戦略的創造研究推進事業　CREST
研究領域
「実用化を目指した組込みシステム用
ディペンダブル・オペレーティングシステム」
研究課題
「ディペンダブルシステムソフトウェア構築技術」

研究終了報告書

研究期間　平成18年10月～平成24年3月

研究代表者：前田　俊行
（東京大学大学院情報理工学系
研究科　助教）

§1 研究実施の概要

(1)実施概要

本研究は、近年発展を遂げた静的プログラム解析技術（プログラムを数学的理論に基づいて解析することで、プログラムを実行することなく、その性質を知る技術)、特に型理論とモデル検査理論に基づき、システムソフトウェアの構築・検証技術を実現することでシステムソフトウェアの安全化・高信頼化を目指した。また、既存のソフトウェア高信頼化の理論研究では、実際的な議論や現実的な検証ツールの開発といった点が不十分であった。これに対し本研究では、既存の C 言語やアセンブリ言語等を対象に、一般の開発者にとって実際に実用的な検証ツールを開発することも目指した。

また当初の研究構想には無かったが、オープンシステムの考え方、すなわち、本質的に未知の障害要因が必ず存在し、事前に全ての障害要因に対処することは不可能であると仮定したシステムの捉え方において、プログラム解析技術を有効に適用する手法の考案も行った。

上記を達成するため、本研究では以下の三つの研究を行った。なお、これら三つの研究において、実際に他研究チームの作成した幾つかのプログラム等の検証が行える程度の実用性を持つツールを実現することができた。また他の商用プログラム検査ツールでは検出できなかったようなバグを検出することもできた。

1. システムソフトウェアを記述可能な型付きアセンブリ言語の設計・実装

型付きアセンブリ言語とは、アセンブリ言語のレベルで型理論を応用することにより、メモリの安全性等を保証できる言語であるが、従来の型付きアセンブリ言語は複数のプログラムが同時に非同期に実行される環境を考慮しておらず、実用的なシステムソフトウェアに応用することは困難であった。

そこで本研究では、複数のプログラムが同時に非同期に実行されるような環境でも、メモリ安全性や制御フロー安全性を保証できるような型付きアセンブリ言語の設計・実装を行った。これを用いてシステムソフトウェアのメモリ管理機構やプロセス管理機構等を構築することで、システムソフトウェアの安全性・信頼性を向上できる。

2. C 言語から型付きアセンブリ言語への変換器の設計・実装

型付きアセンブリ言語などの安全なプログラミング言語を用いてソフトウェアを構築すれば、システムは安全になるが、既存のソフトウェアをその言語で書き直さなければならない。しかし、全てのソフトウェアを新たに異なる言語で書き直すのは現実的では無い。

そこで本研究では、既存の C 言語のソースコードをそのまま(もしくは必要であれば C 言語に僅かな拡張・注釈を加えることによって)ほぼ自動的かつ小さなコストで、型付きアセンブリ言語へ変換する手法の設計・実装を行った。これにより、既存の C 言語プログラムに対して型付きアセンブリ言語の型検査を応用することが可能になる。

3. モデル検査技法に基づくシステムソフトウェアの解析

型付きアセンブリ言語などの安全なプログラミング言語を用いることで、ソフトウェアのメモリ安全性と制御フロー安全性を保証することができるが、現実的なシステムソフトウェアを考えると、より高度で複雑な安全性を保証することが必要な場合がある。

そこで本研究では、C 言語を用いて構築されたシステムソフトウェアに対しても現実的に適用可能なモデル検査手法・ツールの研究開発を行った。具体的には、システムソフトウェア、またそのシステムソフトウェアを利用するプログラムが満たすべき性質を、モデル検査器が直接解釈できる形式で記述し、これを用いてシステムのモデル検査を行うことを目指した。

(2)顕著な成果

1. システムソフトウェアを記述可能な型付きアセンブリ言語の設計・実装

概要: 複数のプログラムが同時に非同期に実行されるような環境でもメモリ安全性や制御フロー安全性を保証できるような型付きアセンブリ言語の設計・実装を行った。また他研究チームの作成した幾つかのプログラムの試験検証が行える程度の実用性を持つ型付きアセンブリ言語の処理系を実装し、C 言語から型付きアセンブリ言語への変換器と組み合わせることにより、他の商用検査ツールでは検出できなかったようなバグを検出できた。

2. C 言語から型付きアセンブリ言語への変換器の設計・実装

概要: 既存の C 言語のソースコードをほぼそのまま自動的に型付きアセンブリ言語へ変換する変換器の設計・実装を行った。また上述(1.)の通り、型付きアセンブリ言語の処理系と組み合わせることにより、他研究チームの作成した幾つかのプログラム中のバグを検出できた。

3. モデル検査技法に基づくシステムソフトウェアの解析

概要: C 言語を用いて構築されたシステムソフトウェアに対しても現実的に適用可能なモデル検査器の設計・実装を行った。また実際に他研究チームの作成した幾つかのプログラムに対して試験検証を行い、他の商用検査ツールでは検出できなかったようなバグを検出できた。

§2. 研究構想

（1）当初の研究構想

計算機（携帯電話、PDA、PC 等）、そして計算機を相互に接続するネットワークは、既に広く一般に普及し、もはや生活に欠かすことのできない社会基盤となった。このため、これらの計算機上で動作するソフトウェアの安全性・信頼性を確保することが非常に重要な課題である、ということが既に広く認識されている。

しかしながら、コンピュータシステムで最も基礎的なソフトウェアであるシステムソフトウェア（オペレーティングシステム等）は、依然として三十年以上前に考案された C 言語やアセンブリ言語などの安全でない言語で構築されており、その安全性・信頼性には大きな疑いがあると言わざるを得ない。また現実に多くの安全性・信頼性上の問題、例えばシステムの異常停止、情報漏洩等の問題が発生している。

従来、これらの安全性・信頼性上の問題を解決するために、様々なシステムソフトウェア高信頼化技術が提案されてきたが、その高信頼化技術自体、C 言語などの安全でない言語で記述されているため、その安全性・信頼性を保証・検証できないという問題が存在する。最悪の場合には、高信頼化技術の実装に問題があり、システムソフトウェア全体の安全性・信頼性が損なわれてしまう可能性すらある。

これに対し本研究では、近年発展を遂げた静的プログラム解析技術（プログラムを数学的理論に基づいて解析することで、プログラムを実行することなく、その性質を知る技術）、特に型理論とモデル検査理論に基づき、システムソフトウェアの構築・検証技術を実現することでシステムソフトウェアの安全化・高信頼化を目指した。具体的には、例えばメモリ安全性（プログラムが不正なメモリ操作を行わないこと）や並行実行安全性（プログラムが複数同時に実行されてもメモリ等に不整合が生じたり、異常停止したりしないこと）を検証することで、システムソフトウェアが正しく実装されていることを保証し、その安全性・信頼性を確保することを目指した。

また、既存のソフトウェア高信頼化の理論研究では、ややもすると机上の議論にとどまり、実際的な議論や現実的な検証ツールの開発といった点が不十分であった。これに対し本研究では、理論に基づき、既存の C 言語やアセンブリ言語等を対象に、システムソフトウェアの開発者にとって実際に実用的な検証ツールを開発することを目指した。

