ソフトウェア工学 103−17
（1995．3．10）

抽象解釈に基づく仕様の段階的具体化法

吉岡信和　鈴木正人　片山卓也

北陸先端科学技術大学院大学 情報科学研究科
〒923-12 石川県能美郡辰口町旭台15

複雑な仕様を満たすソフトウェアを構成するために段階的詳細化の手法が提案されている．しかし，これまでの手法の多くは詳細化の途中段階に作成したプログラムは実行できなかった．また，実行できてもデータが不完全なため，そのプログラムに有効な意味を与えることが困難であった．

本稿ではこの問題を解決するために，データの具象化に基づいてソフトウェアを段階的に構成するISDR法を提案する．ISDR法ではまずデータを抽象化して，その抽象値のみを入力とする最も単純なプログラム(原始ソフトウェア)を作成する．次に，原始ソフトウェアの入出力データを必要な段階まで具象化し，具象化された入出力データに関する仕様を満たすようにソフトウェアを詳細化する．この過程を入出力データが十分に詳細化され最終の仕様を満たすプログラムが得られるまで繰り返す．また，詳細化の中間段階に作成した抽象的なデータ上のプログラムに意味を与えるために抽象解釈の技法を利用した．

An Incremental Specification Methodology Using Abstract Interpretation

Nobukazu YOSHIOKA　Masato SUZUKI　Takuya KATAYAMA

School of Information Science
Japan Advanced Institute of Science and Technology
15 Asahidai, Tatsunokuchi, Ishikawa 923-12, Japan

Stepwise refinement methodology is one of a general technique for making large and complex software products. In most of these methodologies we can't execute programs created during its development until it complete. Further more it is difficult to give a sense to its execution, because of their incompleteness in data.

In this paper, we propose an incremental software creation methodology based on data reification , ISDR. ISDR consists of two parts. First, we create most simplified software (primitive software) in which both input and output set has only one abstract data, then define a program on those data. Secondly, we refine this program repeatedly by reification of those data until the program satisfies the original specification. We use abstract interpretation technique for giving some sense to programs created during its development.

1 はじめに

ソフトウェアの規模が大きくなると, それに対する要求が複雑になり, 実際にソフトウェアを動かしてみないと仕様が細部まで決定できないという問題が生じる. また仕様が完成してもプログラムを作成する段階で, バグが発生している箇所が特定できない, バグが多数発生するためプログラム全体を通して実行できない, などの問題がまとめて発生する. この原因の1つは, これまでのソフトウェア開発の手法では, まずその仕様を細部まで決定した後でないとプログラムを構築できないためである.

この問題を解決するために段階的詳細化の手法がいくつか提案されている. しかし, これまでの段階的詳細化法では, 詳細化はその設計を行うときに限られ, 詳細化の途中段階のプログラムを実行できないものがほとんどであった. これに対し, 途中段階のプログラムを実行できるならば, その段階での抽象的なレベルの誤りの検出に有効であると考えられる.

そのような方法の一つとして鶴巻らによってHAWAII法が提案されている [1]. この方法は, プログラムをその構成要素である各モジュールの機能や性能を段階的に追加, 向上させていくことによって詳細化する. しかしこの方法では, データをどのように具体化するかについては述べていない.

そこで本稿では, データを段階的に具体化し, その途中段階のプログラムを実行しながらソフトウェアを段階的に構成するISDR法[†1]を提案する. ISDR法では, データの集合を抽象的な値の集合と捉え, その抽象値を段階的に具体的な値に変化させ, その変化に従ってソフトウェアを詳細化する. また, 詳細化の途中段階のプログラムを実行するために抽象解釈の技法 [2] を利用している. 抽象解釈とは, プログラム中のデータを抽象化し, その値のもとでプログラムがどう振舞うか調べる事によってプログラムを解析するための技法である. 本稿では, これをプログラムを段階的に構成する方法として用いる.

2 ソフトウェアの段階的構成

これまでの段階的詳細化によるソフトウェア構成法は, 詳細化の途中段階ではプログラムが作成できなかったり, 作成できても実際に実行して動作を確認する事ができなかった. それは, 次の2つの理由からである.

1. プログラム自体が未完成であり, まだ記述していない処理など, 未定義の部分が存在する.
2. 詳細化の途中段階に現れるデータは未完成であり, そのデータに対するプログラムの挙動に十分な意味を与えられない.

しかし, 抽象解釈の技法を応用することにより, 抽象的なデータに対するプログラムに意味のある解釈を与えることが可能である. そこで, 本稿で提案するISDR法は次のことを実現する事を目的とする.

1. ソフトウェアを詳細化する途中の段階で作成したプログラムを実行可能にする.
2. プログラムの一部がまだ定義されていない未完成なプログラムでも実行可能にする.

3 データの具象化に基づくソフトウェアの詳細化

段階的詳細化によってソフトウェアを構成するためには, それが扱うデータとその処理を抽象化する必要がある. データの抽象化とは, 同じ特徴をもつデータの集合をまとめ, これを1つの抽象値として考えることである. また処理の抽象化とは, その抽象値に対してプログラムを定義する事である.

こうして抽象化したソフトウェアを詳細化するためには, 抽象化とは反対の操作を行う. すなわち, 抽象的な値を表す集合を再分割し, より具体的な集合を表す複数の値に具体化する. そして, それぞれの値についてプログラムを定義することによって, 処理の詳細化を行う.

抽象解釈を利用し, このプログラムに意味を与えることによって抽象値に対するソフトウェアを実行することができる.

抽象的なデータ上のソフトウェアに対して, この様に詳細化を繰り返すことにより目的とするソフトウェアを構成する方法が本論文で提案するISDR法である.

