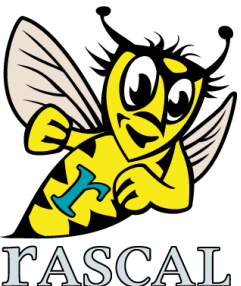
Relations

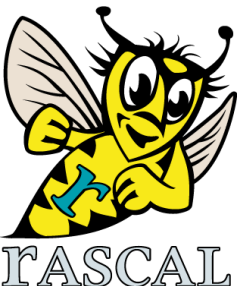
Operators, Functions, Applications

Paul Klint



Overview

- Definition of relation
- Some relation history
- Operators on relations
- Relation comprehension == set comprehension
- Library functions for relations
- Relation matching == Set matching

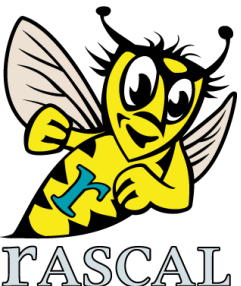


A Relation ...

- Given values of type A, a binary relation on A is a subset of A x A, i.e., a set of pairs, both from A
- A binary relation on values of type Person could be:
  - Abe likes Claire
  - Bob likes Harry
  - Gina likes Pat
- {<"abe", "claire">, <"bob", "harry">, <"gina", "pat">}

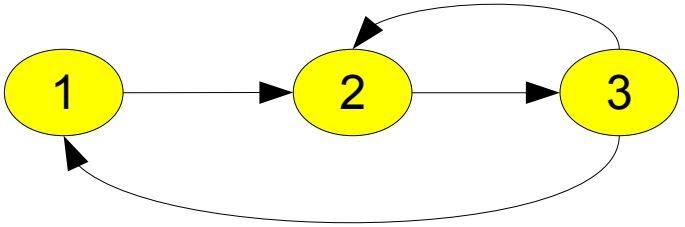
A Relation ...

- A binary relation can also be defined on sets of different values, e.g., between Person and Fruit: {<"abe", "banana">, <"claire", "apple">, <"john", "grape">, <"abe", "orange">}
- Note that
  - this is partial information, only a subset of Person or Fruit need be mentioned.
  - Values can occur more than once in some tuple

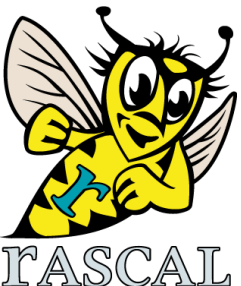


A Relation ...

- Another example is a graph. If we use integers to identify vertices (nodes), then the graph



- can be represented as: {<1,2>,<2,3>,<3,2>,<3,1>}
- Each pair represents an edge in the graph.



A Relation is ...

- An *n*-ary relation just connects values from *n* different domains. Suppose we define a Top2000 relation using the following domains:
  - Position in the song chart
  - Artist name
  - Song name
  - Year of song
- A subset of the Top2000 relation might then be:

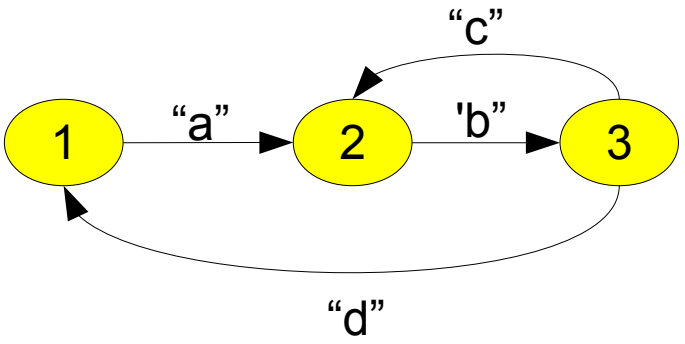
{ <1,"Eagles","Hotel California",1977>,

<2,"Queen","Bohemian rhapsody",1975>,

<3,"Boudewijn de Groot","Avond",1997> }

A Relation ...

- Another example is a labeled graph. The labeled graph



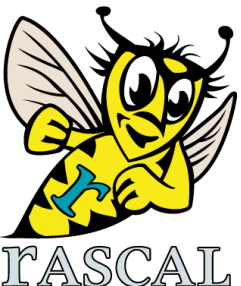
- can be represented as:

  {<1,"a",2>, <2,"b",3>, <3,"c",2>, <3,"d", 1>}

- A triple represents a labeled edge in the graph.

A Relation ...

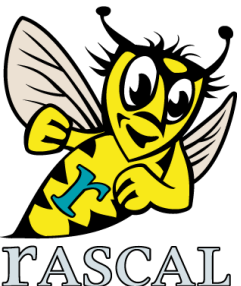
- A function associates a unique value with an argument (a recipe)
- A map also associates a unique value with a key (a finite collection)
- A relation *may* associate multiple values with each other: recall abe who likes banana and orange.



Relations in Rascal ...

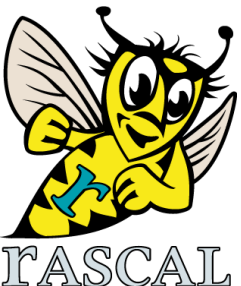
- Are sets of tuples of equal length and type.
- Their type is set[tuple[T_1,T_2,...T_n]] which may be abbreviated to rel[T_1,T_2,...T_n]
- Examples:
  - rel[Person,Person] likePerson;
  - rel[Person,Fruit] likeFruit;
  - rel[int,str,str,int] top2000;