具体的には、上記の目的を達成するため本研究では以下の三つの研究を行うこととした。

1. システムソフトウェアを記述可能な型付きアセンブリ言語の設計・実装

型付きアセンブリ言語とは、アセンブリ言語のレベルで型理論を応用することにより、メモリの安全性（プログラムが不正なメモリアクセスを行わないこと）や制御フローの安全性（プログラムが不正なコード実行を行わないこと）を保証できる言語である。例えば、型付きアセンブリ言語を用いてプログラムを構築することにより、そのプログラムにメモリ操作のバグが存在しないことを保証できる。

ところが従来の型付きアセンブリ言語は、複数のプログラムが同時に非同期に実行される環境を考慮しておらず、このような環境では、メモリ安全性や制御フロー安全性を厳密に保証することはできなかった。このため、実用的なシステムソフトウェアの開発のために従来の型付きアセンブリ言語を直接利用することは困難であった。

そこで本研究では、このような複数のプログラムが同時に非同期に実行されるような環境でも、メモリ安全性や制御フロー安全性を保証できるような型付きアセンブリ言語の設計・実装を行うこととした。これを用いてシステムソフトウェアのメモリ管理機構やプロセス管理機構、デバイスドライバを構築することで、システムソフトウェアの安全性・信頼性を向上できる。

2. C 言語から型付きアセンブリ言語への変換器の設計・実装

型付きアセンブリ言語などの安全なプログラミング言語を用いてソフトウェアを構築すれば、システムは安全になるが、既存のソフトウェアをその言語で書き直さなければならない。しかし、既存のソフトウェア資産は膨大であるため、全てのソフトウェアを新たに異なる言語で書き直すのは現実的では無い。例えば、Linux カーネルのソースコードは約 700 万行にも及ぶため、これを一から書き直

すのは非常に困難である。
そこで本研究では、既存の C 言語のソースコードをそのまま(もしくは必要であれば C 言語に僅かな拡張・注釈を加えることによって)ほぼ自動的かつ小さなコストで、型付きアセンブリ言語へ変換する手法の設計・実装を行うこととした。

3. モデル検査技法に基づくシステムソフトウェアの解析

メモリ安全性と制御フロー安全性を保証できれば、プログラムの不正なメモリ操作や不正なコード実行を完全に防止することができ、ソフトウェアの安全性・信頼性は大きく向上する。しかし、現実的なシステムソフトウェアを考えると、上記のメモリ安全性や制御フロー安全性にとどまらず、より高度で複雑な安全性を保証することが必要な場合がある。

例えば、システムソフトウェア内で複数のプログラムが同時に実行されている場合に、双方のプログラムが、互いに相手のプログラムの処理の終了を待ってしまい、結果として双方のプログラムの実行が異常停止してしまうこと(デッドロック)が生じないようにすることや、システムソフトウェアが機密情報(パスワード、クレジットカード番号、個人情報など)を誤って処理して外部に漏洩しないようにすること、特定のプログラムが計算機資源(メモリや CPU 時間)を専有してしまわないようにすること等を保証することが必要とされる場合がある。

従来、このような複雑な安全性を保証するためには、モデル検査の手法が用いられてきたが、オペレーティングシステム等のシステムソフトウェアの安全性検証には現実的には応用できなかった。これは従来のモデル検査がシステムソフトウェアで用いられるようなプログラム手法を特別に考慮していなかったためである。

これに対し、本研究では C 言語を用いて構築されたシステムソフトウェアに対しても現実的に適用可能なモデル検査手法・ツールの研究開発を行うこととした。具体的には、システムソフトウェア、またそのシステムソフトウェアを利用するプログラムが満たすべき性質を、モデル検査器が直接解釈できる形式で記述し、これを用いてシステムのモデル検査を行うことを目指した。

(2)新たに追加・修正など変更した研究構想

研究を進めていく中で、現代のシステム開発・運用で求められているディペンダビリティとは何かについて他研究チームや研究総括と議論・検討を重ねた結果、「オープンシステム」という考え方が重要であるという認識に至った。ここでいうオープンシステムとは、端的に言えば、未知の部分が本質的に存在するような(すなわち閉じていない、クローズドでない)システム、またはシステムの理解の仕方のことである。従来のシステムのディペンダビリティの考え方では、システム障害等の要因は基本的に事前に全て把握可能という立場であり、この観点から、障害要因に対してどのように対応するかを対象としていた。これに対しオープンシステムの考え方では、事前に全ての障害要因を把握することは不可能であるから、実際に問題が発生してしまったときに、どうやってその問題を検出し、素早く障害回避・復旧をし、障害の要因・責任を明確に説明できるようにし、システムの修繕等を行うのか等が重要となる。

これに対し、上述のプログラム解析(型理論やモデル検査理論)は、プログラムが何らかの既知の性質を満たしていることを示すものであり、オープンシステムの考え方で重視される、未知の問題・性質を直接扱うことは難しい。そこで本研究では、研究領域全体で議論・設計している、オープンシステムにおけるディペンダビリティの向上のための開発・運用プロセス(DEOS プロセス)におけるシステム開発・修正の段階においてプログラム解析理論・技術を適用することによって、プロセスの円滑な実行を実現し、ディペンダビリティの向上に寄与する手法の検討を行った。

§3 研究実施体制

(1) 前田グループ

①研究参加者

氏名	所属	役職	参加時期
前田 俊行	東京大学大学院情報理工学系研究科	助教	H18.10～H24.3
松田 元彦	東京大学大学院情報理工学系研究科	特任准教授	H19.6～H24.3
村田 雅之	東京大学大学院情報理工学系研究科	技術補佐員	H23.4～H24.3
池尻 拓朗	東京大学大学院情報理工学系研究科	技術補佐員	H23.4～H24.3
坂西 由美子	東京大学大学院情報理工学系研究科	技術補佐員	H20.9～H24.3
米澤 明憲	東京大学大学院情報理工学系研究科	教授	H18.10～H23.3
吉野 寿宏	東京大学大学院情報理工学系研究科	技術補佐員	H18.10～H21.3
佐藤 春旗	東京大学大学院情報理工学系研究科	技術補佐員	H19.4～H20.3
小酒井 隆広	東京大学大学院情報理工学系研究科	技術補佐員	H19.4～H21.3
潮田 資秀	東京大学大学院情報理工学系研究科	技術補佐員	H20.4～H22.3
野尻 隆宏	東京大学大学院情報理工学系研究科	技術補佐員	H20.4～H22.3
渡邊 裕貴	東京大学大学院情報理工学系研究科	技術補佐員	H21.4～H23.3
鈴木 友博	東京大学大学院情報理工学系研究科	技術補佐員	H22.10～H23.3
鈴木 由美子	東京大学大学院情報理工学系研究科	技術補佐員	H20.4～H20.9

