# Implementierung von Algebren in SECONDO

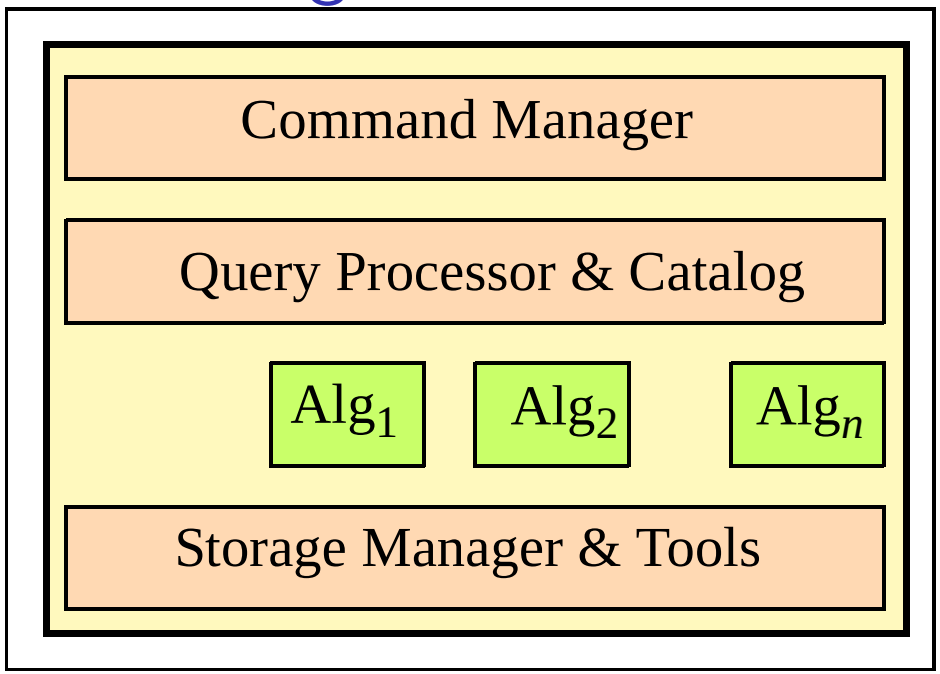
M. Spiekermann

Fakultät für Mathematik und Informatik

12.10.2007

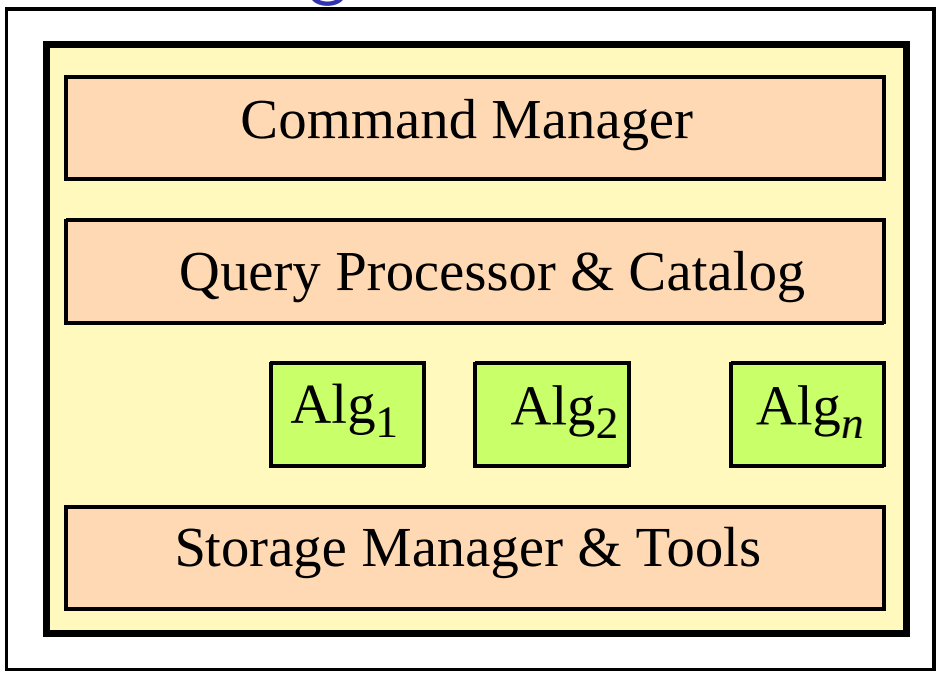
# Outline

# Funktionen eines Algebramoduls



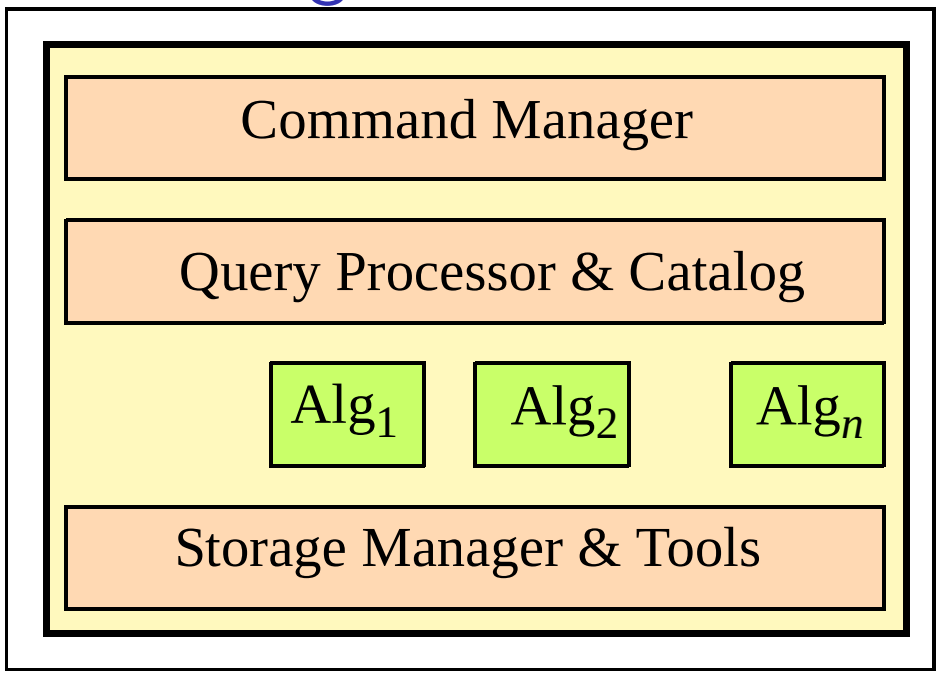

- Implementiert Typkonstruktoren (Datentypen) und Operatoren
- Definiert eine textuelle Darstellung für die Datentypen, in Form einer geschachtelten Liste
- Erlaubt die Definition einer Syntax für die Operatoren. Dies ermöglicht eine für Menschen leserliche Darstellung von Auswertungsplänen.

# Funktionen eines Algebramoduls




- Implementiert Typkonstruktoren (Datentypen) und Operatoren
- Definiert eine textuelle Darstellung für die Datentypen, in Form einer geschachtelten Liste
- Erlaubt die Definition einer Syntax für die Operatoren. Dies ermöglicht eine für Menschen leserliche Darstellung von Auswertungsplänen.

# Funktionen eines Algebramoduls




- Implementiert Typkonstruktoren (Datentypen) und Operatoren
- Definiert eine textuelle Darstellung für die Datentypen, in Form einer geschachtelten Liste
- Erlaubt die Definition einer Syntax für die Operatoren. Dies ermöglicht eine für Menschen leserliche Darstellung von Auswertungsplänen.

# Spezifikation einer Algebra

- Eine Algebra wird nach dem Konzept der Second Order Signature beschrieben.
- Die im folgenden beschriebene Algebra ist recht simpel und enthalt keine Typkonstruktoren, die selbt wieder andere Typen als Parameter haben.

# Spezifikation einer Algebra

- Eine Algebra wird nach dem Konzept der Second Order Signature beschrieben.
- Die im folgenden beschriebene Algebra ist recht simpel und enthalt keine Typkonstruktoren, die selbt wieder andere Typen als Parameter haben.

# Spezifikation einer Algebra

- Eine Algebra wird nach dem Konzept der Second Order Signature beschrieben.
- Die im folgenden beschriebene Algebra ist recht simpel und enthalt keine Typkonstruktoren, die selbt wieder andere Typen als Parameter haben.

