

Lamb.da - Das Spiel

Entwurfsdokument

Farid El-Haddad, Florian Fervers, Kai Fieger,
Robert Hochweiß, Kay Schmitteckert

3. Januar 2015



Inhaltsverzeichnis

1	Einleitung	3
2	Grobentwurf	4
3	Feinentwurf	5
3.1	package lambda	5
3.1.1	public class Observable <Observer>	5
3.2	package lambda.model.lambdaterm	6
3.2.1	public abstract class LambdaTerm implements Observable<LambdaTerm>	8
3.2.2	public interface LambdaTermObserver	8
3.2.3	public class LambdaApplication extends LambdaTerm	8
3.2.4	public abstract class LambdaValue extends LambdaTerm	10
3.2.5	public class LambdaAbstraction extends LambdaValue	11
3.2.6	public class LambdaVariable extends LambdaValue	12
3.3	package lambda.model.lambdaterm.visitor	13
3.3.1	public interface LambdaTermVisitor <R>	13
3.3.2	public class AlphaConversionVisitor implements LambdaTermVisitor<Integer>	13
3.3.3	public class ColorCollectionVisitor implements LambdaTermVisitor<String>	13
3.3.4	public class IsColorBoundVisitor implements LambdaTermVisitor<Boolean>	13
3.3.5	public class ApplicationVisitor implements LambdaTermVisitor<LambdaTerm>	13
3.3.6	public class CopyVisitor implements LambdaTermVisitor<LambdaTerm>	13
3.3.7	public class RemoveTermVisitor implements LambdaTermVisitor<Object>	13
3.3.8	public abstract class BetaReductionVisitor implements LambdaTermVisitor<LambdaTerm>	22
4	Datenstrukturen	25
5	Dynamische Modelle	26
6	Projektplan	27
7	Glossar	28
8	Anhang	29

1 Einleitung

2 Grobentwurf

3 Feinentwurf

3.1 package lambda

3.1.1 public class **Observable**<Observer>

Beschreibung

Repräsentiert ein Objekt, das von Beobachtern überwacht werden kann. Dabei informiert das Objekt alle Beobachter, sobald Änderungen an ihm vorgenommen werden.

Typ-Parameter

- <Observer>
Der Typ eines Beobachters.

Attribute

- private List<Observer> **observers**
Die Liste der Beobachter dieses Objektes.

Konstruktoren

- public **Observable**()
Instanziert ein Objekt dieser Klasse.

Methoden

- public void **addObserver**(Observer o)
Fügt den gegebenen Beobachter diesem Objekt hinzu, sodass dieser bei Änderungen informiert wird.

Parameter

- Observer o
Der neue Beobachter.

Exceptions

- NullPointerException
Falls o == null ist.

- public void **removeObserver**(Observer o)
Entfernt den Beobachter aus der Liste, falls dieser darin existiert, sodass dieser nicht mehr bei Änderungen informiert wird.

Parameter

- Observer o
Der zu entfernende Beobachter.

Exceptions

- NullPointerException
Falls o == null ist.

- public void **notify**(Consumer<Observer> notifier)
Ruft die gegebene Funktion auf allen Beobachtern auf. Wird benutzt, um Beobachter über Änderungen am Objekt zu informieren.

Parameter

- Consumer<Observer> notifier
Die Funktion, die auf allen Beobachtern ausgeführt wird.

3.2 package **lambda.model.lambdaterm**

3.2.1 public abstract class **LambdaTerm** implements
Observable<LambdaTermObserver>

Beschreibung

Repräsentiert einen Term im Lambda-Kalkül bzw. ein Knoten in der Baumstruktur eines Lambda-Terms.

Attribute

- private LambdaTerm **parent**
Der Elternknoten dieses Terms. Kann auch null sein, falls der Knoten eine Wurzel ist.

Konstruktoren

- public **LambdaTerm**(LambdaTerm parent)
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten.

Parameter

- LambdaTerm parent
Der Elternknoten dieses Terms. Kann auch null sein, falls der Knoten eine Wurzel ist.

