独立行政法人情報処理推進機構 委託

2013 年度ソフトウェア工学分野の先導的研究支援事業
「抽象化に基づいた UML 設計検証支援ツールの開発」
成果報告書

平成 26 年 2 月

公立大学法人 岡山県立大学

本報告書は独立行政法人情報処理推進機構　技術本部　ソフトウェア高信頼化センターが実施した「2013 年度ソフトウェア工学分野の先導的研究支援事業」の公募による採択を受けて公立大学法人　岡山県立大学（研究責任者　有本和民）が実施した研究の成果をとりまとめたものである.

目次

研究成果概要

１．背景

情報システムは社会インフラの重要な部分を占めており，社会生活を送るうえで必要不可欠な存在となっていることから，その障害が社会に及ぼす影響はかつてなく大きい．中でも生活家電やスマートフォンなどの通信機器，そして車載システムなどで用いられる組込みソフトウェアは，その重要性から信頼性について極めて高い品質基準が要求される一方で，複雑化・大規模化による開発コストの増加が課題となっている．

ソフトウェア開発では，開発工程の後半に進むほど手戻りによるコストが大きくなるため，設計の無矛盾性や仕様との整合性といった，設計工程における不具合検出の重要性が広く認知されている．一方で，組込みソフトウェアでは１件の不具合が社会に及ぼす影響が甚大であるため，その検証には多くのリソースが割かれることになるが，大規模・複雑化が顕著な組込みソフトウェアにおいては，システムの取り得る状態数は人手でテストを行う限界を大きく超えている．

この課題を解決するための有効な手段と考えられているのがモデル検査技術である．モデル検査ではソフトウェアの振る舞いをクリプキ構造と呼ばれる有向グラフによってモデル化した上で，グラフを網羅的に探索することにより求める特性が満たされるか否かを自動的に証明することが可能となっている．モデル検査は設計の無矛盾性や仕様との整合性の検証を完全に自動化することが可能であり，設計工程における不具合の検出に費やすコストを抑えることができる．また，モデルに対する網羅的検証を行うため，モデル化した振る舞いにおいては求める特性を満たすか否かについて完全な保証を得ることができる．

以上の理由から，組込みソフトウェア設計へのモデル検査の適用に対する産業界からの期待は極めて大きいといえる．しかしながら，モデル検査技術を組込みソフトウェアの設計検証に導入するには，種々の問題点が指摘されており，中でも２つの大きな問題点がある．

まず，第一の問題点として，モデル作成の困難さが挙げられる．モデル検査による設計検証では，ソフトウェアの設計文書をモデル検査ツール固有のモデル化言語で記述する必要がある．しかしながら，モデル化言語の文法や意味論は，組込みソフトウェアの設計技術者が通常使用しているUML図などの設計文書の記述方式とは大きな隔たりがあるため，設計からモデルを生成するには専門的な知識やノウハウが必要となる．

第二の問題点は，状態爆発の危険性である．組込みソフトウェアの設計記述は大規模かつ複雑となる．モデル検査ではモデルの状態空間の網羅的探索を行うため，設計記述を全てモデル化しようとした場合，探索する状態数が爆発的に増加することにより，現実的な時間内で検証が完了しない恐れがある．

これらの問題が，モデル検査を設計検証へ導入する上での大きな障壁となっている．

２．目的

これらの問題はいずれも設計記述から検証モデルを作成する段階において発生する．したがって，モデル検査による設計検証を行う際に，検証モデルの作成をツールによって支援することで，これらの問題を解決し，モデル検査の導入における障壁を低減できると期

待される．

本研究では，設計記述から検証モデルを自動的に作成するための検証支援ツールの開発を行う．対象とする設計記法としては，組込みソフトウェアの開発において広く利用されている形式仕様記法の 1 つである UML を想定する．また，モデル検査ツールとして，NuSMV を用いている．NuSMV は，モデル記述言語である SMV 言語の表現力および検証速度に優れたモデル検査ツールである．したがって，本研究の目的は，以下の 3 つの機能をもつ検証支援ツールを開発することとなる．

機能１．SMV 言語への自動変換を目的とした UML 図の抽象化

機能２．モデルサイズ削減を目的とした，UML 図の抽出，分割，および抽象化

機能３．UML 図から SMV 言語への自動変換

３．研究概要

前述の機能を実現するため，岡山県立大学情報工学部において，ソフトウェア工学を専門とする 4 名の研究者で研究体制を構築した．具体的には，主に，機能 1 を有本和民教授が，機能 2 を佐藤洋一郎准教授および天嵜聡介助教が，機能 3 を横川智教助教が中心となって担当し，研究責任者である有本教授が研究全体を統括することにより，研究開発を実施した．

3 つの機能を実現するにあたり，7 つの研究目標を設定した上で研究開発を行った．

機能 1 について，UML の記法の中には複雑な振る舞いを記述するためのものが存在するが，これらをモデル化言語で記述したときの記述量は著しく大きくなるため，自動変換のコストも大きくなる．しかしながら，ソフトウェア設計者が必ずしもこれらの記法を厳密な意味論に基づいて利用しているとはいえず，設計の解釈に曖昧性が含まれる可能性がある．そこで本研究では，対象とする UML 図の記法の中で，曖昧な解釈を引き起こす可能性の高いものに制約を課すことで，曖昧性の混入を回避する．さらに，制約に沿った記述を支援するため，記述の制約に沿わない箇所については，どのように記述を修正すればよいかという候補の提示や，部分的な修正の自動化を行う．したがって，機能１を実現するための研究目標は以下の 2 つとなる．

研究目標１．自動変換を行う UML 図に対する制約定義書の作成

研究目標２．UML 図の制約違反検出および抽象化手法の開発

機能 2 について，大規模かつ複雑な UML 図をモデル化した場合，検証モデルの状態数が爆発的に増加し，網羅的探索による検証は困難となる．そこで，検査対象システムから検証すべき特性に関連した情報のみを抽出し，モデルの抽象化を行うことで，検証コストを削減する．検証特性に基づいた抽出・分割および抽象化を行うためには，設計が満たすべき仕様を適切な形で検証特性としてツールに与える必要がある．そこで，設計が満たすべき仕様についてテンプレートを用意して入力を定型化することにより，設計者が検証したい特性を容易に入力できるようにする．さらに，仕様に関連する UML 図の要素を明示的に示すようテンプレートを設計することで，仕様に基づく UML 図の抽出・分割および抽象化を効率的に行うことを可能とする．したがって，機能 2 を実現するための研究目標は以下の 3 つとなる．

研究目標３．仕様の入力テンプレートの開発
研究目標４．仕様に基づく UML 図の部分抽出手法の開発
研究目標５．仕様に基づく UML 図の記述抽象化手法の開発

機能 3 について，モデル検査を行うためには，検査対象システムをモデル化言語で記述する必要がある．しかしながら，NuSMV のモデル化言語である SMV 言語の文法や意味論は，検査対象となる UML とは大きく隔たりがある．そこで本研究では，UML 図からその振る舞いを表現する SMV 言語によるモデルへの自動変換を行う．機能 3 および本研究が目的とする検証支援ツール（以下，本ツール）を実現するための研究目標は以下の 2 つとなる．

研究目標６．UML 図および仕様から SMV 言語への自動変換手法の開発
研究目標７．検証支援ツールの入出力インタフェースの作成

以降では，各機能について成果の概要をまとめる．

４．SMV 言語への自動変換を目的とした UML 図の抽象化

機能 1 について，まずは研究目標 1 として自動変換を行う UML 図に対する制約定義書の作成を行った．始めに，先行研究における UML 図の形式的検証手法における制約のリストアップを行った．これまでに我々は，モデル検査を用いた状態マシン図およびシーケンス図の形式的検証に関する研究開発を行ってきた．先行研究における状態マシン図およびシーケンス図に対する記述制約は表 1 に示すとおりである．

表 1:先行研究における UML 図に対する記述制約

		状態マシン図		シーケンス図	
	検証ツール	階層構造	変数	階層構造	生存期間
[HARADA 2009]	SMV	×	×	×	×
[KAWAKAMI 2010]	SMV			△	×
[NIMIYA 2010]	Alloy	△	○		
[ASADA 2011]	LTSA			△	×
[MIYAZAKI 2012]	LTSA			△	×
[YOKOGAWA 2013]	LTSA	×	×	×	×
[片山 2013]	CSP,FDR	○	×	△	×

その上で，先行研究における UML 図の制約の評価および本ツールで採用する制約定義についての検討を行った．表 1 に示すように，UML の状態マシン図およびシーケンス図を対象とした形式的検証を実施する上で定める制約は，階層構造を扱うか否かで大きく分類される．また，状態マシン図については変数を扱うか否か，シーケンス図についてはオブジェクトの生存期間を扱うか否かによって分類される．これまでに取り組んできた UML 図を対象とした形式的検証に関する研究開発から得られた知見として，階層構造を持つ状態マシン図では，実行される遷移の選択について複雑な意味論を持つため，モデルが非常に複雑になりやすい．そのため，SMV 言語によるモデルを生成するための手続きも複雑になることから，本ツールにおいては状態マシン図の階層構造は処理の対象外とする．シーケンス図の階層構造に関して，SMV を用いた先行研究での手法を応用することで階層構造の一部には対応可能であると考えられる．しかしながら，上述のように本ツールでは状態マシ

ン図の階層構造は扱っておらず，階層構造による記述が不要なシステム開発の初期段階での利用を想定している．したがって，シーケンス図の階層構造も状態マシン図と同様に本ツールの処理対象外とする.ただし,相互作用参照は記述の簡略化のためのみに用いられ，参照記述を除外した記述との相互変換も容易であるため，本ツールの処理対象内に含めるものとする．変数については，我々がこれまでに取り組んでいたモデル検査ツール Alloy を用いた設計検証に関する先行研究と同様のアプローチでモデル化が可能であると考えられるが，SMV では扱う変数の型に制約があるため，その制約に基づき，利用できる変数は整数型およびブール型のみに限るものとする．また，シーケンス図の生存期間については先行研究と同様に本ツールでは処理対象外とする．ただし，オブジェクトの生成・終了メッセージについては，他のメッセージと同様に処理を行う．最後に，状態マシン図およびシーケンス図の記述制約を制約定義書として文書化することで，研究目標 1 を達成している．

次に研究目標 2 として UML 図の制約違反検出および抽象化手法の開発を行った．まず既存の UML モデリングツールから，本ツールの入力となる UML 図を作成するためのツールを選定した．UML モデリングツールは数多くのものが公開されているが，そのうち astah* professional や Enterprise Architect など代表的なものを 5 つピックアップし，検討を行った．UML 図の記法への対応状況，出力データ形式の処理の容易さ，ユーザインタフェース，そして導入コストという観点から検討した結果として，本研究ではモデリングツールとして astah* community を採用することにした．そして，選定した UML モデリングツールによって作成されるデータ構造の解析を行った．astah* community で作成したモデリングデータは，Java 環境で利用可能な astah* API を用いることで処理を行うことが可能である．astah* API では状態マシン図の情報は IStateMachineDiagram クラスのオブジェクトとして取得できる．各状態マシンの情報はメソッド getStateMachine()などを用いて取得できる.同様にシーケンス図については相互作用に関する情報を IInteraction クラスのオブジェクトとして取得でき，メソッド getLifelines()などを用いることでライフライン上の結合フラグメントやメッセージの情報を取得できる．さらに，UML モデリングツールによって作成されるデータから，制約違反に対応した部分を抽出するための手法を開発した．上述のとおり，astah* API では UML 図の情報を直接取得できるため，取得した情報に対して，状態マシン図およびシーケンス図の制約定義に違反するか否かを判定し，違反していた場合はモデリングデータ内の情報の該当部分にその旨の印を付加することで違反部分の抽出が可能となる．また，制約に違反する部分に対して，制約を満たすような修正候補を提示するためのアルゴリズムを開発した．修正候補提示アルゴリズムは，(1) UML 図読み込み機能，(2)制約違反検出機能，(3)違反箇所提示機能，そして(4)修正候補提示機能の 4 つから構成されている．UML 図読み込み機能はユーザが UML モデリングツール astah* community によって記述した UML の状態マシン図およびシーケンス図の情報を astah* API を用いて読み込む機能である．制約違反検出機能は UML 図読み込み機能によってモジュール内部のデータ構造に格納された状態マシン図およびシーケンス図の情報を読み込み，制約違反の有無をチェックする機能である．違反箇所提示機能は，制約違反検出機能によってモジュール内部のデータ構造に格納された状態マシン図およびシーケンス図の「違反有

りの印」の情報を読み込み，それぞれの UML 図内の違反箇所を特定して，ユーザにメッセージで提示する機能である．修正候補提示機能は制約違反検出機能によってモジュール内部のデータ構造に格納された状態マシン図およびシーケンス図の「違反有りの印」の情報を読み込み，制約違反をしている記法の修正候補を特定して，ユーザにメッセージで提示する機能である．最後に，上述のアルゴリズムを実行する機能モジュールである，修正候補提示モジュールの設計書を作成することで，研究目標 2 を達成している．

５．モデルサイズ削減を目的とした UML 図の抽出・分割および抽象化

機能 2 について，まずは研究目標 3 として仕様の入力テンプレートの作成を行った．まずは先行研究で提唱された仕様パターンについて調査を行った．仕様パターンは，自然言語で記述された検査項目をいくつかの定められた型へと当てはめることにより，検査式を容易に作成できるよう考案されたものである．次に，特に仕様頻度が高いパターンについて，そのパターンに対応した仕様を記述するためのテンプレートを開発する．本ツールで用いる仕様テンプレートは，検査特性として頻繁に利用される安全性・活性・到達可能性などの特性を容易に記述可能とすることを目的として開発する．仕様パターンの表現能力は非常に高く，これらの特性も仕様パターンを用いて記述が可能であるが，仕様パターンを有効に活用するためには時相論理などの専門的な知識が求められる．そのため，本ツールでは，利用頻度が高い安全性・活性・到達可能性の 3 つの特性を記述するための仕様テンプレートを作成する．作成した仕様テンプレートを表 2 に示す．表 2 に示すように，本研究では，状態・メッセージ・変数という UML 図の重要な要素に関する安全性・活性・到達可能性の 3 つの特性を記述するためのテンプレートを開発し，ユーザに提供する．

表 2:仕様テンプレート

NO	特性	要素	式表現	意味
1	安全性	状態	SAFE(s)	決して状態 s はアクティブにならない
2		メッセージ	SAFE(m)	決してメッセージ m は送信されない
3		変数	SAFE(x,a)	決して変数 x の値が a とならない
4	活性	状態	LIVE(s)	いつか必ず状態 s がアクティブとなる
5		メッセージ	LIVE(m)	いつか必ずメッセージ m が送信される
6		変数	LIVE(x, a)	いつか必ず変数 x の値が a となる
7	到達可能性	状態	REACHABLE(s)	状態 s がアクティブになる可能性がある
8		メッセージ	REACHABLE(m)	メッセージ m が送信される可能性がある
9		変数	REACHABLE(x,a)	変数 x の値が a となる可能性がある

次に研究目標 4 として，仕様に基づく UML 図の部分抽出手法の開発を行った．まず，仕様と UML 図の各要素との関連度の計算式の定義を行った．関連度は，状態に関する要求仕様に対して，入力された状態マシン図の状態および遷移に対して定められる．ここで，要求仕様が参照する状態 s に対して，s が属する状態マシン図を M とする．関連度は，s との関連度が最も高い状態および遷移を関連度 1，最も低いものを関連度 4 として，以下のよ

うに定義される.

- M に含まれる状態の中で遷移によって s へと到達可能である状態および到達するまでに辿る遷移を，関連度 1 とする.
- M に含まれる状態および遷移で，1.に含まれないものを，関連度 2 とする.
- M とメッセージ名や変数名を共有する状態マシン図に含まれる状態および遷移を，関連度 3 とする.
- M とメッセージ名や変数名を共有しない状態マシン図に含まれる状態および遷移を，関連度 4 とする.

そして，関連度に基づく部分抽出アルゴリズムの開発を行った．本アルゴリズムでは，ユーザが関連度の閾値を入力し，その閾値より関連度が高い状態および遷移のみを状態マシン図から抽出して得られた状態マシン図を出力として生成する．そして最後に，部分抽出アルゴリズムを実行する機能モジュールである,部分抽出モジュールの設計書を作成し，研究目標 4 を達成している.

最後に研究目標 5 として,仕様に基づく UML 図の記述抽象化手法の開発を行った.まず，仕様と UML 図の要素との関連性を判定するアルゴリズムの開発を行った．関連性は，変数に関する要求仕様に対して,入力された状態マシン図の変数に対して定められる.ここで，要求仕様が参照する変数を x とする．関連性は，x との関連性があるかないかの 2 値として，以下のように定義される.

- x の値を変化させるアクション，すなわち代入文の右辺に含まれる変数は x と関連性があるとする.
- x と関連性がある変数 y に対して,y と関連性がある変数も同様に x と関連性があるとする.

そして，テンプレートで記述された仕様と状態マシン図の変数との関連性を判定するアルゴリズムを実行する機能モジュールである,関連性判定モジュールの設計書を作成した.次に，関連性判定の結果に基づく UML 図の抽象化アルゴリズムの開発を行った．本アルゴリズムでは，関連性判定の結果に基づき，関連性がないと判定された変数およびその変数が現れるガード条件およびアクションを状態マシン図から削除して得られた状態マシン図を,出力として生成する.最後に,抽象化アルゴリズムを実行する機能モジュールである，抽象化モジュールの設計書を作成し，研究目標 5 を達成している.

６．UML 図から SMV 言語への自動変換

機能 3 について，研究目標 6 として UML および仕様から SMV 言語への自動変換手法の開発を行った．まず，UML 図および仕様テンプレートによる記述と SMV 言語の対応関係の整理を行った．SMV 言語で記述されたモデルは変数宣言部，状態遷移系記述部，そして検査式記述部から構成される．変数宣言部では検査対象となるモデルの要素を変数として宣言する．状態遷移系記述部では，モデルの各要素が他の要素の値に基づいて定義される遷移条件に依存してどのように変化するかを記述する．そして，検査式記述部では，検査式を時相論理の 1 つである CTL によって記述する．UML の状態マシン図は，状態および遷移によって構成されている．まず，これらの要素をどのように SMV 言語によってモデル化する

かについて検討を行った．状態マシン図の振る舞いをモデル化する場合，SMV 内で変数として表すべき要素は，状態，メッセージ，そして変数の 3 つである．遷移の発火によるこれらの要素の値の変化を，SMV 内の遷移関係として記述することで SMV による状態マシン図のモデル化が可能となる．次に，シーケンス図と SMV 言語との対応関係について検討を行った．シーケンス図は，ライフラインとその間で実行されるメッセージ通信によって構成されている．本ツールでは，シーケンス図で記述されているメッセージに着目し，状態マシン図がそのメッセージを正しく実行しているか否かについて検証を行う．したがって，SMV 内で変数として表すべき要素はメッセージのみとなる．シーケンス図で記述されたメッセージの振る舞いは，SMV が検証する検査式として記述する．最後に，仕様テンプレートと SMV 言語との対応関係について検討を行った．仕様テンプレートは，SMV によって検証する検査性質を記述するためのものである．したがって，仕様テンプレートによって記述される仕様は，SMV 言語によるモデルの中では，CTL 式や LTL 式によって記述される検査式に対応する．本ツールにおける仕様テンプレートで記述する検査特性である，安全性，活性，そして到達可能性の 3 つの特性は，それぞれ CTL の演算子によって表現することができる．まず安全性は，CTL における演算子である，AG(Always Globally，全ての場合において常に～が成り立つ)を用いて表現できる．次に活性は，同じく CTL における演算子である AF(Always in the Future，全ての場合において将来いつか～が成り立つ)を用いて表現できる．そして到達可能性は，CTL における演算子である EF(Eventually in the Future，ある場合において将来いつか～が成り立つ)を用いて表現できる．次に，状態マシン図，シーケンス図，そして仕様テンプレートから SMV 言語への自動変換を行うアルゴリズムの開発を行った．本ツールでは，状態マシン図の情報から SMV 言語によるモデルを生成し，シーケンス図および仕様テンプレートの情報から検査式を生成する．状態マシン図およびシーケンス図のモデルデータは UML モデリングツール astah*によって作成される．したがって，このアルゴリズムの概要は図 1 に示す通りである．

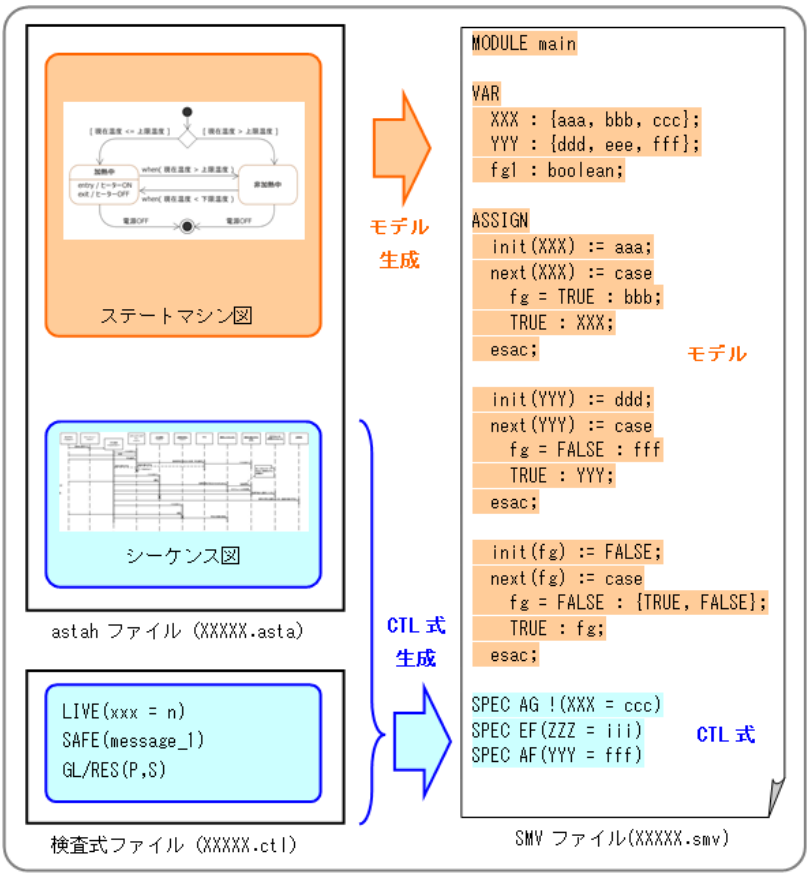


図 1:SMV プログラムへの自動変換アルゴリズムの概要

まず，状態マシン図について，本アルゴリズムでは，状態マシン図から SMV プログラムへの変換を(1)状態マシン図の記法の等価変換(2)状態マシン図から SMV 言語によるモデルの生成の 2 段階で行う．(1)では，状態マシン図の擬似状態（初期擬似状態，終了状態，選択擬似状態，ジャンクション擬似状態）を，等価な単純状態による表現へと変換する．そして，単純状態と遷移経路のラベル（トリガ，[ガード]，アクション）のみが存在するような状態マシン図へと変換した上で，(2)で SMV 言語によるモデルを生成する．シーケンス図について，本アルゴリズムでは，与えられたシーケンス図に対して，シーケンス図に記載されたメッセージの振る舞いに関する CTL 式を生成する．具体的には，シーケンス図に記載されたメッセージの安全性，活性，そして到達可能性を確認するための CTL 式を生成する．そして，仕様テンプレートについて，本アルゴリズムでは仕様テンプレートに基づいて記述された検査項目をもとに CTL による検査式を生成する．最後に，自動変換アルゴリズムを実行する機能モジュールである，自動変換モジュールの設計書を作成し，研究目標 6 を達成している．

7．検証支援ツールの開発

最後に，本研究の目的を達成するため，研究目標 7 として検証支援ツールの開発を行った．まず，これまでに作成した機能モジュールの設計書を元に，ツール作成を外注するための仕様書を作成した．これらの機能モジュールを統合した，検証支援ツールの仕様書を作成した．本研究で開発する検証支援ツールの概要を図 2 に示す．

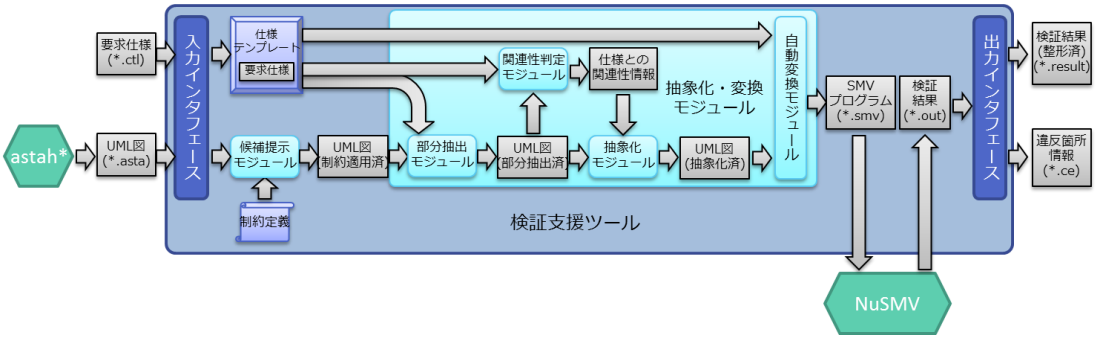


図 2:検証支援ツールの概要

機能モジュールについて，まず候補提示モジュールの設計書をもとに仕様書を作成し，その上で 8 月上旬に開発業者への発注を行い，9 月中旬に納品及び検収を完了した．部分抽出モジュール，関連性判定モジュール，抽象化モジュール，そして自動変換モジュールの 4 つについては抽象化・変換モジュールという 1 つの機能モジュールとして仕様書を作成した．その上で，11 月上旬に開発業者への発注を行い，12 月上旬に納品および検収を完了した．そして，これらの機能モジュールを統合して検証支援ツールを完成するため，入出力インタフェースの設計を行った．本ツールの入力インタフェースは，astah* community で作成された UML の状態マシン図とシーケンス図のモデリングデータ(*.asta)と，仕様テンプレートに基づいて記述された要求仕様(*.ctl)を読み込むことが可能である．*.ctl ファイルはテキストファイル形式で作成するものとする．次に，本ツールは抽象化・変換モジュールが生成した SMV プログラム(*.smv)をモデル検査ツール NuSMV へと入力して検証を行う．NuSMV による検証結果は検証結果ファイル(*.out)として保存される．*.out ファイルはテキストファイル形式で保存されるものとする．本ツールの出力インタフェースは，NuSMV が出力した検証結果ファイルを読み込み，整形済みの検証結果ファイル(*.result)を出力する．また，もし検査式が満たされなかった場合は，NuSMV が生成した反例を違反箇所情報(*.ce)として出力する．検証支援ツールについては 12 月下旬に開発業者への発注を行い，1 月上旬に納品および検収を完了し，これにより研究目標 7 を達成している．