**kinds** SIMPLE

**type constructors**

| | | | |
|---|---|---|---|
| | → | SIMPLE | *xpoint*, *xrectangle* |

**operators**

| | | | |
|---|---|---|---|
| *xpoint* × *xrectangle* | → | *bool* | inside |
| *xrectangle* × *xrectangle* | → | *bool* | inside |
| *xrectangle* × *xrectangle* | → | *bool* | intersects |

# Grundgerüst einer Implementierungsdatei

```
#include "Algebra.h"
#include "NestedList.h"
#include "Symbols.h"

extern NestedList* nl;
extern QueryProcessor *qp;

using namespace symbols;

namespace prt {

class XPoint {};

class XRectangle {};

class PointRectangleAlgebra : public Algebra
{
 public:
  PointRectangleAlgebra() : Algebra()
  {
    AddTypeConstructor( &xpointTC );
    AddTypeConstructor( &xrectangleTC );

    xpointTC.AssociateKind( SIMPLE );
    xrectangleTC.AssociateKind( SIMPLE );

    AddOperator( intersectsInfo(), intersectsFun, RectRectBool );

    ValueMapping insideFuns[] = { insideFun_PR, insideFun_RR, 0 };
    AddOperator( insideInfo(), insideFuns, insideSelect, insideTypeMap );
  }
};
```

# Die Klasse XPoint

```
class XPoint
{
 public:
  XPoint( int x, int y );
  XPoint(const XPoint& rhs);
  ~XPoint();

  int  GetX() const;
  int  GetY() const;
  void SetX( int x );
  void SetY( int y );

  XPoint* Clone();

  static Word     In( const ListExpr typeInfo, const ListExpr instance,
                      const int errorPos, ListExpr& errorInfo,
                      bool& correct );

  static ListExpr Out( ListExpr typeInfo, Word value );

  static Word     Create( const ListExpr typeInfo );

  static void     Delete( const ListExpr typeInfo, Word& w );

  static void     Close( const ListExpr typeInfo, Word& w );

  static Word     Clone( const ListExpr typeInfo, const Word& w );

  static bool     KindCheck( ListExpr type, ListExpr& errorInfo );

  static int      SizeOfObj();

  static ListExpr Property();

 private:
  inline XPoint() {}

  int x;
  int y;
};
```

# Externe/Interne Darstellung

- Beispiele für externe Darstellungen:

```
(OBJECT alexanderplatz
        ()
        point
        (11068.0 12895.0))
(OBJECT eight
        ()
        instant
        "2003-11-20-08:00")
(OBJECT intset_a10
        ()
        (array int)
        (1 2 3 4 5 6 7 8 9 10))
```

- Solche Listendarstellungen können von Secondo eingelesen werden (restore command).

# Externe/Interne Darstellung

- Beispiele für externe Darstellungen:

```
(OBJECT alexanderplatz
        ()
        point
        (11068.0 12895.0))
(OBJECT eight
        ()
        instant
        "2003-11-20-08:00")
(OBJECT intset_a10
        ()
        (array int)
        (1 2 3 4 5 6 7 8 9 10))
```

- Solche Listendarstellungen können von Secondo eingelesen werden (restore command).

# Die Programmierschnittstelle für Listen

- Der Nested List Parser(Compiler) übersetzt textuelle Listen.
- Eine globale Instanz der Klasse `NestedList` stellt diese dann zur Verfügung.
- Einzelne Listen werden durch einen Wert des Typs `ListExpr` referenziert.
- Die Klasse `NestedList` stellt eine Programmierschittstelle zur Analyse und Konstruktion von Listen bereit.

# Die Programmierschnittstelle für Listen

- Der Nested List Parser(Compiler) übersetzt textuelle Listen.
- Eine globale Instanz der Klasse `NestedList` stellt diese dann zur Verfügung.
- Einzelne Listen werden durch einen Wert des Typs `ListExpr` referenziert.
- Die Klasse `NestedList` stellt eine Programmierschittstelle zur Analyse und Konstruktion von Listen bereit.

# Die Programmierschnittstelle für Listen

- Der Nested List Parser(Compiler) übersetzt textuelle Listen.
- Eine globale Instanz der Klasse `NestedList` stellt diese dann zur Verfügung.
- Einzelne Listen werden durch einen Wert des Typs `ListExpr` referenziert.
- Die Klasse `NestedList` stellt eine Programmierschittstelle zur Analyse und Konstruktion von Listen bereit.

# Die Programmierschnittstelle für Listen

- Der Nested List Parser(Compiler) übersetzt textuelle Listen.
- Eine globale Instanz der Klasse `NestedList` stellt diese dann zur Verfügung.
- Einzelne Listen werden durch einen Wert des Typs `ListExpr` referenziert.
- Die Klasse `NestedList` stellt eine Programmierschittstelle zur Analyse und Konstruktion von Listen bereit.

