SECONDO: Implementierung einer Algebra

Simone Jandt

FernUniversität in Hagen
LG Datenbanksysteme f.neue Anwendungen

09.10.2009

Secondo: Implementierung einer Algebra
Inhalt

SECONDO: Implementierung einer Algebra

Einführung einer Algebra für Punkte und Rechtecke

Datentypen

- $\underline{xpoint}$ (x, y) mit $x, y \in \mathbb{Z}$
- $\underline{xrectangle}$ (x_1, x_2, y_1, y_2) mit $x_1, x_2, y_1, y_2 \in \mathbb{Z}$

Operatoren

- inside: $\underline{xpoint} \times \underline{xrectangle} \longrightarrow \underline{bool}$
- inside: $\underline{xrectangle} \times \underline{xrectangle} \longrightarrow \underline{bool}$
- intersects: $\underline{xrectangle} \times \underline{xrectangle} \longrightarrow \underline{bool}$

secondo/Algebras/PointRectangle

- PointRectangleAlgebra.cpp
- PointRectangleAlgebra.h
- PointRectangleAlgebra.spec
- PointRectangle.examples
- makefile

SECONDO: Implementierung einer Algebra

PointRectangleAlgebra.cpp: Definition C++-Klasse `XPoint`

Konstruktoren und Destruktoren

- `public XPoint (int x, int y);`
- `public XPoint (const XPoint& xp);`
- `private XPoint ();`
- `public ~XPoint ();`

Attributdefinition

```
private int x,y;
```

Methoden

- Get- und Set-Methoden für private Attribute
- Methoden zur SECONDO-Integration

NestedList

- Leere Liste: ()
- Ein oder mehrere Elemente: (a) oder (a b) oder ...

Mögliche Elemente

- einfache Werte der Typen:
  `integer`, `boolean`, `string` oder `text`
- mathematische Symbole oder Bezeichner
- NestedList z.B.: ((a (x z) b c) d)

SECONDO: Implementierung einer Algebra

Out-Funktion für Ausgabe `XPoint` als `NestedList`)

Beispiel

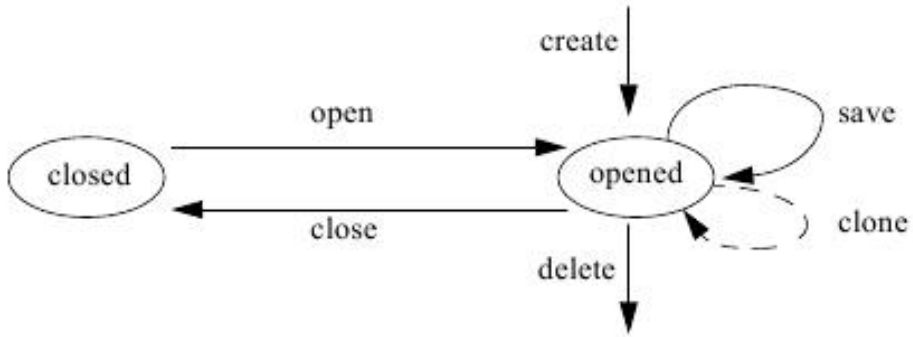
XPoint := (x y) mit $x, y \in \mathbb{Z}$ (integer)

```
ListExpr XPoint::Out(ListExpr typeInfo,
                     Word value)
  {XPoint *point =
    static_cast<XPoint*> (value.addr);
  NList twoElems(NList (point->GetX()),
                 NList (point->GetY()));
  return twoElems.listExpr();}
```

SECONDO: Implementierung einer Algebra

In-Funktion zum Einlesen `XPoint` aus `NestedList`


```
Word XPoint::In(const ListExpr typeInfo,
  const ListExpr instance, const int errorPos,
  ListExpr& errorInfo, bool& correct)
{Word w = SetWord(Address(0));
NList list(instance);
if (list.length() == 2) {
  NList f = list.first();
  NList s = list.second();
  if (f.isInt() && s.isInt()){
    w.addr = new XPoint(f.intval(), s.intval());
    correct = true; return w;}}
cmsg.inFunError("Two int expected.");
correct = false; return w;}
```



- **create** : Erzeugt ein Datenbankobjekt
- **delete** : Löscht ein Datenbankobjekt
- **close** : Entfernt das Objekt aus dem Hauptspeicher
- **clone** : Erzeugt neue Kopie des Objekts
- **open** : Liest das Datenbankobjekt von der Festplatte
- **save** : Speichert Änderungen auf der Festplatte