8．まとめ

以上のように，本研究では以下の 3 つの機能をもつ検証支援ツールの開発を行った．

機能 1．SMV 言語への自動変換を目的とした UML 図の抽象化

機能 2．モデルサイズ削減を目的とした，UML 図の抽出，分割，および抽象化

機能 3．UML 図から SMV 言語への自動変換

この検証支援ツールの実現により，組込みソフトウェア開発における設計検証にモデル検査を導入する上での諸問題の解決に寄与し，開発コストの削減およびソフトウェアの信頼性向上が期待される．さらに，産業界へのモデル検査技術の普及も期待される．今後の課題は，記述制約の緩和などの検証支援ツールの機能拡張ならびにツールのインタフェースの改良を行い，実際の開発現場への普及を進めることである．

1 研究の背景および目的

1.1 背景

計算機の能力向上と低廉化により，社会のあらゆる場面で情報システムが活用されるようになっており，我々の社会生活は情報システムが提供するサービスから多大な恩恵を受けている．それに伴い，情報システムが活用される多くの場面において情報システムの重要性は一段と増してきているといえる．

現在の情報システムは，社会インフラの重要な部分においても活用されており，我々が社会生活を送るうえで必要不可欠な存在となっている．そのため，情報システムの障害が我々の社会生活に及ぼす影響はかつてなく大きい．近年の事例では，銀行の ATM システムの障害によって料金の引き落としや口座振替ができなくなり，大きな混乱が引き起こされた．

情報システムは社会の要請に応えるかたちで高度化しており，これに伴ってその構造は複雑化・大規模化の一途を辿っている．現在の情報システムは，その重要性から信頼性についてかつてなく高い品質基準が要求される一方で，複雑化・大規模化が進んでいることから，どのように信頼性を保証するかが大きな課題となっている．

情報システムを構成するソフトウェアの開発においては，複雑化・大規模化による開発コストの増大が課題となっている．クリティカルな分野のソフトウェアでは１件の不具合が社会に及ぼす影響が甚大であるため特に品質が重要視される．しかしながら，その複雑さや規模のため設計上の一貫性などを担保することが難しく，開発工程の後半で多くの問題が発覚することが多々ある．そして，この問題に対応するための手戻りによるコストが発生している．

ソフトウェア開発における修正のコストは開発工程の後半に進むほど大きくなるといわれている．そのため，開発工程の前半の成果物の品質を確保することがコストの増大を防ぐために有効である．このため，ソフトウェアの設計の無矛盾性や仕様との整合性を確保する活動の重要性が広く認知されてきている．

情報システムの中でも，大規模･複雑化が顕著な組込みソフトウェアにおいては，システムの取り得る状態数は人手でテストを行う限界を大きく超えている．このようなソフトウェアの設計の無矛盾性や仕様との整合性といった設計工程における不具合を検出することが開発現場における喫緊の課題である．

この課題を解決するための有効な手段と考えられているのがモデル検査技術である．モデル検査技術はソフトウェアの振る舞いをクリプキ構造と呼ばれる有向グラフによってモデル化する．その上で，グラフを網羅的に探索することにより，求める特性が満たされるか否かを自動的に証明することが可能となっている．モデル検査技術は設計の無矛盾性や仕様との整合性の検証を完全に自動化することが可能であるため，設計工程における不具合の検出に費やすコストを抑えることができる．また，人間には全体を詳細に把握することが困難となるような大規模な設計に対しても適用可能である．

以上の理由から，ソフトウェア設計へのモデル検査の適用に対する産業界からの期待は極めて大きい．

1.2 研究課題

モデル検査技術はソフトウェアの設計を網羅的に検証し，設計の無矛盾性や仕様との整合性を確認するために有効である．また，モデル検査技術によるソフトウェア設計の検査について産業界からの期待も大きい．このような状況にも関わらず，組込みソフトウェアの設計検証にモデル検査技術を導入した事例は多く報告されていない．一部企業での導入事例は存在するが，産業界からの期待の大きさに比してモデル検査による設計検証を導入している企業の数は少ない．

モデル検査を設計検証に導入している事例が少ない原因として，以下の2つの問題点が指摘されている．

問題1．モデル作成の困難さ

モデル検査による設計検証では，モデル検査ツールの入力形式に沿った形でソフトウェアの設計文書を変換しなければならない．一般にモデル検査ツールの入力はそのツールに固有のモデル化言語で記述する必要がある．例えば，SMV であれば SMV 言語と呼ばれるモデル化言語が，SPIN であれば Promela と呼ばれるモデル化言語が用いられる．

モデル検査ツール特有のモデル化言語の使用が要求されるため，モデル検査を設計検証に使用するためには，ソフトウェアの設計技術者はその言語に習熟する必要がある．他のプログラミング言語や設計の記述方法と同様に，モデル化言語の習熟には実践を交えた十分な時間が必要となる．

以上に加えて，モデル検査ツール特有のモデル化言語の記述方法は，組込みソフトウェアの設計技術者が通常使用している UML 図などの設計文書の記述方法と大きな隔たりがある．UML 図などの設計文書の記述方法は実務的な利便性に重きが置かれており，記述上の意味論について詳細が定められていない．一方で，SMV 言語などで採用されているモデル化言語は意味論が厳密に定められており，その記述方法もその厳密さを反映したものとなっている．そのため，モデル検査ツール特有のモデル化言語の習得は通常のプログラミング言語などの習得よりも難しい．

さらに，モデル検査ツール特有のモデル化言語に習熟したとしても，組込みソフトウェアの設計で通常使用している UML 図などからモデルを生成するには専門的な知識やノウハウが必要となる．

このようなモデル作成に伴う困難がモデル検査による設計検証の導入が進まない理由のひとつである．

問題2．状態爆発の危険性

現在，大規模・複雑化が顕著な組込みソフトウェアにおいては，システムの取り得る状態数は人手でテストを行う限界を超えている．この状況がソフトウェアの設計の無矛盾性や仕様との整合性の検証にモデル検査を導入したい動機となっている．その一方で，大規模・複雑化したソフトウェアの設計を単純にモデル化言語で変換してモデル検査ができるほど計算機の能力は進化していない．

モデル検査ではモデルの状態空間の網羅的探索を行うため，大規模化・複雑化したソフトウェアの設計記述を全てモデル化しようとした場合，探索する状態数が爆発して現実的な時間内で検証が完了しない恐れがある．

モデル検査の専門家はソフトウェアの設計意図を読み取り，適切に設計を抽象化することで巧みに状態爆発を回避している．しかしながら，組込みソフトウェアの設計技術者がこのような技術やノウハウを身に着けるには長期間のトレーニングが必要となる．

このように，大規模化に伴う状態爆発の発生に対処することの困難さがモデル検査による設計検証の導入を妨げている．

これらの問題は，モデル検査の専門家でない組込みソフトウェアの設計技術者が，必要最低限の知識で適切にモデルを作成するにはどうすればよいのか，といった問いに集約できる．

本研究では，この問いへの答えとして，モデル検査による設計検証を行う上での，モデル作成を支援するツールを開発し，これらの問題の解決を目指す．

1.3 研究の意義

1.1 背景および 1.2 研究課題に記したとおり，情報システムの高度化に伴ってその構造は複雑化・大規模化の一途を辿っており，ソフトウェアの開発コストをどのようにして抑えるのかが課題となっている．ソフトウェアの不具合修正に要するコストは開発工程の後半に進むほど大きくなるため，上流工程での設計検証がコスト削減には有効である．ソフトウェアの自動検証技術であるモデル検査を用いることで，設計工程における不具合検出コストを削減できると期待されている．

組込みソフトウェアの開発においては，形式手法の一種である形式仕様記法の利用が広まりつつある．このような形式仕様記法の導入を契機として，形式的に記述された設計を対象としたモデル検査の適用への試みもなされている．例えば，富士ゼロックスにおいて，形式仕様記法の 1 つである UML でモデル化された複合機の組込みソフトウェアの設計に対してモデル検査を適用した事例が報告されている．

しかしながら，現在報告されている研究成果では，モデル検査ツールの入力モデルへの変換手続きが確立しているのは UML の記法のごく一部のみである．また，これらの手法では，大部分が人手によるモデル記述を行っており，変換の自動化を達成しているとはいえない．この点は，大規模化・複雑化している組込みソフトウェアの設計検証にとっては大きな問題である．大規模化による影響については，さらに，状態爆発をどのように抑えていくのかといった課題があるが，これらの提案手法では，設計の大規模化に対する対策はまだ十分とはいえない．

以上のような状況において，本研究に取り組むことは次のような意義をもつ．

1．設計検証の半自動化による開発者への負荷の軽減

従来の技術では，設計検証の完全な自動化は達成されていない．変換を自動化するための前処理は完全に人手で行っている．本研究では，大規模化・複雑化している組込みソフトウェアの設計検証の現場での利用しやすさを狙い，従来人手で行われていた前処理を半自動化できるような手法の開発に取り組む．ここで半自動化とは，従来の労働集約的な作業を定型化することにあたる．半自動化によって，モデル検査の適用における開発者への負荷が大幅に軽減できると考えられる．

2．モデルサイズ削減による状態爆発の回避

現在，大規模・複雑化が顕著な組込みソフトウェアにおいては，システムの取り得る状態数は人手でテストを行う限界を超えている．このことは，単純なモデル検査技術の適用では開発者のニーズを満たせない可能性を示唆している．本研究では，モデル検査の専門家でなくても状態爆発を回避するために，UML 図の抽象化や仕様と関連する部分の抽出・分割を自動で行う手法の開発に取り組む．大部分の処理を自動化できるため，大規模・複雑な設計に対しても漏れなく抽象化が可能となるため，モデルサイズの削減が容易となり，状態爆発を回避できると考えらえる．

本研究で提案する検証支援ツールがもたらす最も大きな効果は，設計検証に対するモデル検査の適用可能性の向上である．前述の通り，設計のモデル化の困難さと大規模化に伴う状態爆発のため，実際の開発現場においては，設計検証へのモデル検査の適用は進められていなかった．本ツールの実現はこれらの問題を解決し，その成果としてモデル検査の適用可能性を大きく向上させることが期待される．

さらに，本研究は，(1)適用範囲の拡大と(2)抽象化の効果の向上という，2つの方向性での発展が考えられる．これらの発展により，UML の自動変換および自動検証を行うための環境が整備され，UML の利便性の向上も期待される．

また，産業界において，本研究の成果により，設計検証に対して形式手法を適用する事例が大きく増加することが期待される．これにより，ソフトウェア開発プロジェクトにおける開発コストの削減ならびに作成されるソフトウェアの信頼性向上という効果が得られるとともに，適用事例やユーザの増加に伴う形式検証分野のさらなる発展が期待される．

2 実施内容

2.1 到達目標

組込みソフトウェアの開発においては，形式仕様記法の 1 つである UML が広く利用されている．そこで，UML を用いたソフトウェア設計に対してモデル検査による形式的検証を適用することにより，人手では発見が困難な誤りであっても容易に検出ことができ，設計コストを大幅に削減することが可能となる．さらに，上流工程で誤りを検出することで，下流工程でのバグ検出による手戻りの防止も期待できる．

本研究の到達目標は，組込みソフトウェア開発プロジェクトの設計工程において①UML 図で記述されたソフトウェア製品の設計図を解析し，②当該製品において検証したい特性に関連する設計情報のみを抽出し，③モデル検査ツールの入力形式に自動で変換・出力する，検証支援ツール（図 2-1）の実現である．公開されているモデル検査ツールは種々存在するが，本研究ではモデル記述言語の表現力および検証速度に優れた検査器である NuSMV を用いるものとする．したがって，①については UML 図を NuSMV のモデル記述言語（以下，SMV 言語という）を用いてモデル化することを前提として，モデルの自動生成を効率的に行うための UML 図への制約定義および制約違反が存在するときはそれを解消する機能によって実現される．②については NuSMV による検証コストを削減するため，検証すべき特性である要求仕様に基づいて UML 図から特性に関連しない部分を抽象化する機能として実現される．③については，抽象化を施した UML 図を SMV 言語による記述へと自動変換する機能として実現される．

すなわち，本研究の到達目標は，以下の 3 つの機能をもつ検証支援ツールを開発することとなる．

- 機能1．SMV 言語への自動変換を目的とした UML 図の抽象化
- 機能2．モデルサイズ削減を目的とした，UML 図の抽出，分割，および抽象化
- 機能3．UML 図から SMV 言語への自動変換

これらの3つの機能とそれを実現するための研究目標について以下に述べる．

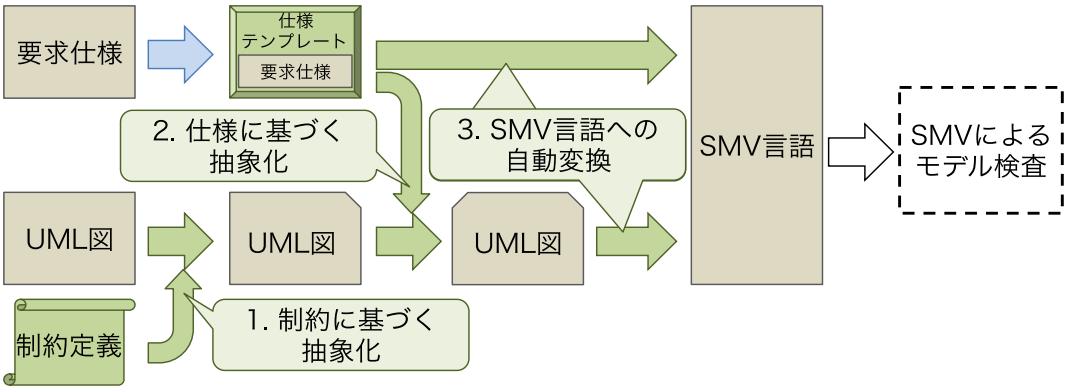


図 2-1：検証支援ツールの枠組み

2.1.1 SMV 言語への自動変換を目的とした UML 図の抽象化

組込みソフトウェア開発プロジェクトでは，製品の設計文書として UML 図がよく用いられる．大

規模化・複雑化した組込みソフトウェアの詳細な動作を記録するためには，特に状態マシン図とシーケンス図が有用である．しかしながら，状態マシン図やシーケンス図はその記述のしやすさから広く普及しているが，厳密な意味論についてよく理解されないまま用いられることで，同じ記述であっても設計者によって解釈が異なる場合がある．モデル検査を UML 設計検証へと適用する上では，このような UML 図の解釈の曖昧さに対処する必要性がある．

本研究では UML によって記述された組込みソフトウェアの設計を NuSMV によって自動検証することを目指しているため，UML 図の解釈の曖昧さへの対処は自動化することが望ましい．しかしながら，UML には多種多様な記法が含まれるため，全ての記法に対処することは難しい．したがって，本研究では現実的な解決方法として，対象とする UML 図の記述に制約を課すことで，曖昧な解釈を引き起こす可能性の高い記述が混入することを回避する．

ここで，UML 図の記述に制約を課すことで，設計者が制約に従って記述を行うために新たなコストが発生する恐れがある．また，記述された UML 図が制約を満たしていることを確認する手段が必要となる．これらの問題点については，まず，UML のもつ記述のしやすさという利点を失わないように配慮した上で制約を定義する．さらに，ソフトウェアの設計者が制約に沿った記述を行うことを支援すべく，記述の制約に沿わない箇所について，どのように記述を修正すればよいかという候補の提示や，部分的な修正の自動化により対処する．

以上の課題を考慮した上で，機能1を実現するため，以下の2つを研究目標として設定した．

研究目標1．自動変換を行う UML 図に対する制約定義書の作成

研究目標2．UML 図の制約違反検出および抽象化手法の開発

2.1.2 モデルサイズ削減を目的とした，UML 図の抽出・分割および抽象化

大規模化・複雑化した組込みソフトウェアの開発では，記述される UML 図も同様に大規模かつ複雑なものとなる．UML 図の規模や複雑さが大きかったとしても SMV 言語によってモデル化することは原理的には可能であるが，モデルの状態爆発により，モデル検査による検証を実施することには困難が伴う．

このような状況への対処手段として，検査対象システムから検証すべき特性に関連した情報のみを抽出することで，モデル検査が実施可能な規模にまでモデルの抽象化を行う．しかしながら，抽象化や部分抽出処理を適切に行うためには，対象システムとモデル検査技術の双方について専門的な知識を要する．

そこで，本研究で実現する検証支援ツールでは，状態爆発に対処するための UML 図の抽出・分割および抽象化処理を自動化する．検証特性に基づいた抽出・分割および抽象化を自動的に行うためには，設計が満たすべき仕様を適切な形で検証特性としてツールに与える必要がある．ここで，ソフトウェアの設計者は設計のどの箇所がどのような仕様を満たすべきかについては明確な要求を持っているが，その要求をモデル検査に適した形式で示すことは困難である．

以上の点については，設計が満たすべき仕様についてテンプレートを用意して入力を定型化することにより，設計者が検証したい特性を容易に入力できるようにする．さらに，仕様に関連する UML 図の要素を明示的に示すようテンプレートを設計することで，仕様に基づく UML 図の抽出・分割および抽象化を効率的に行うことを可能とする．

以上に述べた点を考慮した上で，機能 2 を実現するため，以下の3つを研究目標として設定した．

研究目標3. 仕様の入力テンプレートの開発
研究目標4. 仕様に基づくUML 図の部分抽出手法の開発
研究目標5. 仕様に基づくUML 図の記述抽象化手法の開発

2.1.3 UML 図から SMV 言語への自動変換

モデル検査を実施するためには，検査対象システムをモデル検査ツールに固有のモデル化言語で記述しなければならない. しかしながら，NuSMV のモデル化言語である SMV 言語の文法や意味論は，検査対象となるUMLとは大きく隔たりがある. したがって，UML 図で記述された振る舞いを正しく SMV 言語によって表現するには，モデル検査や NuSMV についての専門的な知識や経験が必要となるため，組込みソフトウェアの設計者にとっては利用への障壁が非常に高い.

そこで，本研究で実現する検証支援ツールでは，UML 図からその振る舞いを表現するSMV 言語によるモデルへの自動変換を行うことで，設計者の本ツールの利用への障壁を解消する. 前述の機能1，機能2によって，定められた制約を満たし，かつ検証特性に基づいて抽象化を施されたUML 図を得ることができる. ここでは，得られたUML 図から，SMV 言語による記述を自動生成するための手法を開発する.

機能 3 および本研究が目的とする検証支援ツールを実現するため，以下の 2 つを研究目標として設定した.

研究目標6. UML 図および仕様からSMV 言語への自動変換手法の開発
研究目標7. 検証支援ツールの入出力インタフェースの作成

2.2 研究アプローチ

2.2.1 研究の全体像

2.1 到達目標にて述べたとおり， 本研究では以下の3つの機能をもつ検証支援ツールの実現を目指す. これにより，1.2 研究課題で述べた，モデル作成の困難さと状態爆発の危険性という，モデル検査を設計検証に適用する上で生じる 2 つの問題の解決を目指す.

- 機能1. SMV 言語への自動変換を目的としたUML 図の抽象化
- 機能2. モデルサイズ削減を目的とした，UML 図の抽出，分割，および抽象化
- 機能3. UML 図からSMV 言語への自動変換

以上の3つの機能を実現するために5つの機能モジュールを定義する. 各モジュールと3つの機能との対応関係を図 2-2 に示す. 以下では各モジュールの働きについて詳細を述べる.

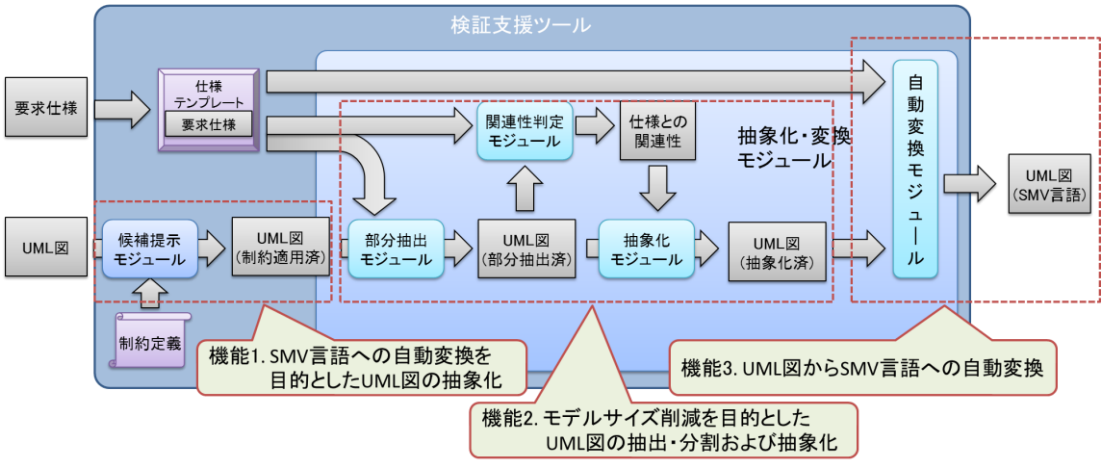


図 2-2:機能モジュールと 3 つの機能との対応関係

モジュール1:候補提示モジュール

このモジュールは機能1の実現を支援するために作成する。機能1では自動変換ツールの入力となる UML 図の記述方法に制約を課すことによって SMV 言語への自動変換を実現可能にしている. さらに, 既存の UML 図作成ツールでは制約を満たす UML 図の作成は必ずしも容易でないため, 機能1の実現には以下に述べる要件が満たされる必要がある。

1. ソフトウェアの設計者が作成した UML 図の記述について, 制約に沿った記述であるかを自動で確認できること
2. 制約に違反した記述について, 修正の方法を提示できること

候補提示モジュールはこれら2つの要求を実現するためのモジュールである. 制約に違反した記述を確認する過程では, どのような制約にどのように違反したかについても特定可能にする. これによって, 違反の種類に応じた適切な修正の候補を提示できるようになる. このモジュールでは astah*などの既存の UML 図の作成ツールとの連携を意識した実装を目指す.

モジュール2:関連性判定モジュール

このモジュールは機能2の実現を支援するために作成する。機能2では大規模化・複雑化した組込みソフトウェアにおいてモデル検査を実用的なものとするために, UML 図の中からモデル検査によって確認したい特性に関連する部分を抽出・分割する.

本研究で作成する自動変換ツールでは, 仕様の入力をテンプレート化することによってモデル検査によって確認したい特性や対象となる変数などはソフトウェアの設計者が特定できるような設計となっている. しかしながら, それらの入力に基づいてモデル検査で特性を確認するために必要な要素を判別できる必要がある.

本モジュールはソフトウェアの設計者からの特性や検証したい箇所といった入力に基づいて, UML 図の他の部分との関連性を計算するモジュールである。

モジュール3:部分抽出モジュール

このモジュールは機能2の実現を支援するために作成する。前述の関連性判定モジュールによって，モデル検査を行うのに必要十分な要素の特定が可能となる．しかしながら，特定された箇所を抽出してモデル検査が可能な形式に整える必要がある．部分抽出モジュールはこの機能を実現するためのものである．

モジュール4:抽象化モジュール

このモジュールは機能2の実現を支援するために作成する．機能2では大規模化・複雑化した組込みソフトウェアにおいてモデル検査を実用的なものとするために，UML 図の部分抽出だけでなく適切な抽象化を行ってモデル検査の際に探索する状態数を削減することも目指している．この目的のためには以下の2つの要件が満たされなければならない。

- 抽象化しても仕様に影響を及ぼさないこと
- UML 図と仕様のみを入力すれば自動で抽象化できること

まず，モデル検査に求められているのは UML 図が仕様にある特性を満たすかどうかの確認であるため，抽象化が仕様に影響を及ぼしてはならず，また影響の有無が判別可能でなくてはならない．次に，本研究で実現する自動変換ツールの想定利用者はモデル検査に習熟していないため，仕様への影響の有無に関して利用者に更なる情報の入力を求めるようなことは極力避ける必要がある．このため，影響の有無の確認および抽象化については完全自動化が不可欠である．

モジュール5:自動変換モジュール

このモジュールは機能3の実現を支援するために作成する．機能3には以下の3つが入力として渡される．

- 抽出・分割・抽象化された UML 図
- テンプレートに従って入力された仕様
- モデル検査で確認したい特性

これらの入力から SMV 言語を自動で作成する．SMV 言語の作成にあたっては，実際にモデル検査を行うソフトウェアである NuSMV と連携できるようにして，利用者が UML 図などの入力からモデル検査の結果の確認までをシームレスに行えるような設計を目指す．

以降の節では各機能の実現に際して採用したアプローチについて詳細を述べる．

2.2.2 SMV 言語への自動変換を目的とした UML 図の抽象化

SMV などのモデル検査ツールでは検査の対象や検査したい特性を表現するために独自の記法が用いられる．記法は従来のプログラミング言語と類似したものも多いが，記述に際してモデル検査に関する知識が必要となる．また，検査したい特性の記述に際しては時相論理に関する知識が要求される．さらに加えて，モデル検査を効率的に行えるように記述するには経験が必要である．このように，ソフトウェアの設計者にモデル検査に関連した知識・経験が要求されるため，SMV によるモデル検査を設計検証への導入が困難であるのが現状である．

本研究では，SMV によるモデルの作成の困難さを克服するために UML 図から SMV 言語への自動変換を考える．UML 図はソフトウェアの設計者がよく利用する記述方法であり，記

述に関する経験も豊富である．UML 図を直接入力できるようにすることでモデル検査の導入が容易になると考えられる．

一方で，UML 図の記述方法は自由度が高く，あらゆる UML 図の記述方法を受理することは現実的には難しい．そこで， UML 図を抽象化することで SMV 言語によるモデルへの自動変換を実現する．このアプローチを採用した理由は，研究者がこれまでに SMV を用いた UML の状態マシン図およびシーケンス図の自動検証について研究開発を行ってきており，その成果を援用することで自動変換を実現できると考えたためである．

状態マシン図およびシーケンス図は組込みソフトウェアの開発現場でも広く利用されている UML 図であり，これまでの研究開発の成果を応用することは研究の効率の面からも妥当だと考えている．

これまでの研究開発において得た知見として，UML 図のように広く設計に用いられている記述方法を入力として自動検証を行うためには，記述方法にある種の制約を課すことが有効だということである．制約を課すことによって記述方法の自由度を制限できるため，自動変換が容易になるメリットがある．また，大規模化・複雑化した組込みソフトウェアの特性を検証するにあたっては，モデルの状態爆発が問題となるが，適切な制約を課すことで検証に必要となる部分を抽出しやすくなると期待できる．

UML 図の記述方法に制約を課すアプローチで重要なことは制約の選定である．制約の選定にあたっては，まず，研究者がこれまでに開発した UML 図から SMV 言語への変換手法で課した制約について，本研究でも有用であるかを検討する．さらに，ソフトウェアの設計検証にモデル検査を利用している，企業の開発者や研究者の意見を幅広く取り入れて，実用性とのバランスを検討する．

本研究では自動変換が前提であるため，UML 図のデータから制約違反が検出可能でなければならない．そこで，ソフトウェア設計において広く用いられている描画ツールが出力する UML 図のデータを対象として，SMV 言語への変換に必要な情報の抽出等を行い，制約された記法に含まれない部分を検出するように自動変換ツールを設計する．また，単に制約違反を検出するだけでなく，制約内の記法への等価変換が可能であれば変換を行うようにする．

2.2.3 モデルサイズ削減を目的とした UML 図の抽出・分割および抽象化

本研究で開発に取り組む自動変換ツールは，実際に組込みソフトウェアを設計している技術者を対象としている．現在の組込みソフトウェアは大規模化・複雑化しているため，自動変換ツールに入力された UML 図をそのまま SMV 言語へ変換してもモデル検査を実施する段階で状態爆発となる．また，状態爆発が発生しない場合でも，実際に検証したい特性に関連しない要素を含んだ SMV 言語であるため，専門家が適切に問題部分を抽出する場合に比べて検証コストが増大する．

本研究では，検査対象となる UML 図から検査特性と関連する部分のみを抽出・分割し，抽象化するアプローチによってこの問題の解決を図る．抽出・分割・抽象化は人手を介さず自動で行えるようにすることで，ソフトウェアの設計者にモデル検査や SMV 言語に関する知識を問うことなく自動変換ツールを利用できるようにする．

抽出・分割・抽象化を行うにあたって問題となるのは，どのような情報をどのような形

式で自動変換ツールに与えるかという点である.まず必要となる情報は以下の２つである.

