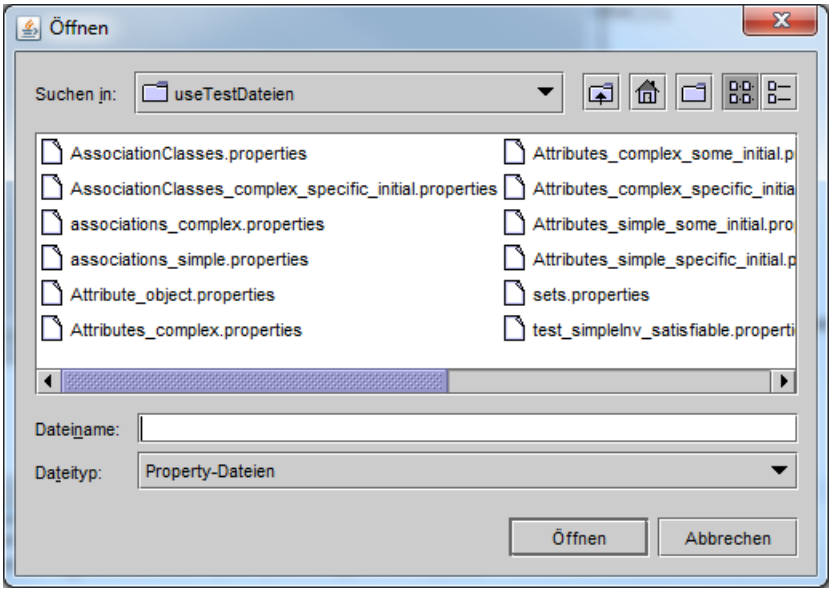
Kurzanleitung UseKodkodModelValidator

Aufruf über die GUI

1. Das Plugin wird über ein Button in der Toolbar von Use oder alternativ über den Menüpunkt Plugins – Validation – Kodkod gestartet

2. Sofern ein Modell geöffnet ist öffnet sich folgender Dateiauswahldialog.



Über den Dialog lassen sich .properties – Dateien laden, die momentan für die Konfiguration des Suchraums verwendet werden. Ein möglicherweise vorhandenes Objektdiagramm wird immer als Teillösung interpretiert.

3. Wenn eine Lösung gefunden wird, wird automatisch ein Objektdiagramm angelegt. In einer neuen Objektdiagrammansicht wird das aktuelle Objektdiagramm dann angezeigt.

Befehle

Der Modellvalidierer lässt sich ebenfalls über die Use-Konsole steuern Im Vergleich zu dem Aufruf über die GUI kann hier die gesamte Funktionalität des Modellvalidierers aufgerufen werden. Folgende Befehle sind momentan definiert:

- *kodkod –modelReset*

  Bedeutung:

  Setzt den Zustand des Modells auf den Ursprungszustand zurück, d.h. es ist nicht konfiguriert.

- *kodkod –validate*

  Parameter:

  - (optional) Pfad zu einer Konfigurationsdatei

  Bedeutung:

  Bei Angabe eines Pfades wird das Modell mit den Daten aus der Konfigurationsdatei konfiguriert. Ein möglicherweise vorhandenes Objektdiagramm wird ggf. als Teillösung interpretiert. Nach der Konfiguration wird ein Validierungsvorgang gestartet und bei einer gefundenen Lösung ein Objektdiagramm erstellt.

  Ohne die Angabe einer Konfigurationsdatei wird der Standardsuchraum verwendet und eine entsprechende Konfigurationsdatei erzeugt.

  - Pro Klasse exakt ein Objekt
  - Pro Assoziation genau ein Link
  - Alle Attributwerte sind definiert
  - String: 1 <= x <= 5
  - Real: -2 <= x <= 2 (Schrittweite 0.5)
  - Integer: -10 <= x <= 10
  - Alle Invarianten sind aktiviert

  Sollte mit dem Standardsuchraum keine Lösung gefunden werden und es sind Invarianten vorhanden, so werden die Invarianten in einem zweiten Durchlauf deaktiviert. Bei einer jetzt möglicherweise vorhandenen Lösung werden die deaktivierten Invarianten auf Erfüllbarkeit geprüft und ggf. wieder aktiviert.