SECONDO: Implementierung einer Algebra

Erforderliche Ergänzungen in der `PointRectangleAlgebra.cpp`

Create-Funktion (Anlegen eines Datenbankobjekts)

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

Delete-Funktion (Löschen eines Datenbankobjekts)

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

SECONDO: Implementierung einer Algebra

Erforderliche Ergänzungen in der `PointRectangleAlgebra.cpp`

Close-Funktion (Schließen eines Datenbankobjekts)

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

Clone-Funktion (Neue Kopie eines Datenbankobjekts)

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

Save-Funktion (Speichern der Änderungen)

```
bool XPoint::Save (SMIRecord& valueRecord,
    size_t& offset, const ListExpr typeInfo,
    Word& value){
  XPoint* x = static_cast<XPoint*>(value.addr);
  size_t size = sizeof(int);
  bool ok = valueRecord.Write(
                &x->GetX(), size, offset);
  offset += size;
  ok = ok && valueRecord.Write(
                &x->GetY(), size, offset);
  offset += size;
  return ok;}
```

Open-Funktion (Lesen des Datenobjekts von Festplatte)

```
bool XPoint::Open (SMIRecord& valueRecord,
    size_t& offset, const ListExpr typeInfo,
    Word& value){
  size_t size = sizeof(int);
  int x = 0; int y = 0;
  bool ok = valueRecord.Read(&x,size,offset);
  offset += size;
  ok = ok && valueRecord.Read(&y,size,offset);
  offset += size;
  value.addr = new XPoint(x,y);
  return ok;}
```

SizeOfObj-Funktion (Objektgröße)

```
int XPoint::SizeOfObj(){
  return sizeof(XPoint);}
```

Cast-Funktion (Für Objekte in Relationen)

```
void* XPoint::Cast(void* addr){
  return (new (addr) XPoint);}
```

CheckKind-Funktion

```
bool XPoint::CheckKind(ListExpr type,
    ListExpr& errorInfo){
  return (nl->IsEqual(type, "xpoint"));}
```

SECONDO: Implementierung einer Algebra
Zuordnung der Datentypfunktionen


```
struct xpointFunctions :
  ConstructorFunctions<XPoint>{
    xpointFunctions(){
      in          = XPoint::In;
      out         = XPoint::Out;
      create      = XPoint::Create;
      deletion    = XPoint::Delete;
      close       = XPoint::Close;
      clone       = XPoint::Clone;
      open        = XPoint::Open;
      save        = XPoint::Save;
      cast        = XPoint::Cast;
      sizeOf      = XPoint::SizeOfObj;
      kindCheck = XPoint::KindCheck;}};
```

```
struct xpointInfo : ConstructorInfo
  {xpointInfo()
    {name        = "xpoint";
    signature   = "->"+ SIMPLE;
    typeExample = "xpoint";
    listRep     = "(<int> <int>)";
    valueExample = "(5 7)";
    remarks     = "Integer Coordinates";}};
```

Erzeugung eigentlicher TypeConstructor

```
xpointInfo xpi;
xpointFunctions xpf;
TypeConstructor xpointTC(xpi,xpf);
```

SECONDO: Implementierung einer Algebra

TypeMapping = Vergleich der übergebenen Argumente mit den zulässigen Datentypen für den Operator

Beispiel

intersects : *xrectangle* × *xrectangle* → *bool*

```
ListExpr intersectsTypeMap (ListExpr args)
  {NList type(args);
  if (type == NList("xrectangle", "xrectangle"))
    return NList(BOOL).listExpr()
  else
    return NList::typeError(
            "Two rectangles expected.");}
```

SECONDO: Implementierung einer Algebra

ValueMapping = Berechnung des Ergebnisses

Beispiel

intersects : *xrectangle* × *xrectangle* → *bool*

```
int intersectsValueMap(Word *args, Word& res,
    int message, Word& local, Supplier s)
{XRectangle *r1 =
  static_cast<XRectangle*>(args[0].addr);
XRectangle *r2 =
  static_cast<XRectangle*>(args[1].addr);
res = qp->ResultStorage(s);
CcBool *b = static_cast<CcBool*>(res.addr);
b->Set(true, r1->intersects(*r2));
return 0;}
```