ここで, データの具象化の例を示す. 図1は正の整数集合を表す抽象値 *Posi*, 0, 負の整数集合を表す抽象値 *Nega* を持つ抽象的なデータ集合の中の抽象値 *Posi* を, 正の奇数の集合を表す抽象値 *PosiOdd* と正の偶数の集合を表す抽象値 *PosiEven* に再分割したことを表している. このとき, *Posi* は *PosiOdd* と *PosiEven* に具象化されたという. この具象化の様子を表現するために, 抽象値の集合に半順序関係を与えた半順序集合(ドメイン)を考える.

ISDR法によってソフトウェアを段階的に詳細化する手順をまとめると次のようになる.

1. データを抽象化する.
2. 抽象化したデータに基づいて仕様を作成し, それに対するプログラムを定義し, 実行する.
3. 以下を目的のソフトウェアが構成されるまで繰り

†1 Incremental Software creation methodology based on Data Reification

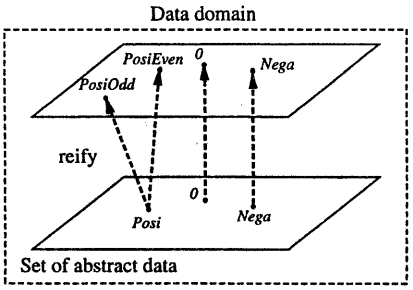

図1　データの具象化

返す.

3-1. 抽象的なデータを必要とする段階まで1段階具象化する.

3-2. データの具象化に基づき仕様とプログラムを詳細化し, それを実行する.

4 準備

この節では本稿で扱う用語の定義を行う.

定義1 ソフトウェア

ソフトウェア$\mathcal{S}$は次のように仕様とプログラムからなる.

$$\mathcal{S} \equiv \langle \mathrm{Spec}, \mathrm{Prog} \rangle$$

定義2 仕様

$\mathcal{S}$の仕様 Spec は次の3つの要素からなる.

$$\mathrm{Spec} \equiv (P^{\mathrm{I}}, P^{\mathrm{O}}, P^{\mathrm{IO}})$$

P^{I}は入力仕様, P^{O}は出力仕様, P^{IO}は入出力間仕様を表す述語である.

P^{I}は$\mathcal{S}$の入力データ集合$\boldsymbol{R}^{\mathrm{I}}$を決定し, P^{O}は$\mathcal{S}$の出力データ集合$\boldsymbol{R}^{\mathrm{O}}$を決定する.

$$\boldsymbol{R}^{\mathrm{I}} \equiv \{d^{\mathrm{I}} \mid P^{\mathrm{I}}(d^{\mathrm{I}})\}$$
$$\boldsymbol{R}^{\mathrm{O}} \equiv \{d^{\mathrm{O}} \mid P^{\mathrm{I}}(d^{\mathrm{O}})\}$$

P^{IO}は入力データと出力データの関係を表し, 次の条件を満たしている.

$$\forall d^{\mathrm{I}}, d^{\mathrm{O}} \quad [P^{\mathrm{IO}}(d^{\mathrm{I}}, d^{\mathrm{O}}) \Rightarrow P^{\mathrm{I}}(d^{\mathrm{I}}) \wedge P^{\mathrm{O}}(d^{\mathrm{O}})]$$
$$\forall d^{\mathrm{I}}, d^{\mathrm{O}}, d'^{\mathrm{O}} \quad [P^{\mathrm{IO}}(d^{\mathrm{I}}, d^{\mathrm{O}}) \wedge P^{\mathrm{IO}}(d^{\mathrm{I}}, d'^{\mathrm{O}}) \Rightarrow d^{\mathrm{O}} = d'^{\mathrm{O}}]$$

定義3 プログラム

$\mathcal{S}$のプログラム Prog は次の4つの要素からなる.

$$\mathrm{Prog} \equiv (F, \mathbf{AFS}, \mathbf{RS}, \mathbf{DS})$$

Fをプログラム関数, **AFS** は補助関数群, **RS** はデータ集合群, **DS** はドメイン群と呼ぶ.

FはP^{IO}を実現するための関数であり,**AFS** はFの中で呼び出される補助関数の集まりである. また **RS** や **DS** は, それらの関数が扱うデータ集合やデータドメインの集合である.

プログラム関数 Fは, 仕様 Spec と次のような関係がある部分関数である.

$$\forall d^{\mathrm{I}}, d^{\mathrm{O}} \quad [P^{\mathrm{IO}}(d^{\mathrm{I}}, d^{\mathrm{O}}) \Leftrightarrow F(d^{\mathrm{I}}) = d^{\mathrm{O}}]$$

本稿で扱う関数は全てカリー化してあり次のような型を持つとする.

定義4 関数の型

1. 任意の$\boldsymbol{R}(\in \mathbf{RS})$ は型である.
2. τ, σが型ならば, $\tau \to \sigma$は型である.

プログラム関数 Fは Spec を P^{IO}を直接実現するための関数であり, その型は$\boldsymbol{R}^{\mathrm{I}} \to \boldsymbol{R}^{\mathrm{O}}$であある. また Fは,P^{IO}が成り立っている入力データと出力データに対する定義である.

ある抽象値 d を d'に具象化した時 $d \preceq d'$と書き, この関係を**近似関係**と呼ぶ. また,d を d'の近似値と呼び,d'を d の具象値と呼ぶ.

この関係を用いて, データ集合の具象化の様子を表現するためにデータドメインを定義する.

定義5 データドメイン

データドメイン$\boldsymbol{D}$とは, データ集合$\boldsymbol{S}$とその要素に成り立つ近似関係$\preceq_S$で構成される.

$$\boldsymbol{D} \equiv \langle \boldsymbol{S}, \preceq_S \rangle$$

データドメインを単にドメインと呼ぶこともある.