1. ソフトウェア設計が満たすべき仕様
2. 仕様に関連するUML図の要素

1について，満たすべき仕様が明らかでなければ，どの部分を抽出する必要があるか判別できない．ソフトウェア開発では仕様の記述にさまざまな記法が用いられる．代表的な形式仕様記法としては，VDM-SLやZ言語，CSPなどが挙げられる．これら全てを入力として扱えることが望ましいが，現実的には全てに対応することはコストの面から難しい．

本研究では，ソフトウェアの設計者が容易に仕様を指定できるように統一された入力方法を整備する．具体的には，仕様の記述方法について調査を行い，記述される仕様の種類や記述方法について共通性を見出し，仕様入力のテンプレートを作成することでこの課題の解決を図る．モデル検査によって検証したい特性は多くの場合共通しており，入力テンプレートに一定の柔軟性を持たせることで大部分の要求仕様に対応できると考えられる．

2について，仕様に関連する要素は仕様を入力する際に同時に入力できるように上記のテンプレートを整備する．具体的には，仕様に関連するUML図の要素である変数，状態，メッセージなどをテンプレートで指定できるように自動変換ツールを設計する．

また，ソフトウェアの設計者が指定したUML図の要素を足掛かりとして，実際にモデル検査を実施するために必要となる要素を特定し分割する仕組みを組み込む.この仕組みは，UML 図のある要素が仕様の検証に関連するか判定する関連度判定のモジュールと，関連ありと判定された要素を抽出するモジュールから構成される.

関連性の判定に関しては,どのような基準に基づいて判定を行うのかという問題がある.この点はモデル検査によって検証したい特性に依存する部分が大きいため，前述の仕様入力のテンプレートと対応付けて調査および計算式の定義を進めていく.

関連性が高い部分の抽出は前節と同様に，ソフトウェア設計において広く用いられている描画ツールが出力するUML図のデータを想定して自動変換ツールを設計する.

2.2.4 UML図からSMV言語への自動変換

本研究で作成する自動変換ツールでは,制約を満たす UML 図が入力されると,機能1によってUML図がSMV言語への変換が容易になるように抽象化される.そして,対象のソフトウェア設計が満たすべき仕様とモデル検査で検証したい部分が追加で入力されると,抽象化された UML図からモデル検査によって検証したい特性に関連する部分が抽出・分割される.また,この過程でさらにUML図は抽象化されてSMV言語への変換の入力となる.

UML図とSMV言語の自動変換に関しては,まず,UML図の各要素および要素間の関連をどのようにSMV言語の要素へマッピングするかが課題である.SMV言語への変換にあたってはテンプレートで入力した仕様についても考慮する必要があり,2つの記述を統合した形での自動変換が必要となる.SMV言語における記述が仕様やUML図における記述の意図と一致しない場合,モデル検査によって検証したい特性を表現できているとは言えず,検証支援ツールの出力が信頼できないものとなる.

次に,自動変換アルゴリズムの計算量が課題として考えられる.機能2の出力であり機能3の入力となるUML図は,機能1と機能2によって抽象化と関連部分の抽象化・分割が行われており,その内部の記法は単純化されている.しかしながら,組込みソフトウェアは大規模化・複雑化してお

り, その記述量については一定以上の規模となることが予想される. したがって, SMV 言語への自動変換アルゴリズムを考えるにあたっては実用的な観点からの計算量の把握および評価が必要となる.

以上の2点の課題に関して, SMV 言語への自動変換アルゴリズムの開発では, 研究者が過去に公表した状態マシン図やシーケンス図向けのアルゴリズムを出発点とするアプローチを採った. まず, マッピングの問題については, 過去に公表したアルゴリズムにおいて一定の考慮がなされており, その信頼性についても検証済みである. このアルゴリズムに基づいて今回の自動変換アルゴリズムを構築していくことが効果的であると考えられる.

一方で, このアルゴリズムでは今回開発する自動変換ツールが対象とする規模を想定しておらず, 実用上の品質を満足できるか不明である. しかし, その計算量は UML 図の規模に対して多項式程度であることは理論的に解明できている. 簡単な試算ではあるが, 機能1と機能2による抽象化およびサイズ削減と組み合わせることで, 組込みソフトウェアの設計規模に対しても実用的な時間で実行できることを確認している. 本研究では詳細な検討を行ったうえでアルゴリズムを改善していく.

以上の3つの取組みで設計した手法を統合して最終的に検証支援ツールとして実装する. 2.2.2 節の取組みで, 候補提示モジュールを含む機能1の設計が完了する. また, 2.2.3 節の取組みで, 関連性判断モジュール, 部分抽出モジュール, 抽象化モジュールを含む機能2の設計が完了する. 2.2.4 節の取組みでは, 自動変換モジュールを含む機能3の設計が完了する. これら5つのモジュールを含む3つの機能の設計をもとに実装に必要となる仕様書を作成する.

上記の仕様書に加えて, 入出力インタフェースや UML 描画ツールへのエクスポート機能, 各機能の呼び出しを司るモジュールなど, 必要な仕様をさらに加えた検証支援ツールの仕様を作成する. 効率性などを考慮して検証支援ツールの試作を外注する.

2.2.5 関連するこれまでの研究について

当研究室では, モデル検査を用いた状態マシン図およびシーケンス図の形式的検証に関する研究開発を行ってきた. 主なテーマとしては以下の 4 つである.

- SMV を用いた状態マシン図とシーケンス図の形式的検証[HARADA 2009]
- SMV を用いたシーケンス図の形式的検証[KAWAKAMI 2010]
- Alloy を用いた状態マシン図の形式的検証[NIMIYA 2010]
- LTSA を用いた状態マシン図とシーケンス図の形式的検証[ASADA 2011][MIYAZAKI 2012][YOKOGAWA 2013]

これらの手法は設計間の検証を目的としたものである. これに対して本研究では, 設計が仕様を満たすか否かの検証を目的としている. 本研究ではモデル検査を適用する上での障壁の一つである検査式の記述に要するコストを削減するために仕様を記述するためのテンプレートを開発している. また, これまでの研究では検証コスト削減を目的とした UML 図の抽象化や分割については十分に検討されていない.

UML 検証に対する SMV の適用事例など, モデル検査を用いた設計検証についての取組みは数多く報告されている. 例えば JAXA は人工衛星の姿勢制御ソフトウェアにモデル検査ツ

ールUPPAALを適用し，仕様記述の検証を行っている．また日立ソリューションズは家電のBlu-ray制御のミドルウェアにモデル検査ツールSPINを適用して設計検証を行っている．しかしながら，これまでに報告された事例はあくまでも個別のケーススタディであり，ツールやノウハウ化もされていないため，現場の開発に即座に応用することは困難であったといえる．

本研究では，設計抽象化や仕様記述についての技術をパッケージ化してツールとして実装することで，現場の開発に対して即座に応用可能な技術の普及を目指している．

2.3 研究の活動実績・経緯

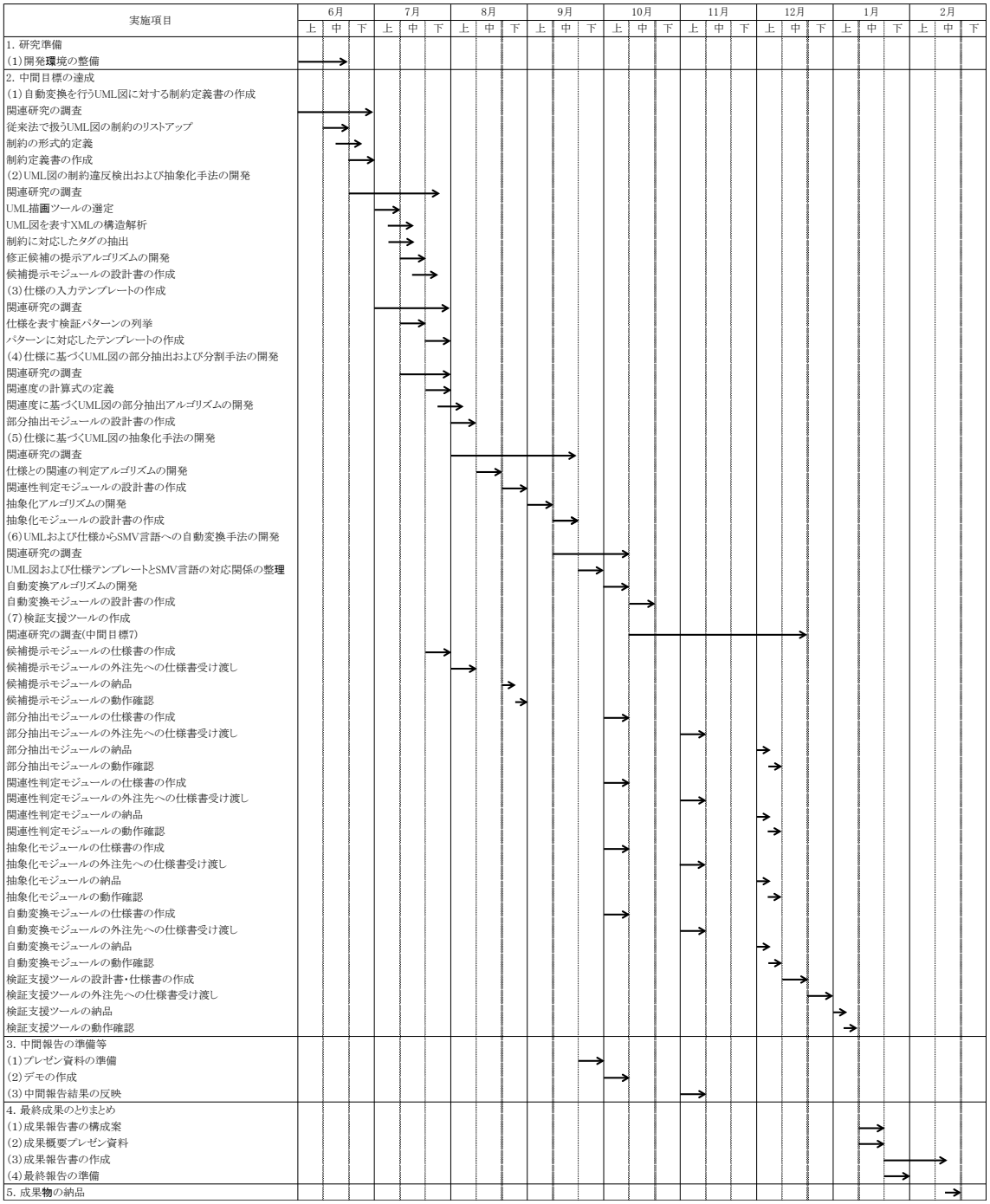
2.3.1 研究活動実績

本研究の活動実績および経緯について述べる．表2-2に研究実施実績を示す．

活動実績の概要について，まず2013年6月上旬～中旬にかけては，研究実施のための準備として，開発環境の整備を行った．2013年6月上旬～下旬にかけて，研究目標1である「自動変換を行うUML図に対する制約定義書の作成」に関する研究開発を行った．6月中旬～7月下旬にかけて,研究目標2である「UML図の制約違反検出および抽象化手法の開発」に関する研究開発を行った．7月下旬に「候補提示モジュール」の開発について株式会社フォーマルテックへと発注を行い，8月上旬に納品および検収作業を完了した．7月上旬～7月下旬にかけて，研究目標3である「仕様の入力テンプレートの作成」に関する研究開発を行った．7月中旬～8月中旬にかけて，研究目標4である「仕様に基づくUML図の部分抽出および分割手法の開発」に関する研究開発を行った．8月上旬～9月下旬にかけて，研究目標5である「仕様に基づくUML図の抽象化手法の開発」に関する研究開発を行った．9月中旬～10月下旬にかけて，研究目標6である「UMLおよび仕様からSMV言語への自動変換手法の開発」に関する研究開発を行った．9月下旬～10月下旬にかけては中間報告の準備を併せて行い，10月31日にIPAにて実施された中間報告会にて報告を行った．11月上旬に「抽象化・変換モジュール」の開発について株式会社フォーマルテックへと発注を行い，12月上旬に納品および検収作業を完了した．10月中旬～12月下旬にかけて，研究目標7である「検証支援ツールの作成」に関する研究開発を行った．12月下旬に「検証支援ツール」の開発について株式会社フォーマルテックへと発注を行い，1月上旬に上品および検収作業を完了した．12月下旬以降は，最終報告の準備および最終成果のとりまとめを行い，1月29日にIPAにて実施された最終成果報告会にて報告を行った．成果報告書の構成案を1月14日に提出し，2月7日に成果報告書の第一案を提出した．そして最終成果報告書を2月14日に提出した．

本研究の実施にあたっては，研究メンバーによるミーティングを月2回開催し，研究の進捗状況を確認するとともに，問題を共有し，課題解決に向けた議論を行った．また，研究目標の内部レビューを計7回実施した．

表 2-2:研究実施実績


実施項目
6月
7月
8月
9月
10月
11月
12月
1月
2月
上 中 下
1. 研究準備
(1)開発環境の整備
2. 中間目標の達成
(1)自動変換を行うUML図に対する制約定義書の作成
関連研究の調査
従来法で扱うUML図の制約のリストアップ
制約の形式的定義
制約定義書の作成
(2)UML図の制約違反検出および抽象化手法の開発
関連研究の調査
UML描画ツールの選定
UML図を表すXMLの構造解析
制約に対応したタグの抽出
修正候補の提示アルゴリズムの開発
候補提示モジュールの設計書の作成
(3)仕様の入力テンプレートの作成
関連研究の調査
仕様を表す検証パターンの列挙
パターンに対応したテンプレートの作成
(4)仕様に基づくUML図の部分抽出および分割手法の開発
関連研究の調査
関連度の計算式の定義
関連度に基づくUML図の部分抽出アルゴリズムの開発
部分抽出モジュールの設計書の作成
(5)仕様に基づくUML図の抽象化手法の開発
関連研究の調査
仕様との関連の判定アルゴリズムの開発
関連性判定モジュールの設計書の作成
抽象化アルゴリズムの開発
抽象化モジュールの設計書の作成
(6)UMLおよび仕様からSMV言語への自動変換手法の開発
関連研究の調査
UML図および仕様テンプレートとSMV言語の対応関係の整理
自動変換アルゴリズムの開発
自動変換モジュールの設計書の作成
(7)検証支援ツールの作成
関連研究の調査(中間目標7)
候補提示モジュールの仕様書の作成
候補提示モジュールの外注先への仕様書受け渡し
候補提示モジュールの納品
候補提示モジュールの動作確認
部分抽出モジュールの仕様書の作成
部分抽出モジュールの外注先への仕様書受け渡し
部分抽出モジュールの納品
部分抽出モジュールの動作確認
関連性判定モジュールの仕様書の作成
関連性判定モジュールの外注先への仕様書受け渡し
関連性判定モジュールの納品
関連性判定モジュールの動作確認
抽象化モジュールの仕様書の作成
抽象化モジュールの外注先への仕様書受け渡し
抽象化モジュールの納品
抽象化モジュールの動作確認
自動変換モジュールの仕様書の作成
自動変換モジュールの外注先への仕様書受け渡し
自動変換モジュールの納品
自動変換モジュールの動作確認
検証支援ツールの設計書・仕様書の作成
検証支援ツールの外注先への仕様書受け渡し
検証支援ツールの納品
検証支援ツールの動作確認
3. 中間報告の準備等
(1)プレゼン資料の準備
(2)デモの作成
(3)中間報告結果の反映
4. 最終成果のとりまとめ
(1)成果報告書の構成案
(2)成果概要プレゼン資料
(3)成果報告書の作成
(4)最終報告の準備
5. 成果物の納品

2.3.2 学会参加

本研究における，学会等の参加状況とその成果について述べる．

6 月

6 月 13 日～14 日にかけて，有本教授，佐藤准教授，横川助教の 3 名が本研究の情報収集および UML 設計の自動検証ツールの展示のため,ETWest2013(組込み総合技術展 関西)に参加した．参加者との情報交換の結果として，UML 設計の自動検証ツールに関して，開発者が求める要件についての知見を得るとともに，参加企業が実際に現場で利用しているソフトウェア検証技術についての情報を得ることができた．

6 月 12 日～14 日にかけて,天嵜助教が本研究の情報収集のため,国際会議 Profes2013(プロダクトに着目したソフトウェア開発プロセスの改善に関する国際会議)に参加した.同会議はソフトウェア工学の中でも実証的な研究を特に対象とした会議であり，他の会議と比較して，産業界からの発表および出席の比率が高いのが特徴である．参加者と，開発中の制約定義書と，実際の UML の利用上における慣習などとの関係について議論を行い，従来法における制約が実用上の問題となるようなものではないことを確認した．また，ソフトウェア開発プロセスに関するセッションに集まった研究者・実務者と，開発プロセスにおいて形式仕様を実施する工程の導入方法についても現状確認を行った．ソフトウェア開発における形式手法の適用に関する最新動向についても情報交換を行った．組込み関係の産業に携わる参加者も多く，実務の観点から意見を伺うことができた．

7 月

7 月 3 日に，横川助教と天嵜助教が，産業技術総合研究所関西センター尼崎支所にて，仕様のテンプレート化手法の最新技術動向に関して，西原秀明研究員からヒアリングを行った．ヒアリングを通して，仕様パターンの技術動向および現場での利用状況に関する情報を得るとともに，モデル検査の適用を前提とした仕様のテンプレート化についてのアドバイスを受けることができた．

7 月 8 日～10 日にかけて，天嵜助教が本研究の情報収集のため，SS2013(ソフトウェアシンポジウム)に参加した．技術者との情報交換を通して，本研究で開発するツールの開発現場への導入を容易にするための施策に関する情報とともに，導入の継続性に関する知見を得ることができた．

7 月 13 日～19 日にかけて，横川助教が本研究の情報収集のため，CAV2013(コンピュータ援用検証に関する国際会議)に出席した．参加者との情報交換および発表の聴講により，ソフトウェア検証における抽象化技術に関する情報を得ることができた．

7 月 24 日～27 日にかけて,横川助教と天嵜助教が電子情報通信学会ソフトウェアサイエンス研究会に出席し，UML によるソフトウェアシステムのモデル化に関する成果発表[落水 2013-1]および，参加者との情報交換を行った．発表を聴講することで，UML によるソフトウェア設計記述を対象とした抽象化手法に関する情報に加えて UML によるソフトウェアの仕様記述および検証に関する情報を得られた．また，成果発表に関する参加者との質疑・議論を通して，技術者がソフトウェアのモデル化において重要視する点に関しての知見が得られた．

7 月 27 日～29 日にかけて，佐藤准教授と横川助教が本研究の情報収集のため，回路とシステムワークショップに参加した．参加者との情報交換を通して，組込みシステム設計の部分抽出・分割手法に関する情報を得るとともに，統計モデル検査アルゴリズムのハードウェア実装に関する発表を聴講し，高速なモデル検査を実現するためのモデル化技術に関する情報を得ることができた．

8 月

8 月 2 日に，横川助教と天嵜助教が，産業技術総合研究所関西センター尼崎支所にて，ソフトウェア設計の抽象化手法の最新技術動向に関して，西原秀明研究員からヒアリングを行った．ヒアリングを通して，従来法を UML の抽象化に適用する上で発生する問題点に関する情報を得るとともに，モデル検査を高速化する上でのソフトウェア設計の抽象化手法についてのアドバイスを受けることができた．

8 月 18 日～26 日にかけて，横川助教と天嵜助教が本研究の情報収集のため，ESEC/FSE 2013(ヨーロッパにおけるソフトウェア工学国際会議/ソフトウェア工学の基礎に関する国際会議)に参加した．研究発表の聴講を通じて，ソフトウェアの設計検証の効率化手段に関する情報および検査対象に特化したソフトウェアの設計検証手法に関する情報を得ることができた．

8 月 27 日～29 日にかけて，横川助教と天嵜助教が本研究の情報収集のため，ICGSE2013(グローバルソフトウェア工学に関する国際会議)に参加した．参加者との意見交換および研究発表の聴講を通じて，ソフトウェア検証において求められる要件や検証効率化に関する情報を得ることができた．

9 月

9 月 4 日～6 日にかけて，横川助教と天嵜助教が本研究の情報収集のため，EuroMicro DSD/SEAA Conferences2013(EuroMicro ディジタルシステム設計/ソフトウェア工学の先進適用に関する会議)に出席した．参加者との意見交換および研究発表の聴講を通じて，ソフトウェア検証における状態空間の抽象化技術やソフトウェア検証を目的とした抽象化および形式化に関する情報を得ることができた．

9 月 8 日～11 日にかけて，横川助教と天嵜助教が本研究の情報収集のため，SES2013(ソフトウェア・エンジニアリング・シンポジウム 2013)に参加した．参加者との情報交換および研究発表の聴講を通じて，ソフトウェアシステムの効率的な抽象化に関する情報を得るとともに，併設ワークショップでの研究発表[横川 2013][片山 2013][落水 2013-2]を通じて，設計検証の効率化に関する知見を得ることができた．

9 月 17 日～20 日にかけて，有本教授と佐藤准教授が本研究の情報収集のため，電子情報通信学会ソサイエティ大会に参加した．参加者との情報交換を通じて，組込みシステム検証における状態爆発の回避方法および自動変換の効率化手法に関する知見が得られた．

9 月 25 日に，横川助教と天嵜助教が，産業技術総合研究所関西センター尼崎支所にて，ソフトウェア設計の自動検証手法の最新技術動向に関して，西原秀明研究員からヒアリングを行った．ヒアリングを通して，従来法を UML の形式的検証に適用する上で発生する問題点に関する情報を得るとともに，UML をモデル検査ツールの入力記述へと効果的に変換

するためのアプローチについてのアドバイスを受けることができた.

10 月

9 月 29 日から 10 月 4 日にかけて，有本教授が本研究の情報収集のため，国際会議 ESWEEK2013(組込みシステム週間)に参加した．同会議は組込みシステムに特化した，ハードウェアからソフトウェアのシステムレベルをカバーするもので，組込みシステムに注力している大学と組込みシステムに係わる IP ベンダー（ハードウェア，ソフトウェア)，EDA ベンダー等のメーカが主な発表者・参加者である．例年２５０名程度の参加者があり，特に今後の組込みシステムに適用される先導的技術の発表が多数行われる．本会議では，参加者との情報交換を通じて UML による組込みシステム設計を対象とした自動処理手法の最新技術動向に関する知見を得るとともに，線形システムの安全性検証に関する発表を聴講し，組込みシステムの抽象化に基づく自動検証に関する情報を得ることができた.

10 月 7 日～8 日にかけて，有本教授が本研究の情報収集のため，情報処理学会のシステム LSI 設計技術研究会(SIG-SLDM)に参加した．ハードウェアの自動設計手法に関する発表を聴講し，ハードウェア検証の自動化に関する知見を得るとともに，再構成型ハードウェアの設計手法に関する発表を行い，参加者との討論を通して，ハードウェアの設計検証における産業界からの要求について情報を得ることができた．また，同発表により，提案する抽象化手法による検証の高速化について，適用可能性という視点から議論を行い，組込みソフトウェアにおいての効率的な抽象化技術に関しての情報を得ることができた.

10 月 7 日から 10 月 14 日にかけて，天嵜助教が本研究の情報収集のため，ESEM2013(実証的ソフトウェア工学および定量的評価に関する国際会議)に参加した．同会議は毎年 9 月もしくは 10 月に行われる,ソフトウェア工学の中でも実証的な研究を特に対象とした会議である.他の会議と異なり,産業界からの発表および出席の比率が高いのが特徴である．今年は基調講演が 3 件，通常の発表が 24 件，ショートペーパーが 12 件であった．発表は全日２つのセッションが並行で行われた．報告者が参加した発表セッションでは，組み合わせテストの手法が開発現場に受け入れられない理由について，習熟の難易度に着目して実証的検証を行った研究が興味深かった．ソフトウェア工学の研究分野では多くの手法が提案されている一方で、開発現場で実際に受け入れられているものは多くない．習熟の容易さは本受託研究における成果物へも反映が必要な視点で大変参考になった．参加者との情報交換を通じて，UML および仕様からの SMV への自動変換手法に関して，実務で想定される仕様規模では従来法のアルゴリズムの応用で問題ないとの意見をいただいた．また，部分的に問題がある場合も計算機の能力の発展によりカバーできるだろうとの意見をいただいた.