Beispiel

intersects : *xrectangle* × *xrectangle* → *bool*

```
struct intersectsInfo : OperatorInfo{
  intersectsInfo(){
    name        = "intersects";
    signature   = "xrectangle × xrectangle
                                -> bool";
    syntax      = "x intersects y";
    meaning     = "true if x intersects y";}};
```

SECONDO: Implementierung einer Algebra
Überladene Operatoren für mehrere Signaturen

- Erweiterung des TypeMappings auf alle zulässigen Argumentkombinationen
- Ergänzung OperatorInfo um weitere Signaturen mit: `appendSignature("a × b -> c")`.
- Berechnungsfunktionen müssen für alle Argumtenkombinationen bereitgestellt werden
- ValueMapping = Zusammenstellung der Berechnungsfunktionen in einem Array von Funktionszeigern
- Funktion zur Selektion der Berechnungsfunktion an Hand der übergebenen Argumentkombination

Beispiel

inside : *xpoint* $\times$ *xrectangle* $\rightarrow$ *bool*
inside : *xrectangle* $\times$ *xrectangle* $\rightarrow$ *bool*

```
const string insidemap[2][3] =
  {{"xpoint","xrectangle",BOOL},
    {"xrectangle","xrectangle",BOOL}};
ListExpr insideTypeMap (ListExpr args)
  {return SimpleMaps<2,3>(insidemap,args);}
```

ValueMapping für inside Operator

```
ValueMapping insideFuns[] =
  {inside_PR, inside_RR, 0}
```

Select-Funktion für inside Operator

```
int insideSelect(ListExpr args){
  NList type(args);
  if (type.first().isSymbol("xpoint"))
    return 0;
  else return 1;}
oder:
int insideSelect(ListExpr args){
  return SimpleSelect<2,3>(insidemap,args);}
```

SECONDO: Implementierung einer Algebra
Definition der PointRectangleAlgebra-Klasse


```
class PointRectangleAlgebra : public Algebra{
  public PointRectangleAlgebra():Algebra(){
    AddTypeConstructor(&xpointTC);
    AddTypeConstructor(&xrectangleTC);
    xpointTC.AssociateKind(SIMPLE);
    xrectangleTC.AssociateKind(SIMPLE);
    AddOperator(intersectsInfo(),
        intersectsValueMap, instersectsTypeMap);
    AddOperator( insideInfo(), insideFuns,
        insideSelect, insideTypeMap);}
  public ~PointRectangleAlgebra(){}};
```

```
#include "Algebra.h"
#include "NestedList.h" (alte Version)
#include "NList.h" (neuere Version)
#include "LogMsg.h"
#include "QueryProcessor.h"
#include "ConstructorTemplates.h"
#include "StandardTypes.h"
extern NestedList *nl;
extern QueryProcessor *qp;
#include "Symbols.h"
using namespace symbols;
#include "TypeMapUtils.h"
using namespace mappings;
#include <string>
using namespace std;
```

SECONDO: Implementierung einer Algebra

Initialisierung der `PointRectangleAlgebra`

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

SECONDO: Implementierung einer Algebra

Aufbau der `PointRectangleAlgebra.spec`-Datei

- Eine Zeile für jeden Operator
- `operator <opname> alias <ALIAS> pattern <pattern>`
- Mögliche `<pattern>`: `secondo/Algebras/specs`
- Gleichnamige Operatoren in unterschiedlichen Algebren müßen identische Spezifikation haben

Beispiel

```
operator inside alias INSIDE pattern
                        _ infixop _
operator intersects alias INTERSECTS pattern
                        _ infixop _
```

SECONDO: Implementierung einer Algebra

Aufbau der `PointRectangle.examples`-Datei

```
Database : prttest
Restore  : NO

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

SECONDO: Implementierung einer Algebra

Algebraaktivierung

Eintrag in `secondo/makefile.algebras`

```
ALGEBRA_DIRS += PointRectangle
ALGEBRAS += PointRectangleAlgebra
```

Eintrag in `secondo/Algebras/Management/AlgebraList.i.cfg`