データドメインは, 近似関係によって半順序集合になる.

$\boldsymbol{D} = \langle \boldsymbol{S}, \preceq_S \rangle$ の時, $\mathrm{dat}(\boldsymbol{D}) = \boldsymbol{S}$, $\mathrm{rel}(\boldsymbol{D}) = \preceq_S$と定義する. また, データドメインの和を次のように定義する.

$$\boldsymbol{D} \cup \boldsymbol{D}' \equiv \langle \mathrm{dat}(\boldsymbol{D}) \cup \mathrm{dat}(\boldsymbol{D}'), \mathrm{rel}(\boldsymbol{D}) \cup \mathrm{rel}(\boldsymbol{D}') \rangle$$

Prog 中のデータ集合群 **RS** とメイン群 **DS** の間には次の関係がある.

$$\mathbf{RS} = \{\sqcup \boldsymbol{D} | \boldsymbol{D} \in \mathbf{DS}\}$$

ここで$\sqcup$は, ドメインの集合の中で最も具体的な値の集合をとる演算であり, 次のように定義する.

$$\sqcup \boldsymbol{D} = \{d \mid d \in \mathrm{dat}(\boldsymbol{D}), d' \in \mathrm{dat}(\boldsymbol{D}), (d' \preceq d) \in \mathrm{rel}(\boldsymbol{D})\}$$

5 ソフトウエアの段階的構成法

ISDR 法によるソフトウェアの構成手順は次のようになる.

1. 入出力データ集合を抽象化し, 原始ソフトウェア$\mathcal{S}_0$を構成する.
2. 抽象化された入出力データ集合の中の値がこれ以上具象化できなくなるまで,$\mathcal{S}_0$の詳細化を繰り返す.

$\mathcal{S}_0$を n 回詳細化したソフトウェアをバージョン n のソフトウェアと呼び, $\mathcal{S}_n$と書く.

入出力データ集合がこれ以上具象化できない最終目標のソフトウェア$\mathcal{S}_\omega$を構成するまでのソフトウェアの詳細化列は次のようになる.

$$\mathcal{S}_0 \Rightarrow \mathcal{S}_1 \Rightarrow \ldots \Rightarrow \mathcal{S}_\omega$$

手順の詳細を以下に述べる.

5.1 原始ソフトウェア

ソフトウェアの抽象化をどのレベルまで行うかは,一意に決定できない. しかし,原始ソフトウェア$\mathcal{S}_0$を最も抽象的なソフトウェアとして捉えると次のようになる.

つまり,ソフトウェアの入出力データの集合をそれぞれ1つの抽象値$\perp_{in}$, $\perp_{out}$とみなし,プログラムを$\perp_{in}$から$\perp_{out}$への写像と抽象化したものが$\mathcal{S}_0$である. $\mathcal{S}_0$を形式的に書くと次のようになる.

$$\begin{aligned}
\mathcal{S}_0 &\equiv \langle \mathrm{Spec}_0, \mathrm{Prog}_0 \rangle \\
\mathrm{Spec}_0 &\equiv (P_0^{\mathrm{I}}, P_0^{\mathrm{O}}, P_0^{\mathrm{IO}}) \\
P_0^{\mathrm{I}}(x) &\equiv (x = \perp_{in}) \\
P_0^{\mathrm{O}}(x) &\equiv (x = \perp_{out}) \\
P_0^{\mathrm{IO}}(x,y) &\equiv (x = \perp_{in}) \wedge (y = \perp_{out}) \\
\mathrm{Prog}_0 &\equiv (F_0, \emptyset, \{R_0^{\mathrm{I}}, R_0^{\mathrm{O}}\}, \{D_0^{\mathrm{I}}, D_0^{\mathrm{O}}\}) \\
D_0^{\mathrm{I}} &\equiv \langle \{\perp_{in}\}, \emptyset \rangle \\
D_0^{\mathrm{O}} &\equiv \langle \{\perp_{out}\}, \emptyset \rangle \\
R_0^{\mathrm{I}} &\equiv \{\perp_{in}\} \\
R_0^{\mathrm{O}} &\equiv \{\perp_{out}\} \\
F_0 &:: R_0^{\mathrm{I}} \rightarrow R_0^{\mathrm{O}} \\
F_0(x) &= \perp_{out}, \ \textbf{if}\ x = \perp_{in}
\end{aligned}$$

5.2 ソフトウェアの詳細化

5.2.1 データの具象化

データの具象化は,抽象値の上のソフトウェアを1段階$\mathcal{S}_\omega$に近づけるために行う.

例えば,原始ソフトウェアの入力ドメインが,次のように整数集合を表す抽象値*Num*のみから構成されているとする.

$$D_0^{\mathrm{I}} \equiv \langle \{Num\}, \emptyset \rangle$$

$\mathcal{S}_\omega$がどんな処理を行うかで,このドメインをどう具象化するか変わってくる. ここでは,抽象値*Num*を正の整数を表す抽象値*Posi*, 0, 負の整数を表す抽象値*Nega*に具象化する. つまり,ドメインD_0^{I}を次のようなD_1^{I}に具象化する.

$$\begin{aligned}
D_1^{\mathrm{I}} &\equiv D_0^{\mathrm{I}} \cup \langle \{Posi, 0, Nega\}, \\
&\qquad \{Num \preceq Posi, Num \preceq 0, Num \preceq Nega\} \rangle \\
&= \langle \{Num, Posi, 0, Nega\}, \\
&\qquad \{Num \preceq Posi, Num \preceq 0, Num \preceq Nega\} \rangle
\end{aligned}$$

入出力ドメインの具象化は抽象値$\perp_{in}$や$\perp_{out}$を具象化する事で進むため,これらのドメインは$\perp_{in}$や$\perp_{out}$をボトムとする半順序集合になる.