- *kodkod –scrolling*

  Parameter:

  - Pfad einer Konfigurationsdatei
  - next -> zur nächsten Lösung blättern
  - previous -> zur vorherigen Lösung blättern
  - show(Lösung) -> eine bestimmte Lösung wählen

  Bedeutung:

  Ermöglicht das Blättern in den Lösungen. Für den ersten Aufruf muss eine Konfigurationsdatei angegeben werden. Vor der Konfiguration wird ein möglicherweise vorhandenes Objektdiagramm ggf. extrahiert und als Teillösung

interpretiert. Bei einer gefundenen Lösung kann über den Parameter „next“ zu einer weiteren Lösung geblättert werden, die sich zumindest minimal von allen vorherigen Lösungen unterscheidet. Eine einmal gefundene Lösung wird dabei für weitere Validierungsvorgänge verboten. Aus diesem Grund ist der Suchraum irgendwann erschöpft und es werden keine weiteren Lösungen gefunden. Der Parameter „previous“ ermöglicht das Zurückblättern zu einer vorherigen Lösung. Mit dem Parameter „show“ kann zu einer bestimmten Lösung gesprungen werden.

- *kodkod –scrollingAll*

  Parameter:

  - Pfad einer Konfigurationsdatei
  - next -> zur nächsten Lösung blättern
  - previous -> zur vorherigen Lösung blättern
  - show(Lösung) -> eine bestimmte Lösung wählen

  Bedeutung:

  Wie kodkod –scrolling, jedoch mit dem Unterschied, dass bei erstmaligem Aufruf mit einer Konfigurations-datei alle möglichen Lösungen gesucht werden.

- *kodkod –invIndep*

  Parameter:

  - (optional) Angabe einer Invariante in der Syntax: *Klasse-Invariante*

  Bedeutung:

  Prüft die Unabhängigkeit einer/aller Invarianten, in dem jeweils die zu prüfende Invariante negiert wird. Ohne konkrete Angabe einer Invariante werden alle Invarianten des Modells nacheinander geprüft.

- *kodkod –config*

  Parameter:

  - satsolver := SatFactory -> Setzen des SatSolvers
  - bitwidth := Bitbreite (1 <= x <= 32) -> Setzen der Bitbreite
  - automaticDiagramExtraction := on/off (Abkürzung: ade) -> Aktivieren/Deaktivieren der Funktionalität zum Extrahieren eines vorhandenen Objektdiagramms
  - save -> Speichern der Einstellungen

  Verschiedene Kombinationen möglich. Parametertrennung über ein Semikolon (;).
  Die Namen bei den Name-Wert Paaren können jeweils auch großgeschrieben werden.

  Bedeutung:

  Ändern des SatSolvers und der Bitbreite sowie die Möglichkeit zum Speichern der Einstellungen.

  Kodkod unterstützt die folgenden SatSolver:

  - DefaultSAT4J
  - LightSAT4J
  - MiniSat
  - MiniSatProver

- ZChaffMincost
- CryptoMiniSat
- Lingeling

Default- und LightSAT4J können immer genutzt werden, wobei DefaultSAT4J standardmäßig genutzt wird. Momentan werden noch MiniSat und MiniSatProver in einer 32 Bit Version mitgeliefert. Sie können also nur verwendet werden wenn die JVM ebenfalls mit 32 Bit arbeitet.

- *kodkod ?*

  Parameter:

  - enable -> Aktivieren der Abfragefunktionalität
  - disable -> Dektivieren der Abfragefunktionalität
  - enabled -> Status der Abfragefunktionalität
  - OCL-Ausdruck -> Abfrage

  Bedeutung:

  Dieser Befehl ermöglicht Abfragen an die relationale Lösung. Damit dies möglich ist, muss die Funktionalität vor Beginn einer Validierung eingeschaltet werden. (Aufgrund zusätzlicher Relationen kann dies die Laufzeit verlängern.) Nach dem Einschalten kann bei einer gefundenen Lösung ein OCL-Ausdruck angegeben werden, der in die relationale Logik übersetzt wird und anschließend gegen die relationale Lösung evaluiert wird. (Der OCL-Ausdruck darf nur Elemente enthalten, die in dem Objektdiagramm vorkommen, welches durch den Modellvalidierer erzeugt wurde. Manuell hinzugefügte Elemente können nicht verwendet werden, da sie in der relationalen Lösung nicht vorhanden sind.)

- *gen load*

  Parameter:

  - Pfad zu einer Datei mit zusätzlichen Invarianten

  Bedeutung:

  Nachladen der Invarianten aus der Datei für den Generator. Der Modellvalidierer kann die zusätzlich geladenen Invarianten ebenfalls verwenden.

- *gen unload*

  Parameter:

  - (optional) Angabe einer Invariante in der Syntax: *Klasse::Invariante*

  Bedeutung:

  Entladen aller zusätzlich geladenen Invarianten. Bei Angabe einer Invariante wird nur diese entladen.

Konfiguration über eine Property-Datei.

Folgende Dinge lassen sich für die einzelnen Elemente konfigurieren.

