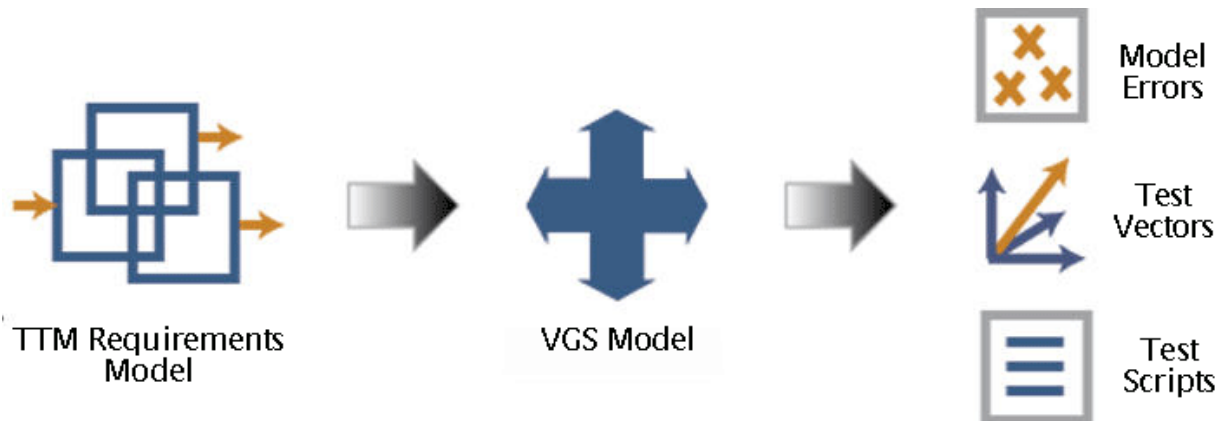




< TTM (T-VEC Tabular Modeler) に関して >

T-VEC Tabular Modeler (TTM)は、要求仕様・設計をモデル化して形式手法を活用するためのツール。モデル化された仕様・設計に対して、形式手法を用いて欠陥の自動解析(モデル検査、定理証明) テスト自動生成が、T-VEC Test Vector Generation System (VGS)を用いて行われます。TTM モデル言語は、米国海軍研究試験所 (Naval Research Lab) にて開発された、SCR (Software Cost Reduction) を基にして開発された。



TTMを用いた自動ベリフィケーション&バリデーションは、要求仕様・設計をモデル化することから始まる。TTMモデルではシステム仕様を、システム入力と期待されるシステム出力間のリレーションシップに表現。そしてモデル検査、テスト自動生成のためにT-VEC独自のモデル(形式的仕様記述)に変換される。これに対して T-VEC Test Vector Generation System (VGS)でテストベクタを生成させることでモデル検査(矛盾を検出)し、要求仕様・設計上に潜在する欠陥を明らかにする。欠陥が修正されれば、VGS からモデル上の各要件に対してテストベクタが生成される。これらテストベクタは、要求仕様の正当性確認や、システム動作の検証に用いることができる(仕様と実装の一致性検証)。また テストベクタをテストスクリプト化する仕組みを持ち、これによりシステムに対して実行させることができる。

自動モデルベーステスト生成を支える実践的でユニークなTTM :

- * モデルの分解 -モデル担当者の協調作業を促進
- * モデルの再利用 -既存モデルを基にして新しいモデルへ拡張できる
- * 要件に対するトレーサビリティ
-要件IDはモデル、テストベクタに反映され、要件からテストに至るトレーサビリティが支援される
- * モデル検査 -要件やモデル上の欠陥を開発早期段階に発見
- * モデル検査とテストベクタ生成の早期段階からの実施
- * 文字列、文字、配列、構造体をサポート
- * SCRツールの SSL 、SCRファイルのインポートをサポート

更なるTTMの機能：

- * 複雑な式・表現の体裁を自動で整える
- * 便利な検索機能
- * キーボードショートカット設定

< TTM (T-VEC Tabular Modeler) 表形式のモデリングについて >

TTM は、要求仕様を表形式にフォーマルモデル化するためのツール。これは、米国海軍研究試験所 (Naval Research Lab) にて開発された、SCR (Software Cost Reduction) を基にして開発された。TTMはSCRを拡張し、モデル解析、自動テスト生成をT-VECツールでできるようにしている。

SCR について (SCR の資料より)