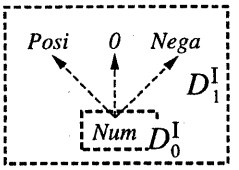

図2 データの具象化

定義6 データドメインの近似関係

データドメイン$\boldsymbol{D}$と$\boldsymbol{D}'$との間に次の様な関係が成り立つ時,$\boldsymbol{D}'$は$\boldsymbol{D}$を**具象化**したドメインと呼び, $\boldsymbol{D} \sqsubseteq \boldsymbol{D}'$と書く.

$$\begin{aligned}
\mathrm{dat}(\boldsymbol{D}) &\subseteq \mathrm{dat}(\boldsymbol{D}') \\
\mathrm{rel}(\boldsymbol{D}) &\subseteq \mathrm{rel}(\boldsymbol{D}') \\
\forall d' \in \mathrm{dat}(\boldsymbol{D}'), \exists d \in \mathrm{dat}(\boldsymbol{D}) \quad & [d \preceq d']
\end{aligned}$$

また, とよぶ.

定義7 データ集合の近似関係

データ集合$\boldsymbol{R}$と$\boldsymbol{R}'$の間に次の関係が成り立つ時, $\boldsymbol{R}'$は$\boldsymbol{R}$を**具象化**したデータ集合と呼び, $\boldsymbol{R} \sqsubseteq \boldsymbol{R}'$と書く.

$$\begin{aligned}
&\forall d \in \boldsymbol{R}, \exists d' \in \boldsymbol{R}' \quad [d \preceq d'] \\
&\forall d' \in \boldsymbol{R}', \exists d \in \boldsymbol{R} \quad [d \preceq d']
\end{aligned}$$

$\mathcal{S}_n$を構成するまで,入出力ドメインや入出力データ集合は次のように具象化される.

$$\begin{aligned}
D_0^{\mathrm{I}} &\sqsubseteq D_1^{\mathrm{I}} \sqsubseteq \ldots \sqsubseteq D_n^{\mathrm{I}} \\
D_0^{\mathrm{O}} &\sqsubseteq D_1^{\mathrm{O}} \sqsubseteq \ldots \sqsubseteq D_n^{\mathrm{O}} \\
R_0^{\mathrm{I}} &\sqsubseteq R_1^{\mathrm{I}} \sqsubseteq \ldots \sqsubseteq R_n^{\mathrm{I}} \\
R_0^{\mathrm{O}} &\sqsubseteq R_1^{\mathrm{O}} \sqsubseteq \ldots \sqsubseteq R_n^{\mathrm{O}}
\end{aligned}$$

5.3 仕様の詳細化

$\mathcal{S}_n$では, 具象化した入出力ドメインD_n^{I}, D_n^{O}のうち,新しく具象化された値を含む最も具体的な値に関してのみ仕様Spec_nを定義すればよい. なぜなら,それ以外の抽象的なデータに関する仕様は前のバージョンで既に定義されているからである.

定義すべき入力データ集合R_n^{I}と出力データ集合R_n^{O}は,入力ドメインD_n^{I}と出力ドメインD_n^{O}の最も具体的な値の集合である.

$$\begin{aligned}
R_n^{\mathrm{I}} &\equiv \sqcup D_n^{\mathrm{I}} \\
R_n^{\mathrm{O}} &\equiv \sqcup D_n^{\mathrm{O}}
\end{aligned}$$

このR_n^{I}とR_n^{O}に対する入力仕様P_n^{I}と出力仕様P_n^{IO}を定義すればよい.

入出力間仕様P_n^{IO}は,入力データ集合R_n^{I}の中の全ての値について定義する必要は無く必要な範囲で定義すればよい.

P_n^{IO}を過去のバージョンの入出力間仕様よりも詳細な定義とするためには，次の関係式が成り立っていなければならない．

$$\forall i < n, \forall d_n^I \in \boldsymbol{R}_n^I, \forall d_i^I \in \boldsymbol{R}_i^I, \exists d_n^O \in \boldsymbol{R}_n^O, \exists d_i^O \in \boldsymbol{R}_i^O$$
$$[P_n^{IO}(d_n^I, d_n^O) \wedge d_i^I \preceq d_n^I \wedge P_i^{IO}(d_i^I, d_i^O) \Rightarrow d_i^O \preceq d_n^O] \quad (*)$$

5.4 プログラムの詳細化

プログラム Prog_n 中で呼び出す関数は，過去のバージョンのプログラムで定義した関数でもよい．つまり，Prog_n の中の任意の関数 f_n は，次のように形式的に表現できる．

$$f_n = H_n(F_0, F_1, \ldots, F_n, g_i, g_j, \ldots)$$

ここで，$F_0, F_1, \ldots, F_{n-1}$ は過去のバージョンで定義したプログラム関数で，$g_i, g_j \ldots$ は n バージョンまでに定義した補助関数である．また，H_n は関数を合成する演算である．

プログラム関数 F_n は入出力間仕様 P_n^{IO} を満足するように定義する．よって F_n には，P_n^{IO} に成り立つ関係式 (*) より次の関係式が成り立つ．

$$\forall i < n, \forall d_n^I \in \boldsymbol{R}_n^I, \forall d_i^I \in \boldsymbol{R}_i^I, \exists d_n^O \in \boldsymbol{R}_n^O, \exists d_i^O \in \boldsymbol{R}_i^O$$
$$[(F_n(d_n^I) = d_n^O) \wedge (d_i^I \preceq d_n^I) \wedge (F_i(d_i^I) = d_i^O) \Rightarrow d_i^O \preceq d_n^O]$$