Some Relation History

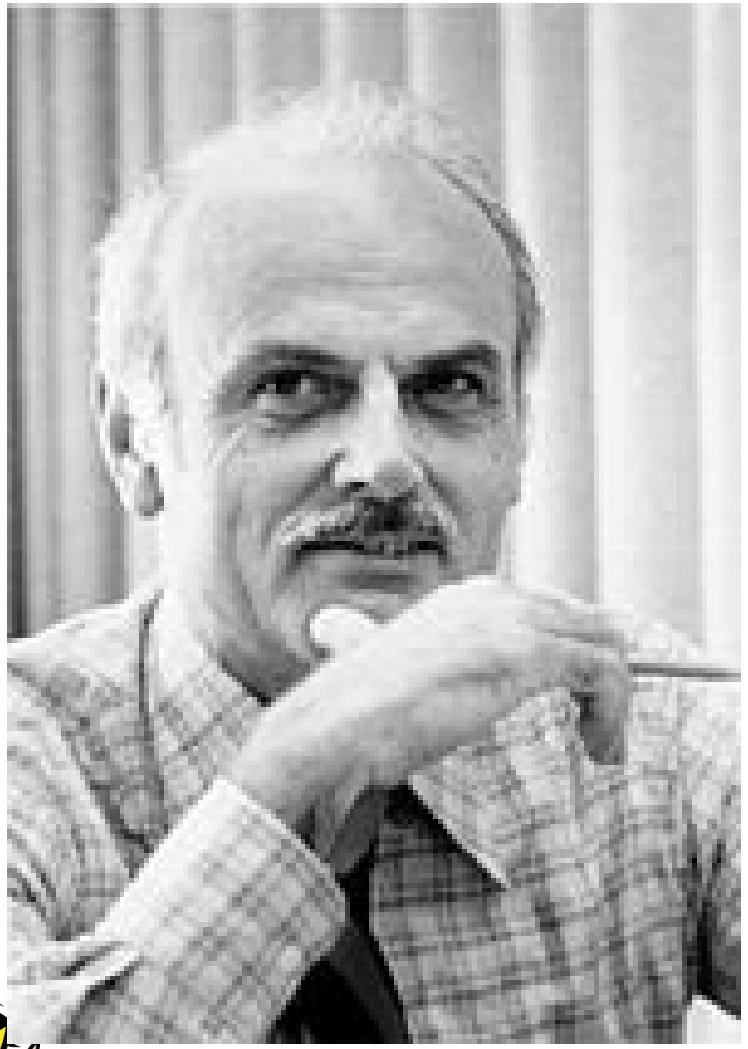


Alfred Tarski (1901-1983)

- Polish Logician
- Inventor of *relational algebra,* i.e., a treatment of relations based on logical operators



Ed Codd (1923-2003)



- English computer scientist
- Invented relational model for database management
- Based on relational calculus
- SQL: Structured Query Language; widely used

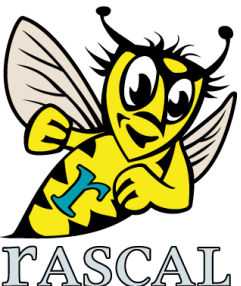
Database as Table

- Recall the labeled graph:
  {<1,"a",2>, <2,"b",3>, <3,"c",2>, <3,"d", 1>}
- Can be represented by the following table:

Named column

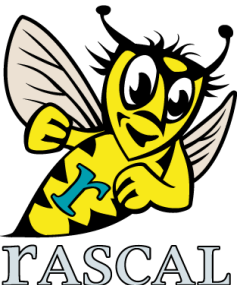
Row

From	Label	To
1	“a”	2
2	“b”	3
3	“c”	2
3	“d”	1



Alain Colmerauer (1941)

- French computer scientist
- Inventor of Prolog, a language based on relations
- Many applications in AI



A Bit more about Tuples

Rascal/Expressions/Values/Tuple

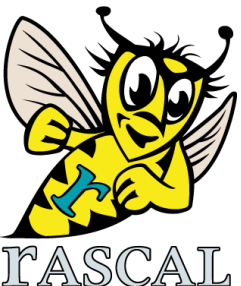
- Tuple types may contain a label for each element. The label can be used to retrieve that element (and to name a column of a relation!)
  - tuple[str first, str last, int age] P = <"Jo","Jones",35>;
  - P.first gives "Jo"
  - P.first = "Bo"; assigns a new tuple to P:
  - tuple[str first,str last,int age]: <"Bo","Jones",35>
- Elements can also be accessed by position:

  P[0] also gives "Jo"

A Bit more about Tuples

Rascal/Expressions/Values/Tuple

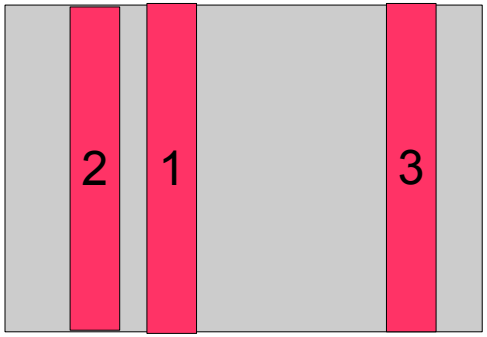
- Tuple values can also be concatenated using +:
  - <"abc", 1, 2.5> + <true, "def"> gives
  - tuple[str,int,real,bool,str]: <"abc", 1, 2.5, true, "def">
- Tuples can also be compared to each other using ==, <, ...



Relational Operators

Rascal/Expressions/Values/Relation

- Relations are sets, all set operators apply:
  - Insert, Union: `+`
  - Difference: `-`
  - Intersection: `&`
  - Product: `*`
  - In: `in`, NotIn: `notin`
  - Compare: `==`, `<`, ...
- Additional operators:
  - FieldSelection: `.`
  - FieldProjection: `< >`
  - Join: `join`
  - Composition: `o`
  - TransitiveClosure: `*`, `+`
  - Subscription: `[ ]`



Field Projection: < >

- Reorganizes the fields in a tuple or relation (uses field labels or field indices)

```
rel[str day, int daynum, int length] Traffic =
   {<"mon", 1, 100>, <"tue", 2, 150>, <"wed", 3, 125>,
    <"thur", 4, 110>, <"fri", 5, 90>};
```

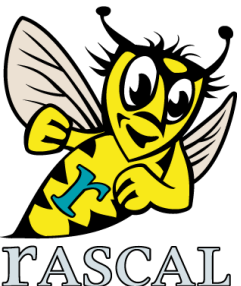
```
rascal>Traffic<length,daynum>
rel[int length, int daynum]: {
  <150,2>,
  <100,1>,
  <110,4>,
  <125,3>,
  <90,5>
}
```

