



Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

Erweiterung von SECONDO

Algebraimplementierung

Thomas Behr

Fakultät für Mathematik und Informatik
Datenbanksysteme für neue Anwendungen



FernUniversität in Hagen

11. Oktober 2014

©2014 FernUniversität in Hagen



Outline

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- 1 Nested Lists
- 2 Algebren - Vorbereitende Schritte
 - Einbindung
 - includes
- 3 Typen in SECONDO implementieren
- 4 Operatoren implementieren
 - einfache Operatoren
 - überladene Operatoren
- 5 Ströme in Secondo
 - Ströme als Argument
 - Ströme als Ergebnis
 - Strom als Argument und als Ergebnis
- 6 Funktionen als Argumente



Verschachtelte Listen

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Kommen an vielen Stellen vor
 - externe Darstellung von Objekten, z.B.

```
[const point value ( 1.0 9.7)]
```
 - Type Mapping
 - Selection Function
 - Display Function
 - ...
- Ein Listenelement ist
 - ein Atom *oder*
 - eine Liste
- atomare Elemente
 - Int (32 bit)
 - Real
 - Bool
 - String (max. 48 Zeichen)
 - Symbol (max. 48 Zeichen)
 - Text (beliebig lang)



Verwendung von Nested List

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

■ Funktionsaufrufe über globalen Listenspeicher `nl`

```
#include "NestedList.h"  
extern NestedList* nl;
```

■ Erzeugen von Atomen

```
ListExpr li = nl->IntAtom(23);  
ListExpr lr = nl->RealAtom(-8.3);  
ListExpr lsy = nl->SymbolAtom("Symbol");  
...
```

■ Erzeugen von kurzen Listen

```
ListExpr l1 = nl->OneElemList(li);  
ListExpr l2 = nl->TwoElemList(li, lr);  
...  
ListExpr l6 = nl->SixElemList(li, lr, lsy, l1, l2, li);  
bool ok = nl->ReadFromString("( a 'b' 1.0 (TRUE 6) )", ls);
```

■ Erzeugen von langen Listen

```
bool ok = nl->ReadFromFile('file.nl', list);  
  
ListExpr l1000 = nl->OneElemList(nl->IntAtom(0));  
ListExpr last = l1000;  
for(int i=1; i<1000; i++){  
    last = nl->Append(last, nl->IntAtom(i));  
}
```



Verwendung von Nested List

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

■ Typ einer Liste

```
nl->AtomType(list)
```

ergibt: NoAtom, IntType, RealType,
BoolType, StringType, SymbolType oder
TextType

■ Wert eines Atoms

```
int i = nl->IntValue(list);  
string s = nl->SymbolValue(list);  
string t = nl->Text2String(list);  
...
```

■ Länge einer Liste

```
nl->IsEmpty(list);  
nl->ListLength(list); // -1 if list is an atom  
nl->HasLength(list, 3);
```

■ Zerlegen einer Liste

```
nl->First(list); nl->Second(list); ... nl->Sixth(list);  
nl->Rest(list); // list without first element
```



Verwendung von Nested List

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

■ Vergleich von Listen

```
bool eq = nl->Equal(list1, list2);  
bool isSym = nl->IsEqual(list, "string", false);  
    // list is an symbol with value string, case insensitive
```

■ Ausgabe von Listen

```
cout << nl->ToString(list) << endl;
```

■ Nützliche Hilfsroutinen sind in ListUtils zu finden

```
#include "NestedList.h"  
#include "ListUtils.h"  
.....  
if( listutils::isNumeric(list)){  
    double d = listutils::getNumValue(list);  
}
```



Verwendung von Nested List

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- **Achtung**
NestedList reagieren hart auf Fehlbenutzung, daher vor Aufruf einer Funktion stets prüfen, ob die Liste das notwendige Format hat
- Wrapperklasse `NList`
 - objektorientiertes Interface für verschachtelte Listen
 - Exceptions statt Assertions (meist)



Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Algebren enthalten
 - Typen
 - Operatoren
- Algebren können Typen anderer Algebren verwenden
- SECONDO bietet wohldefinierte Schnittstellen zur Einbindung neuer Algebren
- Algebren werden mit dem Systemkern zusammengelinkt
- Neue Typen/Operatoren erweitern den Sprachumfang von SECONDO



Algebren definieren und einbinden

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- neues Verzeichnis im Verzeichnis `Algebras` anlegen

- neuen Eintrag in der Datei

`Algebras/Management/AlgebraList.i.cfg`
erstellen

```
ALGEBRA_INCLUDE (<Nummer>, <AlgebraName>)
```