Methoden

- `public abstract <T> T accept(LambdaTermVisitor<T> visitor)`
Nimmt den gegebenen Besucher entgegen und ruft dessen `visit`-Methode auf. Die Rückgabe des Besuchers wird auch von dieser Methode zurückgegeben.

Typ-Parameter

– `<T>`

Der Typ des Rückgabewertes des Besuchers. Wird benötigt, um verschiedene Rückgabewerte von verschiedenen Besucherklassen zu ermöglichen.

Parameter

– `LambdaTermVisitor<T> visitor`

Der Besucher, der entgegen genommen wird.

Rückgabe

– Gibt den Rückgabewert des Besuchers zurück.

Exceptions

– `NullPointerException`

Falls `visitor == null` ist.

- `public boolean isRoot()`
Gibt zurück, ob dieser Knoten die Wurzel ist. Ein Knoten ist eine Wurzel, wenn sein Elternknoten `null` ist.

Rückgabe

– Gibt zurück, ob dieser Knoten die Wurzel ist.

- `public boolean isValue()`
Gibt zurück, ob dieser Term ein Wert - d.h. eine Abstraktion oder Variable - ist. Gibt in der Standard-Implementierung `false` zurück und wird von entsprechenden Unterklassen überschrieben.

Rückgabe

– Gibt zurück, ob dieser Term ein Wert ist.

- `public LambdaTerm getParent()`
Gibt den Elternknoten dieses Knotens wieder oder `null`, falls dieser Knoten eine Wurzel ist.

Rückgabe

- Der Elternknoten dieses Knotens.

3.2.2 public interface **LambdaTermObserver**

Beschreibung

Repräsentiert einen Beobachter eines Lambda-Terms, welcher über Änderungen am Term informiert wird.

Methoden

- public void **replaceTerm**(LambdaTerm old, LambdaTerm new)
Wird aufgerufen um dem Beobachter mitzuteilen, dass der gegebene alte Term durch den gegebenen neuen ersetzt wird. Einer von beiden Parametern kann null sein, niemals aber beide.

Parameter

- LambdaTerm old
Der ersetzte Term.

- LambdaTerm new
Der neue Term.

- public void **setColor**(LambdaValue term, Color color)
Wird aufgerufen um dem Beobachter mitzuteilen, dass die Farbe des gegebenen Terms durch die gegebene neue Farbe ersetzt wird.

Parameter

- LambdaValue term
Der veränderte Term.

- Color color
Die neue Farbe des Terms.

3.2.3 public class **LambdaApplication** extends LambdaTerm

Beschreibung

Repräsentiert eine Applikation im Lambda-Kalkül.

Attribute

- `private LambdaTerm first`
Linker bzw. erster Kindknoten der Applikation.
- `private LambdaTerm second`
Rechter bzw. zweiter Kindknoten der Applikation.

Konstruktoren

- `public LambdaApplication(LambdaTerm parent)`
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten.

Parameter

- `LambdaTerm parent`
Der Elternknoten dieses Terms. `null` ist erlaubt, resultiert aber in einem ungültigen Lambda-Term.

Methoden

- `public <T> T accept(LambdaTermVisitor<T> visitor)`
Siehe `LambdaTerm.accept`
- `public void setFirst(LambdaTerm first)`
Setzt den linken bzw. ersten Kindknoten dieser Applikation und informiert alle Beobachter über diese Änderung.

Parameter

- `LambdaTerm first`
Der neue linke Kindknoten. `null` ist erlaubt, resultiert aber in einem ungültigen Lambda-Term.

- `public LambdaTerm getFirst()`
Gibt den linken bzw. ersten Kindknoten dieser Applikation zurück.

Rückgabe

- Der linke Kindknoten dieser Applikation.

- `public void setSecond(LambdaTerm second)`
Setzt den rechten bzw. zweiten Kindknoten dieser Applikation und informiert alle Beobachter über diese Änderung.

Parameter

- `LambdaTerm second`
Der neue rechte Kindknoten. `null` ist erlaubt, resultiert aber in einem ungültigen Lambda-Term.