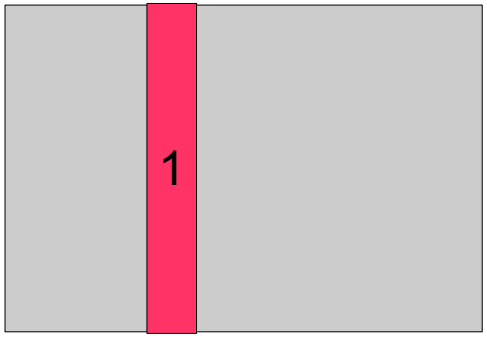
Exercises

- Given

```
rel[str day, int daynum, int length] Traffic =
   {<"mon", 1, 100>, <"tue", 2, 150>, <"wed", 3, 125>,
    <"thur", 4, 110>, <"fri", 5, 90>};
```

- Compute:
  - Traffic<2,0,1>
  - Traffic<2, daynum>





Field Selection:

- Selects a “column” from a relation

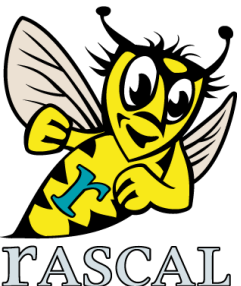
```
rel[str day, int daynum, int length] Traffic =
   {<"mon", 1, 100>, <"tue", 2, 150>, <"wed", 3, 125>,
    <"thur", 4, 110>, <"fri", 5, 90>};
```

```
rascal>Traffic.day
set[str]:
{"fri","mon","wed","tue","thur"}

rascal>Traffic.length
set[int]: {110,125,90,150,100}
```

Join: join

- The relation resulting from concatenating the tuples in each relation.

```
rascal>{<1,2>, <10,20>} join {<2,3>, <20,30>};
rel[int, int, int, int]: {
  <1,2,2,3>,
  <10,20,20,30>,
  <1,2,20,30>,
  <10,20,2,3>
}
```



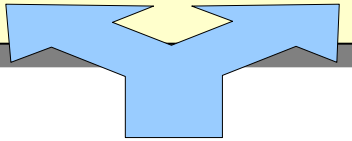
Composition: o

- Recall function composition: given add2 and mul3, mul3(add2(x)) will add 2 and multiply by 3
- Similarly, relation composition composes binary relations L and R such that
  - Each pair <a, b> in L is combined with each pair <b, c> in R to produce a pair <a, c>

```
rascal>{<1,10>, <2,20>, <3,15>} o {<10,100>, <20,200>};
rel[int, int]: {
  <1,100>,
  <2,200>
}
```

Defining composition

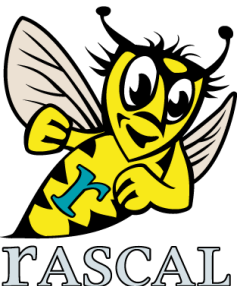
```
public rel[int,int] comp1(rel[int,int] L,rel[int,int] R) =
       { <a, c> | <int a, int b> <- L, <b, int c> <- R};
```



Filter matching tuples

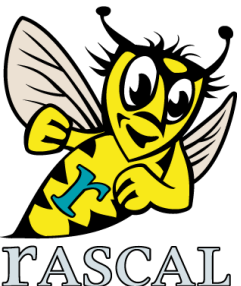
The general case:

```
public rel[&A, &C] comp2(rel[&A, &B] L, rel[&B, &C] R) =
       { <a, c> | <&A a, &B b> <- L, <b, &C c> <- R};
```



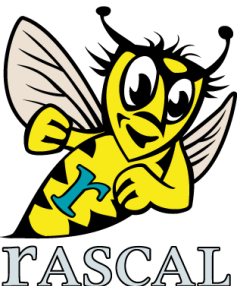
Exercises

- Compute the following compositions:
  - {<1,10>, <2, 20>} o {<10, 101>, <10, 102>, <20, 200>}
  - {<1, "a">, <2, "b">, <3, "c">} o {<"c", "C">, <"b", "B">, <"a", "A">} o {<"A", "AA">, <"B", "BB">}



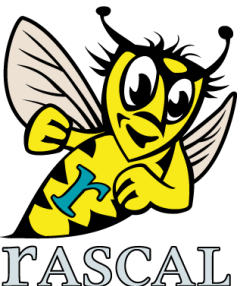
Transitive Closure: * and +

- Power of a relation:
  - $R^1 = R$, $R^2 = R \circ R$, $R^3 = R \circ R \circ R$, ...
  - R^0 = identity relation = {<a, a>,<b,b> | <a,b> <- R}
- Repeatedly compose a binary relation:
  - R+ = $R^1 + R^2 + R^3 + \ldots$ (transitive closure)
  - R* = $R^0 + R^1 + R^2 + R^3 + \ldots$ (reflexive transitive closure)



Applications of transitive closure

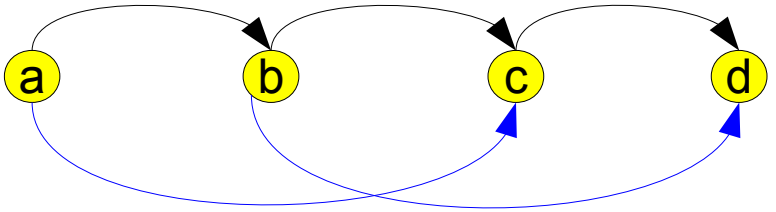
- Compute all the friends of your friends
- Compute all the procedures that a procedure can call
- Compute whether a given node is reachable from the root of a network



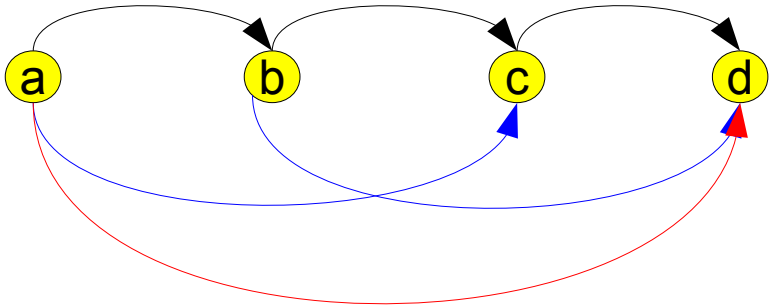
How to compute Transitive Closure?