10 月 16 日に,横川助教と天嵜助教が,産業技術総合研究所関西センター尼崎支所にて，UML を対象とした分析ツールの最新技術動向に関して，西原秀明研究員からヒアリングを行った．ヒアリングを通して，UML 検証ツールのインタフェースにおいて開発者が求める要素ならびに，従来の設計検証ツールに対してユーザが持つ問題意識について，アドバイスを受けることができた.

10 月 16 日～18 日にかけて，佐藤准教授が本研究の情報収集のため，組込みシステムシンポジウム(ESS2013)に参加した．モデル駆動開発に関する発表を聴講し，組込みシステム

開発時における設計検証手法についての知見を得るとともに，組込みシステムの障害検出手法に関する発表を聴講し，組込みシステム設計の自動検証における反例分析手法に関する知見を得ることができた．

10 月 23 日～25 日にかけて，横川助教が本研究の情報収集のため，情報処理学会のソフトウェア工学研究会(SIG-SE)に参加した．ソフトウェアのテスト自動生成手法に関する発表を聴講し，ソフトウェア検証の自動化に対する開発現場からの要求についての知見を得るとともに，組込みソフトウェアの障害分析手法に関する発表を聴講し，組込みソフトウェア設計の自動検証における反例分析手法に関する知見を得ることができた．

11 月

11 月 19 日～22 日にかけて，有本教授，佐藤准教授，横川助教が本研究の情報収集のため，組込み総合技術展(ET2013)に参加し，ポスター展示を行った．組込みシステムメーカ等から多くの訪問者が本展示ブースを訪問し，論理モデルの抽象化技術に高い関心を持っていることが判明した．また，来展者との意見交換を通じて，開発中の自動検証ツールの有効性・新規性を確認するとともに，組込みシステム開発環境，設計手法等に関する情報収集の結果，提案する自動検証手法のさらなる高性能化・高機能化に関する知見が得られた．また，組込みシステム設計者および設計ツール開発者から，現状の設計検証における課題に関する意見を多数収集することができた．

11 月 27 日～29 日にかけて,佐藤准教授が本研究の情報収集のため,デザインガイア 2013 に参加した．NoC システム開発環境，設計手法等に関する情報収集の結果，本研究で開発する検証支援ツールのインタフェースについて開発現場で要求される項目に関する知見が得られた．さらに，参加者との意見交換の結果，開発中の検証支援ツールの有効性を確認できた．

12 月

12 月 1 日～6 日にかけて，横川助教が本研究の情報収集および成果発表[Yokogawa 2013]のため,PEDC2013(環太平洋におけるディペンダブルコンピューティングに関する国際シンポジウム)に参加した．発表に関する質疑および参加者との議論を通じて，UML の自動検証に関する改善点についての情報を得ることができた．

2.3.3 内部・外部打ち合わせの実施状況

本節では，研究チーム内での打ち合わせおよび外部の協力研究者との打ち合わせの実施実績について述べる．

(1) 内部打ち合わせの実施状況

研究メンバー内部では，以下のように研究打ち合わせを実施した．

まず，毎月 2 回ずつ，研究メンバーによるミーティングを行った．このミーティングでは，学会参加等による情報収集を通して得られた結果についての報告や，研究の進捗状況の報告および確認・情報共有を行うとともに，研究推進に際して発生した諸問題についての対策に関して議論を行った．

また，研究推進において発生・発覚した問題については，即座にメール等の手段を用いてメンバー間にて情報共有を行い，後のミーティングでの議論をスムーズに進められるように取り組んだ．

さらに，本研究で定めた研究目標については，達成後にその成果について研究メンバーによる内部レビューを行った．このレビューでは，研究目標の各作業項目に関する達成状況についてメンバー間で互いに確認を行った．

(2) 外部打ち合わせの実施状況

研究推進のため，外部の協力研究者との打ち合わせを以下の通り実施した．

まず，前述のとおり，組込みシステムを対象とした形式手法の専門家である，産業技術総合研究所の西原秀明研究員に，関連技術の最新技術動向に関するヒアリングを計 4 回実施した．

また，UML で記述されたソフトウェアシステムを対象とした形式的検証の専門家である，川崎医療福祉大学の宮崎仁講師に，関連技術や先行研究に関するレクチャを依頼した．計 20 回に渡り，研究メンバーへのレクチャを実施した．また，同講師には研究のレビューも依頼し，研究を実施する上で有益なコメントを多数いただいた．

2.4 研究実施体制

2.4.1 実施体制

研究実施体制を図 2-3 に示す.

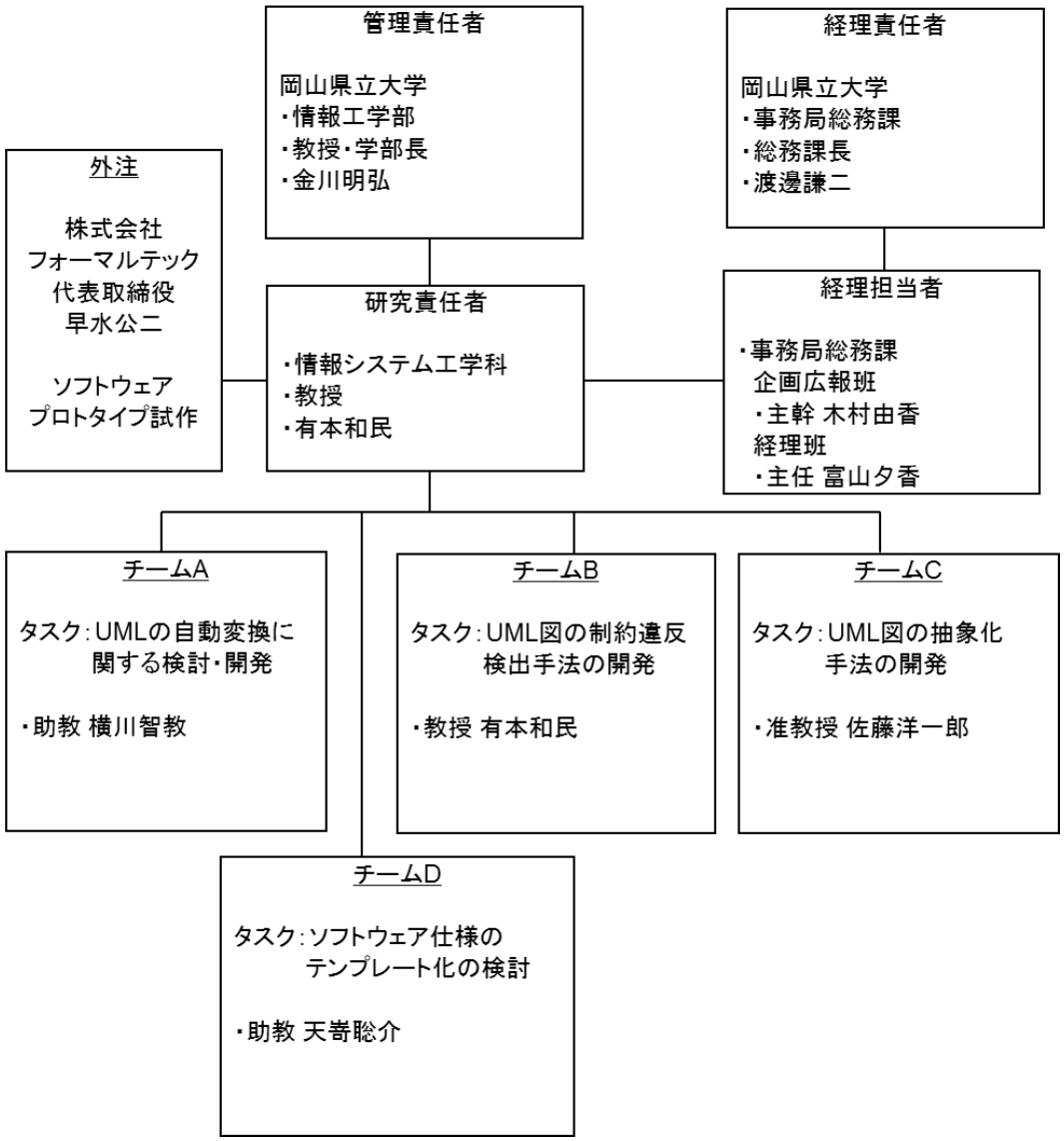


図 2-3:研究実施体制

2.4.2 研究メンバー

研究メンバーのプロフィールは以下の通りである．

有本和民

1979 年大阪大学基礎工学部卒業．1981 年同大大学院修士課程修了．同年三菱電機株式会社入社．2003 年株式会社ルネサステクノロジ継承転籍．現在，岡山県立大学情報工学部教授．博士(工学)．長堤消費電力組込みシステムの開発に関する研究に従事．IEEE，電子情報通信学会各会員．IEEE フェロー．

佐藤洋一郎

1982 年岡山大学工学部電子工学科卒業．1984 年同大大学院修士課程修了．同年株式会社東芝入社．1987 年岡山大学工学部情報工学科助手．1995 年岡山県立大学情報工学部助教授．現在，同大准教授．博士(工学)．大規模ディジタルシステムの高性能化．高信頼化，画像処理の高速化に関する研究に従事．電子情報通信学会，情報処理学会，映像メディア学会，各会員．

横川智教

1999 年大阪大学基礎工学部情報科学科中退．2001 年同大大学院修士課程修了．2004 年同大学院博士課程修了．同年岡山県立大学情報工学部助手．現在，同大助教．博士(工学)．モデル検査による組込みシステム設計の形式的検証に関する研究に従事．IEEE，ACM，電子情報通信学会，情報処理学会，日本ソフトウェア科学会，各会員．

天嵜聡介

2003 年大阪大学大学院修了課程修了．2006 年同大大学院博士課程修了．現在，岡山県立大学情報工学部助教．博士(情報科学)．工数見積もりに関する研究に従事．IEEE，ACM，情報処理学会，各会員．

2.4.3 外注先

本研究では，ツールの開発を株式会社フォーマルテックへと外注した．発注した項目は以下の 3 つである．

１．候補提示モジュールの開発

2.2.1節においてモジュール１として示されている機能モジュールの開発を外注した．発注内容としては，基本・詳細設計および設計書の作成，モジュールの実装，そしてテストおよびテスト報告書の作成である．

２．抽象化・変換モジュールの開発

2.2.1節においてモジュール２～５として示されている機能モジュールがもつ機能を統合したモジュールの開発を外注した．発注内容としては，基本・詳細設計および設計書の作成，モジュールの実装，そしてテストおよびテスト報告書の作成である．

３．検証支援ツールの開発

2.2.1 節で示した 5 つの機能モジュールをそれぞれ呼び出すための機能の実装，そして入出力インタフェースの開発を外注した．発注内容としては，基本・詳細設計，ツールの実装，テストおよびテスト報告書の作成，そして操作マニュアルの作成である．

3 研究成果

3.1 研究目標 1「自動変換を行う UML 図に対する制約定義」

3.1.1 当初の想定

(1) 想定する仮説等

本研究の目的は，UML で記述された設計ドキュメントを対象とした SMV による検証を支援するツールを実現することである．ここで，無制約の UML 図は自動処理を行うのに適していないため，コンピュータによる自動処理を効率化するためには適切な制約を与える必要がある．

ここで，本研究目標を達成する上で，本手法での UML 図への制約をどのように定めるかという課題が想定される．ここでは，研究者らがこれまでに開発した UML 図から SMV 言語への変換手法を応用し，その制約に基づいて制約定義書を作成することで，この課題の解決を目指す．

(2) 当初の到達目標と期待される効果

ここで設定する到達目標としては，UML の状態マシン図およびシーケンス図に対して自動処理を前提とした制約を定義し，制約定義書を作成することである．

UML 図に対して適切な制約定義を与えることにより，効率的な SMV 言語への自動変換が可能となる．

3.1.2 研究プロセスと成果

(1) 研究プロセス

本研究目標を達成するため，以下のプロセスに従って研究開発を実施した．

1. これまでに研究者らが開発してきた UML 図の形式的検証手法における制約をリストアップする．
2. 先行研究における制約を本研究の目的と照らし合わせて評価し，本研究で採用する制約を定義する．
3. 制約を制約定義書という形式で文書化する．

(2) 具体的な研究成果の内容

１．先行研究における UML 図の形式的検証手法における制約のリストアップ

これまでに研究者らは，モデル検査を用いた状態マシン図およびシーケンス図の形式的検証に関する研究開発を行ってきた．主なものとして，［HARADA 2009］では SMV を用いて状態マシン図とシーケンス図の整合性違反を検出するためのツールを開発している．また，[KAWAKAMI 2010]では SMV を用いてシーケンス図を検証するための手法を開発している．[NIMIYA 2010]ではAlloyを用いた状態マシン図とコミュニケーション図の整合性検証における事例報告を行っている．[ASADA 2011][MIYAZAKI 2012]では LTSA を用いてシーケンス

図の詳細化違反を検出する手法を，[YOKOGAWA 2013]では LTSA を用いて状態マシン図とシーケンス図の整合性違反を検出する手法を開発している．[片山 2013]では，CSP を用いて状態マシン図とシーケンス図をモデル化し，FDR ツールによってその整合性を検証する手法を提案している．

これらの手法において，対象となる UML 図に定めた制約として，大きく異なる点は階層構造を扱うか否かである．[HARADA 2009]では単純状態のみからなる状態マシン図と，結合フラグメントや相互作用参照を持たないシーケンス図を扱っており，階層構造を持つ状態マシン図およびシーケンス図は対象としていない．一方で[KAWAKAMI 2010]では結合フラグメントによる階層構造をもつシーケンス図を検証の対象として扱っている．しかしながら，利用可能な結合フラグメントは 12 種類のうち alt，opt，par の 3 種のみである．[NIMIYA 2010]では合成状態および直交状態を含む状態マシン図を検証の対象として扱っているが，この状態マシン図は，階層間をまたぐ複雑な処理構造を持つ遷移は含んでおらず，比較的単純な構造を持つものである．[ASADA 2011]では結合フラグメントを持つシーケンス図を扱っているが，利用可能な結合フラグメントは par，seq，strict の 3 種のみである．[MIYAZAKI 2012]の手法では，それに alt および loop を加えた 5 種の結合フラグメントを扱うことができる．[YOKOGAWA 2013]では，シーケンス図は[MIYAZAKI 2012]の手法と同じクラスのものを扱うことができるが,状態マシン図は階層構造を持たないものに限られる．一方で,[片山 2013]では状態マシン図およびシーケンス図ともに階層構造を持つものを扱うことが可能となっているが，シーケンス図においては他の先行研究と同様に結合フラグメントの一部のみとなる．

階層構造以外に着目すると，まず状態マシン図については変数を扱うか否かで分類される．上記の先行研究では，[NIMIYA 2010]では変数を持つ状態マシン図を扱うことができるが，[HARADA 2009]や[YOKOGAWA 2013][片山 2013]では変数の値に基づく状態遷移を持つ状態マシン図は検証の対象としていない．シーケンス図に関しては，メッセージの送受信以外に表現できる内容として，オブジェクトの生存期間や生成・終了の時期についての記述が挙げられるが，上記の先行研究では，いずれにおいてもシーケンス図において各オブジェクトは常に生存しているものとして扱っており，生成・終了を明示的に記述したシーケンス図は検証の対象としていない．

先行研究における状態マシン図およびシーケンス図に対する記述制約を表 3-1-1 にまとめる．

表 3-1-1:先行研究における UML 図の記述制約

		状態マシン図		シーケンス図	
	検証ツール	階層構造	変数	階層構造	生存期間
[HARADA 2009]	SMV	×	×	×	×
[KAWAKAMI 2010]	SMV			△	×
[NIMIYA 2010]	Alloy	△	○		
[ASADA 2011]	LTSA			△	×
[MIYAZAKI 2012]	LTSA			△	×
[YOKOGAWA 2013]	LTSA	×	×	×	×
[片山 2013]	CSP,FDR	○	×	△	×

２．先行研究における UML 図の制約の評価および本ツールで採用する制約定義

上述のように，UML の状態マシン図およびシーケンス図を対象とした形式的検証を実施する上で定める制約は，階層構造を扱うか否かで大きく分類される．また，状態マシン図については変数を扱うか否か，シーケンス図についてはオブジェクトの生存期間を扱うか否かによって分類される．

表 3-1-1 に示すように，SMV を用いた状態マシン図およびシーケンス図の形式的検証に関する先行研究である[HARADA 2009][KAWAKAMI 2010]においては，状態マシン図の階層構造は扱っておらず，シーケンス図の階層構造についても対象とするのは結合フラグメント記法の一部のみとなる．一方で，CSP と FDR を用いた先行研究である[片山 2013]においては階層構造を扱っているが，このモデルでは変数を扱うことが困難であるため，処理の対象からは除外している．Alloy を用いた先行研究である[NIMIYA 2010]においては状態マシン図の階層構造と変数の双方を扱うことができるが，階層構造については制限された記述のみしか扱うことができない．

これまでに取り組んできた UML 図を対象とした形式的検証に関する研究開発から得られた知見として，階層構造を持つ状態マシン図では，実行される遷移の選択について複雑な意味論を持つため，モデルが非常に複雑になりやすい．そのため，SMV 言語によるモデルを生成するための手続きも複雑になることから，本ツールにおいては状態マシン図の階層構造は処理の対象外とする．それに伴い，階層構造をもつ状態マシン図において用いられる記述である履歴，ジョイン・フォーク，入場・退場点の 3 つの擬似状態記法も本ツールの処理対象外とする．また，状態の入退場・状態内アクション記述も，同様の理由でモデルが複雑になることから，処理対象外とする．

次に，変数については SMV を用いた先行研究である[HARADA 2009]では扱っていなかったが，Alloy を用いた先行研究である[NIMIYA 2010]で開発した手法と同様のアプローチでモデル化が可能であると考えられる．したがって，変数に関する遷移のガード条件やアクションといった処理は本ツールの対象内とする．しかしながら，SMV では扱う変数の型に制約があるため，その制約に基づき，利用できる変数は整数型およびブール型のみに限るものとする．また，シーケンス図の生存期間については先行研究と同様に本ツールでは処理対象外とする．ただし，オブジェクトの生成・終了メッセージについては，他のメッセージと同様に処理が可能であるものとする．

シーケンス図に関して，SMV を用いた先行研究である[KAWAKAMI 2010]では結合フラグメントの一部のみは処理可能であるため，この手法を応用することでシーケンス図の階層構造の一部は対応可能であると考えられる．しかしながら，処理が複雑であるため，実装に要するコストが大きくなることが懸念される．また上述のように本ツールでは状態マシン図の階層構造は扱わず，階層構造による記述が不要なシステム開発の初期段階での利用を想定している．この場合，シーケンス図についても階層構造による記述を利用する頻度は極めて少ないと考えられるため，シーケンス図の階層構造も状態マシン図と同様に本ツールの処理対象外とする．ただし，相互作用参照を用いた他のシーケンス図の参照は記述の簡略化のためのみに用いられ，参照記述を除外した記述との相互変換も容易であるため，これについては本ツールの処理対象内に含めるものとする．

３．制約定義書の作成

上述の通り定めた状態マシン図およびシーケンス図の記述制約を制約定義書として文書化した．まず，状態マシン図の制約定義は表 3-1-2 に示す通りとなる．ここで，線形制約(*1)とは，「mx～n」の形(x は変数，m,n は整数，～は<,>,＝のいずれか)をもつ式と，否定(¬)，論理和(∨)および論理積(∧)からなる論理式のことを指す．また，シーケンス図の制約定義は表 3-1-3 に示す通りとなる．

表 3-1-2:状態マシン図の制約定義

要素	記法	対応	備考
状態	初期擬似状態	○	
	単純状態	○	
	合成（サブマシン）状態	×	
	終了状態	○	
	浅い履歴擬似状態	×	
	深い履歴擬似状態	×	
	ジャンクション擬似状態	○	
	選択擬似状態	○	
	フォーク擬似状態	×	
	ジョイン擬似状態	×	
	入場動作／実行活動／退場動作	×	
	直交状態（領域）	×	
遷移	トリガ	○	
	ガード	○	
	アクション	○	

表 3-1-3:シーケンス図の制約定義

記法	対応	備考
ライフライン	○	注 1)
同期メッセージ	○	
非同期メッセージ	○	
create(生成)メッセージ	×	注 2)
destroy(終了)メッセージ	×	注 2)
Reply メッセージ	○	
停止	×	無視
複合フラグメント	×	
相互作用の利用	○	
状態不変式	○	

注 1) オブジェクトは常時生存しているものとして扱う.
注 2) 非同期メッセージとして扱う.

3.1.3 課題とその対応

本研究では, UML で記述された設計ドキュメントを対象とした SMV による検証を支援するツールの実現のため, コンピュータによる自動処理を効率化するための制約を定めた. ここで想定される課題は, 本ツールにおいて状態マシン図およびシーケンス図に対してどのような制約を定められるか, となる.

そこで, 研究者らがこれまでに取り組んできた状態マシン図およびシーケンス図の形式的検証に関する先行研究における記述制約をリストアップし, その制約に基づいて制約定義書を作成することで, この課題を解決することに成功した.

今後の課題としては, 実際のソフトウェアおよび組込みシステムの開発プロジェクトに対して本ツールの適用を行う上で, ここで定義した制約が適切であるか, 十分な表現能力を持っているかについて, 利用者へのレビューなどに基づく評価が必要である.

将来の応用方法としては, ここで定めた制約定義は, アクティビティ図やコミュニケーション図などの他の UML 図に対して形式的検証を前提とした自動処理を行う上でも応用が可能であると考えられる.

3.2 研究目標 2「UML 図の制約違反検出および抽象化手法の開発」

3.2.1 当初の想定

(1) 想定する仮説等

3.1 節で示した制約定義に基づいて UML 図を記述することで,SMV 言語によるモデルへの

効率的な自動変換が可能となる．ここで，ツールへと入力されたUML図が制約を満たしていなかった場合は，制約を満たすようUML図を修正する必要がある．

ここで，本研究目標を達成する上で，入力されたUML図の制約違反をどのように検出するかという問題が想定される．多くのUMLモデリングツールの多くは，UML図の構造をXMLなどの形式化されたデータとして出力することが可能であるため，出力されたデータを解析し，対応する情報を抽出・評価することで違反検出の自動化を実現する．

(2) 当初の到達目標と期待される効果

ここで設定する到達目標としては，制約定義書に基づき，与えられた状態マシン図およびシーケンス図に対して制約に違反する部分を検出し，修正するためのアルゴリズムを開発することである．

制約違反の自動検出アルゴリズムを開発し，ツールに実装することにより，制約を満たすUML図を半自動的に得ることが可能となる．

3.2.2 研究プロセスと成果

(1) 研究プロセス

本研究目標を達成するため，以下のプロセスに従って研究開発を実施した．

1. 既存のUMLモデリングツールから，本ツールの入力となるUML図を作成するためのツールを選定する．
2. 選定したUMLモデリングツールによって作成されるデータについて，その構造を解析する．
3. UMLモデリングツールによって作成されるデータから，制約違反に対応した部分を抽出するための手法を開発する．
4. 制約に違反する部分に対して，制約を満たすような修正候補を提示するためのアルゴリズムを開発する．
5. 上述のアルゴリズムに基づく修正候補提示モジュールの設計書を作成する．

(2) 具体的な研究成果の内容

１．UMLモデリングツールの選定

現在，UMLモデリングツールは数多くのものが開発され，公開されている．本研究では，そのうち代表的なものを５つピックアップし，検討を行った．ピックアップしたUMLモデリングツールは以下の通りである．

- astah* community
- Enterprise Architect
- Microsoft Visio
- AmaterasUML
- ArgoUML

選定基準としては，まず，UML の多くの記法に対応していることが挙げられる．本ツールではUMLの状態マシン図とシーケンス図について，記述に制約を加えた上で扱っている

が，今後の拡張の可能性も考慮して，本研究で採用するモデリングツールは多くの記法に対応していることが望ましい．また，ツールで自動処理を行うため，テキスト形式や XML などの処理しやすい形式でモデリングデータを出力可能であることが求められる．また，本ツールの開発を通して形式手法の普及を目指すという観点から，ここで採用するモデリングツールも十分に普及しているか，または今後普及を目指す上での障壁が少ない方が望ましい．そのため，機能が同等であった場合は，ユーザインタフェースが優れているモデリングツールを優先して選択するべきである．同様の目的から，モデリングツールは無償であるか，有償であってもある程度の機能が無償で利用できることが望ましい．以下，これらの観点から検討を行った結果について述べる．

まず，astah* community（図 3-2-1）は，クラス図，ユースケース図，シーケンス図，アクティビティ図，コミュニケーション図，状態マシン図，コンポーネント図，配置図，合成構造図，オブジェクト図，パッケージ図など，UML の多くの記法に対応している．また，ユーザインタフェースにも優れている．出力形式として，XML による出力が可能であるが，これは有償版の astah* professional を導入する必要がある．

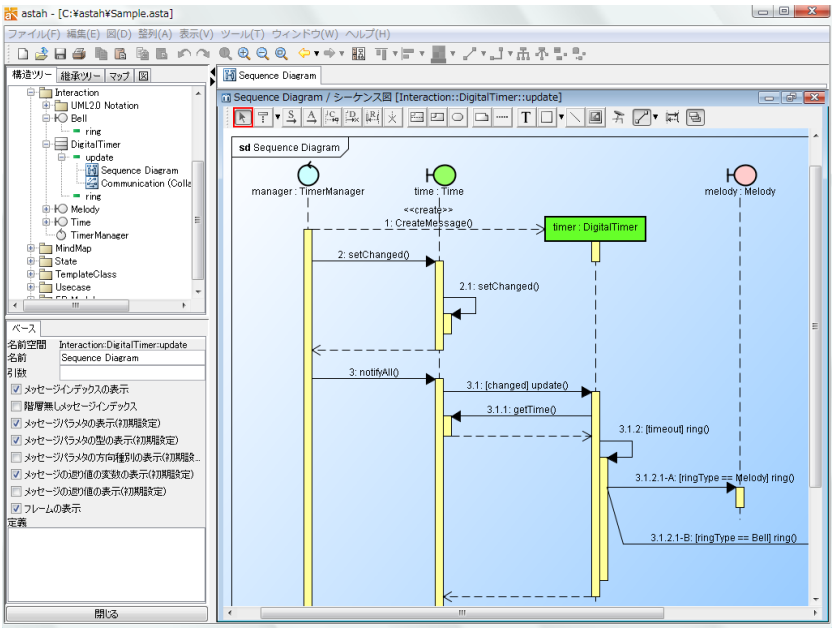


図 3-2-1:astah* community

次に Enterprise Architect（図 3-2-2）については，ユースケース図，クラス図，シーケンス図，状態マシン図，コミュニケーション図，アクティビティ図，タイミング図，パッケージ図，コンポーネント図，配置図を記述できる．出力形式として，Microsoft Word 互換の形式である RTF ドキュメントや HTML での出力が可能である．評価版は入手可能であるが，基本的には有償である．