SCRによる要件記述手法は、正確で、曖昧でなく、テストが容易な、フォーマルな要求仕様を構築するためのアプローチとしてNRLにて開発された。研究者によりフォーマルメソッドは、産業界のシステムの、要件上の問題・欠陥 (曖昧さ、不完全性、不正確さ) などの検証に向いていることを提唱してきたが、大規模で複雑なシステムへ適用することは困難で実践的ではなく、採用されることが殆ど無いのが現状。これに対しSCRは、産業界の大規模なリアルタイムシステムでもフォーマルメソッドの効果を享受できるように開発された。使い易さ、スケーラビリティ、コスト効率の良さなどを踏まえたアプローチであり、産業界の需要を満たすもの。実践的なシステムに幅広く採用されている。(防衛、電話通信網、潜水艦通信、宇宙、原子力発電プラントの制御系 などのシステム)

TTMによる表形式のモデリングは、以下の特有な表現で要求をモデル化。

- ・システムインターフェイス
入力、出力変数、それらのデータタイプ、値の範囲 を十分に定義する
- ・リレーションシップ
入力変数と媒介するリレーションシップにて、システム出力を定義するもの。媒介するリレーションシップは、複雑なリレーションシップを分かりやすい単位に分解することで得る。
- ・コンディション
ある特定の出力変数と入力間のリレーションシップが、正当となる一定期間。
- ・イベント
ある特定の出力変数と入力間のリレーションシップが、正当となるユニークなタイミング。
- ・モード
イベントやコンディションへの経緯を示し、それらに意味のある名前を与え、要件を経緯 history として表現できるようにする。

入力と出力は、システムとそれが動作する環境の相互作用を描写する。Term には中間値を設定し、モデルの階層化のための分解を支える。Mode Machine は、単純な有限ステートマシンとして内部状態を表現する。アサーションは、条件式で、ユニークな名前を持ち、モデル内の如何なる場所でも参照することができる。

TTMによるモデリングは、ソフトウェアやシステムの要件を、従来式のテキストによる要求仕様より、フォーマル（形式的）で厳密な形式に記載すること。これにより従来式の要求仕様の問題 - 曖昧さ、不一致、不完全性などの侵入が未然に防がれることになり、また TTM モデルは自動解析され、これら問題に対する検査が行われる。

TTMに要件を仕様化する上で、テキストエディタや Telelogic DOORS など、インフォーマルなテキスト記載は、フォーマルな表現に変更されるか、改善される必要がある。人間のあつかう自然言語では、多くの異なる記載方法とその解釈が、仕様記述者と読む側とのバックグラウンドによって生じる。結果、誤った解釈、間違いとなる。TTMにフォーマルな表現を定義することで、これら誤った解釈、間違いを防ぐことができる。TTMでは、フォーマルに記載される要件は、シンプルである。要件の情報をフォーマルに記載するための表記法は、C言語のような一般に知られたプログラミング言語を書くことや解釈することほど複雑ではない。それこそ C++ や Ada など複雑な言語と比較すれば、より簡単である。

TTMのフォーマルな要件記述表記とプログラミング言語の主な違いは、要件の表現方法であり、それはシステム実行中のあらゆるタイミングの、システムの入力と出力間に存在するべきリレーションシップの表現。プログラミング言語では、特定ターゲットの動作コンディション下で、それら要求されるリレーションシップを成すシーケンシャルな動作を表記する。対して TTM の要件の記述は、プログラミング言語同様に数式やリレーショナルオペレータを使うものの、各式は、他の全ての式に独立であり、要件には特定の順位などもない。TTM内の各要件式は、特定のシステム出力、あるいは中間 team 変数に対して記述される。TTM内の各要件は、出力値を取るための式、条件式の2つの式からなる。

そして TTM にモデル化された要求仕様やデザインは、独自の階層的 Disjunctive Normal Form (DNF) に変換され、この形式に於ける各機能要求（入出力間の振舞い）の入力域に対するコンストレインツは Domain Conversion Path (DCP) として抽出されます。このパスに対し、テストベクタ生成システムは、上流の制約を受けて伝播されてくる入力範囲で期待出力との組合せを生成します。ここで、テストベクタを生成できない DCP は、モデル上の矛盾として検出されます。また、このテストベクタ生成機能は、フロート、ダブルなど各種データ型、リニア/ノンリニア式に対応しているため、矛盾無く生成されたテストベクタは、要求仕様の正当性確認や、実システムに対する入力と期待値としてテストに用いられ、モデルに対するコードの一致性の検証が行うことができます。