# In- und Out-Funktionen

```
ListExpr
XPoint::Out( ListExpr typeInfo, Word value )
{
  XPoint* point = static_cast<XPoint*>( value.addr );

  return nl->TwoElemList(nl->IntAtom(point->GetX()),
                         nl->IntAtom(point->GetY()));
}

Word
XPoint::In( const ListExpr typeInfo, const ListExpr instance,
            const int errorPos, ListExpr& errorInfo, bool& correct )
{
  Word w = SetWord(Address(0));
  if ( nl->ListLength( instance ) == 2 )
  {
    ListExpr First = nl->First(instance);
    ListExpr Second = nl->Second(instance);

    if ( nl->IsAtom(First) && nl->AtomType(First) == IntType
      && nl->IsAtom(Second) && nl->AtomType(Second) == IntType )
    {
      correct = true;
      w.addr = new XPoint(nl->IntValue(First), nl->IntValue(Second));
      return w;
    }
  }
  correct = false;
  cmsg.inFunError("Expecting a list of two integer atoms!");
  return w;
}
```

# Persistenz

- Die interne Darstellung von Objekten (hier z.B. Klasse `XPoint`) wird vom Storage Manager in Dateien geschrieben.
- Große Objekte können nicht komplett im Hauptspeicher gehalten werden.
- Ein Objekt kann im Zustand `open` oder `closed` sein.
- Mögliche Zustandsübergänge:

# Persistenz

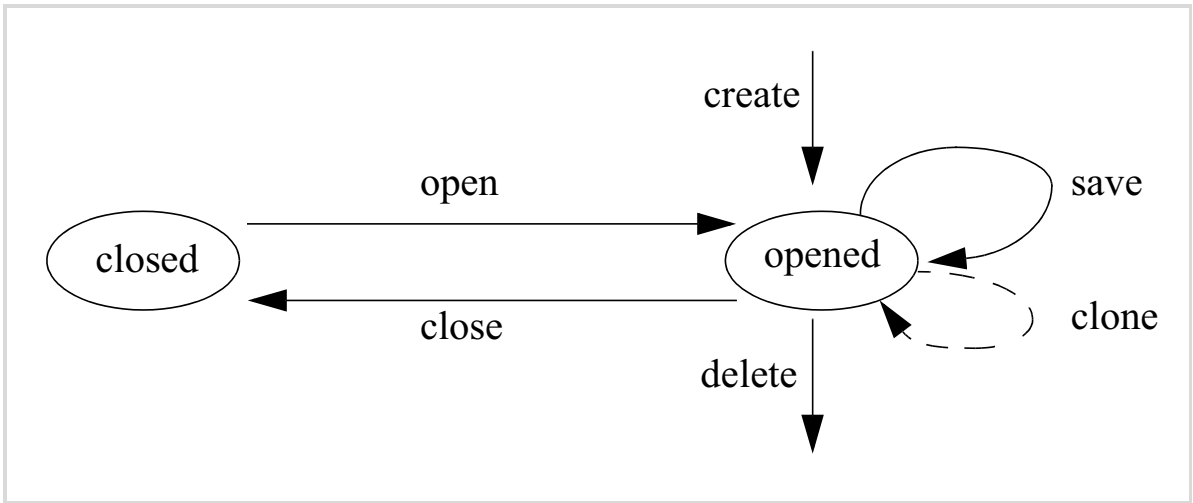
- Die interne Darstellung von Objekten (hier z.B. Klasse `XPoint`) wird vom Storage Manager in Dateien geschrieben.
- Große Objekte können nicht komplett im Hauptspeicher gehalten werden.
- Ein Objekt kann im Zustand `open` oder `closed` sein.
- Mögliche Zustandsübergänge:

# Persistenz

- Die interne Darstellung von Objekten (hier z.B. Klasse `XPoint`) wird vom Storage Manager in Dateien geschrieben.
- Große Objekte können nicht komplett im Hauptspeicher gehalten werden.
- Ein Objekt kann im Zustand `open` oder `closed` sein.
- Mögliche Zustandsübergänge:

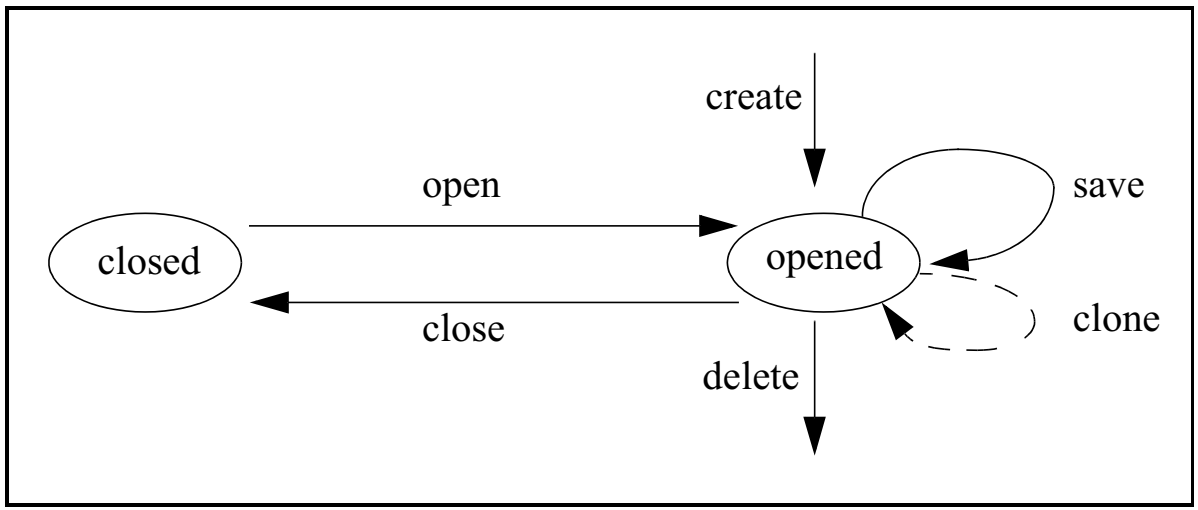
# Persistenz

- Die interne Darstellung von Objekten (hier z.B. Klasse `XPoint`) wird vom Storage Manager in Dateien geschrieben.
- Große Objekte können nicht komplett im Hauptspeicher gehalten werden.
- Ein Objekt kann im Zustand `open` oder `closed` sein.
- Mögliche Zustandsübergänge:



# Persistenz

- Die interne Darstellung von Objekten (hier z.B. Klasse `XPoint`) wird vom Storage Manager in Dateien geschrieben.
- Große Objekte können nicht komplett im Hauptspeicher gehalten werden.
- Ein Objekt kann im Zustand `open` oder `closed` sein.
- Mögliche Zustandsübergänge:



# Implementierungs-Beispiele

```
Word
XPoint::Create( const ListExpr typeInfo )
{
  return (SetWord( new XPoint( 0, 0 ) ));
}

void
XPoint::Delete( const ListExpr typeInfo, Word& w )
{
  delete static_cast<XPoint*>( w.addr );
  w.addr = 0;
}

void
XPoint::Close( const ListExpr typeInfo, Word& w )
{
  delete static_cast<XPoint*>( w.addr );
  w.addr = 0;
}

Word
XPoint::Clone( const ListExpr typeInfo, const Word& w )
{
  XPoint* p = static_cast<XPoint*>( w.addr );
  return SetWord( new XPoint(*p) );
}
```

# Kind Checking

- Jeder Typkonstruktor kann selbst auch Parameter haben.
- Die komplette Typbeschreibung, z.B. `(array int)`, ist daher ggf. zu prüfen.
- Argumentlose Typen, wie z.B. `xpoint` müssen daher nur prüfen, ob die Liste aus einem Listenatom vom Typ Symbol mit Wert `xpoint` besteht.

# Kind Checking

- Jeder Typkonstruktor kann selbst auch Parameter haben.
- Die komplette Typbeschreibung, z.B. `(array int)`, ist daher ggf. zu prüfen.
- Argumentlose Typen, wie z.B. `xpoint` müssen daher nur prüfen, ob die Liste aus einem Listenatom vom Typ Symbol mit Wert `xpoint` besteht.

# Kind Checking

- Jeder Typkonstruktor kann selbst auch Parameter haben.
- Die komplette Typbeschreibung, z.B. `(array int)`, ist daher ggf. zu prüfen.
- Argumentlose Typen, wie z.B. `xpoint` müssen daher nur prüfen, ob die Liste aus einem Listenatom vom Typ Symbol mit Wert `xpoint` besteht.

# Überblick

Jeder Operator benötigt:

- Ein Type-Mapping, welches die Typen der eingehenden Parameter prüft und ggf. den Rückgabetyp berechnet.
- Überladene Operatoren benötigen zusätzlich eine Selection-Funktion, die bestimmt welche Variante benutzt werden soll.
- Ein oder mehrere Value-Mapping Funktionen.
- Eine Beschreibung für den Benutzer.
- Eine Syntaxspezifikation und Beispiele.