```
ALGEBRA_INCLUDE(xx,PointRectangleAlgebra)
```

Eingabe von `make` im Verzeichnis `secondo` übersetzt SECONDO neu und integriert alle aktiven Algebren.

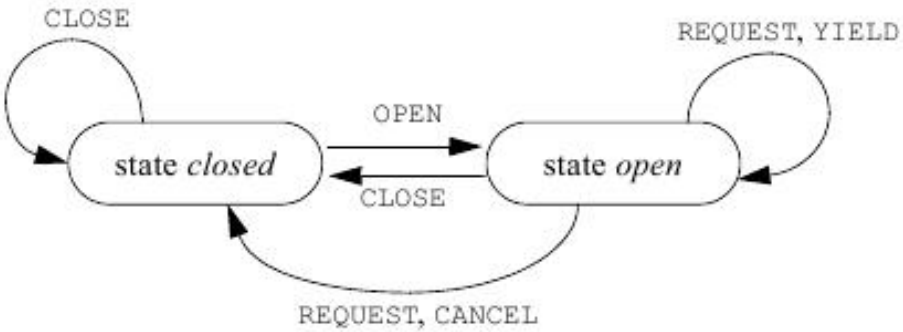
Wozu?

- Verarbeitung großer Datenmengen
- Zwischenergebnisse müßen nicht komplett materialisiert werden
- Erstellung effizienter Ausführungspläne
- Beispiele in `StreamExampleAlgebra`

Arten

- Erzeugung von Datenströmen
- Verarbeiten und Erzeugen von Datenströmen
- Verarbeiten und Verbrauchen von Datenströmen

SECONDO: Implementierung einer Algebra
Zustände von Strömen in SECONDO



- `OPEN` : Öffnet einen Datenstrom
- `CLOSE` : Schließt einen Datenstrom
- `REQUEST` : Fordert das nächste Element des Datenstroms an
- `YIELD` : Anforderung war erfolgreich
- `CANCEL` : Anforderung war nicht erfolgreich

SECONDO: Implementierung einer Algebra

TypeMapping für Stromerzeugende Operatoren

Beispiel

intstream : $\underline{int} \times \underline{int} \longrightarrow \underline{stream}(\underline{int})$

```
ListExpr intstreamsTypeMap (ListExpr args){
  NList type(args);
  if (type != NList(INT, INT))
    return NList::typeError("Two int expected");
  else
    return NList(STREAM, INT).listExpr();}
```

SECONDO: Implementierung einer Algebra

ValueMapping für Stromoperatoren Aufbau interne Struktur

```
int intstreamValueMap (Word *args, Word& res,
  int message, Word& local, Supplier s){
struct Range {
  int current, last;
  Range(CcInt* i1, CcInt* i2){
    if (i1->IsDefined() && i2->IsDefined()){
      current = i1->GetIntval();
      last = i2->GetIntval();}
    else {current = 1; last = 0;}}};
Range* range =
  static_cast<Range*>(local.addr);
switch(message){
```

SECONDO: Implementierung einer Algebra

Fortsetzung `intstreamValueMap case OPEN und CLOSE`


```
case OPEN:{
  i1 = ((CcInt*)args[0].addr);
  i2 = ((CcInt*)args[1].addr);
  range = new Range(i1,i2);
  local.addr = range;
  return 0;}
```

```
case CLOSE:{
  if (range != 0){
    delete range;
    local.addr = 0;}
  return 0;}
```

SECONDO: Implementierung einer Algebra

Fortsetzung `intstreamValueMap case REQUEST`


```
case REQUEST:{
  if (range->current <= range->last){
    CcInt* elem =
      new CcInt(true, range->current++);
    res.addr = elem;
    return YIELD;}
  else {
    res.addr = 0;
    return CANCEL;}}

default: return -1;}}
```

SECONDO: Implementierung einer Algebra

ValueMapping für Stromverbrauchende Operatoren

count: *stream*(*int*) → *int*


```
int countValueMap (Word *args, Word& res,
  int message, Word& local, Supplier s){
qp->Open(args[0].addr);
Word elem(Address(0));
int count = 0;
qp->Request(args[0].addr, elem);
while (qp->Received(args[0].addr)){count++;
  static_cast<CcInt*>(elem.addr)->
                DeleteIfAllowed();
  qp->Request(args[0].addr, elem);}
res = qp->ResultStorage(s);
static_cast<CcInt*>(res.addr)->Set(true,count);
qp->Close(args[0].addr);
return 0;}
```

Secondo: Implementierung einer Algebra

Vielen Dank für Ihre Aufmerksamkeit!