②研究項目

- システムソフトウェアを記述可能な型付きアセンブリ言語の設計・実装
- C 言語から型付きアセンブリ言語への変換器の設計・実装
- モデル検査技法に基づくシステムソフトウェアの解析

§４ 研究実施内容及び成果

４．１ システムソフトウェアを記述可能な型付きアセンブリ言語の設計・実装（東京大学前田グループ）

(1)研究実施内容及び成果

型付きアセンブリ言語とは、アセンブリ言語のレベルで型理論を応用することにより、メモリの安全性（プログラムが不正なメモリアクセスを行わないこと）や制御フローの安全性（プログラムが不正なコード実行を行わないこと）を保証できる言語である。例えば、型付きアセンブリ言語を用いてプログラムを構築することにより、そのプログラムにメモリ操作のバグが存在しないことを保証できる。

ところが、従来の型付きアセンブリ言語は、複数のプログラムが同時に非同期に実行される環境を考慮しておらず、このような環境では、メモリ安全性や制御フロー安全性を厳密に保証することはできなかった。このため、実用的なシステムソフトウェアを開発するためには、従来の型付きアセンブリ言語を直接利用することは困難であった。

そこで本研究では、このような複数のプログラムが同時に非同期に実行されるような環境でも、メモリ安全性や制御フロー安全性を保証できるような型付きアセンブリ言語の設計・実装を行った。これを用いてディペンダビリティ支援機構を構築することで、ディペンダビリティ支援機構そのもののメモリ安全性・制御フロー安全性を検証することが可能となり、結果としてシステム全体のディペンダビリティの向上に寄与する。

具体的に各年度に実施した研究内容・成果は以下の通りである。まず平成 18 年度は、我々が設計した、メモリ管理機構やスレッド管理機構の記述が可能な程度に強力な型付きアセンブリ言語の理論を更に拡張し、SMP 環境のようにプログラムが複数個同時に実行される環境においても、メモリ安全性が保証できる型付きアセンブリ言語の理論の検討を行った。

次いで平成 19 年度は、ハードウェア割込みやマルチコア・SMP に対応した型付きアセンブリ言語の理論設計を行った。ハードウェア割込みに対しては、対応する割込みハンドラが、プログラムの任意の実行ポイントで実行され得ると仮定して型検査を行う。また、マルチコア・SMP に関しては、複数コア（CPU）間で共有され得るメモリ領域に対して、型を変更するようなメモリ操作を基本的に禁止して、CPU ハードウェアが提供する atomicity を保った命令のみ、命令の開始から終了までの間、一時的に型を変更することを許すことで、共有され得るメモリ領域の型安全性を保つ。また、マルチコア・SMP 特有のメモリコンシステンシの問題に対しては、存在型（existential type）の理論を応用することで、幾つかの既存の同期プリミティブを実現できることを確かめた。

また、型付きアセンブリ言語の型検査器を、異なる CPU アーキテクチャごとに一から作成するのは非効率であるため、CPU アーキテクチャごとのアセンブリ言語から、ある共通の中間表現へ変換し、その中間表現上で型検査を行う仕組みを検討した。さらに、中間表現上で行った型検査の結果が、CPU アーキテクチャごとに行った型検査の結果と同等になるための条件についての考察を行った。

また、柔軟なメモリ管理の記述をプログラマに許しつつもメモリ安全性を損なわず、かつ、C 言語からのコンパイルも可能な型付きアセンブリ言語の実現を目指して、本年度は、ポインタ間のエイリアス情報を追跡するエイリアス型システムを、分離論理（separation logic）で用いられている分離含意（separating implication）で拡張することで、従来のエイリアス型システムでは型付けできなかったループ構造の記述が可能になることを示した。

更に平成 20 年度は前年度より引き続き、システムソフトウェアを記述可能な型付きアセンブリ言語の実装を進めた。具体的には、前年度までに行った型付きアセンブリ言語を異なる CPU アーキテクチャへ移植することを容易とするための手法検討に基づき、本研究領域全体で設計・実装中であるディペンダビリティ支援のためのシステム機構を記述することを目標としてプロトタイプ実装を行った。加えて、前年度までに行ったハードウェア割込みやマルチコア・SMP に対応した型付きアセンブリ言語の設計を更に進め、実際に簡単な試験実装も行った。

また、従来よりも柔軟なメモリ管理の記述を可能とするために、参照カウント方式のメモリ管理に対応した型付きアセンブリ言語の設計を行った。従来の型付きアセンブリ言語では、ガーベジコレクション等の外部のメモリ管理機構に依存するか、もしくはポインタ(メモリ参照)のエイリアス(複数のポインタが同一のアドレスを指すこと)に制限を課す必要があったため、参照カウント方式のようなメモリ管理機構を型付きアセンブリ言語で記述することは困難であった。これに対し、実際のメモリ中の(あるアドレスを指す)ポインタの数と、参照カウント値としてメモリ中に保存されている値との誤差を型システムで追跡することによって、参照カウント方式のメモリ管理を型付きアセンブリ言語で記述できることを示した。

また、副作用のない高階関数と副作用のある計算とを含むプログラムの正当性を検証するための型システムの一つであるホーア型理論に対して、従来の手法ではプログラム内で更新されるロケーションの数が静的に決まっている必要があり、ロケーション(メモリアドレス)の数が動的に変化する場合を扱うことができなかったのに対し、ロケーションの扱い方を変更し、配列などのように更新されるロケーションの数が動的に決まるデータ構造に対応できるように拡張した型システムの設計を行った。

次いで平成 21 年度は、前年度より引き続き、より現実的な型付きアセンブリ言語のプロトタイプ実装を行った。またこの実装を用いて幾つかのディペンダビリティ支援機構の検証を試みた。具体的には、他研究チームによって実装された幾つかのディペンダビリティ支援機構のソースコードを型付きアセンブリ言語へ変換したものに対して型検査を行い、メモリ安全性や制御フロー安全性の検証を行った。その結果、あるプログラムのメモリ初期化部分に問題があることを検出することができた。この問題は、一見正しく動作しているように見えて、メモリの使用状況によっては稀に致命的な障害を生じる可能性があるという比較的深刻なものであったため、プロトタイプ実装の有効性を部分的に示せた上、ディペンダビリティ支援機構そのもののディペンダビリティの向上に貢献できた。また型付きアセンブリ言語の静的型検査に要する実行時間は数秒～数十秒の範囲であった。また、この結果を他の検査ツールと比較するために、商用検査ツールを導入し比較検討したところ、上述の我々が検出した問題を検出することはできなかった。

また平成 22 年度は、前年度までに作成した型付きアセンブリ言語のプロトタイプ実装を用いて、我々の型付きアセンブリ言語の理論・実装の問題点について検討を行った。具体的には、他研究チームが作成したディペンダビリティ支援機構のソースコードや Linux カーネルのデバイスドライバのソースコード等を我々の型付きアセンブリ言語へ変換することを試み、またプロトタイプ実装を用いて型検査を試みた。この結果、連結リスト構造(またはそれに似たデータ構造)をループ処理によって操作するようなプログラムの変換・型検査を我々の型付きアセンブリ言語で上手く扱えないことが分かったため、この問題を解決するための型システムの拡張の形式的定義・議論を行った。