Basistypen

- *String = Set{'ada'}*
  Konkrete Werte des Typs String, die in einer Lösung vorhanden sein müssen.
- *String_min = 1*
  Minimale Anzahl an vorhandenen String-Werten. Konkrete Werte haben Vorrang.
- *String_max = 2*
  Maximale Anzahl an vorhandenen String-Werten. Konkrete Werte haben Vorrang.

- *Integer = Set{1,2}*
  Konkrete Werte des Typs Integer, die in einer Lösung vorhanden sein müssen.
- *Integer_min = -10*
  Untere Grenze des Intervalls für die Integer-Werte.
- *Integer_max = 10*
  Oberer Grenze des Intervalls für die Integer-Werte.

- *Real = Set{15.5}*
  Konkrete Werte des Typs Real, die in einer Lösung vorhanden sein müssen.
- *Real_min = -2*
  Untere Grenze des Intervalls für die Real-Werte.
- *Real_max = 2*
  Oberer Grenze des Intervalls für die Real-Werte.
- *Real_step = 0.5*
  Schrittweite der Werte in dem Intervall für die Real-Werte.

Klassen

- *Klassenname = Set{ada,bob}*
  Konkrete Objekte der Klasse, die in einer Lösung vorhanden sein müssen.
- *Klassenname_min = 2*
  Minimale Anzahl an vorhandenen Objekten der Klasse. Konkrete Werte haben Vorrang.
- *Klassenname_max = 4*
  Maximale Anzahl an vorhandenen Objekten der Klasse. Konkrete Werte haben Vorrang.

Attribute

- *Klassenname_Attributname = Set{'Landstr','Allee'}*
  Wertebereich für das Attribut der Klasse.
- *Klassenname_Attributname_min = 1*
  Minimale Anzahl an definierten Werten für das Attribut über alle Objekte der Klasse. Wert -1 fordert definierte Atributwerte für alle Objekt der Klasse (hat Vorrang vor Klassenname_Attributname_max = -1).

- *Klassenname_Attributname_max = 3*
  Maximale Anzahl an definierten Werten für das Attribut über alle Objekte der Klasse. Wert -1 schränkt maximale Anzahl nicht ein (default).
- *Klassenname_Attributname_minSize = 1*
  Minimale Anzahl an Elementen bei einem Attribut mit einem Mengentypen.
- *Klassenname_Attributname_maxSize = 3*
  Minimale Anzahl an Elementen bei einem Attribut mit einem Mengentypen
  Wert -1 schränkt maximale Anzahl nicht ein (default)

Assoziationen

- *Assoziationsname = Set{(ada,bob),(cyd,dan)}*
  Konkrete Links der Assoziation, die in einer Lösung vorhanden sein müssen.
- *Assoziationsname_min = 2*
  Minimale Anzahl an vorhandenen Links der Assoziation. Konkrete Links haben Vorrang.
- *Assoziationsname_max = 6*
  Maximale Anzahl an vorhandenen Links der Assoziation. Konkrete Links haben Vorrang. Wert -1 schränkt maximale Anzahl nicht ein (default).

Assoziationsklassen

- *Assoziationsklassenname_ac = Set{b1}*
  Konkrete Objekte der Assoziationsklasse, die in einer Lösung vorhanden sein müssen.

- *Assoziationsklassenname_Attributname = Set{}*
  siehe Attribute
- *Assoziationsklassenname_Attributname_min = 1*
  siehe Attribute
- *Assoziationsklassenname_Attributname_max = 3*
  siehe Attribute

- Assoziationsklassenname = Set{(b1,bsag,hvv)}
  Konkrete Links der Assoziationsklasse, die in einer Lösung vorhanden sein müssen.
- *Assoziationsklassenname_min = 1*
  Minimale Anzahl an vorhandenen Links der Assoziation. Konkrete Links haben Vorrang.
- *Assoziationsklassenname_max = 1*
  Maximale Anzahl an vorhandenen Links der Assoziation. Konkrete Links haben Vorrang. Wert -1 schränkt maximale Anzahl nicht ein (default).

Invarianten

- *Klassenname_Invariantenname = active/inactive/negate*
  Aktivieren, Deaktivieren und Negieren einer Invariante.

Sonstiges

- *aggregationcyclefreeness = on/off*
  Definiert, ob eine Lösung Zyklen enthalten darf. Bei on sind Zyklen verboten.

- forbiddensharing = *on/off*

  Definiert, ob sharing erlaubt ist, also ob ein Objekt bei einer Komposition zu mehr als einem „Ganzem“ gehört. Bei on ist dies verboten. (Default = on)