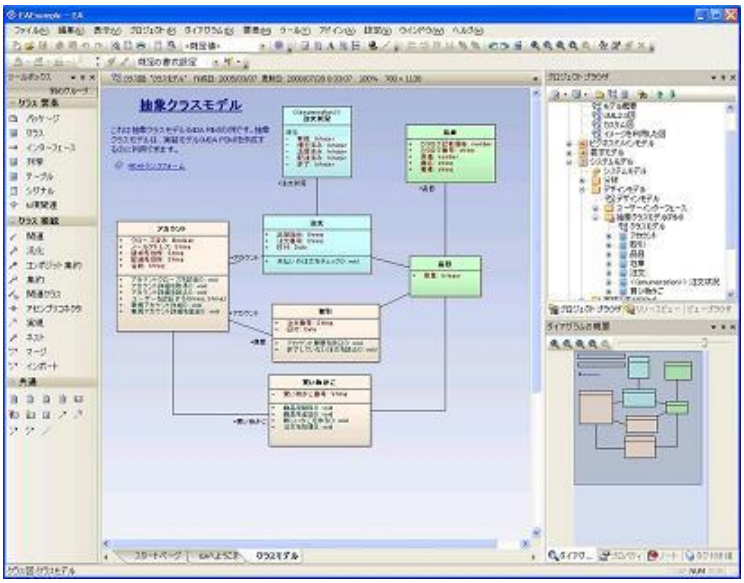


図 3-2-2:Enterprise Architect

Microsoft Visio（図 3-2-3）は，UML 記述のためのテンプレートが用意されており，描画のための UI に優れている．テンプレートに用意されていない図は，基本図形を組み合わせることで記述可能である．出力形式は Visio 図面の形式や JPG のような画像形式がある．XML での出力も可能となっているが，一般的な XML とは異なるため再利用は難しい．

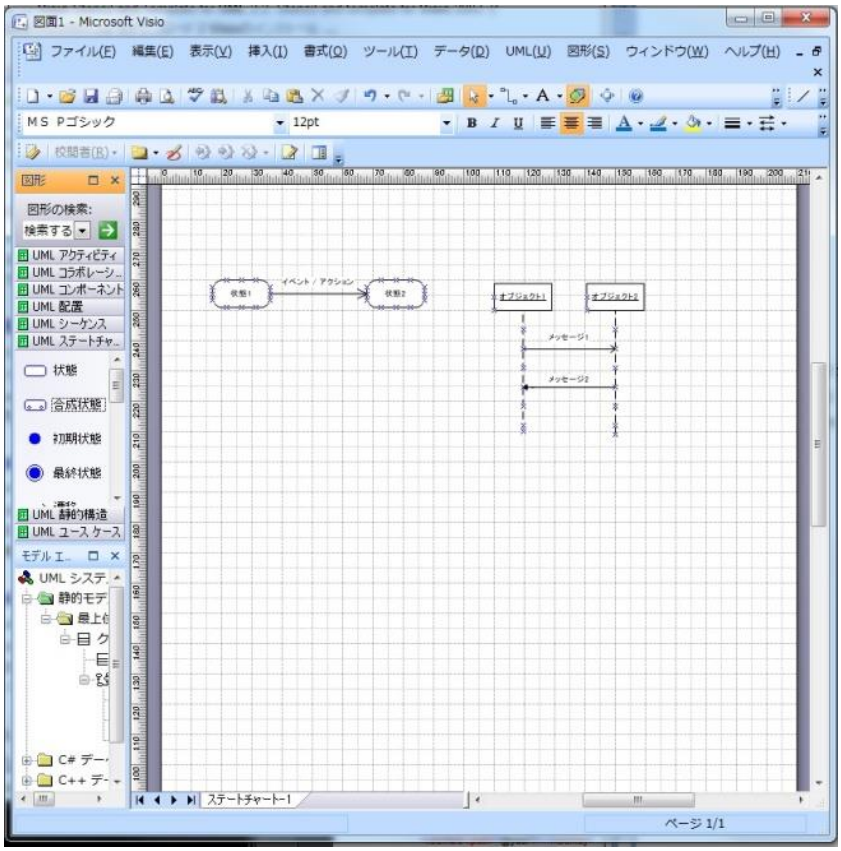


図 3-2-3:Microsoft Visio

AmaterasUML（図 3-2-4）は，Eclipse のプラグインであり，クラス図，ユースケース図，アクティビティ図，シーケンス図を記述できる．XML ベースの XMI 形式での出力が可能である．無償での利用が可能である．

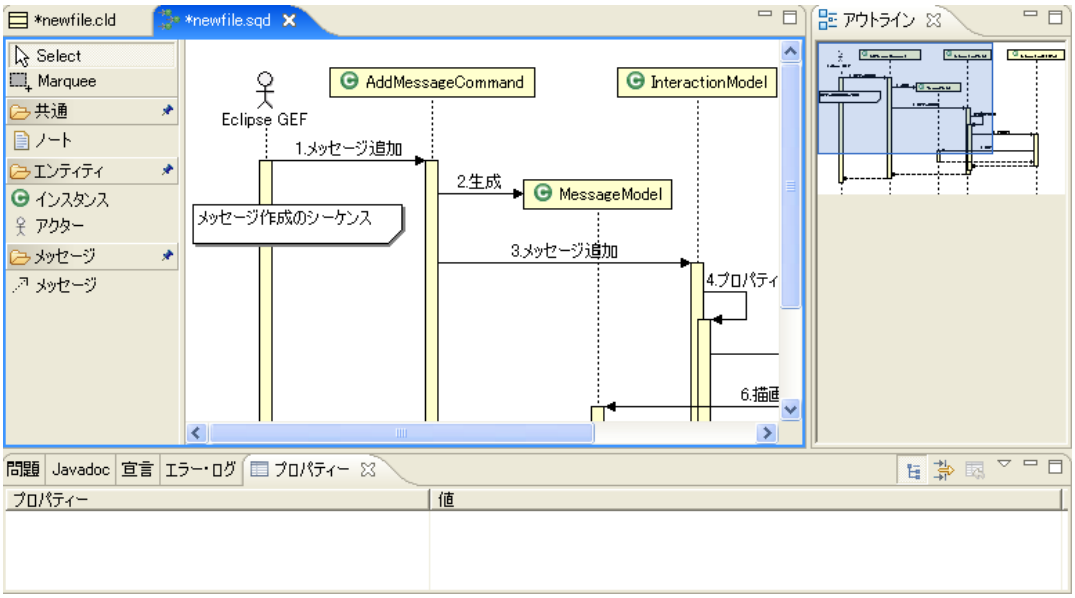


図 3-2-4:AmaterasUML

ArgoUML（図 3-2-5）は，クラス図，状態マシン図，ユースケース図，アクティビティ図，コミュニケーション図，配置図，シーケンス図が記述できる．XML による出力が可能であり，無償での利用が可能である．

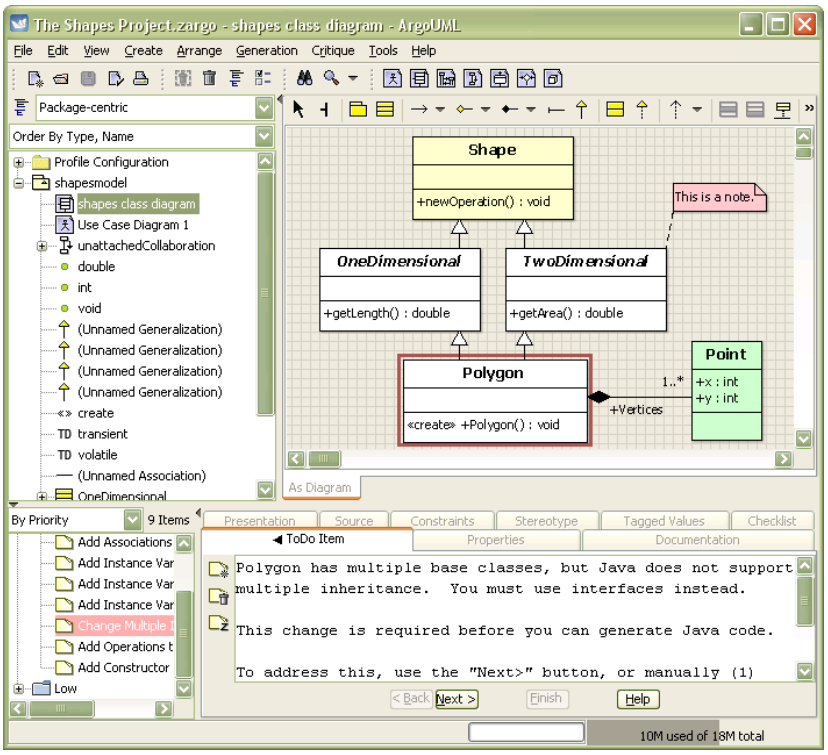


図 3-2-5:ArgoUML

まず， UML 図の記法への対応状況という観点から判断すると，astah* community，Enterprise Architect，そしてMicrosoft Visioの3つが候補して考えられる．次に，出力データ形式の処理の容易さでは，astah* communityかEnterprise Architectとなる．ユーザインタフェースはいずれも十分である．しかしながら，Enterprise Architectは他のツールに比べて導入コストが大きいため，本研究の目的の一つであるモデル検査の導入障壁の低下という観点からは他のツールを優先すべきだと考えられる．また，astah* communityはXML出力には有償版を導入する必要が生じるが，Java環境で利用できるAPIが提供されており，無償版で作成したモデリングデータであっても直接扱うことが可能である．これらの検討に基づき，本研究ではモデリングツールとしてastah* communityを採用するものとする．

２．UMLモデリングデータの構造解析

上述の通り，本ツールではJava環境で利用可能なastah* APIを用いることでastah* communityで作成したモデリングデータの処理を行う．astah* APIについてはWeb上にドキュメントが公開されている（http://astah.change-vision.com/ja/astah-api.html）．

astah* APIでは状態マシン図の情報はIStateMachineDiagramクラスのオブジェクトとして取得できる．各状態マシンの情報はメソッドgetStateMachine()を用いてIStateMachineクラスのオブジェクトとして取得し,状態の情報はメソッドgetVertexes()を用いてIStateクラスの配列として取得できる．さらにメソッドisSubmachineState()を用いることで合成状態であるか否かの判定が可能となる．同様に，擬似状態であるか否かもIStateクラスのオブジェクトにisForkPseudostate()などのメソッドを用いることで判定が可能である．

シーケンス図については相互作用に関する情報をIInteractionクラスのオブジェクトとして取得できる．その上でライフラインはメソッドgetLifelines()を用いることでILifelineクラスの配列として取得できる．また，getFragments()を用いることでライフライン上の結合フラグメントやメッセージの情報をICombinedFragmentやIMessageクラスの配列として取得できる．

３．モデリングデータから制約違反部分の抽出

上述の手順に従って，astah* communityで作成されたモデリングデータからUML図の情報を取得することができる．そして，取得した情報に対して，状態マシン図の制約定義（表3-1-2）およびシーケンス図の制約定義（表3-1-3）に違反するか否かを判定し，違反していた場合はモデリングデータ内の情報の該当部分にその旨の印を付加する．

４．修正候補提示アルゴリズムの開発

修正候補提示アルゴリズムは，(1) UML 図読み込み機能，(2)制約違反検出機能，(3)違反箇所提示機能，そして(4)修正候補提示機能の4つから構成される．

(1)UML図読み込み機能の処理の流れを図3-2-6に示す．ユーザがUMLモデリングツールastah* communityによって記述したUMLの状態マシン図およびシーケンス図の情報を

astah* API を用いて読み込む機能である.

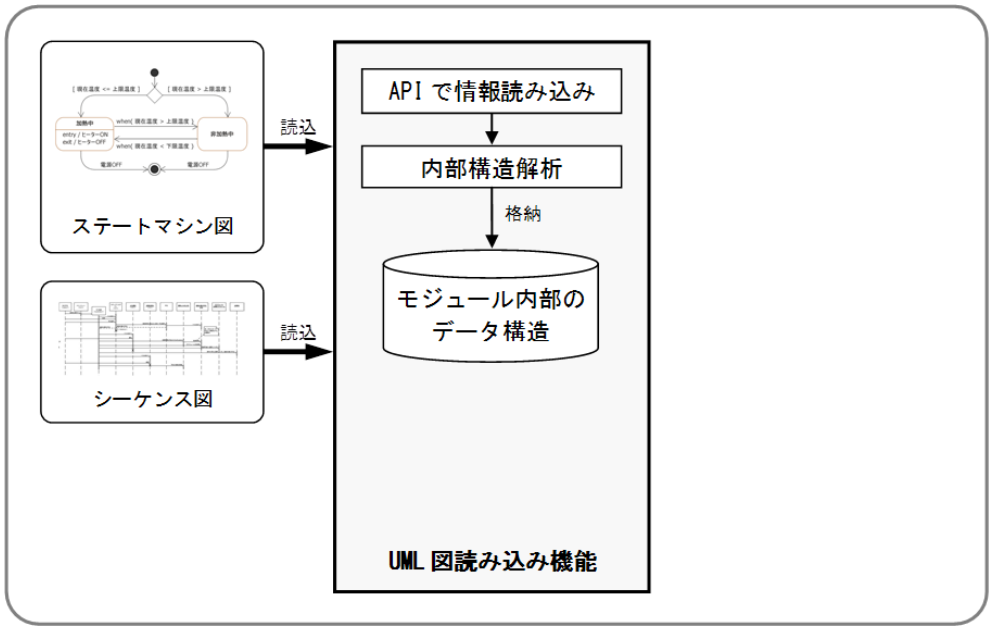


図 3-2-6:UML 図読み込み機能

(2)制約違反検出機能の処理の流れを図 3-2-7 に示す. UML 図読み込み機能によってモジュール内部のデータ構造に格納された状態マシン図およびシーケンス図の情報を読み込み, 表 3-1-2 および表 3-1-3 の制約と比較して, 制約違反（本ツールで対応不可の記法）の有無をチェックする機能である. チェックの結果, 制約違反が存在しない場合は, 本機能および本モジュールを終了する (以下の(3)(4)の機能は動作しない). 制約違反が存在する場合はデータ構造内の情報に, その旨の印を付加する.

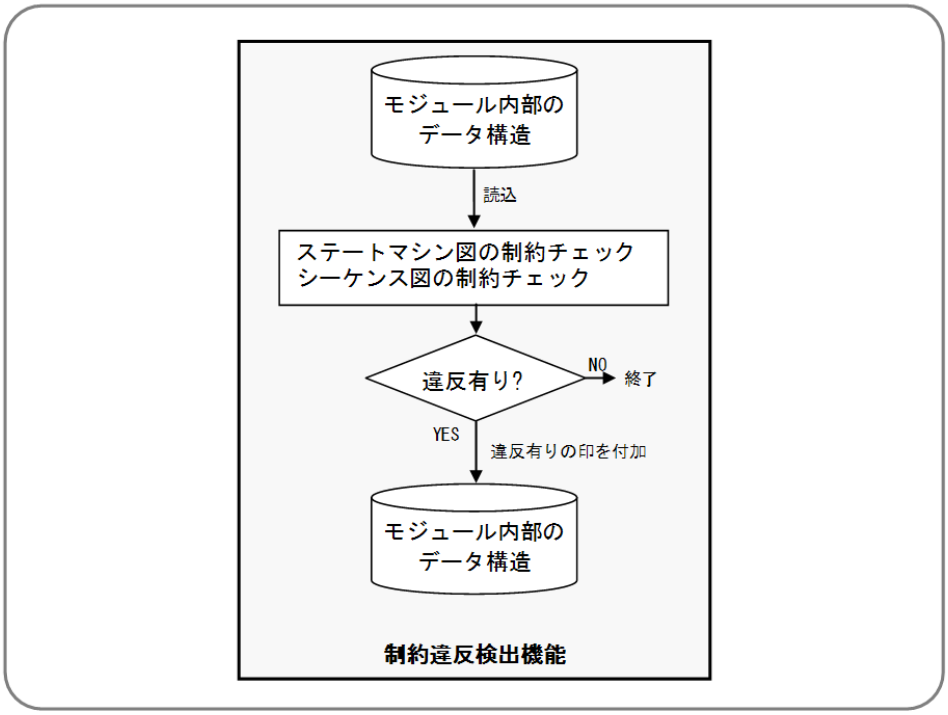


図 3-2-7:制約違反検出機能の処理の流れ

(3)違反箇所提示機能の処理の流れを図 3-2-8 に示す.制約違反検出機能によってモジュール内部のデータ構造に格納された状態マシン図およびシーケンス図の「違反有りの印」の情報を読み込み，それぞれの UML 図内の違反箇所を特定して，ユーザにメッセージで提示する機能である．本機能は次の修正候補提示機能と連動して動作する.

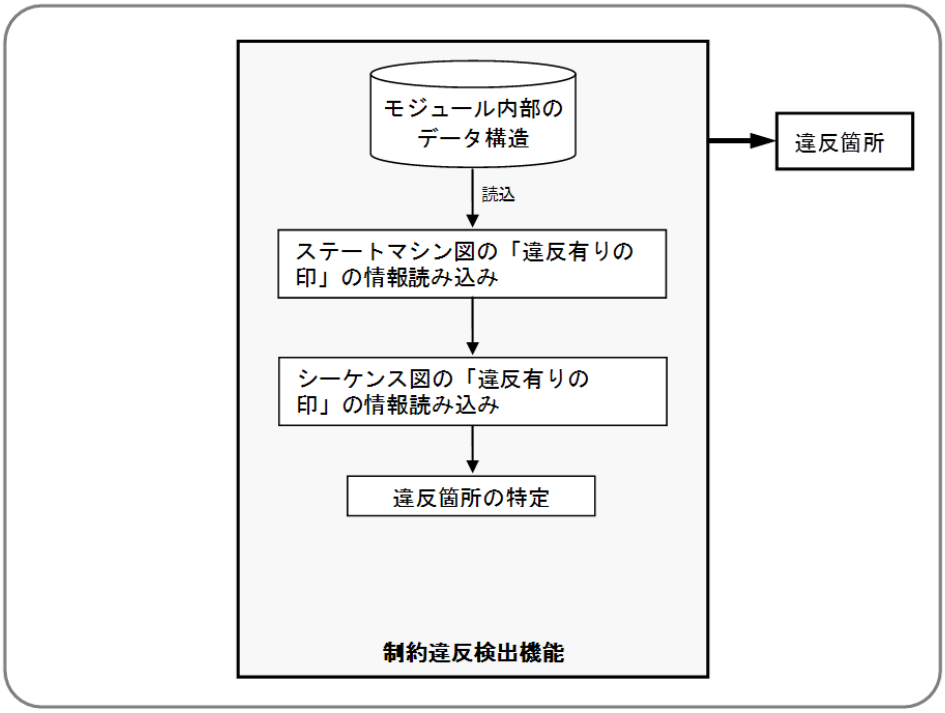


図 3-2-8:違反箇所提示機能の処理の流れ

(4)修正候補提示機能の処理の流れを図 3-2-9 に示す.制約違反検出機能によってモジュール内部のデータ構造に格納された状態マシン図およびシーケンス図の「違反有りの印」の情報を読み込み，制約違反をしている記法の修正候補を特定して，ユーザに表 3-2-1 および表 3-2-2 に示すメッセージで提示する機能である．本機能は前述の違反箇所提示機能と連動して動作する.

表 3-2-1:状態マシン図の違反検出時のメッセージ

NO	記法	メッセージ
1	合成状態	[名称] 図－ [名称] 状態：合成状態は利用できません
2	浅い履歴擬似状態	[名称] 図－ [名称] 状態：浅い履歴擬似状態は利用できません
3	深い履歴擬似状態	[名称] 図－ [名称] 状態：深い履歴擬似状態は利用できません
4	フォーク擬似状態	[名称] 図－ [名称] 状態：フォーク疑似状態は利用できません
5	ジョイン擬似状態	[名称] 図－ [名称] 状態：ジョイン疑似状態は利用できません
6	入場動作	[名称] 図－ [名称] 状態：入場動作は利用できません
7	実行活動	[名称] 図－ [名称] 状態：実行活動は利用できません
8	退場動作	[名称] 図－ [名称] 状態：退場動作は利用できません
9	直交状態（領域）	[名称] 図－ [名称] 状態－ [名称] サブ状態：直行状態は利用できません

表 3-2-2:シーケンス図の違反検出時のメッセージ

NO	記法	メッセージ
1	create メッセージ	［名称］図－［名称］：非同期メッセージとして扱いました
2	destroy メッセージ	［名称］図－［名称］：非同期メッセージとして扱いました
3	複合フラグメント	［名称］図－［名称］－［名称］ライフライン：複合フラグメントは利用できません

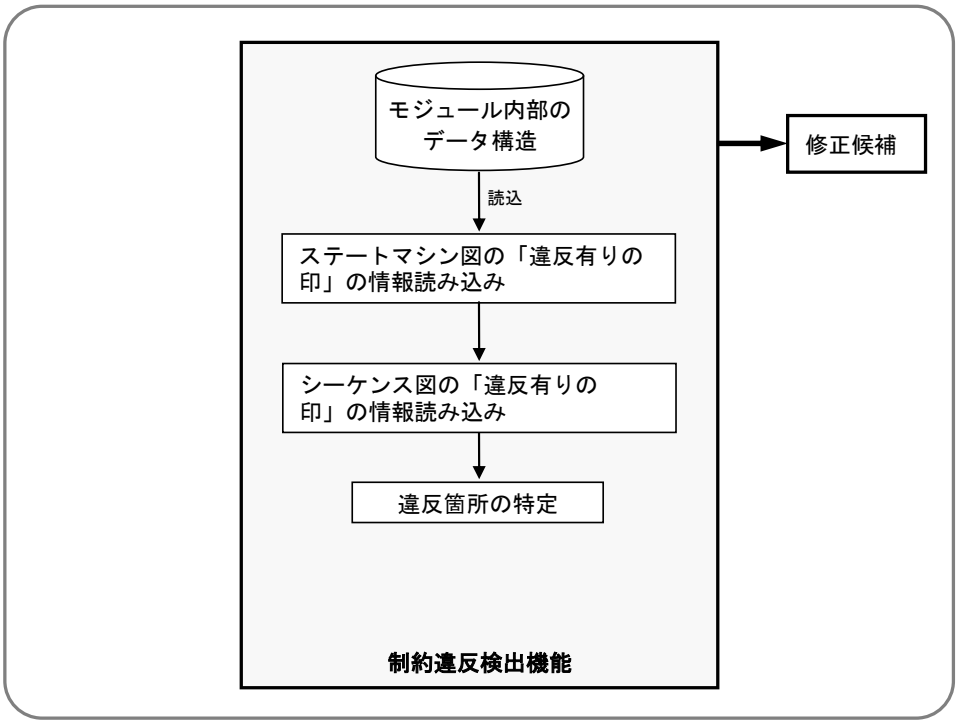


図 3-2-9:修正候補提示機能の処理の流れ

以上の機能を組み合わせることにより，本研究の目的であった修正候補提示アルゴリズムが実現される.

５．修正候補提示モジュールの設計書の作成

これまでに述べた astah* community で記述された UML 図のモデリングデータを読み込み，状態マシン図およびシーケンス図の記述制約に違反する部分を検出し，修正候補を提示するアルゴリズムを文書化し，修正候補提示モジュールの設計書を作成した.