また、予期せぬシステム障害(オープンシステム障害)からの復旧・問題修正等に対して、型検査技術がどのような役割を果たせるかについて、システム開発・運用プロセスの観点から議論・検討を行い、ホワイトペーパー等の形で発表した。具体的には、外部要因の変化や利用者等の要求の変化、事前に想定できなかった未知の障害等に伴って、システム分析(D-Case 等)の修正・改善が行われ、システムのソフトウェアの修正・新規開発が必要となった場合に、基礎的な安全性(プログラムが不正なメモリ操作が行わないことなど)を短時間で検証できるという型検査技術の特性を活かして、ソフトウェア開発者がソフトウェアの修正を行うのに追従して、基礎的な安全性を継続的に保証することができる。これにより、プログラム開発者は、些細なプログラミングミスなどに煩わされること無く、ソフトウェアの修正に集中することができる。一方、システム分析の修正に伴うソフトウェアの修正が期待通りに行われたかどうかの検証・テストは、モデル検査技術やベンチマーキング技術を用いて行うことができる。

次いで平成 23 年度には、前年度までに実装した型付きアセンブリ言語の処理系の完成度の向上を目指した。具体的には、実際のシステムソフトウェア開発者による利用を念頭にした実行環境の整備を進めた。また前年度より引き続き、他研究チームの作成したディペ

ンダビリティ支援機構のソースコードやLinuxカーネルのドライバのソースコードを型付きアセンブリ言語へ変換したものに対して型検査を試みた。更に、前年度までに行った、予期せぬ障害(オープンシステム障害)へ対応するためのシステム運用・開発プロセス全体の中における型検査の位置付けを明確にする議論を更に進め、他研究チームで研究を行なっている運用時のロギング機構・モニタリング機構との連携手法について検討を行い簡単な試験実装を進めた。また、オープンシステムにおけるシステム分析手法(D-Case)との連携について理論検討を行った。

本研究領域全体における上述の研究実施内容の具体的な位置付け・役割は、本研究領域全体が提案する、オープンシステムにおけるディペンダビリティの向上を実現するための開発・運用プロセス(DEOS プロセス)において、実際に利用可能なツールを設計・実装することである。より具体的には DEOS プロセスにおけるシステム開発・修正の段階において、プログラムの新規開発や修正の際に不用意に単純なバグを混入させないように、C 言語から型付きアセンブリ言語への変換器と組み合わせた型検査によって、プログラムのメモリ安全性・制御フロー安全性を継続的に保証し続けることで、システム開発・修正の安定的な土台を提供することである。また別の役割は、DEOS プロセスを実践するためのディペンダビリティ支援機構の具体的な実装自体のメモリ安全性・制御フロー安全性を保証・検証を行うことであり、実際に前述の様に、幾つかのコンポーネント・ドライバに対して検証を行い、既存の商用ツールが検出できなかったようなバグを検出することもできた。

また、従来研究と比べて本研究の新しい点の一つは、型付きアセンブリ言語の理論を拡張することにより、従来の型安全なプログラミング言語では不可能であった、システムソフトウェアの核となる部分であるメモリ管理やスレッド管理等のさまざまな実装に対して型検査を行うことを可能にした点である。また従来の型付きアセンブリ言語の研究では検証の対象とはなっていなかった、複数のスレッドが同時に実行されうる実行環境での同期機構(ロックなど)そのものの検証を行うことを可能にした点も、従来研究と異なる点である。

(2)研究成果の今後期待される効果

本研究では、C 言語から型付きアセンブリ言語への変換器と組み合わせることにより、実際に他研究チームの作成した幾つかのプログラムの試験検証が行える程度の実用性を持つツールを実現できたが、今後の展開として、実行環境やマニュアルの整備、サポート体制の構築等を進めることによって、一般の開発者への普及が期待できると考えられる。

また本研究では複数のスレッドが同時に実行されうる実行環境でのシステムソフトウェアの検証を主眼に置いているが、結果として得られた型システム等はシステムソフトウェアに限定されず、より一般の並列プログラムに拡張可能なものとなっている。近年、また少なくとも今後数年の計算機の傾向として、多数のコアで構成される CPU・GPU を用いる構成が主流になると考えられているため、本研究の成果をアセンブリ言語以外の高級プログラミング言語へ応用することで、一般の並列プログラミング、また特に開発者が(最適化等の意図のもとで)特殊な同期機構を独自に実装しているようなプログラムにおいても解析・検証が可能になると考えられる。

4.2 C 言語から型付きアセンブリ言語への変換器の設計・実装(東京大学前田グループ)

(1)研究実施内容及び成果

型付きアセンブリ言語などの安全なプログラミング言語を用いてソフトウェアを構築すれば、システムは安全になるが、既存のソフトウェアをその言語で書き直さなければならない。しかし、既存のソフトウェア資産は膨大であるため、全てのソフトウェアを新たに異なる言語で書き直すのは現実的では無い。

そこで本研究では、既存の C 言語のソースコードをそのまま(もしくは必要であれば C 言語に僅かな拡張・注釈を加えることによって)ほぼ自動的に、型付きアセンブリ言語へ変換する手法の考案・実装を行った。

具体的に各年度に実施した研究内容・成果は以下の通りである。まず平成 18 年度は、

C 言語からの変換が可能な型付きアセンブリ言語の設計と、型検査器の簡単な試験実装を行った。具体的には、依存型（dependent type）の理論を応用することで、C 言語のほとんどの機能（ポインタ演算、整数・ポインタ間のキャスト、構造体、共用体、可変長引数関数、関数ポインタなど）が型付きアセンブリ言語で表現可能となることが明らかになった。

また、C 言語で記述されたプログラムを型付きアセンブリ言語へ変換する手法の検討を行った。具体的には、C のソースプログラムを解析し、検査コード挿入などのコード変換を施してプログラムの安全性を保証しようとする既存手法を、型付きアセンブリ言語への変換に応用する方法についての検討を行った。この検討に基づき、実際に C 言語から上述の型付きアセンブリ言語への変換器の試験実装を行った。

次いで平成 19 年度は、前年度から継続して行った C 言語からの変換が可能な型付きアセンブリ言語の設計を完成させ、また型検査器のプロトタイプ実装を行った。具体的には、従来の型付きアセンブリ言語を依存型の理論に基づいて改良することで、C 言語特有の複雑な言語機構を型付きアセンブリ言語上で実現した。 また、実際に C 言語で記述されたプログラムを型付きアセンブリ言語へ変換する変換器のプロトタイプ実装を行った。具体的には、前年度までに検討した安全化コンパイラの手法を応用・改良して、型安全性が保証された動的検査コードを挿入する手法を考案した。また、実際に幾つかのプログラムをこの変換器を用いて型付きアセンブリ言語へ変換し、有用性・問題点の検討を行った。

また、C 言語プログラムのコンパイルに広く用いられているコンパイラ GCC の中間言語 RTL に対する型システムの検討も行った。具体的には、C 言語のコンパイル上、本質的な部分を RTL から抜き出して簡略化した言語である Tiny RTL を定義し、これに対して型システムを与えた。