# Überblick

Jeder Operator benötigt:

- Ein Type-Mapping, welches die Typen der eingehenden Parameter prüft und ggf. den Rückgabetyp berechnet.
- Überladene Operatoren benötigen zusätzlich eine Selection-Funktion, die bestimmt welche Variante benutzt werden soll.
- Ein oder mehrere Value-Mapping Funktionen.
- Eine Beschreibung für den Benutzer.
- Eine Syntaxspezifikation und Beispiele.

# Überblick

Jeder Operator benötigt:

- Ein Type-Mapping, welches die Typen der eingehenden Parameter prüft und ggf. den Rückgabetyp berechnet.
- Überladene Operatoren benötigen zusätzlich eine Selection-Funktion, die bestimmt welche Variante benutzt werden soll.
- Ein oder mehrere Value-Mapping Funktionen.
- Eine Beschreibung für den Benutzer.
- Eine Syntaxspezifikation und Beispiele.

# Überblick

Jeder Operator benötigt:

- Ein Type-Mapping, welches die Typen der eingehenden Parameter prüft und ggf. den Rückgabetyp berechnet.
- Überladene Operatoren benötigen zusätzlich eine Selection-Funktion, die bestimmt welche Variante benutzt werden soll.
- Ein oder mehrere Value-Mapping Funktionen.
- Eine Beschreibung für den Benutzer.
- Eine Syntaxspezifikation und Beispiele.

# Überblick

Jeder Operator benötigt:

- Ein Type-Mapping, welches die Typen der eingehenden Parameter prüft und ggf. den Rückgabetyp berechnet.
- Überladene Operatoren benötigen zusätzlich eine Selection-Funktion, die bestimmt welche Variante benutzt werden soll.
- Ein oder mehrere Value-Mapping Funktionen.
- Eine Beschreibung für den Benutzer.
- Eine Syntaxspezifikation und Beispiele.

# Type Mapping und Selection Funktionen

```
ListExpr
insideTypeMap( ListExpr args )
{
  NList type(args);
  const string errMsg = "Expecting two rectangles "
                        "or a point and a rectangle";

  // first alternative: xpoint x xrectangle -> bool
  if ( type == NList(XPOINT, XRECTANGLE) ) {
    return NList(BOOL).listExpr();
  }

  // second alternative: xrectangle x xrectangle -> bool
  if ( type == NList(XRECTANGLE, XRECTANGLE) ) {
    return NList(BOOL).listExpr();
  }

  return NList::typeError(errMsg);
}

int
insideSelect( ListExpr args )
{
  NList type(args);
  if ( type.first().isSymbol( XRECTANGLE ) )
    return 1;
  else
    return 0;
}
```

# Value Mapping

```
int
insideFun_PR (Word* args, Word& result, int message,
              Word& local, Supplier s)
{
  //cout << "insideFun_PR" << endl;
  XPoint* p = static_cast<XPoint*>( args[0].addr );
  XRectangle* r = static_cast<XRectangle*>( args[1].addr );

  result = qp->ResultStorage(s);
                                  //query processor has provided
                                  //a CcBool instance for the result

  CcBool* b = static_cast<CcBool*>( result.addr );

  bool res = ( p->GetX() >= r->GetXLeft()
            && p->GetX() <= r->GetXRight()
            && p->GetY() >= r->GetYBottom()
            && p->GetY() <= r->GetYTop() );

  b->Set(true, res); //the first argument says the boolean
                     //value is defined, the second is the
                     //real boolean value)
  return 0;
}
```

# Operatorbeschreibungen

```
struct insideInfo : OperatorInfo {

  insideInfo()
  {
    name       = INSIDE;

    signature = XPOINT + " x " + XRECTANGLE + " -> " + BOOL;

    // since this is an overloaded operator we append
    // an alternative signature here

    appendSignature( XRECTANGLE + " x " + XRECTANGLE
                                          + " -> " + BOOL );
    syntax     = "_" + INSIDE + "_";
    meaning    = "Inside predicate.";
  }
};
```

# .spec and .example Dateien

```
<PointRectangleAlgebra.spec>:
-----------------------------
operator intersects alias INTERSECTS pattern _ infixop _
operator inside alias INSIDE pattern _ infixop _

<PointRectangle.example>:
-------------------------
Database : prttest
Restore  : YES

Operator : inside
Number   : 1
Signature: xpoint x xrectangle -> bool
Example  : query p1 inside r1
Result   : TRUE

Operator : inside
Number   : 2
Signature: xrectangle x xrectangle -> bool
Example  : query r2 inside r1
Result   : TRUE

Operator : intersects
Number   : 1
Signature: xrectangle x xrectangle -> bool
Example  : query r1 intersects r2
Result   : TRUE
```