3.2.3 課題とその対応

本研究では，3.1 節で示した制約定義に基づいて UML 図を記述することで，SMV 言語によるモデルへの自動変換を行うことを前提とした UML モデリングツールの選定を行い，そのモデリングデータの構造解析を行った上で，制約違反の検出および修正候補の提示アルゴリズムの開発を行った．ここで想定される課題は，入力された UML 図の制約違反をどのように検出するか，となる．

本研究で選定した UML モデリングツールである astah* community はモデリングデータへアクセスするためのインタフェースとして Java の API を提供しているため，この API を用いることでモデリングデータの情報を取得し，制約違反を検出することができた．

今後の課題としては，本アルゴリズムが提示する修正候補がユーザにとってどの程度有用であるか，さらなる評価実験が必要である．

将来の応用方法としては，SMV 言語でのモデル化に限らず，UML 図の自動処理の前処理として本アルゴリズムは応用できると考えられる．

3.3 研究目標 3「仕様の入力テンプレートの開発」

3.3.1 当初の想定

(1) 想定する仮説等

モデル検査を用いて設計検証を行うためには，対象となるシステムが満たすべき仕様をモデル検査ツールの入力となる検査式として記述する必要がある．ここで，多くのモデル検査ツールにおいて，検査式は CTL(Computational Tree Logic，計算木論理)や LTL(Linear Time Logic，線形時間論理)などの時相論理によって記述される．本ツールでモデル検査ツールとして利用する NuSMV では CTL と LTL による検査式の記述が可能である．しかしながら，ソフトウェアが満たすべき仕様を，時相論理により検査式として記述するためには，論理学や離散数学などの専門的な知識を要する．そこで，本ツールでは，ソフトウェアが満たすべき仕様を入力するためのテンプレートを定めることにより，専門的な知識を有しないユーザであっても容易に検査式の記述が可能であるように環境の整備を行う．

ここで，本研究目標を達成する上で，テンプレートにどこまでの柔軟性を持たせるかという問題点が想定される．ここでは，異常状態への到達可能性や不変条件によって記述される活性などの検証パターンを用いて記述された検証性質を仕様として想定し，これらの検証性質を記述可能なようにテンプレートを設計する．異常状態や不変条件は与えられた UML 図上の変数・状態やメッセージを用いて記述するものとする．

(2) 当初の到達目標と期待される効果

ここで設定する到達目標としては，検証パターンとして記述されるようなソフトウェアの代表的な特性が記述可能なテンプレートを開発することとする．

テンプレートを開発することにより，モデル検査の専門的な知識を持たないユーザであっても仕様を記述することが可能となる．

3.3.2 研究プロセスと成果

(1) 研究プロセス

本研究目標を達成するため，以下のプロセスに従って研究開発を実施した．

1. 先行研究で提唱された仕様パターンについて調査を行う．
2. 仕様パターンの中から，特に使用頻度が高いと思われるパターンについて，そのパターンに対応した仕様を記述するためのテンプレートを作成する．

(2) 具体的な研究成果の内容

１．仕様パターンの調査

仕様パターンは，自然言語で記述された検査項目をいくつかの定められた型へと当てはめることにより，検査式を容易に作成できるよう考案されたものである．仕様パターンは 5 つの指定範囲（スコープ）と 10 種の事象（パターン）によって構成されており，「どのような範囲で，どのような事象が発生するか」を表している．

仕様パターンで指定できる範囲を表 3-3-1 に示し，記述できる事象を表 3.3-2 に示す．

表 3-3-1:仕様パターンの指定範囲

NO	指定範囲	式表現	意味
1	Globally	GL	全ての範囲で
2	Before R	BF(R)	R より前で
3	After Q	AF(Q)	Q 以後で
4	Between Q and R	BA(Q,R)	Q 以後で R より前の間で
5	After Q until R	AU(Q,R)	Q 以後で R より前の間で

表 3-3-2:仕様パターンの事象

NO	検査事象	式表現	意味
1	Universality	UNI(P)	P が常に成立する
2	Existence	EXT(P)	P がいつか成立する。
3	Bounded Existence	BEX(P)	P への遷移が、最大でも２回起こる。
4	Precedence	PRC(P,S)	P に先立って S が成立する
5	Precedence Chain(1)	PC1(S,T,P)	S,T に先立って P が成立する
6	Precedence Chain(2)	PC2(P,S,T)	P に先立って S,T が成立する
7	Response	RES(P,S)	P に対して応答 S が成立する
8	Response Chain(1)	RC1(P,S,T)	P に対して応答 S,T が成立する
9	Response Chain(2)	RC2(S,T,P)	S,T に対して応答 P が成立する
10	Constrained Response Chain	CRC(P,Z,S,T)	P に対して Z 無しに応答 S,T が成立する

２．仕様テンプレートの作成

本ツールで用いる仕様テンプレートは，実システムにモデル検査を適用する際に，検査特性として頻繁に利用される安全性・活性・到達可能性などの特性を容易に記述可能とすることを目的として開発する．仕様パターンの表現能力は非常に高く，これらの特性も仕様パターンを用いて記述が可能である．しかしながら，仕様パターンを有効に活用するためには時相論理などの専門的な知識があることが望ましく，それ以外のユーザではその記述能力を十分に活用できない恐れがある．そのため，本ツールでは，利用頻度が高い安全性・活性・到達可能性の 3 つの特性を記述するための仕様テンプレートを作成する．本研究では，状態・メッセージ・変数という UML 図の重要な要素に関する安全性・活性・到達可能性の 3 つの特性を記述するためのテンプレートを提供する．

本研究で作成した仕様テンプレートを表 3-3-3 に示す．

表 3-3-3:仕様テンプレート

NO	特性	要素	式表現	意味
1	安全性	状態	SAFE(s)	決して状態 s はアクティブにならない
2		メッセージ	SAFE(m)	決してメッセージ m は送信されない
3		変数	SAFE(x,a)	決して変数 x の値が a とならない
4	活性	状態	LIVE(s)	いつか必ず状態 s がアクティブとなる
5		メッセージ	LIVE(m)	いつか必ずメッセージ m が送信される
6		変数	LIVE(x, a)	いつか必ず変数 x の値が a となる
7	到達可能性	状態	REACHABLE(s)	状態 s がアクティブになる可能性がある
8		メッセージ	REACHABLE(m)	メッセージ m が送信される可能性がある
9		変数	REACHABLE(x,a)	変数 x の値が a となる可能性がある

3.3.3 課題とその対応

本研究では，専門的な知識を有しないユーザであっても検査式の記述が可能となるように，ソフトウェアが満たすべき仕様を入力するためのテンプレートを開発した．ここで想定される課題は，テンプレートに対して，どこまで記述の柔軟性を持たせるべきか，となる．

本研究では仕様パターンに関する先行研究を調査した上で，専門的な知識を有しないユーザでも検査式の記述が可能となるように，安全性・活性・到達可能性といった利用頻度の高い特性を記述可能なテンプレートを提供することで，記述の柔軟性と利用の容易さのバランスを取ったテンプレートを実現した．

今後の課題として，テンプレートの表現能力が十分であるかについて，現場での開発者の意見を踏まえたさらなる議論が必要である．

将来の応用方法としては，本研究で作成した仕様テンプレートは，他のモデル検査ツールの入力となる検査式への変換も比較的容易であると期待される．

3.4 研究目標 4「仕様に基づく UML 図の部分抽出手法の開発」

3.4.1 当初の想定

(1) 想定する仮説等

モデル検査による設計検証を実施する際に発生する状態爆発問題を回避するために，検査対象となる特性に関連する部分のみを抽出してモデル化し，状態空間の探索を行うという手法がこれまでにも開発されている．本研究で対象とする UML 図の設計検証でも，同様のアプローチによって検証コストを削減できると期待される．そこで本ツールでは，テンプレートによって記述された仕様をもとに，UML 図の要素に対して仕様との関連度を求め，関連度の高い部分のみを抽出した上で SMV 言語によるモデル化を行う．

ここで，本研究目標を達成する上で，何を基準として仕様との関連度を判定するかという問題および関連度をどのように計算するかという問題が想定される．ここでは，検証性質に含まれる状態遷移への影響に基づいて定量的に関連度を求める．

(2) 当初の到達目標と期待される効果

ここで設定する到達目標としては，テンプレートを用いて与えられた仕様に対して，与えられた UML 図の要素との関連度を計算し，関連度に基づいて部分抽出を行うアルゴリズムを開発することとする．

仕様との関連度に基づく UML 図の部分抽出を行うことで，検証に用いるモデルのサイズを削減することができ，検証コストの削減が可能である．

3.4.2 研究プロセスと成果

(1) 研究プロセス

本研究目標を達成するため，以下のプロセスに従って研究開発を実施した．

1. 仕様テンプレートで記述された仕様と，状態マシン図およびシーケンス図の各要素との関連度の計算式を定義する．
2. 計算した関連度に基づいて，状態マシン図およびシーケンス図から関連度の高い部分を抽出するアルゴリズムを開発する．
3. 上述のアルゴリズムに基づく部分抽出モジュールの設計書を作成する．

(2) 具体的な研究成果の内容

１．仕様と UML 図の関連度の計算式の定義

関連度は，状態に関する要求仕様に対して，入力された状態マシン図の状態および遷移に対して定められる．ここで，要求仕様が参照する状態 s に対して，s が属する状態マシン図を M とする．関連度は，s との関連度が最も高い状態および遷移を関連度 1，最も低いものを関連度 4 として，以下のように定義される．

- M に含まれる状態の中で遷移によって s へと到達可能である状態および到達するまでに辿る遷移を，関連度 1 とする．

- M に含まれる状態および遷移で，1.に含まれないものを，関連度 2 とする．
- M とメッセージ名や変数名を共有する状態マシン図に含まれる状態および遷移を，関連度 3 とする．
- M とメッセージ名や変数名を共有しない状態マシン図に含まれる状態および遷移を，関連度 4 とする．

例として，仕様として safe(s)が与えられたときの，状態および遷移の仕様との関連度を図 3-4-1 に示す．

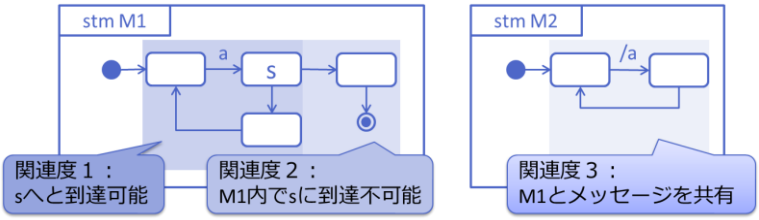


図 3-4-1：safe(s)との関連度

s へと到達可能な状態および s へと到達するまでに辿る遷移は関連度が 1 となっている．また s が含まれる状態マシン M1 について，s に到達不可能でかつ M1 に含まれる状態および遷移は関連度が 2 となっている．そして，M1 とメッセージを共有し，相互作用を行う対象である状態マシン M2 に含まれる状態および遷移については，関連度は 3 となる．それ以外の状態マシンに含まれる状態および遷移は，関連度は 4 となる．

２．関連度に基づく部分抽出アルゴリズムの開発

関連度に基づいて，状態および遷移を入力された状態マシン図から抽出する．本ツールでは，ユーザが関連度の閾値を入力し，その閾値より関連度が高い状態および遷移のみを状態マシン図から抽出して得られた状態マシン図を部分抽出モジュールの出力として生成する．

図 3-4-1 の例において関連度の閾値を 3 とした場合は，M1 および M2 以外の状態マシンは部分抽出モジュールからは出力されない．

３．部分抽出モジュールの設計書の作成

これまでに述べた，テンプレートで記述された仕様と状態マシン図の状態および遷移との関連度を計算し，関連度に基づいて部分抽出を行うアルゴリズムを文書化し，部分抽出モジュールの設計書を作成した．

3.4.3 課題とその対応

本研究では，モデル検査における状態爆発問題を回避するために，検査特性と関連する部分のみを抽出してモデル化を行うための手法を開発した．本ツールでは，テンプレートによって記述された仕様を基に，状態マシン図の状態および遷移の関連度を求め，関連度

の高い部分のみを抽出した上で SMV 言語によるモデル化を行う.ここで想定される課題は,何を基準として仕様との関連度を判定するか，そして関連度をどのように計算するか，となる.

本研究では，検証性質に含まれる状態遷移への影響に基づいて，定量的に関連度を求めている.

今後の課題として，本アルゴリズムにおける仕様との関連性の判定が妥当であるか，例題を用いた評価が必要である.

将来の応用方法としては，本研究で開発した部分抽出手法は，設計検証やソースコード検証におけるモデルサイズの削減への応用が可能である.

3.5 研究目標 5「仕様に基づく UML 図の記述抽象化手法の開発」

3.5.1 当初の想定

(1) 想定する仮説等

状態爆発問題の回避を目的とした 3.4 節で述べたモデルの部分抽出とは別のアプローチとして，検査対象となる特性と関連しない要素を抽象化することで状態を集約し，状態数を削減するという手法が開発されている．本ツールでは，テンプレートによって記述された仕様をもとに，UML 図の要素が仕様と関連しているか判定し，関連しない部分は抽象化した上で SMV 言語によるモデル化を行う.

ここで，仕様と関連しているか否かをどう判定するかという問題および関連しない要素をどう抽象化するかという問題が想定される．ここでは，変数の定義域の単純化や繰り返しの縮約など，従来の抽象化手法を応用する.

(2) 当初の到達目標と期待される効果

ここで設定する到達目標としては，テンプレートを用いて与えられた仕様に対して，与えられた UML 図の要素との関連性を判定するアルゴリズムを開発し，さらに，関連性に基づいて，UML 図の記述を抽象化するアルゴリズムを開発することとする.

仕様との関連性に基づく記述抽象化を行うことで，モデルサイズの削減が可能となり，検証コストのさらなる削減を実現できる.

3.5.2 研究プロセスと成果

(1) 研究プロセス

本研究目標を達成するため，以下のプロセスに従って研究開発を実施した.

1. 仕様テンプレートで記述された仕様をもとに，状態マシン図およびシーケンス図の要素に対して，抽象化による影響があるか否かを判定するアルゴリズムを開発する.
2. 上述のアルゴリズムに基づく，関連性判定モジュールの設計書を作成する.
3. 関連性判定の結果に基づいて，状態マシン図およびシーケンス図の要素について抽象化を行うためのアルゴリズムを開発する.

4.　上述のアルゴリズムに基づく抽象化モジュールの設計書を作成する．

(2) 具体的な研究成果の内容

１．仕様と UML 図との関連性判定アルゴリズムの開発

関連性は，変数に関する要求仕様に対して，入力された状態マシン図の変数に対して定められる．ここで，要求仕様が参照する変数を x とする．関連性は，x との関連性があるかないかの 2 値として，以下のように定義される．

- x の値を変化させるアクション，すなわち代入文の右辺に含まれる変数は x と関連性があるとする．
- x と関連性がある変数 y に対して，y と関連性がある変数も同様に x と関連性があるとする．

例として，仕様として safe(x,2)が与えられたときの，変数の仕様との関連度を図 3-5-1 に示す．

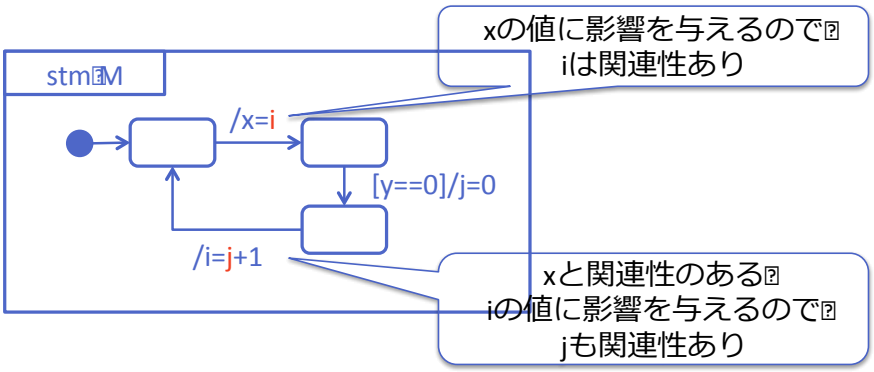


図 3-5-1：safe(x,2)との関連性

この状態マシン図の遷移は x=i というアクションを含むため i は x と関連性があると判定される．また，i=j+1 というというアクションも含んでいるため，j は i と関連性があり，すなわち x とも関連性があると判定される．y については，x，i および j のいずれの代入文でも右辺に含まれていないため，x との関連性はないと判定される．

２．関連性判定モジュールの設計書の作成

上述の，テンプレートで記述された仕様と状態マシン図の変数との関連性を判定するアルゴリズムを文書化し，関連性判定モジュールの設計書を作成した．

３．関連性判定の結果に基づく抽象化アルゴリズムの開発

関連性判定の結果に基づき，関連性がないと判定された変数およびその変数が現れるガード条件およびアクションを状態マシン図から削除して得られた状態マシン図を，抽象化

モジュールの出力として生成する．

図 3-5-1 の例では，変数 y に関連するガード条件[y==0]が削除される．

４．抽象化モジュールの設計書の作成

上述の，関連性判定の結果に基づいて，関連性のない変数に関連する記述を削除することで抽象化を行うアルゴリズムを文書化し，抽象化モジュールの設計書を作成した．

3.5.3 課題と対応

本研究では，3.4 節と同様に状態爆発問題の回避を目的として，検査対象となる特性と関連しない要素を抽象化することで状態数を削減するための手法を開発した．本ツールでは，テンプレートによって記述された仕様をもとに，状態マシン図の変数が仕様と関連しているか判定し，関連しない変数を抽象化した上で SMV 言語によるモデル化を行う．ここで想定される課題は，仕様と関連しているか否かをどう判定するか，そして，関連しない要素をどう抽象化するか，である．

本研究では,変数の定義域の単純化や繰り返しの縮約など,従来の抽象化手法を応用し，仕様に含まれる変数との依存関係をチェックすることで,仕様との関連性を判定している．

今後の課題としては，本アルゴリズムによる抽象化によって，どの程度の検証コストの削減が可能となったかについて，適用実験による評価が必要である．

将来の応用方法としては，本研究で開発した抽象化手法は，設計検証におけるモデルサイズの削減への応用が可能である．

3.6 研究目標 6「UML 図および仕様から SMV 言語への自動変換手法の開発」

3.6.1 当初の想定

(1) 想定する仮説等

モデル検査ツールを用いて検証を行うためには，検査対象のシステムをモデル検査ツールの入力形式に従ってモデル化する必要がある．しかしながら，モデル検査ツールの入力言語は一般的なプログラム言語とは異なった構造を持つため，専門的な知識のないユーザが扱うことは困難である．また，検査対象のシステムの規模が大きくなるに従って，モデルの記述量も大きくなり，人手でモデル化を行うことは難しい．そこで，UML 図から SMV 言語によるモデルへの自動変換を行うことで，人手での処理と比較してモデル作成のコストを大幅に削減可能であると期待される．

ここで，変換に用いるアルゴリズムによっては計算量やモデルの記述量が爆発的に増加する危険性がある．ここでは，研究者がこれまでに開発した変換アルゴリズムを応用することで，計算量・記述量を UML 図の規模に対して多項式時間程度に押さえることが可能である．

(2) 当初の到達目標と期待される効果

ここで設定する到達目標としては，UML 図および入力テンプレートで記述された仕様から，SMV 言語によるモデルを自動的に作成するアルゴリズムを開発することとする．

SMV 言語によるモデルを自動生成することにより，人手の処理と比較して，モデル作成コストの大幅な削減が可能である．

3.6.2 研究プロセスと成果

(1) 研究プロセス

本研究目標を達成するため，以下のプロセスに従って研究開発を実施した．

1. 状態マシン図およびシーケンス図，そして仕様テンプレートによる記述と，SMV 言語による記述の対応関係を整理する．
2. 状態マシン図およびシーケンス図と仕様テンプレートで記述された仕様を，SMV 言語によるモデルへと変換するアルゴリズムを開発する．
3. 上述のアルゴリズムに基づく自動変換モジュールの設計書を作成する．

(2) 具体的な研究成果の内容

１．UML 図および仕様テンプレートによる記述と SMV 言語の対応関係の整理

本ツールでは，UML 図および仕様テンプレートで与えられた検査対象のシステムと検証すべき仕様を SMV 言語によってモデル化し，NuSMV を用いて検証する．SMV 言語は図 3-6-1 に示す構成をもつ．変数宣言部では検査対象となるモデルの要素を変数として宣言する．なお，本ツールでは，変数はブール型と列挙型の二通りのみを用いる．状態遷移系記述部では，モデルの各要素が，他の要素の値に基づいて定義される遷移条件に依存して，どのように変化するかを記述する.そして検査式記述部では,検査式を CTL によって記述する．

```
MODULE main

VAR
  変数宣言部である。
  記述形式は「変数名 : 変数の型;」である。
  本モジュールでは変数の型は以下の 2 つのみとする。
    □BOOL 型 (例  flag : boolean;) ← FALSE と TRUE の 2 値である。
    □LIST 型 (例  CPU  : {WAIT, READY, BUSY};

ASSIGN
  状態遷移系記述部である。
  本モジュールでの記述形式は以下である。
    init(変数名) := 初期状態;
    next(変数名) := case
      遷移条件_1 : 遷移先の状態_1;
      遷移条件_2 : 遷移先の状態_2;


      TRUE : 変数名;
    esac;



SPEC CTL 式_1
SPEC CTL 式_2
SPEC CTL 式_3
```

図 3-6-1:SMV 言語によるモデルの基本構成

UML の状態マシン図は，状態および遷移によって構成されている．まず，これらの要素をどのように SMV 言語によってモデル化するかについて検討を行った．状態マシン図の振る舞いを SMV の状態遷移システムとして表す場合，SMV 内で変数として表すべき要素は，

- 状態
- メッセージ
- 変数

の 3 つとなる．そして，遷移の発火によるこれらの要素の値の変化を，SMV 内の遷移関係として記述することで SMV による状態マシン図のモデル化が可能となる．

次に，シーケンス図と SMV 言語との対応関係について検討を行った．シーケンス図は，ライフラインとその間で実行されるメッセージ通信によって構成されている．本ツールでは，シーケンス図で記述されているメッセージに着目し，状態マシン図がそのメッセージを正しく実行しているか否かについて検証を行う．したがって，SMV 内で変数として表すべき要素は，

- メッセージ

のみとなる．シーケンス図で記述されたメッセージの振る舞いは，SMV が検証する検査

式として記述する.

最後に，仕様テンプレートと SMV 言語との対応関係について検討を行った．仕様テンプレートでは，SMV によって検証する検査性質を記述するためのものである．したがって，仕様テンプレートによって記述される仕様は，SMV 言語によるモデルの中では，CTL 式や LTL 式によって記述される検査式に対応する．本ツールにおける仕様テンプレートで記述する検査特性である，安全性，活性，そして到達可能性の 3 つの特性は，それぞれ CTL の演算子によって表現することができる．まず安全性は，CTL における演算子である，AG(Always Globally，全ての場合において常に～が成り立つ)を用いて表現できる．次に活性は，同じく CTL における演算子である AF(Always in the Future，全ての場合において将来いつか～が成り立つ)を用いて表現できる．そして到達可能性は，CTL における演算子である EF(Eventually in the Future，ある場合において将来いつか～が成り立つ)を用いて表現できる.

以上のように，状態マシン図，シーケンス図，そして仕様テンプレートによって記述される情報と，SMV 言語による記述との対応関係が整理できた．この対応関係に基づいて，状態マシン図，シーケンス図，そして仕様テンプレートから SMV 言語によるモデルへの自動変換を行うアルゴリズムを設計する.

２．状態マシン図およびシーケンス図，仕様テンプレートから SMV 言語への変換アルゴリズムの開発

上述の対応関係に基づいて，状態マシン図およびシーケンス図，そして仕様テンプレートから SMV 言語によるモデルを生成する．本手法では，状態マシン図の情報から SMV 言語によるモデルを生成し，シーケンス図および仕様テンプレートの情報から検査式を生成する．状態マシン図およびシーケンス図のモデルデータは UML モデリングツール astah*によって作成される．したがって，このアルゴリズムの概要は図 3-6-2 に示す通りとなる.

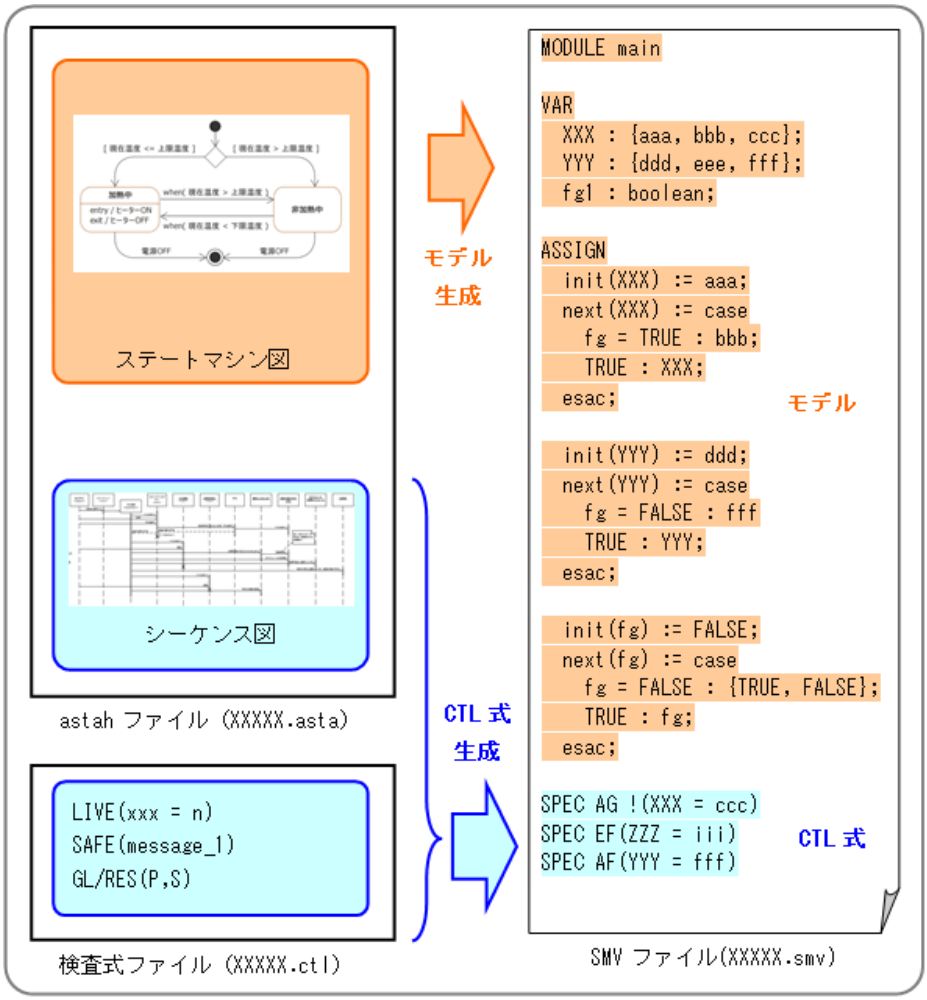


図 3-6-2:SMV プログラムへの自動変換アルゴリズムの概要

以下，状態マシン図，シーケンス図，そして仕様テンプレートから SMV プログラムへの変換を行うための手法について順に述べる．

本アルゴリズムでは,状態マシン図から SMV プログラムへの変換を(1)状態マシン図の記法の等価変換(2)状態マシン図から SMV 言語によるモデルの生成の 2 段階で行う．(1)では，状態マシン図の擬似状態（初期擬似状態，終了状態，選択擬似状態，ジャンクション擬似状態）を，等価な単純状態による表現へと変換する．その上で(2)で SMV 言語によるモデルを生成する．

まず，初期擬似状態については，以下の記述規則が存在する．

- 1 つのステートマシン図には 1 つの初期擬似状態のみが存在し，省略することは不可とする．
- 初期擬似状態に入力する遷移経路は存在しない．
- 初期擬似状態から出力する遷移経路は 1 本のみである．
- 遷移経路には[ガード]とアクションを記述できるが，トリガとなるイベントは記述できない．

したがって，図 3-6-3 に示すように，初期擬似状態を単純状態へと置き換えた後，変換

した単純状態の値を，SMV プログラム内での当該変数の初期状態として定める．

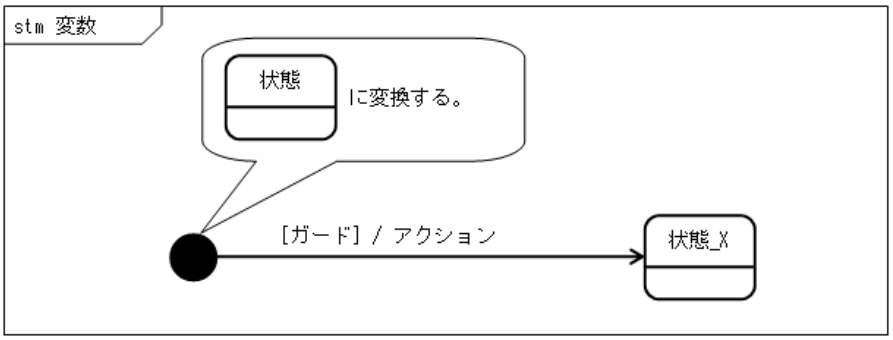


図 3-6-3:状態マシン図の変換（初期擬似状態の排除）

終了状態については，以下の記述規則が存在する．

- 1 つのステートマシン図には複数の終了状態が存在しても良いく，省略も可能とする．
- 終了状態に入力する遷移経路は複数存在しても良い．
- 終了状態から出力する遷移経路は存在しない．
- 遷移経路にはトリガとなるイベント，[ガード]，そしてアクションを記述できる．

したがって，図 3-6-4 に示すように，終了状態を単純状態へと置き換えた後，変換した単純状態へと到達した後は，以降の遷移は前回値を保持する遷移のみとなるよう SMV プログラムを記述する．

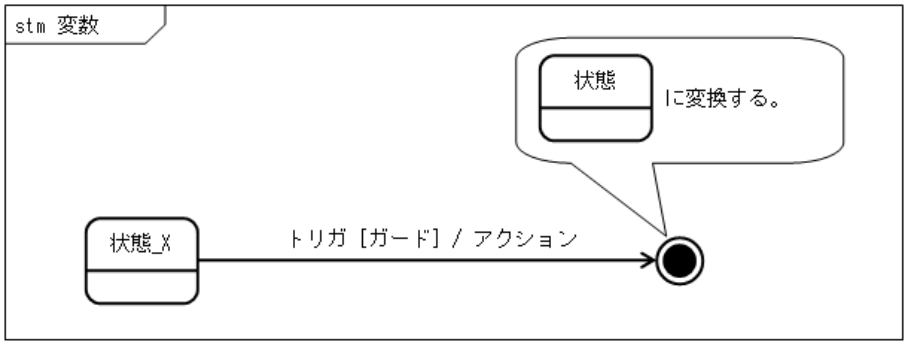


図 3-6-4:状態マシン図の変換（終了状態の排除）

選択擬似状態については，以下の記述規則が存在する．

- 状態遷移系として，1 つの状態として扱われる．
- 選択擬似状態に入力する遷移経路は複数存在しても良い．
- 初期擬似状態，単純状態，ジャンクション擬似状態，選択擬似状態から入力できる．
- 終了状態，単純状態，ジャンクション擬似状態，選択擬似状態に出力できる．
- 選択擬似状態から出力する遷移経路は複数存在しても良い．
- 遷移経路にはトリガとなるイベント，[ガード]，そしてアクションを記述できる．
- 選択擬似状態から出力する遷移経路は，最低でも 1 本が遷移可能でなければならな

い.