Nummer und Name müssen eindeutig sein

- Aktivieren der Algebra in der Datei

`makefile.algebras`

```
ALGEBRA_DIRS += Verzeichnisname  
ALGEBRAS     += Algebraname
```



Algebren definieren und einbinden

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

■ Erstellen einer `makefile` Datei im neuen Algebraverzeichnis

- Kopieren der Datei `makefile` aus dem Verzeichnis `Algebras/Standard-C++`
- Abhängigkeiten zu anderen Algebren einfügen nach der Zeile `include ../../makefile.env`

```
CURRENT_ALGEBRA := AlgebraName
ALGEBRA_DEPENDENCIES := StandardAlgebra
ALGEBRA_DEPENDENCIES += XYZAlgebra
...
```

■ Erstellen der Algebradatei(en)

- Header
- Implementierung(en)
- `spec`-Datei
- `examples`-Datei



Algebradatei(en)

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- PD-Kommentare

<http://dna.fernuni-hagen.de/papers/PDSystem.pdf>

- Includes

- globale (externe) Variablen

- Typkonstruktoren

- Operatoren

- Algebradefinition

- Algebrainitialisierung



includes und externe Variablen

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
#include "Attribute.h"
#include "Algebra.h"
#include "NestedList.h"
#include "QueryProcessor.h"
#include "AlgebraManager.h"
#include "Operator.h"
#include "StandardTypes.h"
#include "NList.h"
#include "Symbols.h"
#include "ListUtils.h"
.....
extern NestedList *nl;
extern QueryProcessor *qp;
extern AlgebraManager *am;
```



Erstellung eines einfachen¹ neuen Typs

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- C++-Klasse, die den Typ darstellt
 - muss einen Konstruktor besitzen, der **nichts** tut
 - erweitert um zusätzliche Funktionen, die das spätere Leben einfacher machen
- Funktionen zur Integration in Secondo
 - Typbeschreibung, Import, Export
 - Erzeugen, Löschen, Öffnen, Speichern, Schließen
 - Duplizieren, Cast, Größe, Typprüfung
- Erstellung einer Typkonstruktor-Instanz
- Hinzufügen zur Algebra
- Zuordnen von Kinds
- Erweiterung der Benutzungsschnittstellen

¹kein Attributdatentyp



C++-Klasse für einen Secondo-Typ, Beispiel: SCircle

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
class SCircle{
public:
    SCircle() {} // Konstruktor, der nichts tut
    SCircle(const double _x, const double _y, const double _r):
        x(_x), y(_y), r(_r) {}
    ~SCircle(){}
    static const string BasicType(){ return "scircle"; }
    static const bool checkType(const ListExpr list) {
        return listutils::isSymbol(list, BasicType());
    }

    double perimeter() const{
        return 2*M_PI*r;
    }
    double getX() const{ return x; }
    double getY() const{ return y; }
    double getR() const{ return r; }
private:
    double x;
    double y;
    double r;
};
```



Typbeschreibung

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- dient der Beschreibung des Typs für den Benutzer
- Funktion, die eine Nested List liefert
- Aufbau der Liste vorgegeben

```
ListExpr SCircleProperty() {  
  return ( nl->TwoElemList (  
    nl->FourElemList (  
      nl->StringAtom("Signature"),  
      nl->StringAtom("Example Type List"),  
      nl->StringAtom("List Rep"),  
      nl->StringAtom("Example List")),  
    nl->FourElemList (  
      nl->StringAtom("-> SIMPLE"),  
      nl->StringAtom(SCircle::BasicType()),  
      nl->StringAtom("(real real real) = (x,y,r)"),  
      nl->StringAtom("(13.5 -76.0 1.0)"),  
    ));  
}
```



Import

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
Word InSCircle( const ListExpr typeInfo, const ListExpr instance,  
                const int errorPos, ListExpr& errorInfo, bool& correct );
```

- result** Typ `Word`, Objekt wird in Member `addr` (einem `void-Pointer`) gespeichert
- typeInfo** Typbeschreibung, wird für zusammengesetzte Typen (z.B. `Tuple`) benötigt
- instance** Liste mit der Objektbeschreibung (muss nicht korrekt sein)
- errorPos** Positionsangabe eines aufgetretenen Fehlers (für mengenwertige Typen)
- errorInfo** Rückgabeparameter, der einen aufgetretenen Fehler beschreibt
- correct** Rückgabeparameter, der den Erfolg des Aufrufs angibt



Import, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
Includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
Word InSCircle( const ListExpr typeInfo, const ListExpr instance,
                const int errorPos, ListExpr& errorInfo, bool& correct ){
    Word res((void*)0);
    correct = false;
    if(!nl->HasLength(instance,3)){
        return res;
    }
    if( !listutils::isNumeric(nl->First(instance))
        || !listutils::isNumeric(nl->Second(instance))
        || !listutils::isNumeric(nl->Third(instance))){
        return res;
    }
    double x = listutils::getNumValue(nl->First(instance));
    double y = listutils::getNumValue(nl->Second(instance));
    double r = listutils::getNumValue(nl->Third(instance));
    if(r<=0){
        return res;
    }
    correct = true;
    res.addr = new SCircle(x,y,r);
    return res;
}
```