更に平成 20 年度は、昨年度より引き続き C 言語から型付きアセンブリ言語への変換のための理論の設計と、実際に C 言語で記述されたシステムソフトウェアを変換するための変換器のプロトタイプ実装を行った。具体的には、他研究チームのディペンダビリティ支援機構の実装の一部分を実際に型付きアセンブリ言語に変換可能なプロトタイプ実装を行った。ただし、GCC（システムソフトウェア開発で広く用いられている C 言語コンパイラ）のインラインアセンブリ機能等は実装されていないため、現状ではソースコードを修正する等の対応が必要である。

次いで平成 21 年度は、前年度より引き続き、C 言語から型付きアセンブリ言語への変換器のより現実的なプロトタイプ実装を行った。またこの実装を用いて他研究チームの作成したディペンダビリティ支援機構の変換を試みた。具体的には、他研究チームによって実装される幾つかのプログラムの一部のソースコードを、この変換器のプロトタイプ実装を用いて型付きアセンブリ言語の型検査器のプロトタイプ実装で検証できる形式に変換した。この変換では、メモリ安全性等を保証するために、実行時に動的検査を行うためのコードが挿入されるが、これに伴うオーバーヘッドは、実行時間に関して数倍～十倍程度であった。また、この動的検査コードとディペンダビリティ支援機構との連携について、基礎的な検討を行った。

また平成 22 年度は、前年度までに作成した C 言語から型付きアセンブリ言語への変換器のプロトタイプ実装を用いて、我々の変換手法の理論・実装の問題点について検討を行った。具体的には、他研究チームの作成したディペンダビリティ支援機構のソースコードや Linux カーネルのデバイスドライバのソースコード等を我々の変換器で変換することを試みた。その結果、型付きセンブリ言語へ変換したプログラムとそれ以外の部分との相互の関数呼び出しやデータ受け渡しの処理を記述する手間が無視できないことが分かったため、この処理を行うプログラムを自動的に生成する手法について設計を行った。

また、動的検査技術、特にロギング機構（モニタリング機構）との連携について理論検討を行った。具体的には、プログラム中にログデータの取得を行う箇所を明示的に指定する場合において、冗長なログを取得しようとしている箇所を検出する手法や、変換器に実装した動的検査コード挿入の仕組みを応用して、ログデータを取得するコードをプログラム中に自動挿入する手法について検討を行った。

次いで平成 23 年度には、前年度までに実装した C 言語から型付きアセンブリ言語への変換器の完成度の向上を目指した。具体的には（型付きアセンブリ言語の処理系と同様に）、実際のシステムソフトウェア開発者による利用を念頭に実行環境の整備を進めた。また前年度より引き続き、この変換器を用いて他研究チームが作成したディペンダビリティ支援機構のソースコードや Linux カーネルのデバイスドライバ等のソースコードの変換を試みることで、実装の問題点の検出・修正を行った。更に、動的検査技術（ロギング機構・モニタリング機構等）と変換器の挿入する実行時検査コードの連携機能についての試験的実装を進めた。

本研究領域全体における上述の研究実施内容の具体的な位置付け・役割は、（型付きアセンブリ言語の処理系と同様に）本研究領域が提案するオープンシステムにおけるディペンダビリティの向上を実現するための開発・運用プロセス（DEOS プロセス）において、実際に利用可能なツールを設計・実装することである。より具体的には DEOS プロセスにおけるシステム開発・修正の段階において、プログラムの新規開発や修正の際に不用意に単純なバグを混入させないように、型付きアセンブリ言語の処理系と組み合わせた型検査によって実装のメモリ安全性・制御フロー安全性を継続的に保証し続けることで、システム開発・修正の安定的な土台を提供することである。また、C 言語から型付きアセンブリ言語への変換に伴って挿入される動的検査コードと実行時のディペンダビリティ支援機構（ロギング機構やモニタリング機構等）とを連携させることで、静的・動的両側面から実装の信頼性の向上することも可能となる。また別の役割は、DEOS プロセスを実践するためのディペンダビリティ機構の具体的な C 言語のソースコードを型付きアセンブリ言語に変換することで、実装の開発効率に影響することなく（つまり、開発者が型付きアセンブリ言語を直接記述する必要なしに）、実装のディペンダビリティを向上することであり、実際に前述の様に、幾つかのコンポーネントに対して検証を行い、既存の商用ツールが検出できなかったようなバグを検出することもできた。

また、従来研究と比べて本研究の新しい点は、C 言語のソースプログラムを安全な型付きアセンブリ言語プログラムへ変換することにより、変換後の機械語プログラムのメモリ安全性等を型検査によって保証・検証することができる点である。従来の研究では、安全でない C 言語のソースプログラムに対して、動的に安全性を検査するコードを挿入して安全化するものが幾つかあったが、これらの研究は C 言語から C 言語への変換を行うものであり、コード挿入後のソースプログラムの安全性やソースプログラムから機械語プログラムへ変換した後の安全性を検証することが困難であった。

(2)研究成果の今後期待される効果

本研究では、型付きアセンブリ言語の処理系と組み合わせることにより、実際に他研究チームの作成した幾つかのプログラムの試験検証が行える程度の実用性を持つツールを実現できたが、今後の展開として、実行環境やマニュアルの整備、また特に他の処理系・コンパイラと組み合わせて開発を行う際に生じる処理系の差異を橋渡しするような仕組みの整備や変換後のプログラムの実効性能の向上などを進めることによって、一般の開発者への普及が期待できると考えられる。

またオープンシステムのディペンダビリティを向上するための DEOS プロセス・アーキテクチャとの関係において、実行時のモニタリング機構・ロギング機構との連携の試験を行ったが、今後の展開としては、DEOS プロセス・アーキテクチャとの連携をツールレベルでより深めること、例えば現状では D-Case ダイアグラムと検証ツールとの間で直接データの受け渡しなどは行われず、人手を介する必要があるが、これを自動化するために D-Case エディタ等のシステムとの連携を行うための仕組みを構築する等の展開が考えられる。

4．3 モデル検査技法と型理論による解析の統合（東京大学前田グループ）

(1)研究実施内容及び成果

メモリ安全性と制御フロー安全性を保証できれば、プログラムの不正なメモリ操作や不正

なコード実行を完全に防止することができ、ソフトウェアの安全性・信頼性は大きく向上する。しかし、現実的なシステムソフトウェアを考えると、上記のメモリ安全性や制御フロー安全性にとどまらず、より高度で複雑な安全性（例えば、複数のプログラムが同時に実行されている場合に、それらのプログラムが正しく同期機構を使っていることや、システムによって提供されている API をプログラムが正しく使っていることなど）を保証することが必要な場合がある。

本研究では、このような複雑な安全性を検証するために、モデル検査の技法を用いて、ディペンダビリティ支援機構の解析を行う。具体的には、ディペンダビリティ支援機構等の満たすべき性質やその機構等を用いるプログラムが満たすべき性質を、モデル検査器が直接解釈できる形式で記述し、これを用いてディペンダビリティ支援機構やシステムのモデル検査を行うことで、システム全体のディペンダビリティの向上を目指した。