したがって，図 3-6-5 に示すように，選択擬似状態を単純状態に置き換えた後，SMV プログラム内では当該変数の 1 つの状態として扱う.

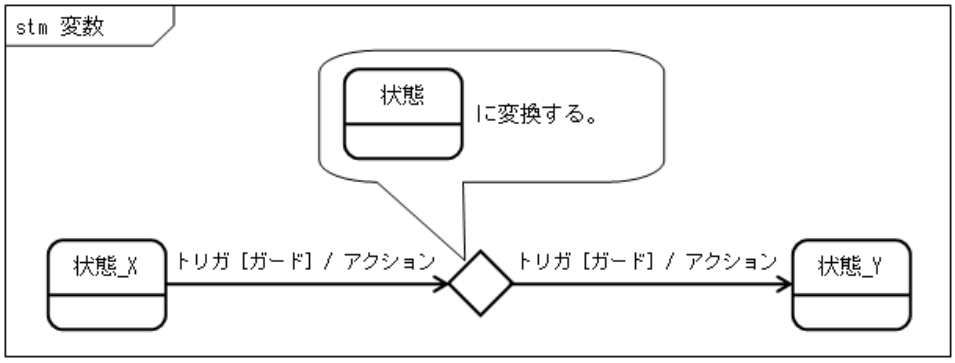


図 3-6-5:状態マシン図の変換（選択擬似状態の排除）

最後にジャンクション擬似状態については，以下の記述規則が存在する.

- 状態遷移系として，1 つの状態としては扱われない.
- ジャンクション擬似状態に入力する遷移経路は複数存在しても良い.
- ジャンクション擬似状態から出力する遷移経路は複数存在しても良い.
- 初期擬似状態，単純状態，接合点，選択擬似状態から入力できる。
- 終了状態，単純状態，ジャンクション擬似状態，選択擬似状態に出力できる。
- ジャンクション擬似状態に入力する遷移経路にはトリガとなるイベント,[ガード], そしてアクションを記述できる.
- ジャンクション擬似状態から出力する遷移経路には[ガード]およびアクションのみ記述できる．トリガとなるイベントは記述できない.

したがって，図 3-6-6 に示すように，ジャンクション擬似状態を削除した上で，前後の遷移のトリガ，ガード，アクションを統合して一つの遷移へと変換する.

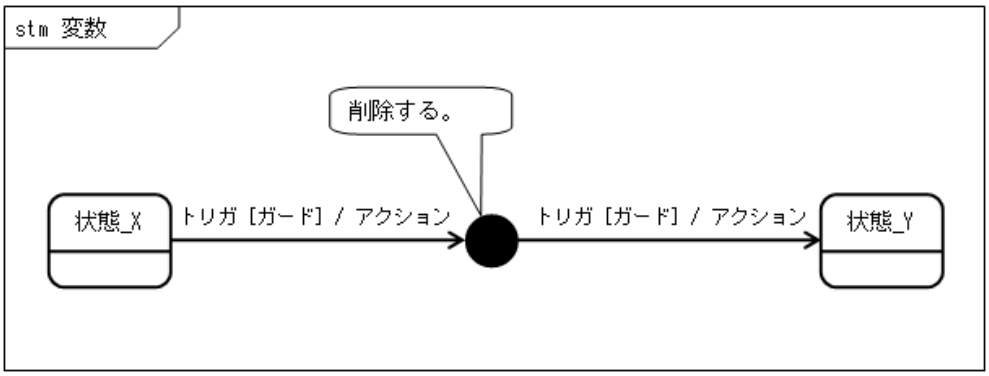


図 3-6-6:状態マシン図の変換（ジャンクション擬似状態の排除）

以上の手順に従い，単純状態と遷移経路のラベル（トリガ，[ガード]，アクション）の

みが存在するような状態マシン図へと変換した上で，SMV 言語によるモデルを生成する．まず，変換の概要について示す．図 3-6-7 に示すのは，得られた状態マシン図の基本的な構成である．

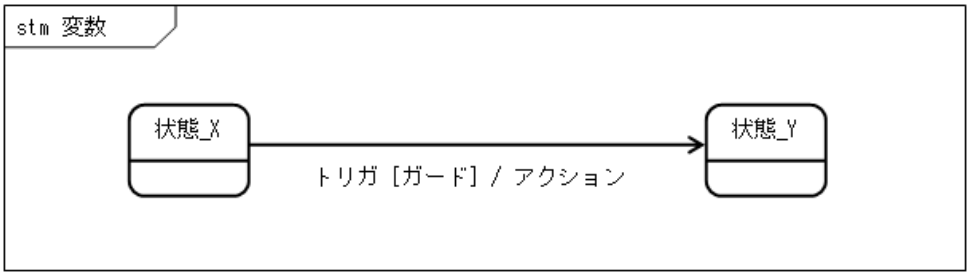


図 3-6-7:状態マシン図の基本構成

本ツールでは，この状態マシン図から図 3-6-8 に示すような SMV 言語によるモデルが生成される．

```
MODULE main

VAR
  変数                    : {初期状態, ・・・ ,状態_X, 状態_Y};
  アクションの左辺の変数 : 型を入力してください;

ASSIGN
  init(変数) := 初期状態;
  next(変数) := case
    初期状態からの遷移条件              : 初期状態の次状態;
    変数 = 状態_X & (トリガ) & (ガード) : 状態_Y;
    状態 Y からの遷移条件               : 状態_Y の次状態;
    TRUE : 変数;
  esac;

  init(アクションの左辺の変数) := 初期状態を入力して下さい;
  next(アクションの左辺の変数) := case
    変数 = 状態_X & next(変数) = 状態_Y : アクションの右辺の式;
    TRUE : アクションの左辺の変数;
  esac;
```

図 3-6-8:状態マシン図から生成したモデル

灰色で網掛けした部分はユーザが直接入力する．状態マシン図には，変数の型や初期状態に関する記述を記法としてサポートしていないため，これらの値はユーザが直接入力する必要がある．

以下，一般化した生成手順について，図 3-6-9 の状態マシン図を例として述べる．本手

法では，状態遷移と遷移の発火によるアクションの実行を個別に状態遷移系としてモデル化することで，SMV プログラムとして状態マシン図の振る舞いのモデルを得る．以下では，図の簡略化のため，「トリガ_N［ガード_N］/ アクション_N」を省略し，「ト_N［ガ_N］/ ア_N」と記載することがある．

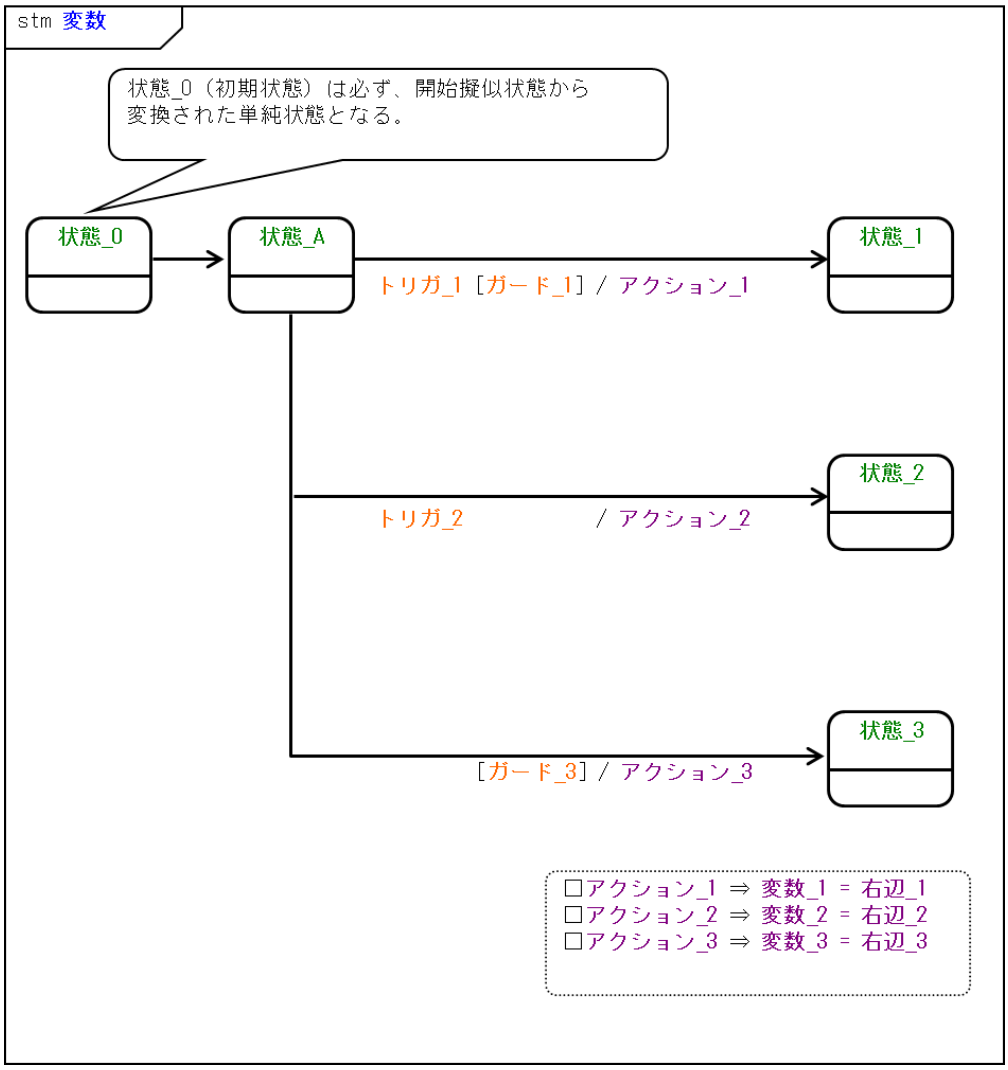


図 3-6-9：状態マシン図の例

この状態マシン図における状態遷移は，図 3-6-10 のようにモデル化できる．図の①は遷移経路の遷移条件が全て成立する場合を表し，②は 2 つの遷移経路の遷移条件のみが成立する場合を表し，③は 1 つの遷移経路の遷移条件のみが成立する場合を表している．

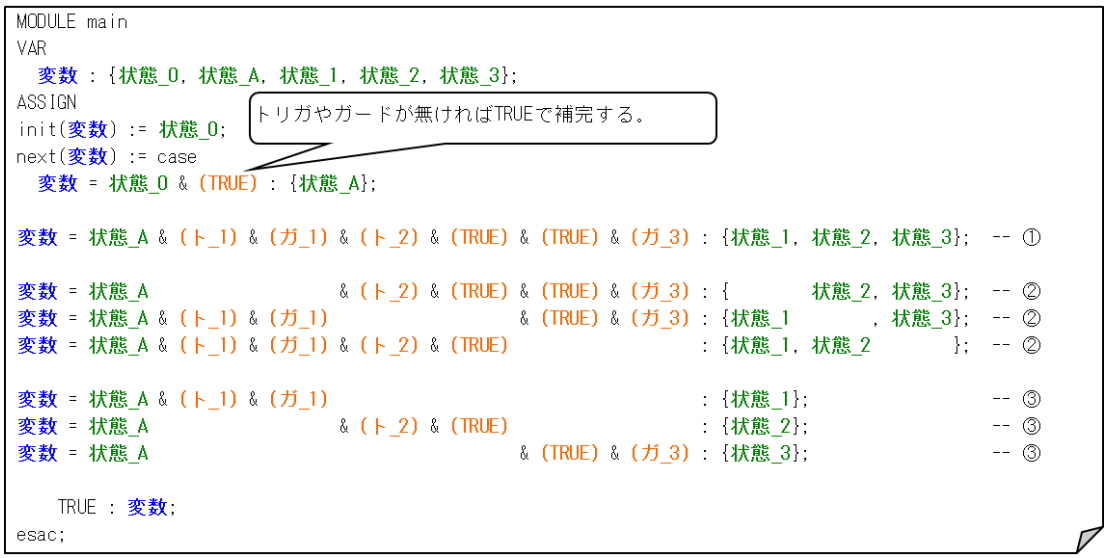


図 3-6-10:状態遷移のモデル生成

遷移の発火によるアクションの実行は,図 3-6-11～図 3-6-13 のようにモデル化される.アクションのモデルを生成する際には，アクションの左辺の変数に着目して，異なるアクションでも左辺の変数が同じであれば同一の変数としてモデルを生成する.

図 3-6-11 は 3 つのアクションの左辺の変数が全て異なる場合であり，図 3-6-12 は 2 つのアクションの左辺の変数が同じ場合である（例では変数_1 と変数_3 が同じ）．また，図 3-6-13 は 3 つのアクションの左辺の変数が全て同じ場合である.

```
MODULE main

VAR
  変数_1 : 型を入力してください;
  変数_2 : 型を入力してください;
  変数_3 : 型を入力してください;

ASSIGN
  init(変数_1) := 初期状態を入力して下さい;
  next(変数_1) := case
    変数 = 状態_0 & next(変数) = 状態_1 : 右辺_1;
    TRUE : 変数_1;
  esac;

  init(変数_2) := 初期状態を入力して下さい;
  next(変数_2) := case
    変数 = 状態_0 & next(変数) = 状態_2 : 右辺_2;
    TRUE : 変数_2;
  esac;

  init(変数_3) := 初期状態を入力して下さい;
  next(変数_3) := case
    変数 = 状態_0 & next(変数) = 状態_3 : 右辺_3;
    TRUE : 変数_3;
  esac;
```

図 3-6-11:アクションのモデル生成（変数が全て異なる場合）

```
MODULE main

VAR
  変数_1 : 型を入力してください;
  変数_2 : 型を入力してください;

ASSIGN
  init(変数_1) := 初期状態を入力して下さい;
  next(変数_1) := case
    変数 = 状態_0 & next(変数) = 状態_1 : 右辺_1;
    変数 = 状態_0 & next(変数) = 状態_3 : 右辺_3;
    TRUE : 変数_1;
  esac;

  init(変数_2) := 初期状態を入力して下さい;
  next(変数_2) := case
    変数 = 状態_0 & next(変数) = 状態_2 : 右辺_2;
    TRUE : 変数_2;
  esac;
```

図 3-6-12:アクションのモデル生成（変数_1 と変数_3 が同じ場合）

```
MODULE main

VAR
  変数_1 : 型を入力してください;

ASSIGN
  init(変数_1) := 初期状態を入力して下さい;
  next(変数_1) := case
    変数 = 状態_0 & next(変数) = 状態_1 : 右辺_1;
    変数 = 状態_0 & next(変数) = 状態_2 : 右辺_2;
    変数 = 状態_0 & next(変数) = 状態_3 : 右辺_3;
    TRUE : 変数_1;
  esac;
```

図 3-6-13:アクションのモデル生成（変数が全て同じ場合）

シーケンス図から SMV プログラムを生成する手順について述べる．本ツールでは，与えられたシーケンス図に対して，シーケンス図に記載されたメッセージの振る舞いに関する CTL 式を生成する．具体的には，シーケンス図に記載されたメッセージの安全性，活性，そして到達可能性を確認するための CTL 式を生成する．安全性，活性，到達可能性について表 3-6-1 に示す．

表 3-6-1:安全性・活性・到達可能性

特性	形式
安全性	システムで発生してはならない事象が発生しないこと
活性	システムで必ず発生すべき事象が発生すること
到達可能性	システムが一度は到達すべき状態に到達すること

表 3-6-1 に従って，シーケンス図に記載されたメッセージ m の安全性・活性・到達可能性は表 3-6-2 に示すとおり定義される．

表 3-6-2:メッセージ m の安全性・活性・到達可能性

特性	形式
安全性	決してメッセージ m は送信されない
活性	いつか必ずメッセージ m が送信される
到達可能性	メッセージ m が送信される可能性がある

したがって，メッセージ m の安全性・活性・到達可能性を表す CTL 式を生成する際の規則は表 3-6-3 に示す通りとなる．

表 3-6-3:シーケンス図のメッセージに関する CTL 式を生成する際の規則

特性	規則
安全性	CTL 式の AG 演算子を用いて生成する
活性	CTL 式の AF 演算子を用いて生成する
到達可能性	CTL 式の EF 演算子を用いて生成する

最後に，仕様テンプレートから SMV プログラムを生成する手順について述べる．本ツールでは，3.3 節で定義した仕様テンプレートに基づいて記述された検査項目をもとに CTL による検査式を生成することが可能である．仕様テンプレートは表 3-3-3 に示した通りである．仕様テンプレートから CTL 式を生成する際の規則を表 3-6-4 に示す．

表 3-6-4:仕様テンプレートに関する CTL 式を生成する際の規則

特性	規則
安全性	CTL 式の AG 演算子を用いて生成する
活性	CTL 式の AF 演算子を用いて生成する
到達可能性	CTL 式の EF 演算子を用いて生成する

規則に関しては，シーケンス図のメッセージに関する CTL 式の生成規則と同様である．この規則に従って得られる CTL 式の形式を，表 3-6-5 に示す．

表 3-6-5:CTL 式の形式

特性	形式
安全性	SPEC AG !(発生してはならない事象)
活性	SPEC AF (発生すべき事象)
到達可能性	SPEC EF (一度は到達すべき状態)

したがって，仕様テンプレートから得られる CTL 式は，表 3-6-6 に示す通りとなる．

表 3-6-6:仕様テンプレートから得られる CTL 式

NO	式表現	CTL 式
1	safe(変数 = 状態)	SPEC AG !(変数 = 状態)
2	safe(メッセージ)	SPEC AG !(メッセージ = TRUE)
3	safe(変数，値)	SPEC AG !(変数 = 値)
4	live(変数 = 状態)	SPEC AF(変数 = 状態)
5	live(メッセージ)	SPEC AF(メッセージ = TRUE)
6	live(変数，値)	SPEC AF(変数 = 値)
7	reachable(変数 = 状態)	SPEC EF(変数 = 状態)
8	reachable(メッセージ)	SPEC EF(メッセージ = TRUE)
9	reachable(変数，値)	SPEC EF(変数 = 値)

以上の手順に従って，状態マシン図およびシーケンス図，そして仕様テンプレートから SMV 言語によるモデルへの自動変換が可能となる．

３．自動変換モジュールの設計書の作成

これまでに述べた状態マシン図およびシーケンス図，そして仕様テンプレートから SMV 言語によるモデルへと変換を行うアルゴリズムを文書化し，自動変換モジュールの設計書を作成した．

3.6.3 課題とその対応

本研究では，状態マシン図およびシーケンス図，そして仕様テンプレートから SMV 言語によるモデルへと自動変換を行うためのアルゴリズムを開発した．状態マシン図は SMV プログラムにおける状態遷移系へと変換され，シーケンス図および仕様テンプレートは SMV プログラムにおける検査式へと変換される．ここで想定される課題は，状態マシン図およびシーケンス図，そして仕様テンプレートから SMV プログラムへの変換が現実的な時間で実施できるか否か，となる．

そこで，研究者らがこれまでに開発した変換アルゴリズムを応用することで，入力となる状態マシン図およびシーケンス図，仕様テンプレートのサイズに対して線形から多項式時間程度での変換処理を実現できた．

今後の課題としては，大規模な UML 図に対して，実際にどの程度の時間で SMV プログラムへ変換可能かに関する評価を行う必要がある．

将来の応用方法としては，本アルゴリズムは，SPIN の Promela 言語や並行システムのモデル化言語である CSP など，SMV 以外のモデル検査ツールの入力モデルへの変換にも応用

が可能であると期待される.

3.7 研究目標 7「検証支援ツールの入出力インタフェースの開発」

3.7.1 当初の想定

(1) 想定する仮説等

これまでに設計した機能モジュールを実装し，統合することにより，本研究の目的である UML で記述された設計ドキュメントを対象とした SMV による検証を支援するツールを実現することが可能となる.

ここで，どのような環境上でツールを実装するかについて検討する必要がある．ここでは，Java などの，UML を用いた開発に広く用いられている環境での実装を行う.

(2) 当初の到達目標と期待される効果

ここで設定する到達目標としては，各機能モジュールの実装ならびに機能モジュールの統合による検証支援ツールの開発とする．また，検証支援ツールの開発の一環として，UML モデリングツールの出力を直接扱うための入力インタフェースおよび，NuSMV の出力を整形するための出力インタフェースについても併せて開発する.

検証支援ツールの実装により，本研究の目的であるモデル検査を用いたソフトウェアの設計支援環境が実現される．そして，検証支援ツールの入出力インタフェースを開発することにより，ツールの利便性の向上が期待される.

3.7.2 研究プロセスと成果

(1) 研究プロセス

本研究目標を達成するため，以下のプロセスに従って研究開発を実施した.

1. 機能モジュールの設計書を元に，ツール作成を外注するための仕様書を作成する.
2. 上述の機能モジュールを統合した，検証支援ツールの仕様書を作成する.

(2) 具体的な研究成果の内容

１．機能モジュールの仕様書の作成

本研究で開発する検証支援ツールの概要を図 3-7-1 に示す．本ツールは 5 つの機能モジュールと，入出力インタフェースから構成される．まず，機能モジュールの 1 つである候補提示モジュールを実装するため，候補提示モジュールの設計書をもとに仕様書を作成した.その上で 8 月上旬に開発業者への発注を行い,9 月中旬に納品および検収を完了した.