- Umwandeln einer Objektinstanz in eine Liste
- wird nur mit dem richtigen Typ aufgerufen, d.h. Cast ist stets erfolgreich

```
ListExpr OutSCircle( ListExpr typeInfo, Word value ) {  
    SCircle* k = (SCircle*) value.addr;  
    return nl->ThreeElemList(  
        nl->RealAtom(k->getX()),  
        nl->RealAtom(k->getY()),  
        nl->RealAtom(k->getR()));  
}
```

typeInfo enthält den Typen als Liste

value enthält einen Zeiger auf die Instanz



Erzeugen und Löschen eines Objekts

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Erzeugen einer Objektinstanz mit einem beliebigen Wert

```
Word CreateSCircle( const ListExpr typeInfo ) {  
    Word w;  
    w.addr = (new SCircle(0,0,1.0));  
    return w;  
}
```

- Löschen eines Objekts inklusive evtl. vorhandener Teile auf der Festplatte

```
void DeleteSCircle( const ListExpr typeInfo, Word& w ) {  
    SCircle *k = (SCircle *)w.addr;  
    delete k;  
    w.addr = 0;  
}
```



Speichern eines Objekts

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren
einfache
überladene

Ströme
Argument
Ergebnis
Hybride

Funktionen

- Serialisierung des Objekts
- wird in SMI-Record geschrieben

```
bool SaveSCircle( SmiRecord& valueRecord, size_t& offset,  
                  const ListExpr typeInfo, Word& value );
```

result Typ `bool`, gibt über den Erfolg der Funktion Auskunft

valueRecord Hier wird das Objekt hineingeschrieben

offset Ab dieser Stelle im Record soll geschrieben werden

typeInfo Typ als Nested List

value das zu speichernde Objekt



Objekt speichern, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
bool SaveSCircle( SmiRecord& valueRecord, size_t& offset,  
                  const ListExpr typeInfo, Word& value ) {  
    SCircle* k = static_cast<SCircle*>( value.addr );  
    size_t size = sizeof(double);  
    double v = k->getX();  
    bool ok = valueRecord.Write( &v, size, offset );  
    offset += size;  
    v = k->getY();  
    ok = ok && valueRecord.Write(&v, size, offset);  
    offset += size;  
    v = k->getR();  
    ok = ok && valueRecord.Write(&v, size, offset);  
    offset += size;  
    return ok;  
}
```



Öffnen eines Objekts

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- holt ein Objekt (teilweise) von der Festplatte in den Speicher

```
bool OpenSCircle( SmiRecord& valueRecord,  
                  size_t& offset, const ListExpr typeInfo,  
                  Word& value );
```

result Type `bool`, gibt über den Erfolg der Funktion Auskunft

valueRecord Hier liegen die einzulesenden Daten

offset ab hier sind die Daten des Objekts zu finden

typeInfo Typ als Liste

value Rückgabeparameter



Öffnen eines Objekts

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
Includes

Typen

Operatoren

einfache
Überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
bool OpenSCircle( SmiRecord& valueRecord,
                 size_t& offset, const ListExpr typeInfo,
                 Word& value ){
    size_t size = sizeof(double);
    double x,y,r;
    bool ok = valueRecord.Read(&x,size,offset);
    offset += size;
    ok = ok && valueRecord.Read(&y,size,offset);
    offset += size;
    ok = ok && valueRecord.Read(&r, size, offset);
    offset += size;
    if(ok){
        value.addr = new SCircle(x,y,r);
    } else {
        value.addr = 0;
    }
    return ok;
}
```

Wichtig: Open- und Save-Funktion müssen symmetrisch sein



Schließen eines Objekts

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Der Hauptspeicheranteil eines Objekts wird gelöscht
- Ggf. existierende zusätzliche (SMI-)Dateien bleiben unberührt (im Gegensatz zu Delete)

```
void CloseSCircle( const ListExpr typeInfo, Word& w ) {  
    SCircle *k = (SCircle *)w.addr;  
    delete k;  
    w.addr = 0;  
}
```