具体的に各年度に実施した研究内容・成果は以下の通りである。まず平成 18 年度は、C 言語プログラムを対象にした既存のモデル検査器の調査を行った。その結果、それらのモデル検査器ではプログラムのメモリ安全性を前提としている（メモリ安全性そのものは検証できない）等の制限があることが分かった。そこで、既存の手法を改良し、プログラム中にアサーションを自動的に挿入することでメモリ安全性を検証するモデル検査の手法の検討を行った。また、モデル検査の応用可能性を検討するために、既存の OS のソースコードの調査も行った。

次いで平成 19 年度は、前年度の調査結果に基づき、モデル検査器のプロトタイプ実装を行った。モデル検査器は、主なプログラム要素として、C 言語構文解析、意味解析、充足性検査器、抽象実行処理の四つの部分から構成されるが、平成 19 年度は抽象実行以外の基本要素である構文解析、意味解析、充足性検査器を作成した。抽象実行は検査器の中心であり、検証精度や検証時間といった検証能力に関わるため、様々な手法について調査・検討を行い、その結果、C 言語で記述された大規模プログラムの検査に対して現状で実用可能性が高いと考えられるブール値プログラムへの変換に基づく手法をプロトタイプ実装では採用することとした。また、モデル検査器は処理量が大きく大規模プログラムを扱うために、モデル検査自体の並列化が可能な実装を行っている。なお、プロトタイプ実装に際して、既存の検査ツールを流用することも検討したが、既存ツールはモデル検査の並列化や型検査との組み合わせなどを考慮せずに実装されており、また継続的に実用するには困難が伴うと考えられた等、自ら実装することとした。

更に平成 20 年度は、前年度より引き続き C 言語で記述されたプログラムの安全性を検証するためのモデル検査理論の構築と、それに基づいたモデル検査器のプロトタイプ実装を進めた。具体的には、検査するプログラムの仕様を記述するための仕様記述言語を定義し、これを解釈してプログラム中に注釈として挿入する仕様記述変換器と、挿入された検査条件が満たされているかを検査するモデル検査器のプロトタイプ実装を進めた。このプロトタイプ実装を用いて、他研究チームのディペンダビリティ支援機構の一部分に対して、実際に簡単な検査を試行することができた。

また、C 言語のメモリ安全性を保証する手法の一つとして、依存型システムを導入して配列の境界検査などを行う手法があるが、この依存型の推論を、モデル検査器を用いて行う方法を考案した。具体的には、与えられた C 言語プログラムに対してモデル検査を行い、配列の境界違反を生じうる実行パスを得て、このパスを解析してプログラム中の変数が満たすべき性質を依存型として抽出し、これを元のプログラムに型注釈として挿入する。また、この手法を既存のモデル検査器を用いて実装し実験を行った。また、モデル検査器を並列化し、大規模計算機上で実行するための予備実験として既存のモデル検査器を改造して並列化し実際に大規模計算機上で動作を確認した。

次いで平成 21 年度は、C 言語で記述されたプログラムの安全性を検証するためのモデル検査器のプロトタイプ実装を行った。またこの実装を用いて、他研究チームの作成したディペンダビリティ支援機構の検証を試みた。具体的には、他研究チームによって実装された幾つかのディペンダビリティ支援機構のソースコードに対して、ロックの整合性や API

使用方法の正当性等についてモデル検査を行った。この検査を行うために、他研究チームと連携して、API 仕様を記述するための仕様記述言語を策定し、また実際に仕様記述言語を用いて API 仕様を記述した。この検査の結果、あるディペンダビリティ支援機構のロックの獲得・解放処理やタイマーの初期化・終了処理に問題があることを検出することができた。この問題はシステムの動作が停止したり、メモリ中のデータを破壊したりする深刻なものであり、更に稀なエラー処理時にのみ生じる、通常のテストでは検出することが難しい問題であったため、プロトタイプ実装の有効性を示せた上、ディペンダビリティ支援機構そのもののディペンダビリティの向上に貢献できた。なお、この検査に要する実行時間は、約三千行のコードに対して数分～三十分程度であった。

また、この結果を他の検査ツールと比較するために、実際の商用検査ツールと比較検討したところ、上述の我々が検出した問題を検出することはできなかった。ただし、当該商用検査ツールの検査実行時間は概ね数分の範囲にとどまっており、利便性を考慮してあえて検査の精度を落としている可能性もある。また、ツールの使い勝手に関しては、商用検査ツールに一日の長があるが、使い勝手の向上によって他研究チームからのフィードバックを得易くするという研究開発の観点から、比較的広く用いられている GUI 統合開発環境との連携機能の実装を行った。

また、モデル検査をより現実的なシステム開発環境で行えるようにするために、プログラムの安全性検証を分割して複数計算機上で実行するための手法の実装・実験を行った。具体的には、プログラム中の特定のデータに注目して処理を分割することで、均等に処理を分割し、かつ、計算機間の通信頻度・通信量を低く抑える方法を検討した。更に、実際にクラスタ計算機システムを用いて実験を行い、その有効性・問題点の検討も行った。また、プログラムの修正に際して、全体を再検査するのではなく、部分的に再検査するような差分モデル検査手法について基礎的な理論の検討を行った。

また平成 22 年度は、前年度までに作成したモデル検査器の実装の改善を行った。具体的には、他研究チームの作成したディペンダビリティ支援機構のモデル検査を、実際に他の研究チームのメンバーに利用してもらうことで、モデル検査器の機能・性能・利便性に関する問題点の検討を行った。この結果、連結リストを用いるような、ポインタ周りの操作を頻繁に行うようなプログラムにおいて、検査の結果が期待通りにならなかったり、検査時間が長引いたりすることが分かった。この問題に対処するため、ポインタ解析を強化したモデル検査器の検討と、またそれにもとづく試験的な実装を行った。また、仕様記述言語において、同期ロックで保護されたデータに関する仕様の記述が上手く表現できないということが分かったため、この点を改良する仕様記述の方法についても理論検討・設計を行った。

さらに、検査に要する時間を短縮するための差分モデル検査（プログラムの修正に際して、全体を再検査するのではなく、部分的に再検査するような手法）の理論的背景として、自己適応計算の理論にもとづいて、通常どおり記述されたプログラムを自己適応計算的な形式のプログラムに自動的に変換する手法についての理論的検討・設計を行った。

また、予期せぬシステム障害（オープンシステム障害）からの復旧・問題修正等に対してモデル検査がどのような役割を果たせるかについて、システム開発・運用プロセスの観点から議論・検討を行い、ホワイトペーパー等の形で発表した。具体的には、外部要因の変化や利用者等の要求の変化、事前に想定できなかった未知の障害等に伴って、システム分析（D-Case 等）の修正・改善が行われ、システムのソフトウェアの修正・新規開発が必要となった場合に、利用者が指定した比較的詳細な仕様を、時間を掛けて検査するというモデル検査技術の特性を活かして、システム分析（D-Case）の修正から生じるソフトウェアに対する要求の修正・追加を、モデル検査器に与える仕様として可能なかぎり表現することで、プログラム開発者が行ったソフトウェアの修正が、システム分析の修正から期待される通りに行われたかどうかを検証することができる。またモデル検査器で検証できないような性質（性能要求など）やシステム全体として修正が正しく行われたかどうかの確認には、ベンチマーキングなどのテストで対応する必要がある。