- `public LambdaTerm getSecond()`
Gibt den rechten bzw. zweiten Kindknoten dieser Applikation zurück.

Rückgabe

- Der rechte Kindknoten dieser Applikation.

3.2.4 `public abstract class LambdaValue extends LambdaTerm`

Beschreibung

Repräsentiert einen Wert - d.h. Abstraktion oder Variable - im Lambda-Kalkül.

Attribute

- `private Color color`
Die Farbe dieses Wertes, äquivalent zum Variablennamen.

Konstruktoren

- `public LambdaValue(LambdaTerm parent, Color color)`
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten und der gegebenen Farbe.

Parameter

- `LambdaTerm parent`
Der Elternknoten dieses Terms. `null` ist erlaubt, falls der Term eine Wurzel ist.
- `Color color`
Die Farbe dieses Wertes.

Exceptions

- `NullPointerException`
Falls `color == null` ist.

Methoden

- `public boolean isValue()`
Gibt zurück, ob dieser Term ein Wert ist. Überschreibt die Funktion in `LambdaTerm` und gibt hier immer `true` zurück.

Rückgabe

- Gibt zurück, ob dieser Term ein Wert ist.

- `public void setColor(Color color)`
Setzt die Farbe dieses Wertes und informiert alle Beobachter über diese Änderung.

Parameter

- `Color color`
Die neue Farbe.

Exceptions

- `NullPointerException`
Falls `color == null` ist.
- `public Color getColor()`
Gibt die Farbe dieses Wertes zurück.

Rückgabe

- Die Farbe dieses Wertes.

3.2.5 `public class LambdaAbstraction extends LambdaValue`

Beschreibung

Repräsentiert eine Abstraktion im Lambda-Kalkül.

Attribute

- `private LambdaTerm inside`
Der Term innerhalb der Applikation. Kann `null` sein, resultiert aber in einem ungültigen Term.

Konstruktoren

- `public LambdaAbstraction(LambdaTerm parent, Color color)`
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten und der gegebenen Farbe.

Parameter

- `LambdaTerm parent`
Der Elternknoten dieses Terms. Kann `null` sein, falls der Term eine Wurzel ist.
- `Color color`
Die Farbe der in dieser Abstraktion gebundenen Variable.

Exceptions

- `NullPointerException`
Falls `color == null` ist.

Methoden

- `public <T> T accept(LambdaTermVisitor<T> visitor)`
Siehe `LambdaTerm.accept`
- `public void setInside(LambdaTerm inside)`
Setzt den Term innerhalb der Abstraktion und informiert alle Beobachter über diese Änderung.

Parameter

- `LambdaTerm inside`
Der neue innere Term. Kann `null` sein, resultiert aber in einem ungültigen Term.
- `public LambdaTerm getInside()`
Gibt den Term innerhalb der Abstraktion zurück.

Rückgabe

- Der innere Term.

3.2.6 `public class LambdaVariable extends LambdaValue`

Beschreibung

Repräsentiert eine Variable im Lambda-Kalkül.

Konstruktoren

- `public LambdaVariable(LambdaTerm parent, Color color)`
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten und der gegebenen Farbe.

Parameter

- `LambdaTerm parent`
Der Elternknoten dieses Terms. Kann null sein, falls der Term eine Wurzel ist.
- `Color color`
Die Farbe der Variable.

Exceptions

- `NullPointerException`
Falls `color == null` ist.

Methoden

- `public <T> T accept(LambdaTermVisitor<T> visitor)`
Siehe `LambdaTerm.accept`

3.3 package `lambda.model.lambdaterm.visitor`

3.3.1 public interface `LambdaTermVisitor<R>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur. Der Besucher kann Operationen an der Datenstruktur ausführen und hat optional einen Rückgabewert.

Typ-Parameter

- `<R>`
Der Typ des Rückgabewertes.

Methoden

- `public void visit(LambdaApplication node)`
Besucht die gegebene Applikation.

Parameter

- `LambdaApplication node`
Die besuchte Applikation. Ist nie null.
- `public void visit(LambdaAbstraction node)`
Besucht die gegebene Abstraktion.