Objekt Klonen

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
Word CloneSCircle( const ListExpr typeInfo, const Word& w ){
    SCircle* k = (SCircle*) w.addr;
    Word res;
    res.addr = new SCircle(k->getX(), k->getY(), k->getR());
    return res;
}
```



Cast und Size Funktionen

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Cast vom Void-Pointer zum Objekt zum Void-Pointer
- dient der Aktualisierung von persistierten Funktionspointern

```
void* CastSCircle( void* addr ) {  
    return (new (addr) SCircle);  
}
```

- Größenangabe = wieviel Platz ist zum Speichern notwendig

```
int SizeOfSCircle() {  
    return 3*sizeof(double);  
}
```



- prüft, ob eine gegebene Liste dem Typ entspricht
- **errorInfo** für genauere Fehlermeldungen
- ähnlich zur Funktion `checkType`

```
bool SCircleTypeCheck(ListExpr type, ListExpr& errorInfo){  
    return nl->IsEqual(type, SCircle::BasicType());  
}
```



Der Typkonstruktor

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
TypeConstructor SCircleTC(  
    SCircle::BasicType(),  
    SCircleProperty,  
    OutSCircle, InSCircle,  
    0, 0, // deprecated, don't think about it  
    CreateSCircle, DeleteSCircle,  
    OpenSCircle, SaveSCircle,  
    CloseSCircle, CloneSCircle,  
    CastSCircle,  
    SizeOfSCircle,  
    SCircleTypeCheck);
```



Algebradefinition und -initialisierung

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Typen werden im Algebrakonstruktor hinzugefügt
- Typen werden Kinds zugeordnet
- Algebradefinition

```
class SCircleAlgebra : public Algebra {  
public:  
    SCircleAlgebra() : Algebra() {  
        AddTypeConstructor( &SCircleTC );  
        SCircleTC.AssociateKind( Kind::SIMPLE() );  
    }  
};
```

- Algebrainitialisierung
- Name der Funktion muss Initialize<AlgebraName> sein

```
extern "C"  
Algebra*  
    InitializeSCircleAlgebra( NestedList* nlRef,  
                             QueryProcessor* qpRef ) {  
    return new SCircleAlgebra;  
}
```



Linken der Algebra

Erweiterung
von `SECONDO`

Thomas Behr

Nested Lists

Algebren

Einbindung
`includes`

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Übersetzen der Algebra
 - im Algebraverzeichnis
`make`
eintippen
- Linken der Algebra
 - im `SECONDO`-Verzeichnis
`make oder make TTY`
eintippen



Erste Tests

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- `SecondoTTYBDB` starten
- `list algebras`
`SCircleAlgebra` sollte in der Liste erscheinen
- `list Algebra SCircleAlgebra`
Typkonstruktor für `scircle` sollte auftauchen
- Öffnen einer Datenbank, dann:
 - `query [const scircle value (9.0 10.0 20.0)]`
ein entsprechendes `SCircle`-Objekt wird als Liste dargestellt
 - `query [const scircle value (a + b)]`
eine Fehlermeldung wird angezeigt
 - `let k1 = [const scircle value (9.0 10.0 20.0)]`
meldet Erfolg
 - `query k1`
zeigt das definierte Objekt an



Erweiterung von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Freuen
- Durchatmen
- Operatoren definieren



Operatoren

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Type Mapping
- Value Mapping(s)
- Value Mapping Array²
- Selection Function²
- Beschreibung
- Operatorinstanz
- Syntax ³
- Beispiel ⁴

²bei überladenen Operatoren

³spec-Datei

⁴examples-Datei



Type Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- prüft, ob die Argumenttypen vom Operator verarbeitet werden können (inklusive Überladungen)
- gibt den Ergebnistyp zurück⁵
- Argumente und Ergebnis als Nested Lists

```
ListExpr perimeterTM(ListExpr args) {  
  string err = "scircle expected";  
  if(!nl->HasLength(args,1)) {  
    return listutils::typeError(err + " (wrong number of arguments)");  
  }  
  if(!SCircle::checkType(nl->First(args))) {  
    return listutils::typeError(err);  
  }  
  return listutils::basicSymbol<CcReal>();  
}
```