この時，$F_i \sqsubseteq F_n$ と書く．

プログラム関数の中で呼び出される補助関数は，(1) 過去のバージョンの関数，(2) 過去のバージョンの補助関数を詳細化した関数，(3) 新たに定義する関数のいずれかである．このうち後の 2 つの関数の集合が $\mathbf{AFS}_n$ である．

$\mathrm{Prog}_i (i < n)$ の中の補助関数 g_i を詳細化する手順は，プログラム関数の場合と同様 g_i のドメインを具象化し，そのドメインの中で最も具体的な値の集合 $\boldsymbol{R}_n$ に対する関数 g_n を定義する．

型 σ と τ につぎの関係が再帰的に成り立つ時，$\sigma \sqsubseteq \tau$ と書く．

1. $\sigma = \boldsymbol{R}$ の時，
   $\tau = \boldsymbol{R}'$ でかつ $\boldsymbol{R} \sqsubseteq \boldsymbol{R}'$ である．
2. $\sigma = \sigma' \to \sigma''$ の時，
   $\tau = \tau' \to \tau''$ でかつ $\sigma' \sqsubseteq \tau' \wedge \sigma'' \sqsubseteq \tau''$ である．

過去の補助関数 $g_i :: \sigma_i$ とそれを詳細化した関数 $g_n :: \sigma_n$ の間には必ず $\sigma_i \sqsubseteq \sigma_n$ が成り立っている．

新しく定義した f_n に関するドメインとデータ集合は，それぞれドメイン群 $\mathbf{DS}_n$ とデータ集合群 $\mathbf{RS}_n$ に追加しておく必要がある．

6 プログラムの解釈

詳細化したソフトウェア $\mathcal{S}_n$ の中のプログラム Prog_n を実行するためには，過去のプログラム $\mathrm{Prog}_0, \mathrm{Prog}_1, \ldots, \mathrm{Prog}_{n-1}$ が全て必要になる場合がある．なぜなら Prog_n の中の関数は，以前のバージョンのプログラムの中の補助関数を呼び出しているかもしれないからである．

Prog_n の中のプログラム関数 F_n を，定義されていない入力データに対しても何らかの近似データを返すような全関数に変換する事で，まだ具象化が進んでいない抽象値の入力に対しても値を決める事が可能になる．

プログラム関数 F_n を全関数に変換した関数を $\tilde{F}_n$ とすると，その関数は次の 3 つの条件を満たしていなければならない．

条件 1 出力データとして $\boldsymbol{R}_n^O$ 中の値の近似値を許すので，値域は $\mathrm{dat}(\boldsymbol{D}_n^O)$ になる．

$$\tilde{F}_n :: \boldsymbol{R}_n^I \to \mathrm{dat}(\boldsymbol{D}_n^O)$$

条件 2 バージョン n の入出力間仕様 P_n^{IO} を満たしている．

$$\forall d^I \in \boldsymbol{R}_n^I, \exists d^O \in \boldsymbol{R}_n^O \quad [P_n^{IO}(d^I, d^O) \wedge (\tilde{F}_n(d^I) = d^O)]$$

条件 3 F_n で定義されていない入力データに対しては，過去の仕様を満たしている．

$$\forall d^I \in \boldsymbol{R}_n^I, \exists i \leq n, \exists d \in \mathrm{dat}(\boldsymbol{D}_n^I) \quad [P_i^{IO}(d, \tilde{F}_n(d^I)) \wedge d \preceq d^I]$$

$\tilde{F}_n$ は F_n で定義されていない入力に対しては，条件 3 が成り立つように，過去のプログラム関数を呼び出せばよい．また，その時の $\tilde{F}_n$ の値を過去のプログラム関数が返す値の中で一番具体的な値とすれば，入力データ集合の中の値のうち，どの値に対する出力がまだ十分に詳細化されていないかを知る事ができる．

よって，$\tilde{F}_n$ の値を次のように定義する．

定義 8 $\tilde{F}_n$ の値

$$\tilde{F}_n(d_n^I) = \max_{\mathrm{rel}(\boldsymbol{D}_n^O)} \bigcup_{i=0\ldots n} \{F_i(d_i^I) | d_i^I \preceq d_n^I \wedge d_i^I \in \boldsymbol{R}_i^I\}$$

ここで max は，集合の中の一番詳細な値を返す次のような演算である．

$$d^\top = \max_{\preceq_D} \boldsymbol{S} \equiv \forall d \in \boldsymbol{S} \ [d \preceq_D d^\top]$$

ただし，$\boldsymbol{S} \subseteq \mathrm{dat}(\boldsymbol{D})$ でなければならない．

過去の任意のプログラム関数の間に $F_i \sqsubseteq F_j$ が成り立つので，集合 $\bigcup_{i=0\ldots n} F_i(d_i^I)$ の任意の要素に対して近似関係が成り立つ．よって $\max \bigcup_{i=0\ldots n} F_i(d_i^I)$ の値は一意に決定できる．

7 具体例

この節では,ISDR 法を用いて実際にソフトウェアを構築した例を示す.

例として, 金属元素と非金属元素のイオン結合の度合を調べるソフトウェアを段階的に構成する. ここで, 金属元素とは Li, Be, Na, Mg,... のような元素であり, 非金属元素とは He, B, C, N, O, ... の様な元素である.

まず, 金属元素, 非金属元素の集合を抽象化することで原始ソフトウェアを作成する. ここでは, 金属元素の集合を表す抽象値として *Metal*, 非金属元素の集合を表す抽象値として *NonMetal*を定義し, イオン結合度はどう分類できるか分からないので, とりあえず抽象値を *Unknown*とする.