Parameter

- `LambdaAbstraction node`
Die besuchte Abstraktion. Ist nie null.

- `public void visit(LambdaVariable node)`
Besucht die gegebene Variable.

Parameter

- `LambdaVariable node`
Die besuchte Variable. Ist nie null.

- `public R getResult()`
Gibt das Resultat der Besucheroperation zurück. Wird nur nach einem Besuch ausgeführt. Gibt in der Standard-Implementierung null zurück.

Rückgabe

- Das Resultat der Besucheroperation.

3.3.2 `public class AlphaConversionVisitor implements
LambdaTermVisitor<LambdaTerm>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher eine Alpha-Konversion auf ihr ausführt.

Attribute

- `private LambdaTerm result`
Der Term nach der Alpha-Konversion.

Konstruktoren

- `public AlphaConversionVisitor(Color old, Color new)`
Instanziert ein Objekt dieser Klasse mit der gegebenen ersetzten und ersetzenden Farbe.

Parameter

- `Color old`
Die zu ersetzende Farbe.

- `Color new`
Die neue Farbe.

Methoden

- `public void visit(LambdaApplication node)`
Besucht die gegebene Applikation und traversiert weiter zu beiden Kindknoten.

Parameter

- `LambdaApplication node`
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`
Besucht die gegebene Abstraktion. Dabei wird die Farbe wenn nötig ersetzt und weiter zum Kindknoten traversiert.

Parameter

- `LambdaAbstraction node`
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`
Besucht die gegebene Variable und ersetzt die Farbe wenn nötig.

Parameter

- `LambdaVariable node`
Die besuchte Variable.

- `public LambdaTerm getResult()`
Gibt den Term zurück, der besucht wurde.

Rückgabe

- Der besuchte Term.

3.3.3 `public class ColorCollectionVisitor implements
LambdaTermVisitor<Set<Color>>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der die Menge der benutzten Farben in diesem Term zurückgibt.

Attribute

- `private Set<Color> result`
Die Menge aller benutzten Farben.

Konstrukturen

- `public ColorCollectionVisitor()`
Instanziert ein Objekt dieser Klasse.

Methoden

- `public void visit(LambdaApplication node)`
Besucht die gegebene Applikation und traversiert weiter zu beiden Kindknoten.

Parameter

- `LambdaApplication node`
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`
Besucht die gegebene Abstraktion. Dabei wird die Farbe zur Menge hinzugefügt und weiter zum Kindknoten traversiert.

Parameter

- `LambdaAbstraction node`
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`
Besucht die gegebene Variable und fügt die Farbe zur Menge hinzu.

Parameter

- `LambdaVariable node`
Die besuchte Variable.

- `public Set<Color> getResult()`
Gibt die Menge der Farben zurück, die in dem besuchten Term benutzt werden.

Rückgabe

- Die Menge der benutzten Farben.

3.3.4 `public class IsColorBoundVisitor implements LambdaTermVisitor<Boolean>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der zurückgibt, ob eine Variable mit der gegebenen Farbe in diesem Term gebunden ist.

Attribute

- `private Color color`
Die zu überprüfende Farbe.
- `private boolean result`
Der Rückgabewert des Besuchs.

Konstruktoren

- `public IsColorBoundVisitor(Color color)`
Instanziert ein Objekt dieser Klasse mit der zu überprüfenden Farbe.

Parameter

- `Color color`
Die zu überprüfende Farbe.

Exceptions

- `NullPointerException`
Falls `color == null` ist.

Methoden

- `public void visit(LambdaApplication node)`
Besucht die gegebene Applikation und traversiert wenn möglich weiter zum Elternknoten.

Parameter

- `LambdaApplication node`
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`
Besucht die gegebene Abstraktion und überprüft, ob die Farbe hier gebunden ist. Traversiert wenn nötig und möglich weiter zum Elternknoten.

Parameter

- `LambdaAbstraction node`
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`
Besucht die gegebene Variable und traversiert weiter zum Elternknoten.