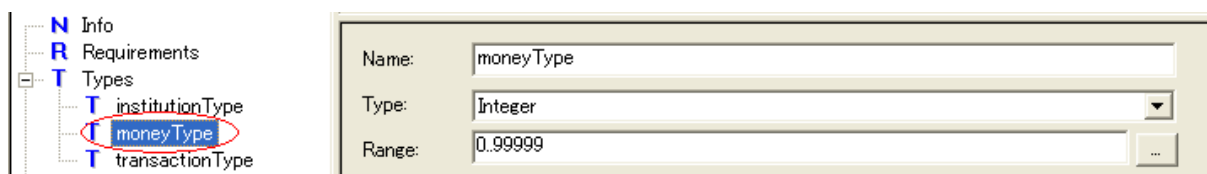
以下、TTMの各フィールドについて説明しています。

【各フィールドの詳細】

[Types]

モデルで使用する型を定義します。

例：Integer 型で、範囲が 0~99999 の moneyType という型を定義



[Constants]

モデルで使用する定数を定義します。

例：Integer 型で、値 180 の HIGH_TEMP という定数を定義

The screenshot shows a tree view on the left and a configuration panel on the right. In the tree view, the 'Constants' folder is expanded, and 'HIGH_TEMP' is selected and circled in red. The configuration panel on the right has the following fields:

- Name: HIGH_TEMP
- Type: Integer (selected from a dropdown menu)
- Value: 180

At the bottom of the configuration panel, there is a table with two columns: 'Requirement ID' and 'Heading (right-click below to edit)'.

[Inputs]

モデルの入力を定義します。

例：Types で定義した tempType 型で。入力変数 temperature を定義

The screenshot shows a tree view on the left and a configuration panel on the right. In the tree view, the 'Inputs' folder is expanded, and 'temperature' is selected and circled in red. The configuration panel on the right has the following fields:

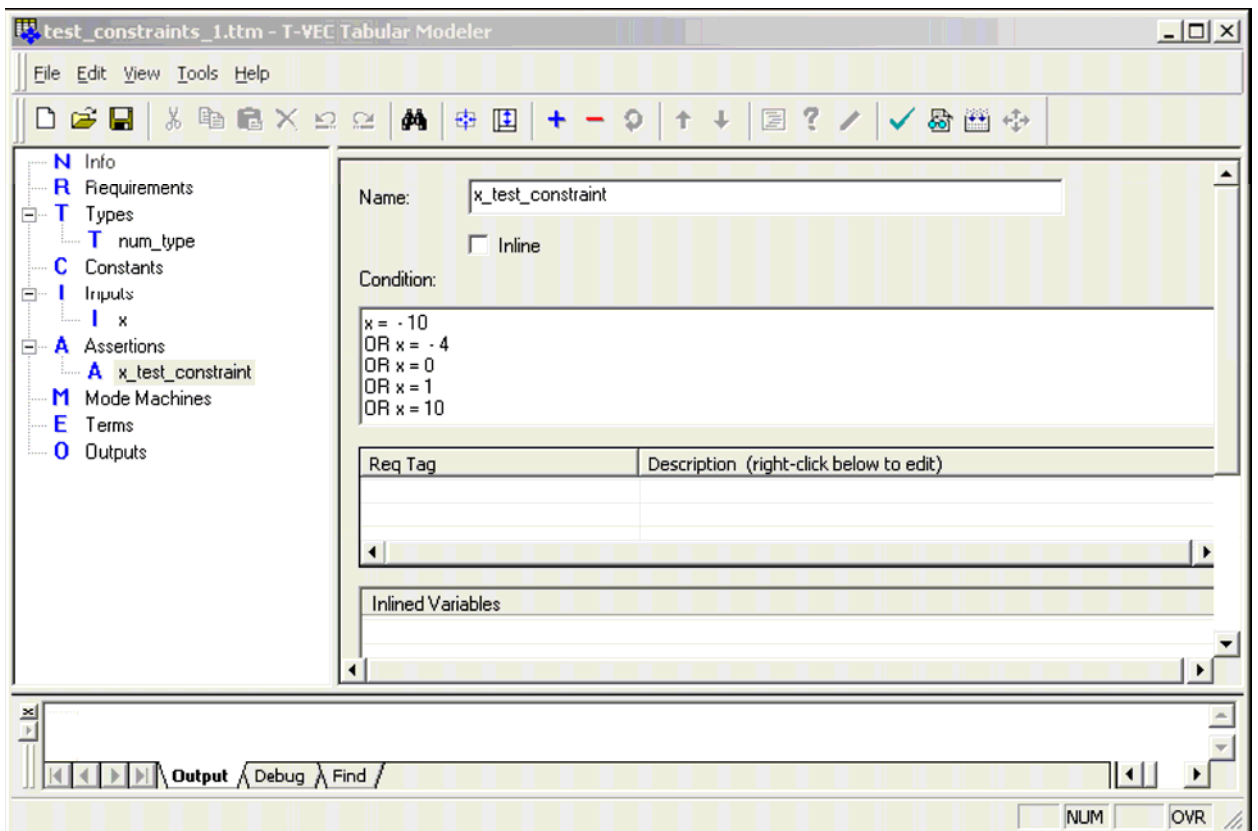
- Name: temperature
- Type: tempType (selected from a dropdown menu)
- Initial Value: -
- Range: 0.300 (with a small menu icon to the right)
- Accuracy: (empty field)