⁵und ggf. weitere Infos



Value Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- nimmt Werte entgegen und berechnet daraus das Ergebnis

```
int perimeterVM (Word* args, Word& result, int message,  
                Word& local, Supplier s);
```

result Typ `int`: bei Nicht-Strom-Operatoren stets 0

args Array von `Word`, deren `addr`-Member auf die Argumente zeigen

result Rückgabe des Ergebnis, bei Nicht-Strom-Operatoren muss `result` auf den Ergebnisspeicher des Operatorknosens `s` im Operatorbaum gesetzt werden

message nur für Stromoperatoren interessant

local lokaler Speicher. Bleibt über mehrere Operatoraufrufe hinweg erhalten.
nur für Stromoperatoren interessant

s Der Operatorknoten dieses Operators



Value Mapping Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
int perimeterVM (Word* args, Word& result, int message,  
                 Word& local, Supplier s) {  
    SCircle* k = (SCircle*) args[0].addr;  
    result = qp->ResultStorage(s);  
    CcReal* res = (CcReal*) result.addr;  
    res->Set(true, k->perimeter());  
    return 0;  
}
```



■ Beschreibung des Operators für den Benutzer

```
OperatorSpec (  
    const string& signature, const string& syntax,  
    const string& meaning, const string& example,  
    const string& remark="");
```

signature Signatur des Operators als String (siehe Type Mapping)

syntax wie wird der Operator aufgerufen

meaning was berechnet der Operator

example eine Beispielanfrage

remark zusätzliche Bemerkungen



Operatorspezifikation, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
OperatorSpec perimeterSpec(  
    "scircle -> real",  
    "perimeter(_)",  
    "Computes the perimeter of an scircle.",  
    "query perimeter([const scircle value (1.0 8.0 16.0)])"  
);
```



Instanz eines Operators

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Definiert einen Operator
- verschiedene Konstruktoren für überladene und nicht-überladene Operatoren⁶

```
Operator(  
    const string& name, const string& specification,  
    ValueMapping valueMapping, SelectFunction selection,  
    TypeMapping typeMapping);  
);
```

name Name des Operators
specification Spezifikation als String
valueMapping die Value Mapping Funktion
selection für nicht überladene Operatoren:
`Operator::SimpleSelect`
typeMapping die Type Mapping Funktion

⁶hier nicht überladen



Instanz eines Operators, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
Operator perimeterOp(  
    "perimeter",  
    perimeterSpec.getStr(),  
    perimeterVM,  
    Operator::SimpleSelect,  
    perimeterTM  
);
```




Einbinden des Operators

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- innerhalb des Algebrakonstruktors `AddOperator` aufrufen

```
class SCircleAlgebra : public Algebra {  
public:  
    SCircleAlgebra() : Algebra() {  
        AddTypeConstructor( &SCircleTC );  
        SCircleTC.AssociateKind( Kind::SIMPLE() );  
  
        AddOperator(&perimeterOp);  
    }  
};
```



Syntax des Operators festlegen

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- neue Datei `AlgebraName.spec` im Algebraverzeichnis anlegen
- Syntax von **jedem** Operator sollte beschrieben werden
- wenn Operatorname in anderer Algebra bereits verwendet wird, muss die Syntax die gleiche wie dort sein
- mögliche Einträge in der Datei

```
# Kommentar
```

```
operator <opname> alias <opalias> pattern  
    <pattern> [<implicit parameters>]
```

opname Name des Operators

opalias Name des Operators als Text

pattern Syntax des Operators

implicit parameters Festlegung von bestimmten Typen
für Funktionsargumente



Mögliche Pattern

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

`op(_)`

Präfixoperator mit einem Argument

`op(_, _)`

Präfixoperator mit beliebig vielen Argumenten

`_ infixop _`

Infixoperator

`_ op`

Postfixoperator mit einem Argument

`__ op`

Postfixoperator mit zwei Argumenten

...

`_ op [<paralist>]`

Postfixoperator mit einem Argument und Parametern

`__ op [<paralist>]`

Postfixoperator mit zwei Argumenten und Parametern

...



Parameterlisten

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

<code>_, _</code>	gewöhnliche Parameter
<code>list</code>	Liste von Parametern
<code>fun</code>	Parameter, der Funktion darstellt (nur zusammen mit impliziten Parametern)
<code>funlist</code>	Liste von Funktionen
<code>X;Y</code>	getrennte Listen, X und Y sind Parameterlisten



Implizite Parameter

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- werden für einfache Benutzung benötigt
- beziehen sich auf Funktionsparameter
- Beispiel filter

```
operator filter alias FILTER pattern _ op [ fun ]  
    implicit parameter streamelem type STREAMELEM
```

automatische Extraktion des Tupeltyps mittels
Typoperatoren



Operatorsyntax, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
operator perimeter alias PERIMETER pattern op(_)
```