Parameter

- `LambdaVariable` node
Die besuchte Variable.

- `public Boolean getResult()`
Gibt zurück, ob die Variable mit der gegebenen Farbe im Term gebunden ist.

Rückgabe

- Gibt zurück, ob die Variable mit der gegebenen Farbe gebunden ist.

3.3.5 `public class ApplicationVisitor implements
LambdaTermVisitor<LambdaTerm>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher eine Applikation ausführt.

Attribute

- `private Color color`
Die Farbe der zu ersetzenden Variablen.
- `private LambdaTerm applicant`
Das Argument der Applikation.
- `private LambdaTerm result`
Der Term nach der Applikation.
- `private boolean hasCheckedAlphaConversion`
Initialisiert mit `false`. Speichert, ob bereits überprüft wurde, ob eine Alpha-Konversion vor der Applikation notwendig ist.

Konstruktoren

- `public ApplicationVisitor(Color color, LambdaTerm applicant)`
Instanziert ein Objekt dieser Klasse mit der gegebenen Variablenfarbe und dem gegebenen Argument.

Parameter

- `Color color`
Die Farbe der zu ersetzenden Variablen.

- LambdaTerm applicant
Das Argument der Applikation.

Exceptions

- NullPointerException
Falls color == null oder applicant == null ist.

Methoden

- public void **visit**(LambdaApplication node)
Besucht die gegebene Applikation und traversiert weiter zu beiden Kindknoten. Dabei werden die Kindknoten auf die Rückgabewerte beider Besuche gesetzt. Speichert als Rückgabewert den besuchten Term.

Parameter

- LambdaApplication node
Die besuchte Applikation.

- public void **visit**(LambdaAbstraction node)
Besucht die gegebene Abstraktion und traversiert weiter zum Kindknoten. Dabei wird der Kindknoten auf den Rückgabewert des Besuchs gesetzt. Speichert als Rückgabewert den besuchten Term.

Parameter

- LambdaAbstraction node
Die besuchte Abstraktion.

- public void **visit**(LambdaVariable node)
Besucht die gegebene Variable und speichert wenn nötig als Rückgabewert applicant.

Parameter

- LambdaVariable node
Die besuchte Variable.

- public LambdaTerm **getResult**()
Gibt den Term nach der Applikation zurück.

Rückgabe

- Der besuchte Term.

- private void **checkAlphaConversion**()

Überprüft, ob eine Alpha-Konversion notwendig ist, falls dies noch nicht getan wurde, und führt diese wenn nötig aus. Entfernt danach das Argument der Applikation aus dem LambdaTerm.

3.3.6 `public class CopyVisitor implements
LambdaTermVisitor<LambdaTerm>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher die Datenstruktur kopiert und die Kopie zurückgibt.

Attribute

- `private LambdaTerm result`
Die Kopie.

Konstruktoren

- `public CopyVisitor()`
Instanziert ein Objekt dieser Klasse.

Methoden

- `public void visit(LambdaApplication node)`
Besucht die gegebene Applikation und erstellt eine Kopie. Traversiert zu beiden Kindknoten und speichert die Rückgabewerte dieser Besuche in den Kindknoten der Kopie.

Parameter

- `LambdaApplication node`
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`
Besucht die gegebene Abstraktion und erstellt eine Kopie. Traversiert zum Kindknoten und speichert den Rückgabewert dieses Besuchs im Kindknoten der Kopie.

Parameter

- `LambdaAbstraction node`
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`
Besucht die gegebene Variable und speichert als Rückgabewert eine Kopie

dieser Variable.

Parameter

- `LambdaVariable node`
Die besuchte Variable.

- `public LambdaTerm getResult()`
Gibt die Kopie zurück.

Rückgabe

- Die Kopie.

3.3.7 `public class RemoveTermVisitor implements LambdaTermVisitor<Object>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher den besuchten Term aus der Datenstruktur entfernt.

Attribute

- `private LambdaTerm removed`
Der zu entfernende Term. Initialisiert mit `null`.

Konstruktoren

- `public RemoveTermVisitor()`
Instanziert ein Objekt dieser Klasse.