次いで平成 23 年度には、前年度までに実装したモデル検査器の完成度の向上を目指

した。具体的には（型付きアセンブリ言語の処理系や C 言語から型付きアセンブリ言語への変換器と同様に）、実際のシステムソフトウェア開発者による利用を念頭に実行環境の整備を進めた。また、前年度より引き続き、このモデル検査器を用いて他研究チームの作成したディペンダビリティ支援機構やLinuxカーネルのデバイスドライバ等の検証を試みることで、実装の問題点の検出・修正を進めた。更に、前年度行った、同期ロックによって保護されたデータに関する仕様記述言語の拡張について実際に試験実装を行った。更に型検査についてと同様に、前年度までに行った予期せぬ障害（オープンシステム障害）へ対応するためのシステム運用・開発プロセス全体の中におけるモデル検査の位置付けを明確にする議論を更に進め、ロギング機構やモニタリング機構などの運用時の機構との相互補完の仕組みや、D-Caseとの連携手段について検討を行った。

本研究領域全体における上述の研究実施内容の具体的な位置付け・役割は、（型付きアセンブリ言語の処理系や C 言語から型付きアセンブリ言語への変換器と同様に）本研究領域全体が提案する、オープンシステムにおけるディペンダビリティの向上を実現するための開発・運用プロセス（DEOS プロセス）において、実際に利用可能なツールを設計・実装することである。より具体的には DEOS プロセスにおけるシステム開発・修正の段階において、開発すべき機能等を仕様としてモデル検査を行うことで、D-Case ダイアグラムにまとめられたDEOSプロセスの上流工程における合意事項がシステムに反映されていること、また予期せぬ障害（オープンシステム障害）に対しては仕様の再検討・再定義を行い、再度モデル検査を行うことによって、システム修正の迅速化・確実化をはかることである。また別の役割は、DEOSプロセスを実践するためのディペンダビリティ機構の具体的なC言語のソースコードに対してモデル検査を行うことで、API 仕様にもとづく実装の正当性の検証、ロック等の同期機構の正当性等の性質を保証・検証し、ディペンダビリティ機構自体のディペンダビリティを向上することであり、実際に前述の様に、幾つかのコンポーネントに対して検証を行い、既存の商用ツールが検出できなかったようなバグを検出することもできた。

また従来研究に対する本研究の新しい点は、検証の基本的なアルゴリズム自体は従来研究を踏襲しているものの、実際のシステムソフトウェアを検証の対象として、その仕様を形式的（プログラムで機械解釈可能な形式）に記述し、システムソフトウェアで重要となる性質（ロックの整合性や実行コンテキストの整合性など）の検証を行っている点である。また、本研究の直接の新規性というよりは本研究領域全体としての新規性であるが、ディペンダビリティを支援する機構の実装自体がディペンダブルであるかの検証を試みること自体、そもそも従来のシステムソフトウェア研究ではあまり省みられてこなかった新しい点である。

⑵研究成果の今後期待される効果

本研究では、実際に他研究チームの作成したプログラムやLinuxカーネルの幾つかのドライバの試験検証が行える程度の実用性を持つツール・仕様記述言語を実現できたが、今後の展開として、マニュアルの整備、実行性能の改善等を進めることで、一般の開発者への普及が期待できると考えられる。

またオープンシステムのディペンダビリティを向上するための DEOS プロセス・アーキテクチャとの関係における今後の展開としては、C 言語から型付きアセンブリ言語への変換器と同様、DEOS プロセス・アーキテクチャとの連携をツールレベルでより深めること、例えばD-Caseエディタ等のシステムとの連携を行うためのツール・システムを構築し、D-Caseダイアグラムと検証ツールとの間でエビデンス等のデータを自動で受け渡しするようにすることなどが考えられる。

§５ 成果物等

(1)ソフトウェア/ハードウェア

型付きアセンブリ言語の処理系・C 言語から型付きアセンブリ言語への変換ソフトウェアのプロトタイプ実装を下記 URL より公開

http://www.yl.is.s.u-tokyo.ac.jp/~tosh/dsdt/

(2)規格

なし

(3)知財出願

①国内出願 (0 件)

②海外出願 (0 件)

③その他の知的財産権

なし

(4)原著論文発表 (国内(和文)誌 0 件、国際 (欧文) 誌 5 件)

1. Takahiro Kosakai, Toshiyuki Maeda and Akinori Yonezawa, “Compiling C Programs into a Strongly Typed Assembly Language”, In Proc. of ASIAN’07, Lecture Notes in Computer Science 4846, pp. 17-32, Dec. 2007.
2. Toshiyuki Maeda and Akinori Yonezawa, “Writing an OS Kernel in a Strictly and Statically Typed Language”, Lecture Notes in Computer Science 5458, pp. 181-197, May 2009.
3. Toshiyuki Maeda and Akinori Yonezawa, “Typed Assembly Language for Implementing OS Kernels in SMP/Multi-Core Environments with Interrupts”, In Proc. of the 5th International Workshop on Systems Software Verification, Oct. 6, 2010. (URI: http://www.usenix.org/events/ssv10/tech/full_papers/Maeda.pdf)
4. Junya Sawazaki, Toshiyuki Maeda, and Akinori Yonezawa, “Implementing a Hybrid Virtual Machine Monitor for Flexible and Efficient Security Mechanisms”, In Proc. of the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2010), pp. 37-46, Dec. 14, 2010. (DOI: http://doi.ieeecomputersociety.org/10.1109/PRDC.2010.32)
5. Toshiyuki Maeda, Haruki Sato, and Akinori Yonezawa, “Extended Alias Type System using Separating Implication”, In Proc. of the 6th ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI 2011), pp. 29-42, Jan. 25, 2011. (DOI: http://dx.doi.org/10.1145/1929553.1929559)

(5)その他の著作物(総説、書籍など)

① 前田俊行, “プログラム検証によるディペンダビリティ支援”, Software Design 2010 年 2 月号, 技術評論社, 2010 年 1 月 18 日.

(6)国際学会発表及び主要な国内学会発表

① 招待講演 (国内会議 0 件、国際会議 0 件)

なし

② 口頭発表 (国内会議 14 件、国際会議 4 件)

1. 吉野寿宏(東大), 前田俊行(東大), 米澤明憲(東大), “低級言語のプログラムを検証するための共通言語の設計とプログラム変換”, The 5th JSSST SIGDSW Workshop on Dependable Systems (DSW’07 summer). Jul. 2007.

