組み合わせテストツール Calot の開発に向けて

山 田 晃 久[†1] 北 村 崇 師[†1]

組み合わせテストはソフトウェア・システムテストのコスト削減，品質向上に効果が期待されるが，既存の組み合わせテストツールは産業界での利用で現れる規模のテストモデルの効果的な設計，高速なテスト生成を行うことができない．本研究では，これらの課題を解決する組み合わせテストツール Calot の開発に取り組んでいる．

Towards Developing the Combinatorial Testing Tool Calot

Akihisa Yamada[†1] and Takashi Kitamura[†1]

Combinatorial testing is expected to reduce the cost and improve the quality of software and system tests, but current combinatorial testing tools are not efficient on designing and handling test models of industrial scale. In this work, we consider solving these problems by developing a combinatorial testing tool, Calot.

1. はじめに

組み合わせテスト (cf.[6]) とは，テスト対象システム (System Under Test) を構成するパラメータの相互作用に着目したテスト技法である．比較的小さい t について，任意の t 個のパラメータについてすべての値の組み合わせをテストすることで，大部分の不具合が検出されることが実験的に示されている．[4]

例として，以下のように様々な環境下での動作を要求されるウェブアプリケーションを考える．

パラメータ	値
CPU	Intel, AMD
OS	Windows, Linux, Mac
Browser	IE, Firefox, Safari

例えば $t = 2$ の時 (pair-wise，all-pair 法ともいう)，考慮すべき組み合わせは {CPU=Intel, OS=Windows}，{Browser = Safari, CPU = Intel} 等がある．

現実には，すべての組み合わせが実現可能というわけではない．例えば，前述のウェブアプリケーションにおいては，以下のような**制約** (禁則) がある．

(1) IE は Windows 上でのみ動作する．
(2) Safari は Mac 上でのみ動作する．
(3) Mac は AMD の CPU をサポートしていない．

以上の 3 つの制約を満たしつつ，任意の 2 パラメータの (制約を満たす) 組み合わせをすべて網羅したテストケースの集合を以下に示す．

No.	CPU	OS	Browser
1	Intel	Windows	IE
2	Intel	Mac	Safari
3	Intel	Linux	Firefox
4	AMD	Windows	Firefox
5	AMD	Linux	Firefox
6	AMD	Windows	IE
7	Intel	Mac	Firefox

産業技術総合研究所では，産業界と協力し，組み合わせテストの実用化に向けた研究を行っている．この中で行っている適用事例において，PICT[3] や ACTS[1] などの既存の組み合わせテストツールではテスト対象を容易にモデル化できない例や，効率的にテストケースを生成できない例が見つかっている．

本研究では，木構造を用いたテストモデリング手法，高速なテストケース生成，テストケース数の最適化の機能を持つ組み合わせテスト生成ツール Calot の実現に取り組んでいる．

2. テストモデルの構造化

前述のウェブアプリケーションの例はきわめて簡素化されているが，実用上はパラメータ数が 100 を超える例も珍しくない．テスト対象のモデル・制約は複雑であり，テストの品質を担保することが難しい．

Calot は，テストモデルの構造的設計を可能にする．

†1 独立行政法人 産業技術総合研究所 (National Institute of Advanced Industrial Science and Technology)

表 1　既存ツールと Calot の比較．ベンチマークは Cohen et al.[2] による 5 例と適用事例より 1 例．$t = 2$．*は最適解であることの証明を含む．

			PICT		ACTS		CASA		Calot (greedy)		Calot (optimum)	
Name	Model	Constraints	#	time	#	time	#	time	#	time	#	time
SPIN-S	$2^{13}4^5$	2^{13}	26	0.2	26	0.9	19	2.7	26	0.0	19*	0.6
SPIN-V	$2^{42}3^24^{11}$	$2^{47}3^2$	63	0.2	45	1.5	38	19.2	43	0.0	31	169.6
GCC	$2^{189}3^{10}$	$2^{37}3^3$	30	0.3	23	1.7	21	1317.1	22	0.3	15	164.5
Apache	$2^{158}3^84^45^16^1$	$2^33^14^25^1$	40	0.3	33	1.1	30	115.3	33	0.1	30*	166.2
Bugzilla	$2^{49}3^14^2$	2^43^1	20	0.2	19	0.5	16	2.5	18	0.0	16*	1.5
Industrial 1	$2^53^{20}4^25^26^110^1$	2^{45}	–	>1h	95	15.4	(N/A)		96	0.1	78	31.9