At the bottom of the configuration panel, there is a table with two columns: 'Requirement ID' and 'Heading (right-click below to edit)'.

[Assertions]

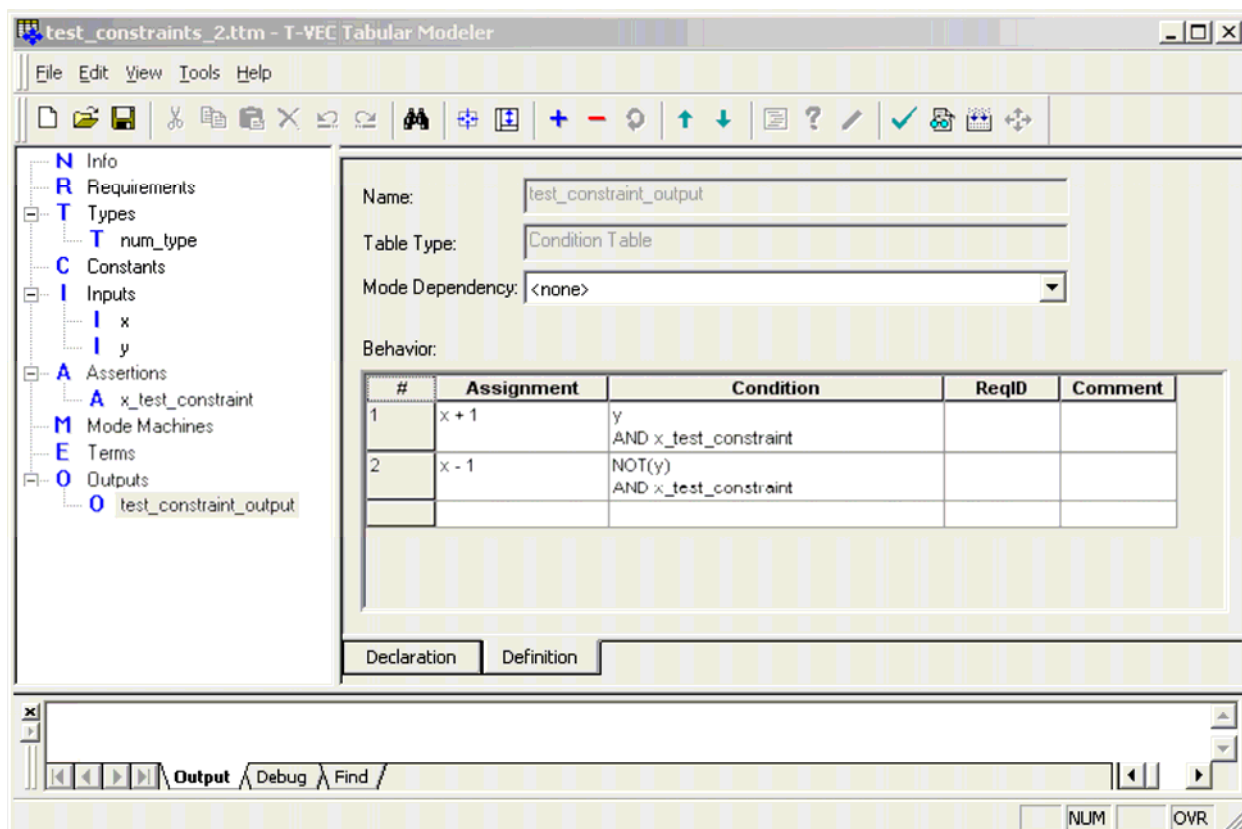
モデル全体で適用される前提条件を定義します。

例： < Assertion の設定 >



x は、 - 1 0 から 1 0 までの値を取り得る数値型変数です。上記の様に Assertions を設定することで、この変数の値を - 1 0、 - 4、 0、 1、 1 0 だけに限定できます。