Example

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- neue Datei `AlgebraName.examples` im Algebraverzeichnis
- wird für automatische Tests verwendet
- **Achtung**: ohne Beispiel wird der Operator abgeschaltet

```
Database : berlintest
Restore  : No

Operator : perimeter
Number   : 1
Signature: scircle -> real
Example  : query perimeter[const scircle value (0.0 0.0 1.0)]
Result   : 6.28
Tolerance : 0.01
```



Übersetzen, Linken, Testen

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- make im Algebraverzeichnis: Übersetzen
- make im SECONDO-Verzeichnis: Linken
- im Verzeichnis `secondo/bin`

```
Selftest tmp/<AlgebraName.examples>
```

eintippen

- Speicherfehler finden durch

```
Selftest --valgrind tmp/<AlgebraName.examples>  
Selftest --valgrindlc tmp/<AlgebraName.examples>
```




Überladene Operatoren

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- haben mehrere Signaturen, d.h. mehrere Typkombinationen können verarbeitet werden
- Anzahl der Argumente darf variieren
- Ergebnistyp kann unterschiedlich sein
- Überladung über mehrere Algebren durch Queryprozessor aufgelöst
- Überladung innerhalb einer Algebra durch den Operator selbst aufgelöst



Überladene Operatoren, Unterschiede

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- TypeMapping muss jede Typkombination akzeptieren und den passenden Ergebnistyp berechnen
- es gibt für jede Typkombination ein eigenes Value Mapping (oder Value Mapping ist als Template realisiert)
- alle ValueMappings werden in einem Array zusammengefasst
- Selection-Funktion wählt richtiges TypeMapping aus
- anderer Konstruktor zur Erzeugung der Operatorinstanz



Überladener Operator Beispieloperator `distN`

Erweiterung
von `SECONDO`

Thomas Behr

Nested Lists

Algebren

Einbindung
`includes`

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- berechnet den Abstand zweier Zahlen
- als Eingabe werden zwei Integer oder zwei reelle Zahlen akzeptiert
- Ergebnis hat den gleichen Typ wie beide Eingaben
- Operator soll ein Infixoperator sein



Type Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
ListExpr distNTM(ListExpr args){
    string err = "int x int or real x real expected";
    if(!nl->HasLength(args,2)){
        return listutils::typeError(err + " (wrong number of arguments)");
    }
    if( CcInt::checkType(nl->First(args))
        && CcInt::checkType(nl->Second(args))){
        return listutils::basicSymbol<CcInt>();
    }
    if( CcReal::checkType(nl->First(args))
        && CcReal::checkType(nl->Second(args))){
        return listutils::basicSymbol<CcReal>();
    }
    return listutils::typeError(err);
}
```



ValueMapping (als Template)

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
Includes

Typen

Operatoren

einfache
Überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
template<class T>
int distNVMT( Word* args, Word& result, int message,
             Word& local, Supplier s ){
    T* a1 = (T*) args[0].addr;
    T* a2 = (T*) args[1].addr;
    result = qp->ResultStorage(s);
    T* res = (T*) result.addr;
    if(!a1->IsDefined() || !a2->IsDefined()){
        res->SetDefined(0);
        return 0;
    }
    res->Set(true, a1->GetValue() - a2->GetValue());
    if(res->GetValue() < 0){
        res->Set(true, res->GetValue() * -1);
    }
    return 0;
}
```



ValueMapping Array und Selection-Funktion

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
ValueMapping distNVM[] = {
    distNVMT<CcInt>,
    distNVMT<CcReal>
};

int distNSelect(ListExpr args){
    if ( CcInt::checkType(nl->First(args)) ){
        return 0;
    }
    if ( CcReal::checkType(nl->First(args)) ){
        return 1;
    }
    return -1;
}
```



Operator-Instanz

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
Operator distNOp(  
    "distN",  
    distNSpec.getStr(),  
    2, // number of Value Mappings  
    distNVM, // value mapping array  
    distNSelect, // selection function  
    distNTM  
);
```



Operatorsyntax und example

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

■ Festlegen der Syntax

```
operator distN alias DISTN pattern _ infixop _
```

■ Empfohlen: für jede Typkombination 1 Beispiel

```
Operator : distN  
Number  : 1  
Signature: int x int -> int  
Example  : query 1 distN 3  
Result   : 2
```

```
Operator : distN  
Number   : 2  
Signature: real x real -> real  
Example  : query 3.0 distN 1.0  
Result   : 2.0
```




Ströme in Secondo

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Vermeiden der Materialisierung von großen Mengen, z.B. Relationen
- Elemente (z.B. Tupel einer Relation) werden einzeln durch die Operatoren gezogen
- Ströme können Argument oder Ausgabe von Operatoren sein
- Ströme können nicht das Ergebnis einer Anfrage sein
- Strom als Argument: Elemente müssen explizit angefordert werden
- Strom als Ergebnis: Einzelne Elemente werden im Value Mapping angefordert