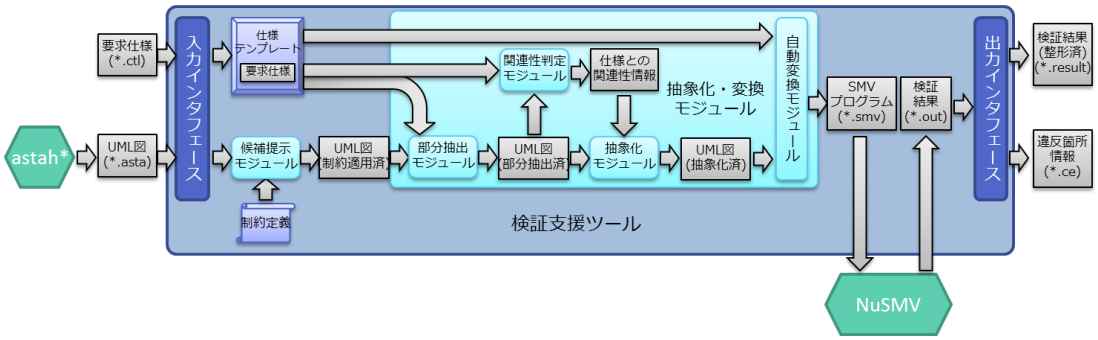


図 3-7-1:検証支援ツールの概要

次に，残りの 4 つの機能モジュールについては 1 つの機能モジュール（抽象化・変換モジュール）として仕様書を作成した．その上で，11 月上旬に開発業者への発注を行い，12 月上旬に納品および検収を完了した．

２．検証支援ツールの仕様書の作成

上述の機能モジュールを統合し，検証支援ツールを完成させるため，まず入出力インタフェースの設計を行った．

本ツールの入力インタフェースは，astah* community で作成された UML の状態マシン図とシーケンス図のモデリングデータ(*.asta)と，仕様テンプレートに基づいて記述された要求仕様(*.ctl)を読み込むことが可能である．*.ctl ファイルはテキストファイル形式で作成するものとする．

次に，本ツールは抽象化・変換モジュールが生成した SMV プログラム(*.smv)をモデル検査ツール NuSMV へと入力して検証を行う．NuSMV による検証結果は検証結果ファイル(*.out)として保存される．*.out ファイルはテキストファイル形式で保存されるものとする．

本ツールの出力インタフェースは，NuSMV が出力した検証結果ファイルを読み込み，整形済みの検証結果ファイル(*.result)を出力する．また，もし検査式が満たされなかった場合は，NuSMV が生成した反例を違反箇所情報(*.ce)として出力する．

また，本ツールはコマンドラインで起動する．起動方法を図 3-7-1 に示す．

```
java -jar umltool.jar プロジェクトファイル名.asta 検査式ファイル名.ctl
```

図 3-7-1:検証支援ツールの起動方法

検証支援ツールについては 12 月下旬に開発業者への発注を行い，1 月上旬に納品および検収を完了した．

3.7.3 課題とその対応

(1) 課題と問題点

本研究では,機能モジュールの実装のため,作成した設計書をもとに仕様書を作成した.また，検証支援ツールの実装のため，入出力インタフェースの設計を行った．ここで想定される課題は，入出力インタフェースにどこまでの機能を持たせるか，となる．

ここではツールの最低限の機能として，入力インタフェースには，モデリングデータおよび仕様テンプレートによる検査特性の読み込み機能を持たせることとする．また出力インタフェースは，NuSMV が出力した検証結果の整形機能を持たせることとする．

今後の課題として，開発したツールの適用実験を通して，ユーザから意見や要望をいただくことで，入出力インタフェースの評価を行う必要がある．

将来の応用方法としては，本研究で開発した入出力インタフェースは，同様の用途をもつ検証ツールを開発する場合，そのインタフェースへの応用が可能である．

4 考察

4.1 研究により判明した効果や問題点等

4.1.1 設定した到達目標の達成と今後の課題

モデル検査ではモデル化したソフトウェアの振る舞いを網羅的に探索することにより求める特性が満たされるか否かを自動的に証明することが可能であるため，設計検証に導入することで開発コストを大幅に削減することが可能である．また，証明されたアルゴリズムに基づいて検証が行われるため，モデル化した振る舞いにおいては求める特性を満たすか否かについて完全な保証を得ることができる．しかしながら，モデル検査技術を組込みソフトウェアの設計検証に導入するには，モデル作成の困難さと状態爆発の危険性という 2 つの問題点がある．そこで，本研究では以下に示す 3 つの機能をもつ設計支援ツールの実装を到達目標として設定している．

機能１．SMV への自動変換を目的とした UML 図の抽象化

機能２．モデルサイズ削減を目的とした UML 図の抽出・分割および抽象化

機能３．UML 図から SMV 言語への自動変換

これにより， 1.3 研究の意義にて挙げた以下の 2 つの目的を達成し，上記の 2 つの問題点を解決することが可能となる．

１．設計検証の半自動化による開発者への負荷の軽減

２．モデルサイズ削減による状態爆発の回避

まず，検証支援ツールの開発について，これまでに述べた通り，上記の 3 つの機能の実装が完了している．機能１については，研究目標１～２として達成した制約定義書および制約違反の検出・修正アルゴリズムの開発によって実現した．機能２については，研究目標３～５として達成した，仕様記述テンプレート，関連度計算・部分抽出アルゴリズム，そして抽象化アルゴリズムの開発によって実現した．そして機能３については，研究目標６～７として達成した，SMV 言語への自動変換アルゴリズムの開発および開発したアルゴリズムを統合することによる検証支援ツールの実装によって実現した．

次に，検証支援ツールの実装によって，上記の 2 つの目的がどこまで達成できたかについて述べる．

まず，目的１について，検証支援ツールの機能１および機能３によって，ユーザは astah* によって UML 図を作成すれば，検証支援ツールを用いて SMV 言語による検証のためのモデルを生成することができる．ここで，記述制約に違反する UML 図を作成した場合は人手での修正が必要となるが，制約を満たしている限りはほぼ完全に自動的に検証モデルを生成し，NuSMV による検証が可能となる．ここで「ほぼ」と述べた理由は，状態マシン図の変数の初期値や値の範囲など，UML の記法としてサポートされていない部分はユーザが直接入力する必要があるためであるが，ユーザによる記述量は入力する UML 図のサイズと比してごくわずかである．また，記述制約に違反した場合であっても，検証支援ツールは違反検出時に UML 図の違反箇所および理由をユーザへと提示するため，修正も比較的容易であ

ると考えられる．以上のように，検証支援ツールを用いることで，UML 図の NuSMV による検証はほぼ自動的に実現され，ユーザの負担も極めて小さい．したがって，目的１は十分に達成されていると考えられる．

次に，目的２について，検証支援ツールの機能２によって，ユーザが仕様記述テンプレートによって入力した仕様に基づいて，UML 図のそれぞれの要素について仕様との関連の度合いを評価したうえで，それに応じて UML 図の部分的に抽出し，さらに振る舞いを抽象化することが可能である．ここで，UML 図の部分抽出により UML 図の規模は大きく削減され，抽象化により振る舞いのパターンも削減することが可能となる．これにより，モデル検査で探索される状態空間のサイズ削減が実現され，状態爆発の回避が期待される．これにより，目的２はひとまずは達成されていると考えられる．しかしながら，本手法の定量的な評価についてはまだ十分になされておらず，開発現場で作成されたドキュメントへの適用など，実プロジェクト上での評価もまだ完了していない．これらは今後の課題として取り組むべきであると考えられる．

また，７つの研究目標への取組みを通して発見された課題について述べる．

まず，研究目標 1「自動変換を行う UML 図に対する制約定義書の作成」について，当初想定していた目標である,状態マシン図とシーケンス図の制約定義の作成は完了したため，研究目標自体は達成されたと考えられる．しかしながら，本ツールでは変換の効率化のために UML 図の記法の多くの部分に制約を掛けたため，トレードオフとして利便性はある程度低下してしまう．これについて，後述する本ツールの適用事例における事例提供元との意見交換においても，提供元からは状態マシン図の全ての記法に対応してほしいという要望を受けている．したがって，目標は達成できたが，取り組むべき課題はまだ残っているといえる．

研究目標２「UML 図の制約違反検出および抽象化手法の開発」について，当初想定していた目標である，制約定義書に基づく状態マシン図およびシーケンス図の制約違反の検出アルゴリズムの開発，ならびに修正候補の提示アルゴリズムの開発は完了しており，研究目標は達成されたと考えられる．ここで，制約違反の多くについては違反が検出された旨をユーザに伝えた上で修正はユーザ自身で行う必要があり，修正候補を提示することはできなかった．これは，制約違反を解消するための修正を自動で行った場合は，ユーザの望まない修正をせざるを得ないケースが多いと判断したためであったが，ユーザによっては人手による修正を煩わしく感じる可能性も存在する．この点については，ユーザの意見を吸い上げつつ，対応を進めていきたい．

研究目標３「仕様の入力テンプレートの開発」について，当初想定していた目標としては，仕様パターンで記述されるソフトウェアの代表的な特性を記述可能なテンプレートを開発することであった．本研究の成果として，安全性・活性・到達可能性という登場頻度の極めて高い検査特性をテンプレートによって記述することが可能である．また，状態マシン図およびシーケンス図における重要な要素である状態・メッセージ・変数について扱うことが可能であることから，本研究で開発した仕様テンプレートは，ソフトウェアの代表的な特性を記述するにあたって十分な表現能力があると考えられる．したがって，研究目標は達成されたと判断している．本研究では，モデル検査の専門知識を持たないユーザ

を想定していたため仕様パターンによる検査特性の記述への取組みは保留していたが，より幅広い層のユーザへの普及のためにも，仕様パターンへの対応は至急取り組むべき課題であると考えられる．

研究目標４「仕様に基づく UML 図の部分抽出手法の開発」について，当初想定していた目標は，テンプレートで記述された仕様に対して，UML 図の要素との関連度を計算し部分抽出を行うアルゴリズムを開発することであった．本研究の成果として，仕様との関連度に基づいて部分抽出を行うアルゴリズムは完成しており，研究目標はひとまず達成できている．しかしながら，検査特性への影響や状態数の削減効果といった部分抽出手法の評価は十分にされているとは言えない．検査特性に影響を与えないという理論的な証明に加えて，開発現場での適用実験などを通した実証的な測定を通して，提案する部分抽出の枠組みの評価を進めていく必要があると考えられる．

研究目標５「仕様に基づく UML 図の記述抽象化手法の開発」について，当初想定していた目標は，テンプレートで記述された仕様に対して，UML 図の要素との関連性を判定するアルゴリズムを開発し，さらに，関連性に基づいて UML 図の記述を抽象化するアルゴリズムを開発することであった．本研究の成果として、仕様との関連性を判定し，関連性のない要素を抽象化するアルゴリズムは完成しているため，研究目標は達成できている．しかし，研究目標４と同様に，抽象化手法の評価は十分ではない．こちらのアルゴリズムについても，適用実験など効果や精度など様々な観点から評価を行う必要がある．

研究目標６「UML 図および仕様から SMV 言語への自動変換手法の開発」について，当初想定していた目標は，UML 図およびテンプレートで記述された仕様から，SMV 言語によるモデルを自動的に作成するアルゴリズムを開発することである．本研究の成果として，状態マシン図およびシーケンス図，仕様テンプレートから SMV プログラムを生成するアルゴリズムは完成しており，研究目標は達成できている．本研究で定義した記述制約を想定するならば，本研究で開発した変換アルゴリズムは効率的であり，完成しているといえる．しかしながら，記述制約を変更し，合成状態など他の記法への対応を行う場合は，本研究で開発した変換アルゴリズムは必ずしも効率的とはいえない可能性もある．したがって，対応できる状態マシン図およびシーケンス図の記法の拡張を前提とした変換アルゴリズムについてさらなる検討を行っていく必要がある．

最後に，研究目標７「検証支援ツールの入出力インタフェースの開発」について，当初想定していた目標は，機能モジュールの統合による検証支援ツールの開発および検証支援ツール入出力インタフェースの開発である．本研究の成果として検証支援ツールおよびその入出力インタフェースの実装は完成している．したがって，この研究目標ならびに本受託研究の目標は達成されている．しかし，本ツールでは現在のところコマンドラインでの実行のみが可能であり，GUI はサポートされていない．利便性を向上し，普及を進める上では GUI の実装が必要不可欠である．

4.1.2 類似研究との比較

UML を対象とした設計検証に関する先行研究での成果と本研究で達成した研究成果の比較について簡単に述べる．

UML の状態マシン図およびシーケンス図による設計検証に関する研究開発は，これまでにも国内外で行われている．まず，[大西 2001]では，UML 図間の静的構造に関する整合性を検証するためのツールを開発している．また，[Egyed 2011]では，UML 図の描画時にモデルの不整合をリアルタイムに検出するツールを開発している．これらは静的な構造に関する設計間の整合性検証を目的としたものである．

一方で，[Bernardi 2002]では，ペトリネットを用いて状態マシン図やシーケンス図といった UML モデルの動作解析を行っている．これは状態マシン図およびシーケンス図で記述されたシステムの速度性能に着目した検証を行うものである．

このほかにも UML を対象とした設計検証手法は数多く報告されているが，その大部分は個別の例題への適用事例であり，開発現場で作成されている設計ドキュメントに対して汎用的に利用できるものではない．また，モデル化手法を一般化したものもあるが，ツールとしての実装はされておらず，現場で即座に利用できる形式で公開されているものは少ない．

これに対して，本研究の成果は制約定義を満たす UML 図であればどのようなものであっても適用することができ，かつモデル化および検証処理も自動化されているため，開発現場への導入は比較的容易であると考えらえる．

4.1.3 新たに発見された課題

4.1.1 節で述べたように，各研究目標への取組みの中で，新たな課題も数多く発見された．

まず，本研究では SMV 言語によるモデル生成を効率化するために，対象とする状態マシン図およびシーケンス図に記述制約を与えていたが，現場のユーザからの要望としては，より多くの記法への対応が求められている．ユーザの利便性向上のため，制約の見直しならびに変換アルゴリズムの修正が求められている．

次に，本研究では修正候補の提示モジュールの機能としては，大部分の制約違反については警告メッセージを出すにとどめ，修正候補の提示や自動修正は基本的には行っていない．しかしながら，制約を満たすよう修正を求めることがユーザの利便性を損ねる可能性もあるため，制約違反については可能な限り自動的に修正を行い，ユーザの作業量を増やさないようにする必要がある．

本研究の仕様テンプレートは，状態マシン図およびシーケンス図の状態・メッセージ・変数の 3 つの要素に関する，安全性・活性・到達可能性の 3 つの特性の， 9 通りの検査特性を記述することが可能である．専門知識のないユーザが利用しやすいよう，利用頻度の高い検査特性のみをテンプレートに含めていたが，より幅広い層のユーザへとアピールするため，仕様パターンの記述が可能となるよう，テンプレートを拡張する必要がある．

また，本研究で提案する部分抽出・抽象化手法については，適用実験がまだ十分ではない．部分抽出・抽象化が検査特性に与える影響や，検証コストに与える影響について定量的な評価をするとともに，ユーザからのフィードバックに基づいてより良い部分抽出・抽象化を実現する必要がある．

最後に，本研究で開発したツールはコマンドラインでのみ実行が可能であり，GUI はサポートされていない．操作性を向上させるとともに，検証結果をグラフィカルに表示することで不具合解析をスムーズに実施できるよう，GUI の実装を始めとするさらなるインタフェースの向上が求められる．

4.1.4 課題全体に対する達成状況と残っている課題について

本研究では，モデル検査を組込みソフトウェアの設計検証に適用する上で生じる，モデル作成の困難さと状態爆発の危険性という 2 つの問題の解決に寄与することを課題として研究開発を行った．まず，モデル作成の困難さという問題について，本研究で開発したツールを用いることで,記述した UML 図から,検証モデルを自動的に作成することができる．また，本ツールでは NuSMV から得られた検証結果を整形して出力している．これにより，モデル検査について専門的な知識をもたないソフトウェア設計技術者であっても，容易に NuSMV による検証を行い，結果をもとに設計の修正を行うことが可能である．一方で，ツールで扱う UML 図の記法や仕様テンプレートの表現能力等にはまだ検討の余地が残されており，インタフェースの向上も併せてモデル検査を用いた組込みソフトウェアの設計検証環境はさらなる改善が可能であると考えられる．

一方で，状態爆発の危険性という問題についても，本ツールでは UML 図に対して抽象化を施すことで検証モデルの状態数を削減している．抽象化に関する開発現場で作成されたドキュメントへの適用実験を通した比較・評価はまだ十分とはいえないが，効果は確認されている．残された課題としては，さらなる適用実験に基づく，部分抽出や抽象化が検証コストに与える影響の定量的な評価や，抽象化を効果的に行うための仕様記述や SMV 言語によるモデル化手法の検討が挙げられる．

4.2 今後の課題

4.2.1 研究成果の産業界への寄与

組込みソフトウェア開発において本ツールが果たす役割

自動かつ網羅的な検証を実現するモデル検査技術の導入は，ソフトウェア開発現場で強く求められている．しかしながら，現在の組込みソフトウェアの開発現場において，技術者がモデル検査技術を設計検証へと導入する上での課題は数多く残されている．中でも，モデル検査ツールに固有のモデル記述言語を用いて，組込みソフトウェアの設計記述をモデル化することや,検査したい性質を特有の論理体系による検査式として表現することは,モデル検査の専門的知識を持たない技術者にとっては非常に困難である．

本ツールは，対象となる UML 図の表現能力やインタフェース面に関しての制限はあるものの，与えられた UML による設計記述からモデル検査ツール NuSMV の入力モデルへの変換をほぼ完全に自動化し，さらにツールによって得られた結果を整形し，わかりやすく表示することが可能である．また，本研究では検査したい特性を記述するための仕様入力テンプレートを開発しており，テンプレートに従って UML 図内の要素を指定すれば，その要素の振る舞いを NuSMV で検査するための検査式を自動的に生成することが可能である．これ

らの機能により，モデル検査の専門的知識を全くもたない技術者であっても，NuSMV を用いた設計検証を実施することができる．

さらに，設計検証へとモデル検査を導入する上でのもう一つの大きな課題は，状態爆発問題への対処である．モデル検査では，検証の対象となるシステムの振る舞いについて，その組み合わせの全てを網羅的にチェックできるようにモデル化を行う必要がある．そのため，多くの場合で，状態爆発と呼ばれる，検証モデルの規模が指数的に増加してしまう現象が発生する．状態爆発が発生した場合，検証に要する時間や計算資源の規模が実用的な範囲を越えてしまうため，検査する性質に関連する部分のみの抽出や，不要な部分の抽象化などの対策により，検証モデルの規模を抑える必要が生じる．本ツールでは，関連部分の抽出や不要な部分の抽象化を自動的に行った上でモデル化を行うため，状態爆発を回避することができる．

このように，本ツールを利用することで，組込みソフトウェアの設計検証へとモデル検査を導入する上での課題への対応が可能となり，導入への障壁が大きく低減されると期待される．

組込みソフトウェア開発において本ツールを導入するメリット

本ツールを利用して，組込みソフトウェアの設計検証にモデル検査を導入することによる最も大きなメリットは，開発コストの削減である．モデル検査における検証の手続きはツールによって完全に自動化されているため，開発コストを増加させることなく設計の無矛盾性の検証や仕様との不整合の検出を行うことが可能である．こうした設計の不具合は後々の開発プロセスにおける手戻りの要因となるため,設計段階での検出が実現されれば,組込みソフトウェア全体の開発コストを大きく削減することができる．

また，モデル検査での検証アルゴリズムは既に証明済みのものである．したがって，モデル検査を用いて検証された性質の正しさについては完全に保証することが可能であり，開発された組込みソフトウェアの信頼性を担保できるということもメリットとして挙げられる．

組込みソフトウェア開発において本ツールを導入するために必要なこと

本ツールを組込みソフトウェア開発へと導入するための前提は，設計に UML を用いている，ということである．本ツールは，UML の状態マシン図およびシーケンス図を用いた組込みソフトウェアの設計プロセスを想定して開発されたものである．さらに，本ツールを利用するためには，UML モデリングツールとして astah* community を利用している必要があるが，astah* community は UML の多くの記法に対応しており，導入のためのコストも比較的小さい．

また，本ツールが扱うことのできる UML 図には制約が設けられており，制約を満たすように設計ドキュメントが記述されている必要がある．そのため，本ツールを利用するためには，制約を満たすように設計を修正しなければならない．本ツールには，制約を満たさない UML 図に対して，制約に違反している箇所や修正候補について提示する機能があるため，制約を満たすよう UML 図を修正することも比較的容易である．

最後に，本ツールを用いたモデル検査による設計検証の枠組みにおいては，検証する特

性はテンプレートによって記述されたものに限定される．このテンプレートでは安全性，活性，そして到達可能性といった組込みソフトウェアが満たすべき基本的な特性の記述がサポートされている．また，テンプレートを利用せず，検証したい特性を直接記述することも可能である．

本ツールと類似ツールとの比較

本ツールと同様に，UML を用いて記述された組込みソフトウェア設計の検証支援を目的としたものとして，例えばUMLモデリングツールのEnterprise Architectは，状態マシン図に対してシミュレーションや状態遷移パスの抽出によって設計検証を行う機能をもっている．また，astah* professional でも同様に状態マシンに対して状態遷移パスの抽出を行う機能がある．

しかしながら，シミュレーションでは得られた動作系列についての評価しかできず，網羅的な検証による保証を得ることはできない.また,状態遷移パスの抽出機能に関しては，与えられた状態マシン図の構造を静的に解析してパスを抽出するため，実際に状態マシン図を動作させたとき，どのパスがどのようなタイミングで実行されるかといった動的な特性についての検証は不可能である．

これに対して本ツールでは，モデル検査ツールとの連携を行うことで，設計に対して動的な特性を網羅的な検証することを可能としている．

4.2.2 研究成果の産業界への展開に向けて

ここまでで述べたように，本研究で得られた成果は組込みソフトウェアの開発現場でもある程度の効果を上げられると期待される．しかし，同時にさらなる改善のための課題も数多く発見されている．これらの考察を踏まえると，本研究の成果を産業界で役立たせるために必要なことは，以下の３つに集約される．

課題１．ツールの利便性の向上

課題２．ツールの機能拡張

課題３．実プロジェクト上での評価実験

まず課題１について，本研究で作成した検証支援ツールは，現在のところコマンドラインでの実行のみが可能であり，GUI はサポートされていない．企業の開発者へと普及するためにはGUIの追加やNuSMVによる検証結果の解析をサポートするための機能の実装など，ツールの利便性を向上させる必要がある．

次に課題２について，検証支援ツールは状態マシン図の記法の一部にしか対応していないため，より多くの開発で利用するためには，機能拡張を行い，より多くの記法へと対応する必要がある．

最後に課題３について，検証支援ツールのもつ，仕様に基づくUML図の部分抽出および抽象化に関する機能については，その効果に関する評価がまだ十分とはいえない．検証支援ツールの有効性を産業界へとアピールし，より多くの開発現場へとツールを普及するためには，実プロジェクト上での評価実験を通してツールの性能の定量的評価を行う必要がある．

これらの課題を解決することにより，本研究の成果を産業界へと普及・発展させること

が可能となる.

4.2.3 開発現場で作成されたドキュメントへの適用

4.1.1 節および 4.1.3 節で述べた新たなる課題の解決を目指して,「開発現場の実プロジェクトへの本ツールの適用」についても着手している．本節では，この取組みとその現時点での成果について簡単に述べる.

まず経緯についてであるが，某ソフトウェア開発企業に事例提供を打診したところ，既存の状態遷移表から UML の状態マシン図を作成して，提供していただけることとなった．ここで適用したシステムは店舗従業員向けの「商品供給指示システム（R-CDS)」である.

本事例で検査対象となる状態遷移表は，4 つの状態と 10 種類のイベントをもつ．そして 13 の遷移をもっている．ここでは従業員が操作する端末数は 2 台とし，2 台の端末とシステムのモニターの表示の 3 つを状態マシン図として記述していただいた.

その上で，仕様テンプレートを用いて仕様を記述し，検証を行った．仕様テンプレートによって記述した 19 種の検査特性について,いずれの特性も NuSMV による検証結果は TRUE となり，誤りがないことが確認できた.

さらに，事例提供元から要望があった特性についても検証を行った．ここで検証した特性は端末の状態と従業員間の通信モードの振る舞いの関係についての仕様である．この特性については，直接 CTL 式によって記述し，NuSMV による検証を行った．NuSMV による検証の結果，特性の一部について FALSE となり，特性が満たされないことが検出された．反例を解析したところ，状態遷移表から状態マシン図への転記ミスが原因であることがわかった.

適用実験の結果をうけて，事例提供元からは，今回は転記ミスではあったものの，このような漏れや間違いを発見できる技術であれば，モデル検査は非常に有用であるというコメントをいただいた．また，ツールとしては状態マシン図の全ての記法に対応して欲しいとの要望をいただくとともに，GUI の実装による利便性の向上についての要望もいただいた.

参考文献

[YOKOGAWA 2013] T.YOKOGAWA, S.AMASAKI, H.MIYAZAKI, K.OKAZAKI, Y.SATO, K.ARIMOTO, ″Consistency verification of UML diagrams based on process bisimulation,″ In The 19th IEEE Pacific Rim Int'l Symp. on Dependable Computing (PRDC 2013), December 2013.

[横川 2013] 横川 智教, 宮崎 仁, 佐藤 洋一郎, 有本 和民, “状態マシン図とシーケンス図の論理式表現に関する検討,” ソフトウェア・エンジニアリング・シンポジウム 2013 ワークショップ,2013 年 9 月.

[片山 2013] 片山 巧,横川 智教,宮崎 仁,佐藤 洋一郎,有本 和民,“CSP を用いた UML 図の整合性検証に関する検討,” ソフトウェア・エンジニアリング・シンポジウム 2013 ワークショップ,2013 年 9 月.

[落水 2013-1] 落水 恭介, 横川 智教,宮崎 仁,佐藤 洋一郎,有本 和民,”状態マシン図を用いたスマートフォンアプリのモデル化,” 電子情報通信学会技術報告, Vol.113, No.159(SS2013 13-35),pp.49-54, 2013 年 7 月.

[落水 2013-2] 落水 恭介, 横川 智教,宮崎 仁,佐藤 洋一郎,有本 和民,”状態マシン図を用いたスマートフォンアプリのモデル化手法に関する検討,” ソフトウェア・エンジニアリング・シンポジウム 2013 ワークショップ,2013 年 9 月.

[MIYAZAKI 2012] H. MIYAZAKI, T. YOKOGAWA, S. AMASAKI, K. ASADA, Y. SATO, ″Synthesis and refinement check of sequence diagrams,″ IEICE TRANS. on INF. & SYST., Vol.E95-D, No.9, pp.2193-2201, Sep. 2012.

[ASADA 2011] K. ASADA, T. YOKOGAWA, S. AMASAKI, H. MIYAZAKI, and Y. SATO, ″Refinement Check of Asynchronous Behaviours of Sequence Diagrams Using LTSA,″ In Proc. of Int'l Symp. on Software Reliability Engineering (ISSRE2011), November 2011.

[NIMIYA 2010] A. NIMIYA, T. YOKOGAWA, H. MIYAZAKI, S. AMASAKI, Y. SATO, and M. HAYASE, ″Model Checking Consistency of UML Diagrams Using Alloy,″ In Int'l Conf. on Computer Science and Software Engineering (ICCSSE2010), November 2010.

[KAWAKAMI 2010] Y. KAWAKAMI, T. YOKOGAWA, H. MIYAZAKI, S. AMASAKI, Y. SATO, and M. HAYASE, ″Symbolic Model Checking of Interactions in Sequence Diagrams with Combined Fragments by SMV,″ In Int'l Conf. on Computer Science and

Software Engineering (ICCSSE2010), November 2010.

[HARADA 2009] S. HARADA, T. YOKOGAWA, H. MIYAZAKI, Y. SATO, and M. HAYASE, ″A Tool Support for Verifying Consistency between UML Diagrams by SMV,″ In Proc. of Int'l Tech. Conf. on Circuits/Systems, Computers and Communications (ITC-CSCC2009), pages 897-900, July 2009.

[大西 2001] 大西淳, “UML におけるモデル整合性検証支援システム,” 電子情報通信学会論文誌, J84-D-1(6), 671-681, 2001 年 6 月.

[Egyed 2011] A. Egyed, “Automatically Detecting and Tracking Inconsistencies in Software Design Models,” IEEE Trans. on Software Engineering, Vol.37, No.2, March/April 2011.

[Bernardi 2002] S. Bernardi, et al., “From UML Sequence Diagrams and Statecharts to analyzable Petri Net models,”In Proc. of Int'l Workshop on Software and Performance (WOSP'02), July 2002.