< Assertions の使用 >



上のテーブルで Assertions の設定を無視 (AND 以降が無いと仮定) すれば、以下の処理が行われます。

- ・ y が true なら、x は - 9 から 1 1 を出力
- ・ y が false なら、x は - 1 1 から 9 を出力

Conditions に AND を使って x_test_constraint の Assertions を適用させることで、行われる処理は以下の様になります。

- ・ y が true なら、x は - 9、 - 3、 1、 2、 1 1 を出力
- ・ y が false なら、x は - 1 1、 - 5、 - 1、 0、 9 を出力

[Functions]

Functions には、与えられたパラメータで構成された処理に応じた値が設定されます。つまり、共通関数のように使用でき、その際パラメータは、引数のような働きをします。同じような処理が複数ある場合に、その処理を Functions で規定することでモデル化を容易にできます。例えば、積分値のような状態変数を取るモデルを容易に組むことも出来ます。

Functions へのパラメータとしては、Inputs または Terms が使えます。

機能的には Terms と似ていますが、Mode Dependency タブが使用できない点と、パラメータが使用できる点で異なります。

例 : Parameters タブで引数を、trackAttributes の処理をテーブルに定義します。

#	Name	Type	Comment
1	trk	trackingDataType	

Main Parameters Requirements Comment

Behavior: trackAttributes

#	Assignment	Condition
1	bearing = trk.bearing; color = WHITE; distanceTau = trk.distanceTau; visible = FALSE;	trackAdvisory(trk) = NONE
2	bearing = trk.bearing; color = YELLOW; distanceTau = trk.distanceTau; visible = TRUE;	trackAdvisory(trk) = TA
3	bearing = trk.bearing; color = RED; distanceTau = trk.distanceTau; visible = TRUE;	trackAdvisory(trk) = RA

下記、Outputs で上記 trackAttributes をコールして処理しています。

Name: trackDisplayMsg
Type: trackDisplayMsgType
Initial Value: -
Range: -
Accuracy: -

Main Mode Dependency Requirements Inlined Variables Comment

Behavior: trackDisplayMsg

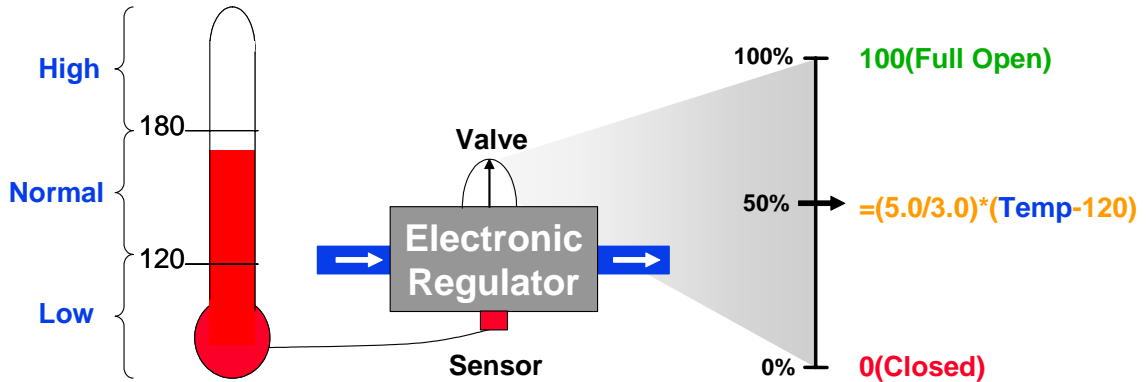
#	Assignment	Condition
1	avoidanceMsg = NONE; trk1.visible = FALSE; trk2.visible = FALSE; trk3.visible = FALSE;	INTERM_adv=advisoryStatus(sensorData) AND INTERM_adv = NONE
2	avoidanceMsg = TA; trk1 = trackAttributes(sensorData.trk1); trk2 = trackAttributes(sensorData.trk2); trk3 = trackAttributes(sensorData.trk3);	INTERM_adv=advisoryStatus(sensorData) AND INTERM_adv = TA
3	avoidanceMsg = RA; trk1 = trackAttributes(sensorData.trk1); trk2 = trackAttributes(sensorData.trk2); trk3 = trackAttributes(sensorData.trk3);	INTERM_adv=advisoryStatus(sensorData) AND INTERM_adv = RA