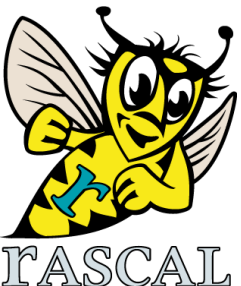
Original relation:

2-Jumps



3-Jumps

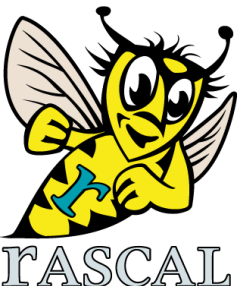




How to compute Transitive Closure?

```
public rel[int,int] tclosure(rel[int,int] R) {
    tc = R;
    while(true){
        tc1 = tc;
        tc += tc o R;
        if(tc1 == tc)
            return tc;
    }
}
```

```
rascal>tclosure({<1,2>,<2,3>,<3,4>})
rel[int, int]: {
  <1,3>, <1,2>,
  <2,3>, <2,4>,
  <3,4>, <1,4>
}
```



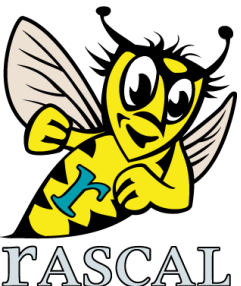
$R^* = R^0 + R+$

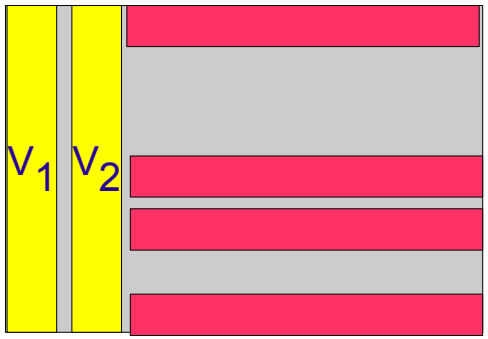
```
public rel[int,int] id(rel[int,int] R) =
  {<a, a>, <b, b> | <int a, int b> <- R};

public rel[int,int] rtclosure(rel[int,int] R) =
   id(R) + tclosure(R);
```

```
rascal>rtclosure({<1,2>,<2,3>,<3,4>})
rel[int, int]: {
  <1,3>,<1,2>,<1,1>,
  <2,2>,<3,3>,<4,4>,
  <2,3>,<2,4>,<3,4>,<1,4>
}
```

Exercises

- Compute the following transitive closures:
  - {<1,2>, <1,3>, <1, 4>, <3, 2>, <4, 5>}+
  - {<1,2>, <1,3>, <1, 4>, <3, 2>, <4, 5>}*



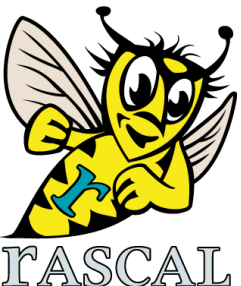


Subscription: [ ]

- R[V_1,V_2,...]: a relation with all tuples that have V_1, V_2, V_3, ... as first elements.

```
rel[str country, int year, int amount] GDP =
{<"US", 2008, 14264600>, <"EU", 2008, 18394115>,
 <"Japan", 2008, 4923761>, <"US", 2007, 13811200>,
 <"EU", 2007, 13811200>, <"Japan", 2007, 4376705>};
```

```
rascal>GDP["Japan"];
rel[int, int]: {
  <2008,4923761>,
  <2007,4376705>
}
rascal>GDP["Japan", 2008];
set[int]: {4923761}
```

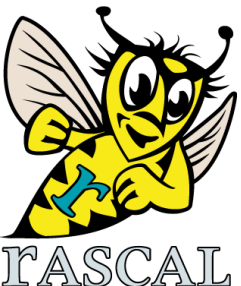


Exercises

- Given

```
rel[str country, int year, int amount] GDP =
{<"US", 2008, 14264600>, <"EU", 2008, 18394115>,
 <"Japan", 2008, 4923761>, <"US", 2007, 13811200>,
 <"EU", 2007, 13811200>, <"Japan", 2007, 4376705>};
```

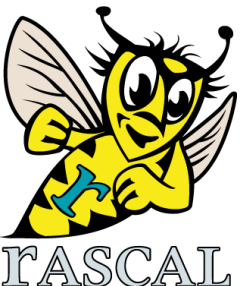
- Compute:
  - GDP["US"]
  - GDP[2008]
  - GDP[_,2008]

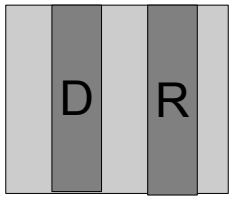


Library Functions for Relations (~15)

Rascal/Libraries/Prelude/Relation

- All set functions
- carrier, carrierR, carrierX
- domain, domainR, domainX
- range, rangeR, rangeX



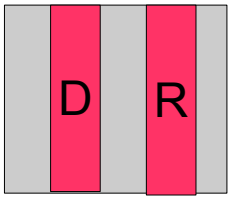


Carrier, Domain, Range

```
R = {<1,10>,<2,20>};
```

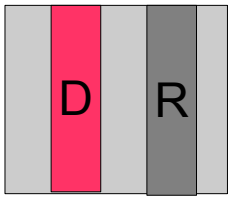
- Carrier: set of elements in any tuple

```
carrier(R) gives
{1,2,10,20}
```



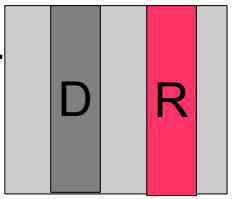
- Domain: set of elements in first element of any tuple

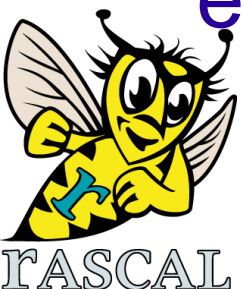
```
domain(R) gives {1,2}
```



- Range: set of elements in second element of any tuple

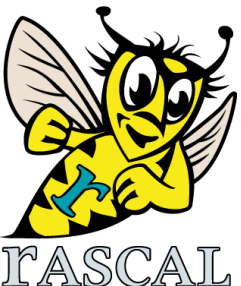
```
range(R) gives {10,20}
```





Exercises

- Give definitions for Rascal functions that implement:
  - carrier
  - domain
  - range