そうすると, 原始ソフトウェアの仕様は次のようになる.

$$\begin{aligned}
\mathrm{Spec}_0 &\equiv (P_0^{\mathrm{I1}}, P_0^{\mathrm{I2}}, P_0^{\mathrm{O}}, P_0^{\mathrm{IO}}) \\
P_0^{\mathrm{I1}}(x1) &\equiv (x1 = \mathit{Metal}) \\
P_0^{\mathrm{I2}}(x2) &\equiv (x2 = \mathit{NonMetal}) \\
P_0^{\mathrm{O}}(y) &\equiv (y = \mathit{Unknown}) \\
P_0^{\mathrm{IO}}(x1, x2, y) &\equiv (x1 = \mathit{Metal}) \\
&\quad \wedge (x2 = \mathit{NonMetal}) \wedge (y = \mathit{Unknown})
\end{aligned}$$

この仕様に対するプログラムを作成すると次のようになる.

$$\begin{aligned}
\mathrm{Prog}_0 &\equiv (F_0, \emptyset, \{R_0^{\mathrm{I1}}, R_0^{\mathrm{I2}}, R_0^{\mathrm{O}}\}, \{D_0^{\mathrm{I1}}, D_0^{\mathrm{I2}}, D_0^{\mathrm{O}}\}) \\
D_0^{\mathrm{I1}} &\equiv \langle \{\mathit{Metal}\}, \emptyset \rangle \\
D_0^{\mathrm{I2}} &\equiv \langle \{\mathit{NonMetal}\}, \emptyset \rangle \\
D_0^{\mathrm{O}} &\equiv \langle \{\mathit{Unknown}\}, \emptyset \rangle \\
R_0^{\mathrm{I1}} &\equiv \{\mathit{Metal}\} \\
R_0^{\mathrm{I2}} &\equiv \{\mathit{NonMetal}\} \\
R_0^{\mathrm{O}} &\equiv \{\mathit{Unknown}\}
\end{aligned}$$

また, プログラム関数 F_0 を仕様を満たすように定義すると次のようになる.

$$\begin{aligned}
F_0 &:: R_0^{\mathrm{I1}} \to R_0^{\mathrm{I2}} \to R_0^{\mathrm{O}} \\
F_0(x, y) &= \mathit{Unknown}, \quad \text{if } x = \mathit{Metal} \\
&\quad \wedge\, y = \mathit{NonMetal}
\end{aligned}$$

原始ソフトウェアでは $\tilde{F}_0$ は F_0 と一致する. 表 1に $\tilde{F}_0$ の値を示す. この表は行に $\tilde{F}_0$ の第一引数の値を並べ, 列に第二引数の値を並べて, それらが交差する点に $\tilde{F}_0$ の値が入っている.

	NonMetal
Metal	*Unknown*

表 1　$\tilde{F}_0$ の値

このソフトウェアを 2 段階詳細化した時, 金属元素のデータ集合は図 3の様に具象化され, 非金属元素のデータ集合は図 4の様に具象化された. またその結果, イオン結合度は図 5の様に具象化できた. 以下にその各段階の詳細化について詳しく述べる.

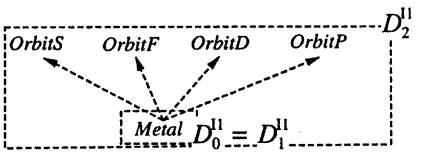

図 3　金属元素データ集合の具象化

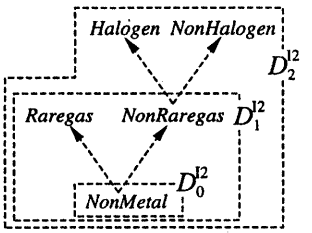

図 4　非金属元素データ集合の具象化

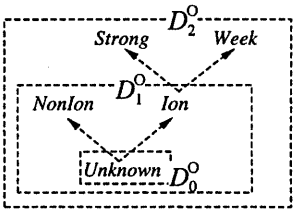

図 5　結合度の具象化

$\mathcal{S}_0$ を詳細化するためにデータ集合を次のように具象化する.

非金属元素は希ガスとそれ以外に分けて考える. なぜなら, 希ガスは他のどの元素とも反応しないからである. それに合わせ, イオンの結合度を結合ありとなしに分けることができる. つまり, 非金属元素の *NonMetal*を希ガスを表す抽象値 *Raregas*と希ガス以外を表す抽象値 *NonRaregas*に具象化する. そして, 結合度の *Unknown*をイオン結合度があることを表す値 *Ion* とイオン結合度が

ないことを表す値 *NonIon* に具象化する．金属元素は以前の値を用いる．

よって，具象化した値に関する仕様を作ると次のようになる．

$$\begin{aligned}
\mathrm{Spec}_1 &\equiv (P_1^{\mathrm{I1}}, P_1^{\mathrm{I2}}, P_1^{\mathrm{O}}, P_1^{\mathrm{IO}}) \\
P_1^{\mathrm{I1}} &= P_0^{\mathrm{I1}} \\
P_1^{\mathrm{I2}}(x2) &\equiv (x2 = \mathit{Raregas}) \\
&\quad \vee (x2 = \mathit{NonRaregas}) \\
P_1^{\mathrm{O}}(y) &\equiv (y = \mathit{Ion}) \vee (y = \mathit{NonIon}) \\
P_1^{\mathrm{IO}}(x1, x2, y) &\equiv (x1 = \mathit{Metal}) \wedge (x2 = \mathit{Raregas}) \\
&\quad \wedge (y = \mathit{NonIon})
\end{aligned}$$