Ströme in Secondo

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

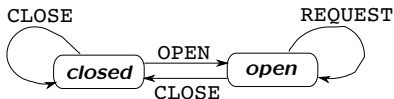
Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen



- Stromoperatoren bekommen Nachrichten (OPEN, REQUEST, CLOSE)
- Antwort auf Nachricht REQUEST: YIELD oder CANCEL
- Nützlicher Wrapper: Klasse `Stream` in `Stream.h`
- Unterschied zu normalen Operatoren: Value Mapping



Ströme als Argument, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument

Ergebnis
Hybride

Funktionen

- Operator `countNumber`
- zählt die Anzahl einer geg. Zahl in einem Integer-Strom
- Postfix-Operator mit Parameter



Type Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
# include "Stream.h"
...
ListExpr countNumberTM(ListExpr args) {
    if(!nl->HasLength(args,2)) {
        return listutils::typeError("wrong number of arguments");
    }
    if( !Stream<CcInt>::checkType(nl->First(args))
        || !CcInt::checkType(nl->Second(args)) ) {
        return listutils::typeError("stream(int) x int expected");
    }
    return listutils::basicSymbol<CcInt>();
}
```



ValueMapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument

Ergebnis
Hybride

Funktionen

```
int countNumberVM( Word* args, Word& result, int message,
                  Word& local, Supplier s ){
    result = qp->ResultStorage(s);
    Stream<CcInt> stream(args[0]);
    CcInt* num = (CcInt*) args[1].addr;
    int count = 0;
    stream.open();
    CcInt* elem;
    while( (elem = stream.request()) ){
        if(num->Compare(elem) == 0){
            count++;
        }
        elem->DeleteIfAllowed();
    }
    CcInt* res = (CcInt*) result.addr;
    res->Set(true, count);
    stream.close();
    return 0;
}
```



Operator Syntax

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument

Ergebnis
Hybride

Funktionen

```
operator countNumber alias countNumber pattern _ op[_]
```



Ströme als Ergebnis

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument

Ergebnis

Hybride

Funktionen

- ValueMapping speichert lokalen Zustand in Variable `local`
- ValueMapping verwendet nicht den Ergebnisspeicher von `s`
- ValueMapping reagiert auf Nachrichten
 - OPEN** Initialisierung, Rückgabe 0
 - REQUEST** nächstes Ergebnis wird berechnet und zurückgegeben (**YIELD**)
kein weiteres Ergebnis (**CANCEL**)
 - CLOSE** lokale Strukturen werden zerstört, Rückgabe 0



Stromoperator, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument

Ergebnis

Hybride

Funktionen

- Operator `getChars`, der die einzelnen Buchstaben eines Strings zurückliefert
- Ergebnis ist ein Strom von Strings
- TypeMapping

```
ListExpr getCharsTM(ListExpr args) {  
  if(!nl->HasLength(args,1)) {  
    return listutils::typeError("wrong number of arguments");  
  }  
  if(!CcString::checkType(nl->First(args))){  
    return listutils::typeError("string expected");  
  }  
  return nl->TwoElemList( listutils::basicSymbol<Stream<CcString > >(),  
                          listutils::basicSymbol<CcString>());  
}
```




LocalInfo-Klasse

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
Includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
class getCharsLI{
public:
    getCharsLI(CcString* arg) : input(""), pos(0){
        if(arg->IsDefined()){
            input = arg->GetValue();
        }
    }

    ~getCharsLI(){}

    CcString* getNext(){
        if(pos >= input.length()){
            return 0;
        }
        CcString* res = new CcString(true, input.substr(pos,1));
        pos++;
        return res;
    }

private:
    string input;
    size_t pos;
};
```