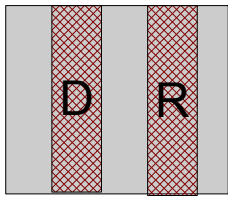


CarrierX, domainX, rangeX

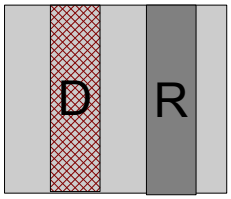
- Exclude tuples from a relation R and given set S
- CarrierX: exclude tuple if any elem in S
- DomainX: exclude tuple if 1st elem in S
- RangeX: exclude tuple if 2nd elem in S

```
R = {<1,10>,<2,20>};
S = {1,20}
```

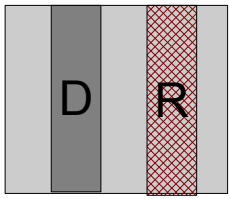
```
carrierX(R,S) = {}
```

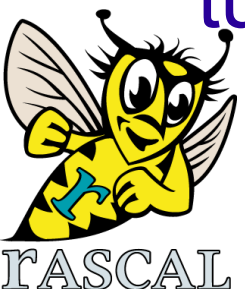


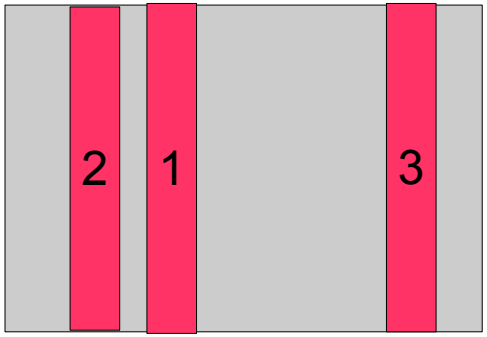

```
domainX(R,S) = {<2,20>}
```




```
rangeX(R,S) = {<1,10>}
```







Field Projection: < >

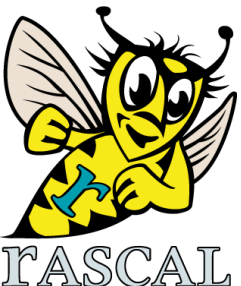
- Reorganizes the fields in a tuple or relation (uses field labels or field indices)

```
rel[str day, int daynum, int length] Traffic =
   {<"mon", 1, 100>, <"tue", 2, 150>, <"wed", 3, 125>,
    <"thur", 4, 110>, <"fri", 5, 90>};
```

```
rascal>Traffic<length,daynum>
rel[int length, int daynum]: {
  <150,2>,
  <100,1>,
  <110,4>,
  <125,3>,
  <90,5>
}
```

Exercises

- Give definitions for Rascal functions that implement:
  - carrierX
  - domainX
  - rangeX

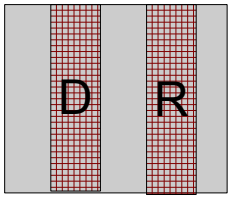


CarrierR, domainR, rangeR

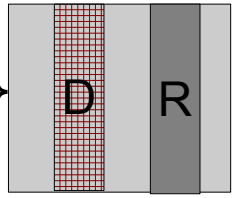
- Include tuples from a relation R and given set S
- CarrierR: include tuple if all elems in S
- DomainR: include tuple if 1st elem in S
- RangeR: include tuple if 2nd elem in S

```
R = {<1,10>,<2,20>};
S = {1,20}
```

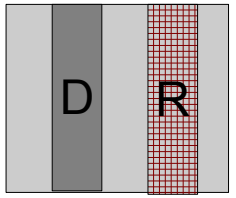
```
carrierR(R,S) = {}
```

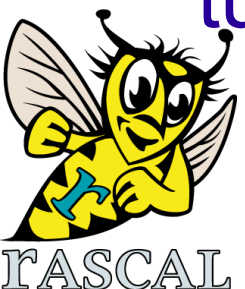



```
domainR(R,S) = {<1,10>}
```



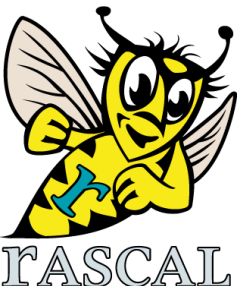

```
rangeR(R,S) = {<2,20>}
```



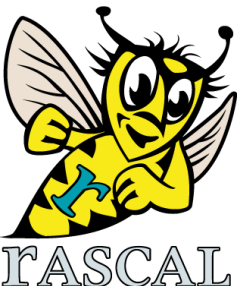


Exercises

- Give definitions for Rascal functions that implement:
  - carrierR
  - domainR
  - rangeR



Applications of Relations



Example

Exploring the AUC Course Catalog

Exploring the AUC Course catalog *themes*

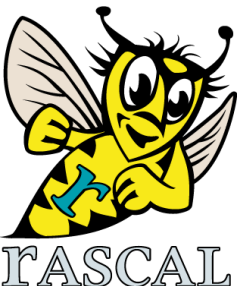
```
public rel[str,str] themeCourses = {
<"Social Systems", "Risk Management and Natural Hazards">,
<"Cities & Cultures", "Modernism and Postmodernism in Theory and Fiction">,
<"Social Systems", "Introductionn to Social Policy">,
<"Energy, Climate and Sustainability", "Introduction to Energy, Climate and Sustainability">,
<"Energy, Climate and Sustainability", "Energy, Climate and Sustainability: a case study">
//, ...
};

public set[str] getThemes() = domain(themeCourses);
```

```
rascal>getThemes()
set[str]: {"Energy, Climate and Sustainability","Cities
& Cultures","Social Systems"}
```

Exploring the AUC Course catalog *tracks*

```
public rel[str,str] trackCourses = {
    <"Environmental Economics", "Risk Management and Natural Hazards">,
    <"Earth and Environment", "Risk Management and Natural Hazards">,
    <"Literature", "Modernism and Postmodernism in Theory and Fiction">
};

public set[str] getTracks() = domain(trackCourses);
```

```
rascal>getTracks()
set[str]: {"Environmental Economics","Earth and
Environment","Literature"}
```