[Mode Machine]

Mode Machine は、状態遷移を定義します。Modes タブで状態を定義し、Transitions テーブルで状態遷移を定義します。Terms と Output の Mode Dependency タブで選択することで、Mode Machine を参照することができます。

例：要求仕様

Temp Range [0...300]



Operational Summary

If Temp is less than Low, then valve is **Closed**

If Temp is greater than High, then valve is **Full Open**

If Temp is between High and Low, then valve is a function of Temp

下記テーブルの1行目、@T(temperature >= LOW_TEMP) @T(式)：式が真
 状態が low の時に、入力 temperature が、定数 LOW_TEMP(120)以上の時、normal へ遷移

- N Info
- R Requirements
- T Types
 - T tempType
 - T valveType
- C Constants
 - C HIGH_TEMP
 - C LOW_TEMP
- I Inputs
 - I temperature
- A Assertions
- F Functions
- M Mode Machines
 - M **sensorMode**
- E Terms
- O Outputs

#	Name	Initial	Final	Enum Value
1	low	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	normal	<input type="checkbox"/>	<input type="checkbox"/>	
3	high	<input type="checkbox"/>	<input type="checkbox"/>	

Main Modes Requirements Inlined Variables Comment

Transitions: sensorMode

#	Source	Guard	Destination	Requirement ID
1	low	@T(temperature >= LOW_TEMP)	normal	
2	normal	@T(temperature > HIGH_TEMP)	high	
3	high	@T(temperature <= HIGH_TEMP)	normal	
4	normal	@T(temperature < LOW_TEMP)	low	

[Terms]

Terms 変数は、システムの入出力の中間値に相当します。上記の Inputs や他のテーブルを使って処理した値を一時的に使用したい場合に有効です。

機能的には Functions と似ていますが、Mode Dependency タブが使用できる点と、パラメータが使用できない点で異なります。(下記、<Table Type> を参照下さい)

Terms テーブルを利用することで、Outputs テーブルが複雑にならないようにすることができます。

[Outputs]

Outputs は、モデル全体の最終的な出力を定義します。この出力は、上記の Inputs やテーブルを使って処理できます。(下記、<Table Type>を参照下さい)

<Table Type >

Terms と Outputs には、Conditions と Event と呼ばれるテーブルがあります。

[Condition]

Inputs や他のテーブルを使って、複数の条件を設定できます。

各条件に応じた適切な値が出力されます。

例：このテーブルでは、以下の処理を行います。

- $x \geq 0$ 且つ y が false の時に x を出力
- $x < 0$ 且つ y が false の時に $-x$ を出力
- y が true の時に 0 を出力

The screenshot shows the T-VEC Tabular Modeler interface. The main window title is '11_modeless_condition_table.ttm - T-VEC Tabular Modeler'. The left sidebar shows a tree view with 'modeless_condition_table' selected under the 'Outputs' category. The main area displays the configuration for this table. The 'Name' is 'modeless_condition_table', 'Table Type' is 'Condition Table', and 'Mode Dependency' is '<none>'. Below this is a 'Behavior' table with the following data:

#	Assignment	Condition	ReqID	Comment
1	x	$x \geq 0$ AND NOT(y)		
2	- x	$x < 0$ AND NOT(y)		
3	0	y		

At the bottom of the configuration area, there are tabs for 'Declaration' and 'Definition'. The bottom status bar shows 'Output', 'Debug', and 'Find' options, along with 'NUM' and 'OVR' indicators.

[Event]

Inputs や他のテーブルを使って、複数の条件を設定できます。

各条件に**変化が発生した時のみ**、適切な値が出力されます。

例：このテーブルでは、以下の処理を行います。

- down_button が false の状態で up_button が true に変化した時に、 $current_value < MAX_VALUE(100)$ であれば $current_value + 1$ を出力

- up_button が false の状態で down_button が true に変化した時に、
current_value > MIN_VALUE(0) であれば current_value - 1 を出力

The screenshot shows the T-VEC Tabular Modeler interface for a component named 'modeless_event_table'. The left sidebar shows a tree view with 'modeless_event_table' selected under the 'Outputs' category. The main window displays the component's configuration and behavior.