# Modul StreamExampleAlgebra

Diese Algebra bietet keine neuen Typen an, aber folgende Operationen:

- `intstream: int x int -> stream(int)`
- `countintstream: stream(int) -> int`
- `printintstream: stream(int) -> stream(int)`
- `filterintstream: stream(int) x (int -> bool) -> stream(int)`

- In Datenbanksystemen spielt Stromverarbeitung eine wichtige Rolle, da dies die Implementierung effizienter Auswertungspläne erlaubt.
- Keine Stromverarbeitung => alle Zwischenergebnisse müssen komplett berechnet werden.

# Modul StreamExampleAlgebra

Diese Algebra bietet keine neuen Typen an, aber folgende Operationen:

- `intstream: int x int -> stream(int)`
- `countintstream: stream(int) -> int`
- `printintstream: stream(int) -> stream(int)`
- `filterintstream: stream(int) x (int -> bool) -> stream(int)`

- In Datenbanksystemen spielt Stromverarbeitung eine wichtige Rolle, da dies die Implementierung effizienter Auswertungspläne erlaubt.
- Keine Stromverarbeitung => alle Zwischenergebnisse müssen komplett berechnet werden.

# Modul StreamExampleAlgebra

Diese Algebra bietet keine neuen Typen an, aber folgende Operationen:

- `intstream: int x int -> stream(int)`
- `countintstream: stream(int) -> int`
- `printintstream: stream(int) -> stream(int)`
- `filterintstream: stream(int) x (int -> bool) -> stream(int)`

- In Datenbanksystemen spielt Stromverarbeitung eine wichtige Rolle, da dies die Implementierung effizienter Auswertungspläne erlaubt.
- Keine Stromverarbeitung => alle Zwischenergebnisse müssen komplett berechnet werden.

# Modul StreamExampleAlgebra

Diese Algebra bietet keine neuen Typen an, aber folgende Operationen:

- `intstream: int x int -> stream(int)`
- `countintstream: stream(int) -> int`
- `printintstream: stream(int) -> stream(int)`
- `filterintstream: stream(int) x (int -> bool) -> stream(int)`

- In Datenbanksystemen spielt Stromverarbeitung eine wichtige Rolle, da dies die Implementierung effizienter Auswertungspläne erlaubt.
- Keine Stromverarbeitung => alle Zwischenergebnisse müssen komplett berechnet werden.

# Modul StreamExampleAlgebra

Diese Algebra bietet keine neuen Typen an, aber folgende Operationen:

- `intstream: int x int -> stream(int)`
- `countintstream: stream(int) -> int`
- `printintstream: stream(int) -> stream(int)`
- `filterintstream: stream(int) x (int -> bool) -> stream(int)`

- In Datenbanksystemen spielt Stromverarbeitung eine wichtige Rolle, da dies die Implementierung effizienter Auswertungspläne erlaubt.
- Keine Stromverarbeitung => alle Zwischenergebnisse müssen komplett berechnet werden.

# Modul StreamExampleAlgebra

Diese Algebra bietet keine neuen Typen an, aber folgende Operationen:

- `intstream: int x int -> stream(int)`
- `countintstream: stream(int) -> int`
- `printintstream: stream(int) -> stream(int)`
- `filterintstream: stream(int) x (int -> bool) -> stream(int)`

- In Datenbanksystemen spielt Stromverarbeitung eine wichtige Rolle, da dies die Implementierung effizienter Auswertungspläne erlaubt.
- Keine Stromverarbeitung => alle Zwischenergebnisse müssen komplett berechnet werden.

# Implementierung eines Stromoperators

Man kann drei Fälle unterscheiden:

- Operatoren, die einen Strom erzeugen.
- Operatoren, die einen Strom verbrauchen.
- Operatoren, die Stromelemente verarbeiten und selbst auch einen Strom erzeugen.
- Ein Strom kann dabei folgende Zustände haben:

# Implementierung eines Stromoperators

Man kann drei Fälle unterscheiden:

- Operatoren, die einen Strom erzeugen.
- Operatoren, die einen Strom verbrauchen.
- Operatoren, die Stromelemente verarbeiten und selbst auch einen Strom erzeugen.
- Ein Strom kann dabei folgende Zustände haben:

# Implementierung eines Stromoperators

Man kann drei Fälle unterscheiden:

- Operatoren, die einen Strom erzeugen.
- Operatoren, die einen Strom verbrauchen.
- Operatoren, die Stromelemente verarbeiten und selbst auch einen Strom erzeugen.
- Ein Strom kann dabei folgende Zustände haben:

# Implementierung eines Stromoperators

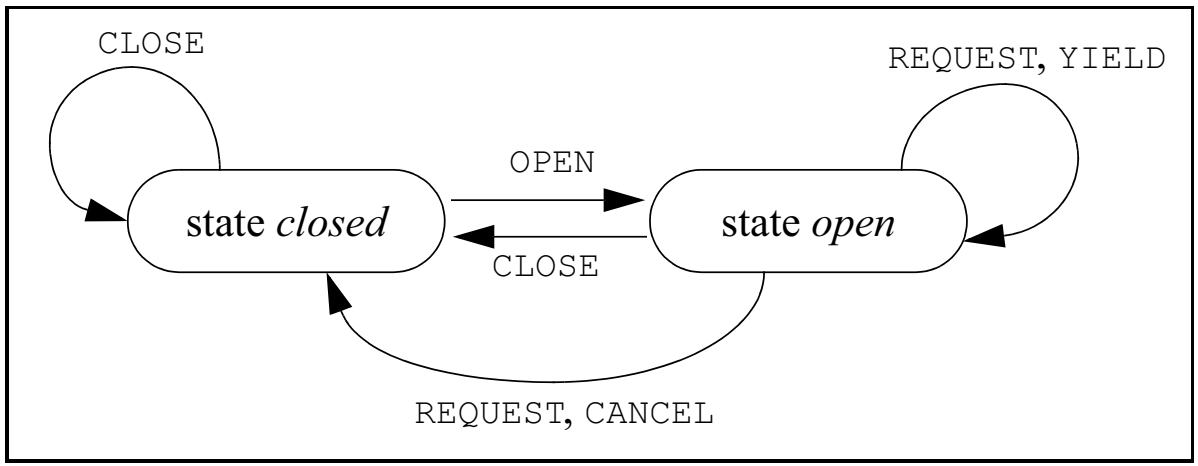
Man kann drei Fälle unterscheiden:

- Operatoren, die einen Strom erzeugen.
- Operatoren, die einen Strom verbrauchen.
- Operatoren, die Stromelemente verarbeiten und selbst auch einen Strom erzeugen.
- Ein Strom kann dabei folgende Zustände haben:

# Implementierung eines Stromoperators

Man kann drei Fälle unterscheiden:

- Operatoren, die einen Strom erzeugen.
- Operatoren, die einen Strom verbrauchen.
- Operatoren, die Stromelemente verarbeiten und selbst auch einen Strom erzeugen.
- Ein Strom kann dabei folgende Zustände haben:



# Operator intstream

```
int
intstreamFun (Word* args, Word& result, int message, Word& local, Supplier s)
{
  Range* range = static_cast<Range*>(local.addr);

  switch( message )
  {
    case OPEN: { // initialize the local storage

      CcInt* i1 = static_cast<CcInt*>( args[0].addr );
      CcInt* i2 = static_cast<CcInt*>( args[1].addr );
      range = new Range(i1, i2);
      local.addr = range;

      return 0;
    }
    case REQUEST: { // return the next stream element

      if ( range->current <= range->last )
      {
        CcInt* elem = new CcInt(true, range->current++);
        result.addr = elem;
        return YIELD;
      }
      else
      {
        // you should always set the result to null
        // before you return a CANCEL
        result.addr = 0;
        return CANCEL;
      }
    }
    case CLOSE: { // free the local storage

      delete range;
      return 0;
    }
    default: { /* should never happen */
      return -1;
}}}
```

# Operator countintstream

```
int
countFun (Word* args, Word& result, int message, Word& local, Supplier s)
{
  qp->Open(args[0].addr); // open the argument stream

  Word elem = SetWord(Address(0)); // retrieve the first element
  qp->Request(args[0].addr, elem);

  int count = 0;
  while ( qp->Received(args[0].addr) )
  {
    count++;
    // consume the stream objects. This will free their
    // memory representation if they are not used any more
    static_cast<Attribute*>(elem.addr)->DeleteIfAllowed();
    qp->Request(args[0].addr, elem);
  }

  // Assign a value to the operations result object which is provided
  // by the query processor
  result = qp->ResultStorage(s);
  static_cast<CcInt*>(result.addr)->Set(true, count);

  qp->Close(args[0].addr); // close the underlying stream

  return 0;
}
```