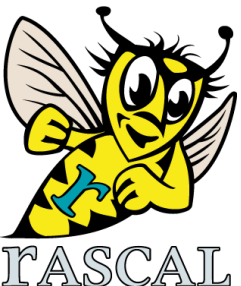


Exploring the AUC Course catalog *courses*

```
public rel[str title, str program, int number, int credits] Courses = {
     <"Risk Management and Natural Hazards", "SSC/SCI", 222, 6>,
     <"Modernism and Postmodernism in Theory and Fiction", "HUM", 121, 6>
     // , ...
};

public rel[str title, str preq] orPrereq = {
<"Risk Management and Natural Hazards", "Urban Economics">,
<"Risk Management and Natural Hazards", "Introduction to Environmental Science">,
*{<"Risk Management and Natural Hazards", tc> | tc <- themeCourses["Energy, Climate and
Sustainability"]}
//, ...
};

public rel[str title, str preq] andPrereq = {
   <"Cancer Biology and Treatment", "Molecular Cell Biology">,
   <"Cancer Biology and Treatment", "Immunology">
   //, ...
};
```

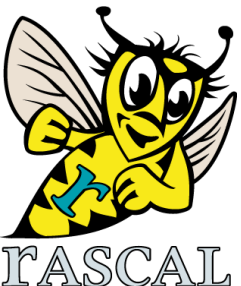
Checking prerequisites

```
// Special purpose checks for individual courses
// public bool satPrereq("xxx", set[str] followedCourses) = ...

public default bool satPrereq(str title, set[str] followedCourses) {
   op = orPrereq[title];
   if(!isEmpty(op) && isEmpty(op & followedCourses))
      return false;
   ap = andPrereq[title];
   if(!(ap <= followedCourses))
      return false;
   return true;
}
```



Checking prerequisites

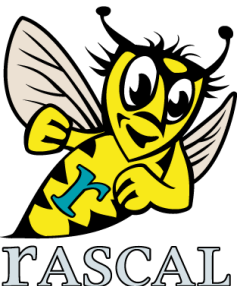
```
// Special purpose checks for individual courses
// public bool satPrereq("xxx", set[str] followedCourses) = ...

public default bool satPrereq(str title, set[str] followedCourses) {
   op = orPrereq[title];
   if(!isEmpty(op) && isEmpty(op & followedCourses))
      return false;
   ap = andPrereq[title];
   if(!(ap <= followedCourses))
      return false;
   return true;
}
```



Consistency Check

Courses that occur in the catalog but
are not part of any theme or track course:

```
public set[str] orphans(){
  allCourses = Courses[0];
  coursesInThemes = themeCourses[1];
  coursesInTracks = trackCourses[1];
  return allCourses - (coursesInThemes + coursesInTracks);
}
```

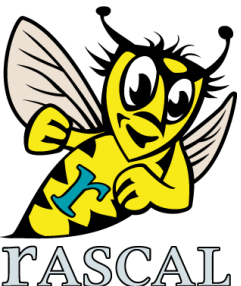


Example

Analyzing the Top2000 Song List

Analyzing the Top2000

- Visit http://top2011.radio2.nl/lijst/ and download the Excel version of the list
- Open in Excel and save as CSV file.
- Read in with `readCSV`
- Write functions to perform the analysis



Top2000 - Radio2.nl - De lijst

http://top2011.radio2.nl/lijst/

top2000 lijst 2011

Google Maps | Read It Later | Mark As Read | Reading List | Import to Mendeley | Send to OmniFocus | Instapaper Read Later | Open in Papers

UMTSE1... | voelter.d... | Alan Tur... | Blackbo... | eapls - ... | Editorial... | Startpag... | Voortga... | Top200...

TOP2000 radio 2 .nl

lorrainvillemsc zegt:

@jabo115 Vrijdag tussen 14&15u live op @Radio2_nl (uitgeklede versie). #Lorrainville #Roodshow

RETWEET

ontdek nl

KIJK

LUISTER

nu live Remco Veldhuis

Frank Boeijen

Muze (akoestische demoversie)

Top 2000 | reageren | de lijst | zoeken

home > de lijst

Dit is de lijst van 2011

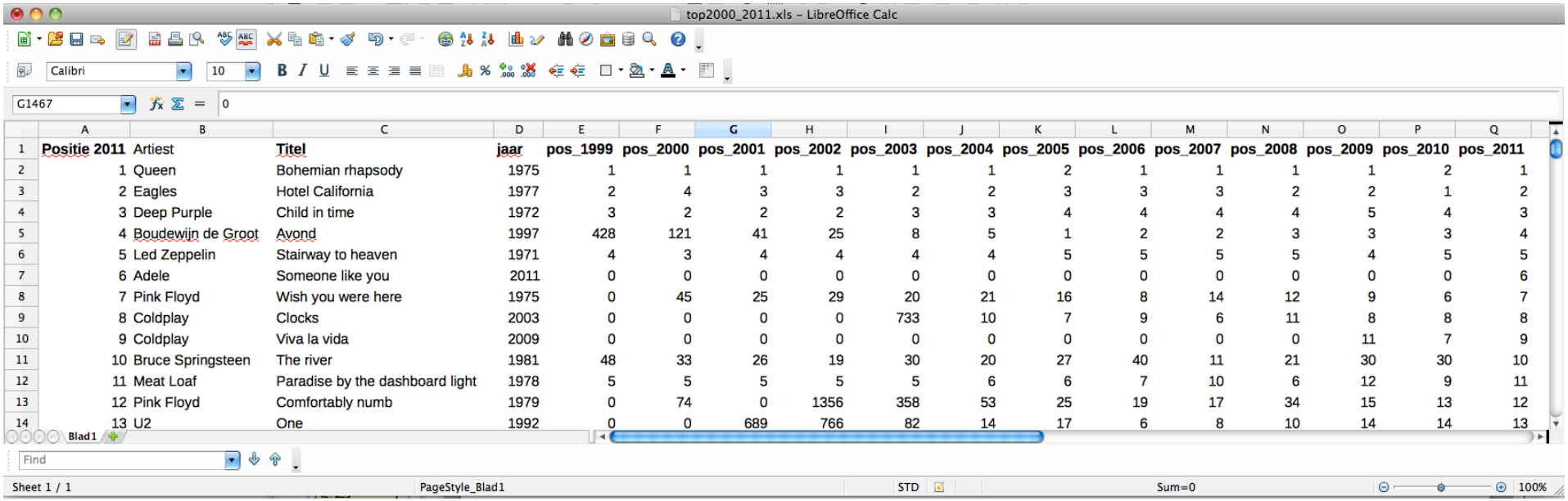
zoek op artiest of titel

Download de lijst van 2011 PDF

▲ gestegen in de lijst ▼ gedaald in de lijst = nieuw in de lijst ● gelijk gebleven