Component Information:

Name	Type	1st value	Table	Inline	Range	Accuracy
modeless_event_table	Integer	-	event	no	-10..10	

Configuration:

Name: modeless_event_table
Table Type: Event Table
Mode Dependency: <none>

Behavior:

#	Assignment	Event	ReqID	Comment
1	current_value + 1	@T(up_button) WHILE(NOT(down_button)) AND (current_value < MAX_VALUE)		
2	current_value - 1	@T(down_button) WHILE(NOT(up_button)) AND (current_value > MIN_VALUE)		

The interface also includes a menu bar (File, Edit, View, Tools, Help), a toolbar with various icons, and a status bar at the bottom with 'Output', 'Debug', and 'Find' options, along with 'NUM' and 'OVR' indicators.

< その他の機能 >

【イベント・コンディションの重複、競合状態の評価機能】

[Disjointness オプション]

Condition テーブルや Event テーブルの Disjointness (非重複性) チェックを行います。

テーブルの各列毎の条件式 (Condition) を AND することで、重複部分があるかチェックします。条件式が重複していれば、テーブルの複数の列が同時に関連することを意味し、それぞれの条件式を満たす入力があるということになります。そうすると、どの結果を返せばよいのか判断できない、つまり要求仕様上の欠陥を検出できます。

例: int 型の X_int、Y_int があり、以下のように条件式が設定されているとします。

- ・ int_disjointness_test_FAIL (条件式の重複あり)

	Assignment	Condition
1	TRUE	X_int > 10 OR Y_int > 10
2	FALSE	X_int > 20
3	FALSE	Y_int > 20

上記は、X_int が 25 の時、1 列目と 2 列目を同時に満たしており、TRUE、FALSE のどちらを出力すればよいか判断できません。同様に Y_int が 25 の時、1 列目と 3 列目を同時に満たしています。

- ・ int_disjointness_test_PASS (条件式の重複なし)

Name:

Type:

Initial Value:

Range:

Accuracy:

Table Type
 Condition
 Event

Inline
 Check for Disjointness

Main | Mode Dependency | Requirements | Inlined Variables | Comment

Behavior: int_disjointness_test_PASS

#	Assignment	Condition
1	TRUE	(X _{jnt} > 10 AND X _{jnt} ≤ 20 AND Y _{jnt} ≤ 20) OR (Y _{jnt} > 10 AND Y _{jnt} ≤ 20 AND X _{jnt} ≤ 20)
2	FALSE	X _{jnt} - 10 > 20 AND Y _{jnt} + 10 ≤ 20
3	FALSE	Y _{jnt} - 10 > 20 AND X _{jnt} + 10 ≤ 20

上記は、各列で条件式の重複はありません。

このモデルの重複有無を T-VEC の Vector Generation System で実行すると以下のような結果が得られます。

Subsystem	Compilation		Test Vectors		Coverage	
	DCPs	Warn/Err	Vectors	Warn/Err	Untested DCPs	Warn/Err
<u>boolean_disjointness_test_FAIL</u>	4	0/0	6	0/0	0	0/0
<u>boolean_disjointness_test_PASS</u>	4	0/0	5	0/0	0	0/0
<u>int_disjointness_test_FAIL</u>	4	0/0	8	0/0	0	0/0
<u>int_disjointness_test_PASS</u>	4	0/0	7	0/0	0	0/0
<u>race_condition_test_FAIL</u>	4	0/0	8	0/0	0	0/0
<u>race_condition_test_PASS</u>	4	0/0	8	0/0	0	0/0
<u>_coreRTS</u>	0	0/0	-	-	-	-
<u>disj_check_boolean_disjointness_test_FAIL</u>	5	0/0	6	0/0	0	0/0
<u>disj_check_boolean_disjointness_test_PASS</u>	5	10/0	0	0/0	5 of 5	0/31
<u>disj_check_int_disjointness_test_FAIL</u>	5	0/0	8	0/0	0	0/0
<u>disj_check_int_disjointness_test_PASS</u>	5	0/0	0	0/10	5 of 5	0/31
<u>raceCond_check_race_condition_test_FAIL</u>	1	0/0	2	0/0	0	0/0
<u>raceCond_check_race_condition_test_PASS</u>	1	1/0	0	0/0	1 of 1	0/7

int_disjointness_test_FAIL(条件式の重複あり)では、テストベクタ(8件)が生成されています。これは、テーブルの各列を AND 条件したロジック(DCP)が抽出されテストベクタが生成できた、つまり AND した条件式に重複部分が存在することを意味しています。

int_disjointness_test_PASS(条件式の重複なし)では、テストベクタが生成されていません。これは、テーブルの各列を AND 条件したロジックが抽出できず(DCP エラー)、テストベクタが生成されなかった、つまり AND した条件式に重複部分が存在しないことを意味しています。