この仕様に対するプログラムを作成すると次のようになる．

$$\begin{aligned}
\mathrm{Prog}_1 &\equiv (F_1, \emptyset, \{R_1^{\mathrm{I1}}, R_1^{\mathrm{I2}}, R_1^{\mathrm{O}}\}, \{D_1^{\mathrm{I1}}, D_1^{\mathrm{I2}}, D_1^{\mathrm{O}}\}) \\
D_1^{\mathrm{I1}} &= D_0^{\mathrm{I1}} \\
D_1^{\mathrm{I2}} &\equiv D_0^{\mathrm{I2}} \cup \langle \{\mathit{Raregas}, \mathit{NonRaregas}\}, \\
&\quad \{\mathit{NonMetal} \preceq \mathit{Raregas}, \\
&\quad \mathit{NonMetal} \preceq \mathit{NonRaregas}\}\rangle \\
D_1^{\mathrm{O}} &\equiv D_0^{\mathrm{O}} \cup \langle \{\mathit{Ion}, \mathit{NonIon}\}, \\
&\quad \{\mathit{Unknown} \preceq \mathit{Ion}, \mathit{Unknown} \preceq \mathit{NonIon}\}\rangle \\
R_1^{\mathrm{I1}} &\equiv \{\mathit{Metal}\} \\
R_1^{\mathrm{I2}} &\equiv \{\mathit{Raregas}, \mathit{NonRaregas}\} \\
R_1^{\mathrm{O}} &\equiv \{\mathit{Ion}, \mathit{NonIon}\}
\end{aligned}$$

また，プログラム関数 F_1 を仕様を満たすように定義すると次のようになる．

$$F_1 :: R_1^{\mathrm{I1}} \rightarrow R_1^{\mathrm{I2}} \rightarrow R_1^{\mathrm{O}}$$

$$F_1(x, y) = \mathit{NonIon}, \ \mathbf{if}\ x = \mathit{Metal} \wedge y = \mathit{Raregas}$$

$\tilde{F}_1$の値を表2に示す．ここで，*Metal* と *Raregas* に対応する値は F_1 で定義されていないので，前のバージョンの F_0 が呼び出され結果は *Unknown* となる．

	Raregas	*NonRaregas*
Metal	*NonIon*	*Unknown*

表2　$\tilde{F}_1$の値

次に$\mathcal{S}_1$を詳細化するために以下のようにデータ集合を詳細化する．

非金属元素の希ガス以外の元素をハロゲンとそれ以外に分ける．そして，金属を軌道で分ける．これはハロゲン元素は，相手の金属元素の最外殻の電子の数によってイオン結合度が変化するからである．それに合わせ，イオン結合が強い場合と弱い場合に分ける．つまり，非金属の *NonRaregas* をハロゲンを表す抽象値 *Halogen* とハロゲン以外を表す抽象値 *NonHalogen* に具象化する．金属の *Metal* はその最外殻の軌道がS軌道の金属を表す抽象値 *OrbitS*，P軌道の金属を表す抽象値 *OrbitP*，D軌道の金属を表す抽象値 *OrbitD*，F軌道の金属を表す抽象値 *OrbitF* に具象化する．そして，結合度の *Ion* をイオン結合度が強いことを表す値 *Strong* とイオン結合度が弱いことを表す値 *Week* に具象化する．

よって，具象化した値に関する仕様を作ると次のようになる．

$$\begin{aligned}
\mathrm{Spec}_2 &\equiv (P_2^{\mathrm{I1}}, P_2^{\mathrm{I2}}, P_2^{\mathrm{IO}}) \\
P_2^{\mathrm{I1}}(x1) &\equiv (x1 = \mathit{OrbitS}) \vee (x1 = \mathit{OrbitP}) \\
&\quad \vee (x1 = \mathit{OrbitD}) \vee (x1 = \mathit{OrbitF}) \\
P_2^{\mathrm{I2}}(x2) &\equiv (x2 = \mathit{Raregas}) \vee (x2 = \mathit{Halogen}) \\
&\quad \vee (x2 = \mathit{NonHalogen}) \\
P_2^{\mathrm{O}}(y) &\equiv (y = \mathit{NonIon}) \vee (y = \mathit{Strong}) \\
&\quad \vee (y = \mathit{Week}) \\
P_2^{\mathrm{IO}}(x1, x2, y) &\equiv \\
&\quad (x1 = \mathit{OrbitS} \wedge x2 = \mathit{Halogen} \wedge y = \mathit{Strong}) \\
&\vee (x1 = \mathit{OrbitP} \wedge x2 = \mathit{Halogen} \wedge y = \mathit{NonIon}) \\
&\quad \vee (x1 = \mathit{OrbitD} \wedge x2 = \mathit{Halogen} \wedge y = \mathit{Week})
\end{aligned}$$

