Edinburgh Univ. (1977).
35) Myers, G. J.: The Art of Software Testing, John Wiley & Sons, Inc. (1979) (長尾真監訳, 松尾正信訳: ソフトウェア・テストの技法, 近代科学社 (1980)).
36) Tavendale, R. D.: A Technique for Prototyping Directly from a Specification, Proc. of ICSE (1985).
37) Heyes, I. J.: Specification Directed Module Testing, IEEE Trans. on SE, Vol. SE-12, No. 1 (1986).
38) Gannon, J., McMullin, P. and Hamlet, R.: Data Abstraction Implementation, Specification, and Testing, ACM TOPLAS, Vol. 3, No. 3 (1981).
39) Horning, J. J.: Combining Algebraic and Predicative Specifications in Larch, in Formal Method and Software Development, LNCS 186 (1985).
40) Tavendale, R. D.: A Technique for Prototyping Directly from a Specification, Proc. of ICSE (1985).
41) Hamilton, M. and Zeldin, S.: The Functional Life Cycle Model and its Automation: USE-IT, Technical Report 37, HOS Inc., Cambridge (1981).
42) Campos, I. M., and Estrin, G.: Concurrent Software System Design Supported by SARA at the Age of One, Proc. 3rd. ICSE (1978).
43) Vidondo, F. Lopez, I. and Girod, J. J.: Galileo System Design Method, Electrical Communications, Vol. 55, No. 4 (1980).
44) Nakajima, R. and Yuasa, T.: The Iota Programming System, LNCS 160 (1983).
45) Wing, J. M.: Writing Larch Interface Language Specifications, ACM TOPLAS, Vol. 9, No. 1 (1987).
46) Beichter, F. W., Herzoc, O. and Perzsch, H.: SLAN-4—A Software Specification and Design Language, IEEE Trans. on SE, Vol. SE-10 No. 2 (1984).
47) Goguen, J. and Moriconi, M.: Formalization in Programming Environments, IEEE Computer (1987).
48) 河岡 司, 荒木哲郎: プロトコルの形式記述法, 電子通信学会誌, Vol. 68, No. 2 (1985).
49) Terwilliger, R. B. and Campbell, R. H.: An Early Report on Encompass, Proc. 10th. ICSE (1988).
50) Luckham, D. C. and von Henke, F. W.: An Overview of Anna, a Specification Language for Ada, IEEE Software (1985).

(平成元年2月9日受付)

付録-1 形式的仕様記述支援ツール一覧

名称（言語名）	開発者（場所）	主な支援ツール	参考文献
Gypsy	テキサス大 ICSCA	構文エディタ パーサ（構文エラーと意味エラーの検出，内部形式への変換） 検証条件生成系（手続きや関数を通過するすべてのパスの決定，検証条件の生成） 定理証明系（ユーザ支援のもとでの検証条件の証明） コンパイラ ユーザインタフェースハンドラ	15)
HDM (SPECIAL)	SRI International	STP（仕様と定理証明を支援するのため対話的支援システム）	13)
HOS（グラフィック形式，AXES）	Higher Order Software Inc.	グラフィック・エディタ エディタ 構文解析プログラム 意味解析プログラム（公理に関して） あらかじめ定義されている抽象データ型のライブラリ コンパイラ（Fortran コードを生成） 対話的シミュレータ （以上総称して，USE-IT と呼ばれる）	41)
SARA	I. M. Campos 他	パーサ 制御グラフ分析系（シミュレーションでは検出できないライブネス問題の予想） シミュレータ トレース，ブレイクポイント機能	42)
GALILEO	ITT SERA	実行のアニメーション化 タイミングの分析ツール	43)
AFFIRM	南カリフォルニア大	検証条件生成系 定理証明系（対話的） 書き換え規則マシン（公理的仕様のチェック）	33)
CIP (CIP-L)	ミュンヘン大	定理証明系 変換系 ユーザインタフェースハンドラ 証明されたプログラムの蓄積と検索のためのデータベース 開発プロセスの記録を保持するセッションログ（以上を総称して，CIP-S と呼ばれる）	26)
IOTA	京都大学 数理解析研究所	エディタ（入力の中間表現，オンラインの文書作成） 定理証明系（階層的かつ対話的） 直接実行系 デバッガ（実行中の状態のモニタ，ダンプの作成）	44)
Larch	V. J. Guttag J. J. Horning	定理証明系（対話的） 構文と静的な意味チェッカ 直構文エディタ 仕様ライブラリ	45)
FDL	R. D. Tavendale	FDL から Prolog への変換系 ヘルプ機能 セッションのロギング ブレイクポイント，トレースなどのデバッグ機能 コマンドファイル化機能 OS とのインタフェースハンドラ	41)
SLAN-4	F. W. Beichter 他	構文エディタ コンパイラ（インタフェース記述の一貫性のチェックなどを含む） 並行性に対する静的なチェック 仕様と設計文書のデータベース	46)
PAISLey	P. Zave 他	インタプリタ 一貫性チェッカ	16)