Methoden

- `public void visit(LambdaApplication node)`
Besucht die gegebene Applikation. Falls noch kein zu entfernender Term gespeichert ist, speichere diese Applikation und traversiere zum Elternknoten, falls dieser nicht `null` ist. Ansonsten ist der Term bereits aus der Baumstruktur entfernt. Falls ein zu entfernender Term - Kindknoten in der Applikation - gespeichert ist, ersetze diesen durch `null`.

Parameter

- `LambdaApplication node`
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`
Besucht die gegebene Abstraktion. Falls noch kein zu entfernender Term gespeichert ist, speichere diese Abstraktion und traversiere zum Elternknoten, falls dieser nicht `null` ist. Ansonsten ist der Term bereits aus der Baumstruktur entfernt. Falls ein zu entfernender Term - Kindknoten der Abstraktion - gespeichert ist, ersetze diesen durch `null`.

Parameter

- `LambdaAbstraction node`
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`
Speichere die Variable als zu entfernenden Term und traversiere zum Elternknoten, falls dieser nicht `null` ist. Ansonsten ist der Term bereits aus der Baumstruktur entfernt.

Parameter

- `LambdaVariable node`
Die besuchte Variable.

3.3.8 `public abstract class BetaReductionVisitor implements LambdaTermVisitor<LambdaTerm>`

Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der eine einzelne Beta-Reduktion gemäß einer Reduktionsstrategie durchführt. Dabei sind Strategien durch Unterklassen dieses Besuchers gegeben.

Attribute

- `protected LambdaTerm result`
Der Term nach der Beta-Reduktion.
- `protected boolean hasReduced`
Speichert, ob von diesem Besucher bereits eine Reduktion durchgeführt wurde. Initialisiert mit `false`.
- `protected LambdaTerm applicant`
Falls der Elternknoten des aktuell besuchten Knotens eine Applikation ist, speichert diese Variable das Argument der Applikation. Initialisiert mit `null`.

Konstrukturen

- `public BetaReductionVisitor()`
Instanziert ein Objekt dieser Klasse.

Methoden

- `public void visit(LambdaApplication node)`
Besucht die gegebene Applikation und ruft die `reduce`-Funktion auf, welche von der Reduktionsstrategie implementiert wird, falls noch keine Reduktion von diesem Besucher durchgeführt wurde. Ansonsten wird als Resultat der besuchte Knoten gespeichert.

Parameter

- `LambdaApplication node`
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`
Besucht die gegebene Abstraktion und ruft die `reduce`-Funktion auf, welche von der Reduktionsstrategie implementiert wird, falls noch keine Reduktion von diesem Besucher durchgeführt wurde. Ansonsten wird als Resultat der besuchte Knoten gespeichert.

Parameter

- `LambdaAbstraction node`
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`
Besucht die gegebene Variable und ruft die `reduce`-Funktion auf, welche von der Reduktionsstrategie implementiert wird, falls noch keine Reduktion von diesem Besucher durchgeführt wurde. Ansonsten wird als Resultat der besuchte Knoten gespeichert.

Parameter

- `LambdaVariable node`
Die besuchte Variable.

- `public LambdaTerm getResult()`
Gibt das Resultat der Reduktion zurück.

Rückgabe

- Der reduzierte Term.

- `public abstract void reduce(LambdaApplication node)`
Reduziert die gegebene Applikation. Implementiert von der Reduktionsstrategie.

Parameter

- `LambdaApplication node`
Die zu reduzierende Applikation.

- `public abstract void reduce(LambdaAbstraction node)`
Reduziert die gegebene Abstraktion. Implementiert von der Reduktionsstrategie.

Parameter

- `LambdaAbstraction node`
Die zu reduzierende Abstraktion.

- `public abstract void reduce(LambdaVariable node)`
Reduziert die gegebene Variable. Implementiert von der Reduktionsstrategie.

Parameter

- `LambdaVariable node`
Die zu reduzierende Variable.

4 Datenstrukturen

5 Dynamische Modelle

6 Projektplan

7 Glossar

8 Anhang