この仕様に対するプログラムを作成すると次のようになる．

$$\begin{aligned}
\mathrm{Prog}_2 &\equiv (F_2, \emptyset, \{R_2^{\mathrm{I1}}, R_2^{\mathrm{I2}}, R_2^{\mathrm{O}}\}, \{D_2^{\mathrm{I1}}, D_2^{\mathrm{I2}}, D_2^{\mathrm{O}}\}) \\
D_2^{\mathrm{I1}} &\equiv D_1^{\mathrm{I1}} \cup \langle \{\mathit{OrbitS}, \mathit{OrbitP}, \mathit{OrbitD}, \mathit{OrbitF}\}, \\
&\quad \{\langle \mathit{Metal} \preceq \mathit{OrbitS}, \mathit{Metal} \preceq \mathit{OrbitP}, \\
&\quad \mathit{Metal} \preceq \mathit{OrbitD}, \mathit{Metal} \preceq \mathit{OrbitF}\}\rangle \\
D_2^{\mathrm{I2}} &\equiv D_1^{\mathrm{I2}} \cup \langle \{\mathit{Halogen}, \mathit{NonHalogen}\}, \\
&\quad \{\mathit{NonRaregas} \preceq \mathit{Halogen}, \\
&\quad \mathit{NonRaregas} \preceq \mathit{NonHalogen}\}\rangle \\
D_2^{\mathrm{O}} &\equiv D_1^{\mathrm{O}} \cup \langle \{\mathit{Week}, \mathit{Strong}\}, \\
&\quad \{\mathit{Ion} \preceq \mathit{Week}, \mathit{Ion} \preceq \mathit{Strong}\}\rangle \\
R_2^{\mathrm{I1}} &\equiv \{\mathit{OrbitS}, \mathit{OrbitP}, \mathit{OrbitD}, \mathit{OrbitF}\} \\
R_2^{\mathrm{I2}} &\equiv \{\mathit{Raregas}, \mathit{Halogen}, \mathit{NonHalogen}\} \\
R_2^{\mathrm{O}} &\equiv \{\mathit{NonIon}, \mathit{Strong}, \mathit{Week}\}
\end{aligned}$$

また，プログラム関数 F_2 は次のように定義できる．

$$F_2 :: R_2^{\mathrm{I1}} \rightarrow R_2^{\mathrm{I2}} \rightarrow R_2^{\mathrm{O}}$$

$$\begin{aligned}
F_2(x, y) &= \mathit{Strong}\ , \ \mathbf{if}\ x = \mathit{OrbitS} \wedge y = \mathit{Halogen} \\
&= \mathit{NonIon}\ , \ \mathbf{if}\ x = \mathit{OrbitP} \wedge y = \mathit{Halogen} \\
&= \mathit{Week}\ , \ \ \mathbf{if}\ x = \mathit{OrbitD} \wedge y = \mathit{Halogen}
\end{aligned}$$

$\tilde{F}_2$の値を表3に示す．

$\mathcal{S}_2$をさらに詳細化していくと，最終的には金属元素は図6の様なドメインに具象化でき，非金属元素は図7の様なドメインに具象化できた．そしてその結果，イオン結合度は図8の様なドメインになった．

	NonRaregas	Halogen	NonHalogen
OrbitS	Unknown	Strong	Ion
OrbitP	Unknown	NonIon	Ion
OrbitD	Unknown	Week	Ion
OrbitF	Unknown	Unknown	Ion

表3 $\widetilde{F}_2$の値

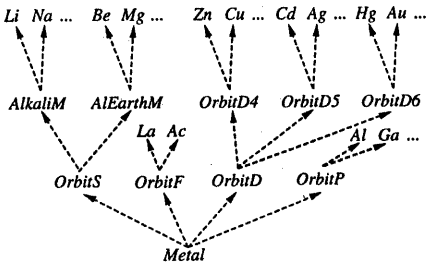

図6 最終的な金属元素のドメイン

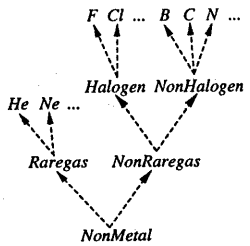

図7 最終的な非金属元素のドメイン

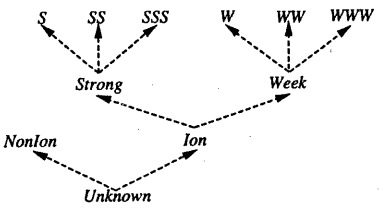

図8 最終的なイオン結合度のドメイン

8 おわりに

本稿では, データを段階的具象化に基づくソフトウェア構成法 ISDR 法を提案した. ISDR 法では, データを具象化する事でソフトウェアを段階的に詳細化し, その途中段階で現れるプログラムに対して抽象解釈を用いて意味を与え, 実行可能にした. これによって, ソフトウェアの詳細化の途中段階で, その概念的な誤りを見つけ出す事が容易になった.

また, 実際にソフトウェアを ISDR 法で構築する例題を述べることで, この方法の妥当性を示した.

複数の入力を持つソフトウェアに対し, 本稿では関数をカリー化することで, それぞれの入力を全く別の集合として扱った. その他にも, 関数をカリー化しないで複数入力を許す場合と, 全ての入力集合を直積した集合を 1 つの入力集合として考える場合がある. 関数に複数入力を許す場合, 注目する引数ごとに関数を定義するプログラムスライシングの技法を応用できる. 注目する引数ごとに関数を定義することで, 関数の機能を段階的に強化したり, 重要な変数から関数を定義する事が可能になる.

今後は ISDR 法を応用した総合的なソフトウェア開発環境を考察し, 実装を行う予定である. その環境には, ドメインの具象化の様子を分かりやすく表示し, 次にどのデータを具象化すべきかの候補を示すようなシステムが必要であろう. また, この環境の上で複雑で大規模なソフトウェアを構築することで, ISDR 法の有効性を実証する.

参考文献

[1] 鶴巻維男, 菊地豊, 片山卓也: ソフトウェア進化に基づくフォールトトレラント手法, 第 11 回大会論文集, pp. 129–132 日本ソフトウェア科学会 (1994).

[2] ABRAMSKY, S. and HANKIN, C. eds.: *ABSTRACT INTERPRETATION OF DECLARATIVE LANGUAGE*, ELLIS HORWOOD LIMITED (1997).

[3] 吉岡信和, 鈴木正人, 片山卓也: データドメインの詳細化に基づくプログラムの段階的構成法, 第 49 回全国大会講演論文集, 第 5 巻, pp. 251–252 情報処理学会 (1994).