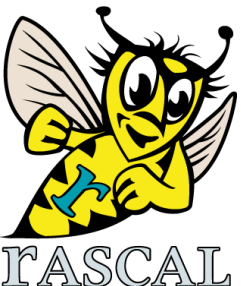
Stijgers	Positie	Artiest	Titel	Uitzending	
▲	1	Queen	Bohemian rhapsody	31 December 23:00 - 00:00 uur	MEER
▼	2	Eagles	Hotel California	31 December 23:00 - 00:00 uur	MEER
▲	3	Deep Purple	Child in time	31 December 23:00 - 00:00 uur	MEER
▼	4	Boudewijn de Groot	Avond	31 December 23:00 - 00:00 uur	MEER
●	5	Led Zeppelin	Stairway to heaven	31 December 23:00 - 00:00 uur	MEER
=	6	Adele	Someone Like You	31 December 23:00 - 00:00 uur	MEER
▼	7	Pink Floyd	Wish you were here	31 December 23:00 - 00:00 uur	MEER
●	8	Coldplay	Clocks	31 December 23:00 - 00:00 uur	MEER

Open top2000_2011.xls as Spreadsheet



Positie 2011	Artiest	Titel	jaar	pos_1999	pos_2000	pos_2001	pos_2002	pos_2003	pos_2004	pos_2005	pos_2006	pos_2007	pos_2008	pos_2009	pos_2010	pos_2011
1	Queen	Bohemian rhapsody	1975	1	1	1	1	1	1	2	1	1	1	1	2	1
2	Eagles	Hotel California	1977	2	4	3	3	2	2	3	3	3	2	2	1	2
3	Deep Purple	Child in time	1972	3	2	2	2	3	3	4	4	4	4	5	4	3
4	Boudewijn de Groot	Avond	1997	428	121	41	25	8	5	1	2	2	3	3	3	4
5	Led Zeppelin	Stairway to heaven	1971	4	3	4	4	4	4	5	5	5	5	4	5	5
6	Adele	Someone like you	2011	0	0	0	0	0	0	0	0	0	0	0	0	6
7	Pink Floyd	Wish you were here	1975	0	45	25	29	20	21	16	8	14	12	9	6	7
8	Coldplay	Clocks	2003	0	0	0	0	733	10	7	9	6	11	8	8	8
9	Coldplay	Viva la vida	2009	0	0	0	0	0	0	0	0	0	0	11	7	9
10	Bruce Springsteen	The river	1981	48	33	26	19	30	20	27	40	11	21	30	30	10
11	Meat Loaf	Paradise by the dashboard light	1978	5	5	5	5	5	6	6	7	10	6	12	9	11
12	Pink Floyd	Comfortably numb	1979	0	74	0	1356	358	53	25	19	17	34	15	13	12
13	U2	One	1992	0	0	689	766	82	14	17	6	8	10	14	14	13

Save as “Text CSV” (Comma Separated Values) file:
top2000_2011.csv



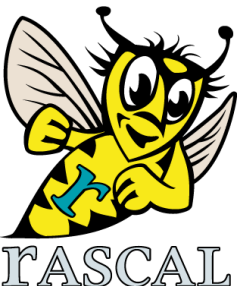
Open top2000_2011.csv as text file

First line: column names

top2000_2011.csv

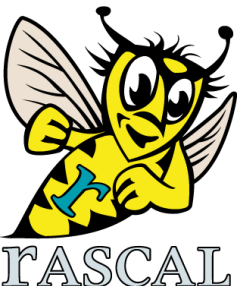
```
Positie 2011,Artiest,Titel,jaar,pos_1999,pos_2000,pos_2001,pos_2002,pos_2003,pos_2004,pos_2005,pos_2006,pos_2007,pos_2008,pos_2009,pos_2010,pos_2011
1,Queen,Bohemian rhapsody,1975,1,1,1,1,1,1,2,1,1,1,1,2,1
2,Eagles,Hotel California,1977,2,4,3,3,2,2,3,3,3,2,2,1,2
3,Deep Purple,Child in time,1972,3,2,2,2,3,3,4,4,4,4,5,4,3
4,Boudewijn de Groot,Avond,1997,428,121,41,25,8,5,1,2,2,3,3,3,4
5,Led Zeppelin,Stairway to heaven,1971,4,3,4,4,4,4,5,5,5,5,4,5,5
6,Adele,Someone like you,2011,0,0,0,0,0,0,0,0,0,0,0,0,6
7,Pink Floyd,Wish you were here,1975,0,45,25,29,20,21,16,8,14,12,9,6,7
8,Coldplay,Clocks,2003,0,0,0,0,733,10,7,9,6,11,8,8,8
9,Coldplay,Viva la vida,2009,0,0,0,0,0,0,0,0,0,0,11,7,9
10,Bruce Springsteen,The river,1981,48,33,26,19,30,20,27,40,11,21,30,30,10
11,Meat Loaf,Paradise by the dashboard light,1978,5,5,5,5,5,6,6,7,10,6,12,9,11
12,Pink Floyd,Comfortably numb,1979,0,74,0,1356,358,53,25,19,17,34,15,13,12
13,U2,One,1992,0,0,689,766,82,14,17,6,8,10,14,14,13
```

Following lines: row of values separated by commas



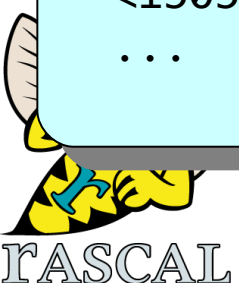
Reading in the CSV File

- `import lang::csv::IO;`
- Use one of the following functions:
  - `value readCSV(loc location)`
    - Read CSV file at location and return a value.
  - `&T readCSV(type[&T] result, loc location)`
    - Read CSV file at location and return required type (if possible).