Value Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
Includes

Typen

Operatoren

einfache
Überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
int getCharsVM( Word* args, Word& result, int message,
               Word& local, Supplier s ){
    getCharsLI* li = (getCharsLI*) local.addr;
    switch(message){
        case OPEN : if(li) {
                        delete li;
                    }
                    local.addr = new getCharsLI( (CcString*) args[0].addr);
                    return 0;
        case REQUEST: result.addr = li?li->getNext():0;
                    return result.addr?YIELD:CANCEL;
        case CLOSE:  if(li){
                        delete li;
                        local.addr = 0;
                    }
                    return 0;
    }
    return 0;
}
```



Ströme als Eingabe und als Ergebnis, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Operator `startsWithS`
- erhält einen Strom von Strings und einen String
- lässt alle Strings durch, die mit dem zweiten Argument anfangen



Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
Includes

Typen

Operatoren

einfache
Überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
class startsWithSLI{
public:
    startsWithSLI(Word s, CcString* st): stream(s), start(""){
        def = st->IsDefined();
        if(def){ start = st->GetValue(); }
        stream.open();
    }

    ~startsWithSLI(){
        stream.close();
    }

    CcString* getNext(){
        if(!def){ return 0; }
        CcString* k;
        while( (k = stream.request()) ){
            if(k->IsDefined() && stringutils::startsWith(k->GetValue(), start)){
                return k;
            }
            k->DeleteIfAllowed();
        }
        return 0;
    }

private:
    Stream<CcString> stream;
    string start;
    bool def;
};
```



Funktionen als Argumente

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Funktionen können als Argumente eines Operators fungieren
- Beispiel: Filter - Operator
 - erhält einen Tupel Strom als erstes Argument
 - erhält eine Funktion `Tupel -> bool` als zweites Argument
 - Tupeltypen müssen übereinstimmen

```
query plz feed filter[.PLZ < 2000] count
```

```
query plz feed  
  filter[ fun(e : STREAMELEM) attr(e,PLZ) < 2000 ]  
  count
```

```
query plz feed  
  filter[ fun(t : tuple([PLZ : int, Ort : string])) attr(t,PLZ) < 2000 ]  
  count
```



Funktionen im Type Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

werden dargestellt als

```
(map Argumenttypen Ergebnistyp)
```

z.B.

```
(map int real real)
```

für eine Funktion mit zwei Argumenten der Typen `int` und `real` und dem Rückgabety `real`



Funktionen im Value Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- haben einen Argumentvektor
- dieser muss zunächst mit den Argumenten gefüllt werden
- Auswertung der Funktion muss explizit angefordert werden
- Ergebnis darf nicht vernichtet oder in einen Strom weitergereicht werden (Ausnahme: Funktion liefert selbst einen Strom)



Funktionsargumente, Beispiel

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

- Operator `replaceElem`
- nimmt einen Strom von Attributen und wendet auf jedes Attribut eine beliebige⁷ Funktion an
- Ergebnis ist der Strom der Ergebniswerte

⁷Rückgabetypp muss ein Attribut sein



Type Mapping

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
Includes

Typen

Operatoren

einfache
Überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
ListExpr replaceElemTM(ListExpr args){
  if(!nl->HasLength(args,2)){
    return listutils::typeError("wrong number of arguments");
  }
  if(!Stream<Attribute>::checkType(nl->First(args))){
    return listutils::typeError("first argument has to be a "
      "stream of attributes");
  }
  if(!listutils::isMap<1>(nl->Second(args))){
    return listutils::typeError("second arg has to be a map with 1 argument");
  }
  ListExpr StreamElem = nl->Second(nl->First(args));
  ListExpr MapArg = nl->Second(nl->Second(args));
  if(!nl->Equal(StreamElem, MapArg)){
    return listutils::typeError("map arg not equal to stream elem");
  }
  ListExpr res = nl->Third(nl->Second(args));
  if(!listutils::isDATA(res)){
    return listutils::typeError("map result is not in kind DATA");
  }
  return nl->TwoElemList( listutils::basicSymbol<Stream<Attribute> >(),
    res);
}
```



LocalInfo-Klasse

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
class replaceElemLI{
public:
    replaceElemLI(Word st, Word f): stream(st), fun(f){
        stream.open();
        funargs = qp->Argument(f.addr);
    }

    ~replaceElemLI(){
        stream.close();
    }

    Attribute* getNext(){
        Attribute* funarg = stream.request();
        if(!funarg){ return 0; }
        (*funargs[0]) = funarg;
        Word funres;
        qp->Request(fun.addr, funres);
        funarg->DeleteIfAllowed();
        Attribute* res = (Attribute*) funres.addr;
        return res->Clone();
    }
private:
    Stream<Attribute> stream;
    Word fun;
    ArgVectorPointer funargs;
};
```



Syntaxdefinition

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

```
operator replaceElem alias REPLACEELEM pattern _ op[ fun ]  
implicit parameter streamelem type STREAMELEM
```

Beispielanfrage:

```
query intstream(1,10) replaceElem[ . + 1.75] transformstream consume
```



Das Ende ist da

Erweiterung
von SECONDO

Thomas Behr

Nested Lists

Algebren

Einbindung
includes

Typen

Operatoren

einfache
überladene

Ströme

Argument
Ergebnis
Hybride

Funktionen

Fragen ?