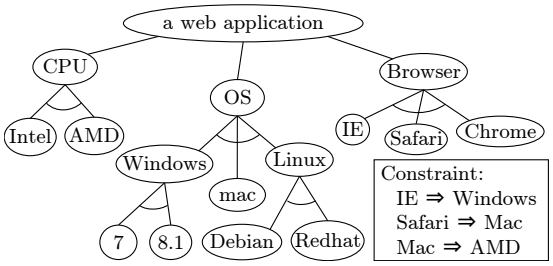

図 1　構造化されたテストモデル

前述の例で，Windows を 7 と 8.1 に，Linux を Debian と Redhat に細分化した例を示す．

```
1  CPU: ( Intel | AMD );
2  OS: (
3   | Windows: ( 7 | 8.1 )
4   | Mac
5   | Linux: ( Debian | Redhat )
6  );
7  Browser: ( IE | Safari | Chrome );
8  $assert (IE => Windows) &&
9          (Safari => Mac) &&
10         (Mac => ! AMD);
```

3. テストケース生成への SAT ソルバの利用

充足可能性 *(Satisfiability; SAT)* **ソルバ**とは，変数を含む CNF 形式の論理式 ϕ を入力し，ϕ を真にするような変数への割り当てが存在するか否か (充足可能性) を自動で判定するツールであり，組み合わせテストへの利用は既に研究されている (cf.[5])．

多くの SAT ソルバは，単に論理式の SAT 判定を行うほかに，*Unsat* **コア**の出力機能や，**逐次的** SAT 判定のための節の追加・削除等の機能を備えている．本研究ではこれらの機能を利用し，組み合わせテスト生成，最適化の効率改善を図る．

3.1 Greedy なテストケース生成

現実的な数のテストケースを作るため，greedy なヒューリスティックスによる手法が研究されている．既存手法では，制約判定を何度も行う必要があり，強い制約を持つテストケース生成の効率を悪化させる要因となっていた．我々は，Unsat コアの情報を用いることで制約判定回数を減らし，PICT ベースの greedy なテストケース生成を効率化できることを発見した．実験的な実装において，既存ツールが効率的にテストケースを生成できない適用事例に対しても効率的にテストケース生成できることを確認している．

3.2 テストケースの最適化

Calot は，SAT ソルバの逐次的機能を用いたテストケース最適化機能を提供する．[7] これにより (カバレッジを保ちつつ) テストケース数を削減でき，特に人間による操作や確認が必要な場合など，テスト実施が高価な場合にコスト削減につながる．

表 1 に既存ツールと Calot の比較実験結果を載せる．

参　考　文　献

1) Borazjany, M.N., Yu, L., Lei, Y., Kacker, R. and Kuhn, R.: Combinatorial Testing of ACTS: A Case Study, *ICST 2012*, pp.591–600 (2012).
2) Cohen, M.B., Dwyer, M.B. and Shi, J.: Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach, *IEEE T. Software Eng.*, Vol.34, No.5, pp.633–650 (2008).
3) Czerwonka, J.: Pairwise testing in real world, *PNSQC 2006*, pp.419–430 (2006).
4) Kuhn, D. R., Wallace, D. R. and Gallo, Jr., A.M.: Software Fault Interactions and Implications for Software Testing, *IEEE T. Software Eng.*, Vol.30, No.6, pp.418–421 (2004).
5) Nanba, T., Tsuchiya, T. and Kikuno, T.: Using Satisfiability Solving for Pairwise Testing in the Presence of Constraints, *IEICE Trans. Fundamentals*, Vol.E95-A, No.9, pp.1501–1505 (2012).
6) Nie, C. and Leung, H.: A Survey of Combinatorial Testing, *ACM Computing Surveys*, Vol.43, No.2, pp.11:1–11:29 (2011).
7) Yamada, A., Kitamura, T., Artho, C., Choi, E., Oiwa, Y. and Biere, A.: Optimization of Combinatorial Testing by Incremental SAT Solving, *ICST 2015*. To appear.