2. 前田俊行(東大), 米澤明憲(東大), “割込みに対応した型付きアセンブリ言語”, The 5th JSSST SIGDSW Workshop on Dependable Systems (DSW'07 summer). Jul. 2007.
3. 野尻隆弘(東大), 前田俊行(東大), 米澤明憲(東大), “モデル検査器を用いた依存型システムのための配列境界の推論”, 第 6 回ディペンダブルシステムワークショップ, 函館, 2008 年 7 月 2 日.
4. 藤田肇(東大), 平野貴仁(東大), 松葉浩也(東大), 前田俊行(東大),菅谷みどり(JST), 石川裕(東大), “安全かつ拡張可能な OS 開発基盤実現に向けて”, 第 6 回ディペンダブルシステムワークショップ, 函館, 2008 年 7 月 2 日.
5. 前田俊行(東大), 米澤明憲(東大), “SMP・マルチコアに対応した型付きアセンブリ言語”, 第 6 回ディペンダブルシステムワークショップ, 函館, 2008 年 7 月 4 日.
6. 松田元彦(東大), 前田俊行(東大), 米澤明憲(東大), “大規模システムソフトウェアのモデル検査器の設計と実装”, 2008 年並列／分散／協調処理に関する『佐賀』サマー・ワークショップ, 佐賀, 2008 年 8 月 6 日.
7. Motohiko Matsuda (Univ. of Tokyo), Toshiyuki Maeda (Univ. of Tokyo) and Akinori Yonezawa (Univ. of Tokyo), “Towards Design and Implementation of Model Checker for System Software”, The 1st International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009), Tokyo, Jan. 2009.
8. 野尻隆弘(東大), 前田俊行(東大), 米澤明憲(東大), “モデル検査器 SPIN における安全性検証手法の並列化”, 第 7 回ディペンダブルシステムワークショップ, 函館, 2009 年 7 月 14 日.
9. 中澤仁(慶大／CREST), 松野裕(産総研), 菅谷みどり(JST), 塙敏博(筑波大), 前田俊行(東大), 藤田肇(東大), 石綿陽一(産総研), 杵渕雄樹(早大), 高村博紀(産総研), 松田元彦(東大), 三浦信一(筑波大), 山田浩史(慶大), “オペレーティングシステムおよび実システムにおけるディペンダビリティの評価と見積り”, 函館, 2009 年 7 月 14 日.
10. 渡邊裕貴(東大), 前田俊行(東大), 米澤明憲(東大), “配列のためのホーア型理論の拡張”, 第 7 回ディペンダブルシステムワークショップ, 函館, 2009 年 7 月 15 日.
11. 前田俊行(東大), “プログラムの作成・修正を考慮したプログラム解析・検証に向けて”, 第 7 回ディペンダブルシステムワークショップ, 函館, 2009 年 7 月 15 日.
12. 加藤真平(東大), 藤田肇(東大), 中澤仁(慶大), 松田元彦(東大), 前田俊行(東大), 杵渕雄樹(早大), 塙敏博(筑波大), 三浦信一(筑波大), 石綿陽一(産総研), 松野裕(産総研), 高村博紀(産総研), 山田浩史(慶大), 吉田哲也(慶大), 倉光君郎(横国大), 菅谷みどり(JST), 石川裕(東大), “ディペンダブルシステム向けベンチマークフレームワークの提案”, 第7回ディペンダブルシステムワークショップ, 函館, 2009 年 7 月 16 日.
13. 松田元彦(東大), 前田俊行(東大), 米澤明憲(東大), “ヒープ中の同期ロックの整合性に関するモデル検査”, 第77回 情報処理学会・プログラミング研究会, 大阪, 2010 年 1 月 27 日.
14. 前田俊行(東大), 米澤明憲(東大), “メモリコンシステンシモデルを考慮した型付きアセンブリ言”, 第 8 回ディペンダブルシステムワークショップ, 函館, 2010 年 7 月 20 日.
15. Toshiyuki Maeda (Univ. of Tokyo), “Dependable System Software Development Technology”, the 2nd JFLI Workshop in Paris, France, Oct. 13, 2010.
16. 前田俊行(東大), “オペレーティングシステム検証の研究動向”, チュートリアル, 組み込みシステムシンポジウム 2010, 東京, 2010 年 10 月 27 日.
17. Hajime Fujita (Univ. of Tokyo), Motohiko Matsuda (Univ. of Tokyo), Toshiyuki

Maeda (Univ. of Tokyo), Shin'ichi Miura (Univ. of Tsukuba), and Yutaka Ishikawa (Univ. of Tokyo), "P-Bus: Programming Interface Layer for Safe OS Kernel Extensions", the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'10), Tokyo, Dec. 15, 2010.

18. Toshiyuki Maeda (Univ. of Tokyo) and Hajime Fujita (Univ. of Tokyo), "Dependability and Verification", DEOS Symposium "Towards Open Systems Dependability" and Workshop "Close Look at DEOS - Demonstration and Discussion -", Dec. 16-17, 2010.

③ ポスター発表　　(国内会議 1 件、国際会議 0 件)

1. 吉野寿宏(東大), 前田俊行(東大), 米澤明憲(東大), "メモリ管理コードの包括的検証のための階層化されたメモリモデル", The 10th JSSST SIGPPL Workshop on Programming and Programming Languages (PPL2008), Poster and Demo session. Mar. 2008.

(7)受賞・報道等

①受賞

なし

②マスコミ(新聞・TV等)報道

なし

③その他

なし

(8)成果展開事例

①実用化に向けての展開

- なし

② 社会還元的な展開活動

- ワクワク IT@あきば(平成 20 年 3 月)にて研究の進捗・成果を展示発表した
- 第 11 回組込みシステム開発技術展(平成 20 年 5 月)にて研究の進捗・成果を展示発表した
- Embedded Technology /組込み総合技術展(毎年 11 月開催)に平成 20 年から平成 23 年まで毎年出展し、研究の進捗・成果を展示発表した
- 本研究で開発したソフトウェアツールを以下の URL で公開中、または公開予定である。
  http:// http://www.yl.is.s.u-tokyo.ac.jp/~tosh/dsdt/

§6 研究期間中の主なワークショップ、シンポジウム、アウトリーチ等の活動

年月日	名称	場所	参加人数	概要
平成 21 年 9 月 4 日	平成21年度公開中間成果報告会	東京大学本郷キャンパス 小柴ホール	100 名ー	研究の中間成果の発表・一般公開
平成 22 年 12 月 16 日・17 日	シンポジウム「オープンシステムディペンダビリティに向けて」	慶應義塾大学三田キャンパス	50 名ー	研究成果の発表および海外研究者との議論

§7 結び

当初の研究構想で想定していた「実用性」は本研究において概ね達成できたと考えている。しかしその一方、研究を遂行する中で、社会一般に普及するために求められる「実用性」と当初想定していた「実用性」には想像以上のギャップがあることが認識され、研究を遂行しつつ如何にそのギャップを埋めて行くかの舵取りが隠れた課題であった。

最後に、（上述の点も含め）研究の遂行等について様々な有益なアドバイスを下さった研究総括、副研究総括、領域アドバイザーの方々、研究推進委員の方々、他研究チームの方々、DEOS 研究開発センターの方々、科学技術振興機構の方々に心より感謝申し上げます。