この機能により重複部分をチェックすることで要求仕様上の欠陥を検出することができます。

[Race Conditions オプション]

ModeMachine (状態遷移) における競合状態のチェックを行います。ある一つの状態 (A) から、条件に応じて複数の遷移先状態 (B、C、...) に移行する場合に、遷移前の状態が同じである移行条件を AND することで、条件に重複部分がないかチェックします。条件が重複していれば、状態遷移が曖昧になっていることを意味し、どの状態へ遷移すればよいのか判断できない、つまり要求仕様上の欠陥を検出できます。

例：

int 型の X_int、Y_int、状態 A、B、C があり、以下のように状態遷移が設定されているとします。

- ・ race_condition_test_FAIL (競合状態あり)

#	Source	Guard	Destination
1	A	@T(X_int > 10)	B
2	A	@T(X_int > 20)	C
3	B	@T(Y_int > 10)	C
4	C	@T(Y_int > 20)	A

上記は、遷移前の状態 A で、X_int が 25 の時、1 列目と 2 列目を同時に満たし、B or C のどちらに遷移すればよいか判断できません。

- ・ race_condition_test_PASS (競合状態なし) (double 型の X_float)

#	Source	Guard	Destination
1	A	@T(X_int > 10) WHEN(X_float > 10.025)	B
2	A	@T(X_int > 20) WHEN(X_float <= 10.025)	C
3	B	@T(Y_int > 10)	C
4	C	@T(Y_int > 20)	A

上記 1 列目と 2 列目は、重複条件はありません。(1 列目と 2 列目は X_float により一意に決まる)
このモデルの重複有無を T-VEC の Vector Generation System で実行すると以下のような結果が得ら

れます。

Subsystem	Compilation		Test Vectors		Coverage	
	DCPs	Warn/Err	Vectors	Warn/Err	Untested DCPs	Warn/Err
<u>boolean_disjointness_test_FAIL</u>	4	0/0	6	0/0	0	0/0
<u>boolean_disjointness_test_PASS</u>	4	0/0	5	0/0	0	0/0
<u>int_disjointness_test_FAIL</u>	4	0/0	8	0/0	0	0/0
<u>int_disjointness_test_PASS</u>	4	0/0	7	0/0	0	0/0
<u>race_condition_test_FAIL</u>	4	0/0	8	0/0	0	0/0
<u>race_condition_test_PASS</u>	4	0/0	8	0/0	0	0/0
<u>__coreRTS</u>	0	0/0	-	-	-	-
<u>disj_check_boolean_disjointness_test_FAIL</u>	5	0/0	6	0/0	0	0/0
<u>disj_check_boolean_disjointness_test_PASS</u>	5	10/0	0	0/0	5 of 5	0/31
<u>disj_check_int_disjointness_test_FAIL</u>	5	0/0	8	0/0	0	0/0
<u>disj_check_int_disjointness_test_PASS</u>	5	0/0	0	0/10	5 of 5	0/31
<u>raceCond_check_race_condition_test_FAIL</u>	1	0/0	2	0/0	0	0/0
<u>raceCond_check_race condition test PASS</u>	1	1/0	0	0/0	1 of 1	0/7

race_condition_test_FAIL (競合状態あり) では、テストベクタ (2 件) が生成されています。これは、テーブルの 1 列目と 2 列目を AND 条件したロジック (DCP) が抽出されテストベクタが生成できた、つまり AND した条件式に重複部分が存在することを意味しています。

race_condition_test_PASS (競合状態なし) では、テストベクタが生成されていません。これは、テーブルの各列を AND 条件したロジックが抽出できず (DCP エラー)、テストベクタが生成されなかった、つまり AND した条件式に重複部分が存在しないことを意味しています。

この機能により、重複部分をチェックすることで要求仕様上の欠陥を検出することができます。

以上

ツールのデモンストレーションを行っています。

ご興味いただける場合は、お手数ですが下記までご依頼頂けると幸いです。



富士設備工業株式会社 電子機器事業部

〒591-8025 大阪府堺市北区長曾根町1928-1

Tel: 072-252-2128 www.fuji-setsu.co.jp