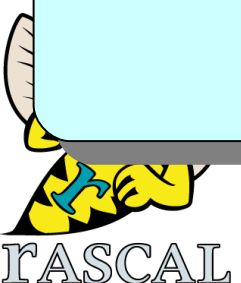
Reading the top2000 CSV File (*as a value*)

```
rascal>loc TOP2000csv = |file:///Users/paulklint/Desktop/top2000_2011.csv|;
loc: |file:///Users/paulklint/Desktop/top2000_2011.csv|

rascal>TOP2000 = readCSV(TOP2000csv);
readCSV inferred the relation type: rel[int Positie2011, str Artiest, str
Titel, int jaar, int pos1999, int pos2000, int pos2001, int pos2002, int
pos2003, int pos2004, int pos2005, int pos2006, int pos2007, int pos2008, int
pos2009, int pos2010, int pos2011]
value: {
  <1317,"Cornelis Vreeswijk","Veronica",1972,0,0,0,0,0,0,1333,885,672,1235,
   1127,1154,1317>,
  <1005,"Beatles","Back in the USSR",1968,0,876,1183,958,898,735,869,1068,
   931,902,974,907,1005>,
  <85,"Ramses Shaffy","Zing vecht huil bid lach werk en bewonder",1971,297,
   584,548,304,295,235,161,164,267,202,7,65,85>,
  <23,"Doors","Riders on the storm",1971,12,14,12,10,12,16,24,26,
   33,22,26,27,23>,
  <1505,"Jamiroquai","Deeper underground",1998,0,0,0,0,0,0,0,0,0,0,0,0,0,1505>,
  ...
```



Reading the Top2000 CSV File (*as a relation type*)

```
rascal>TOP2000 = readCSV(#rel[int Positie2011, str Artiest, str Titel, int
jaar, int pos1999, int pos2000, int pos2001, int pos2002, int pos2003, int
pos2004, int pos2005, int pos2006, int pos2007, int pos2008, int pos2009,
int pos2010, int pos2011], TOP2000csv);
rel[int Positie2011, str Artiest, str Titel, int jaar, int pos1999, int
pos2000, int pos2001, int pos2002, int pos2003, int pos2004, int pos2005,
int pos2006, int pos2007, int pos2008, int pos2009, int pos2010, int
pos2011]: {
  <1317,"Cornelis Vreeswijk","Veronica",1972,0,0,0,0,0,0,1333,885,672,1235,
   1127,1154,1317>,
  <1005,"Beatles","Back in the USSR",1968,0,876,1183,958,898,735,869,1068,
   931,902,974,907,1005>,
  <85,"Ramses Shaffy","Zing vecht huil bid lach werk en bewonder",1971,297,
   584,548,304,295,235,161,164,267,202,7,65,85>,
  ...
```



Analyzing the Top 2000

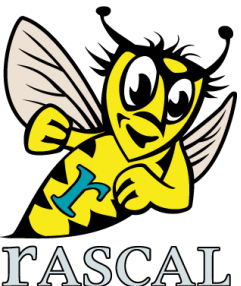
```
module AP8

import Prelude;
import lang::csv::IO;

loc TOP2000csv = |file:///Users/paulklint/Desktop/top2000_2011.csv|;

alias SONG = tuple[int Positie2011, str Artiest, str Titel, int jaar,
           int pos1999, int pos2000, int pos2001, int pos2002,
           int pos2003, int pos2004, int pos2005, int pos2006,
           int pos2007, int pos2008, int pos2009, int pos2010, int pos2011];

alias T2000 = set[SONG];

public T2000 getTOP2000() = readCSV(#T2000, TOP2000csv);
```

```
rascal>TOP2000 = getTOP2000();
T2000: {
  <1317,"Cornelis Vreeswijk","Veronica",1972,0,0,0,0,0,0,1333,885,672,1235,
   1127,1154,1317>,
  <1005,"Beatles","Back in the USSR",1968,0,876,1183,958,898,735,869,1068,
   931,902,974,907,1005>,
    ...
```

Analyzing the Top 2000

```
public int numberOfArtists(T2000 TOP) = size(TOP.Artiest);

public set[str] artistsWithMostHits2011(T2000 TOP) {
  ranking = { <size(TOP[_, artist]), artist> |  artist <- TOP.Artiest};
  highest = max(domain(ranking));
  return ranking[highest];
}
```

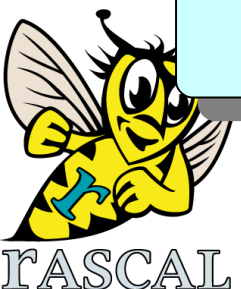
```
rascal>numberOfArtists(TOP2000)
int: 831

rascal>artistsWithMostHits2011(TOP2000)
set[str]: {"Beatles"}
```



Analyzing the Top 2000

```
int inTop10(SONG s){
   cnt = 0;
   for(int i <- [4 .. 16]){
       if(s[i] > 0 && s[i] <= 10)
          cnt += 1;
   }
   return cnt;
}

public set[str] songsInTop10(T2000 TOP) {
  return {song.Titel | song <- TOP, inTop10(song) > 0};
}
```

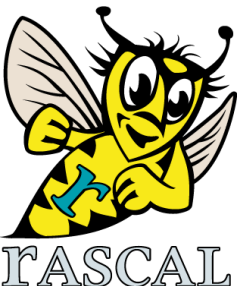
```
rascal>songsInTop10(TOP2000)
set[str]: {"Nothing else matters","The river","Paradise by the dashboard light",
           "Sultans of swing","Viva la vida","Bridge over troubled water", "Imagine",
           "Angie", "Wish you were here", "Hey Jude","Goodnight Saigon","Laat me",
           "Avond","Private investigations","House of the rising sun",
           "A whiter shade of pale","Zing vecht huil bid lach werk en bewonder",
           "Stairway to heaven","Old and wise","One","Brothers in arms",
           "Clocks","Pastorale","Hotel California","Child in time",
           "Someone like you","Bohemian rhapsody","November rain","Yesterday",
           "Riders on the storm"}
```



Analyzing the Top 2000

```
public set[str] songsMostInTop10(T2000 TOP) {
  counts = {<inTop10(song), song.Titel> | song <- TOP};
  highest = max(domain(counts));
  return counts[highest];
}
```

```
rascal>songsMostInTop10(TOP2000)
set[str]: {"Stairway to heaven","Hotel California",
           "Child in time","Bohemian rhapsody"}
```

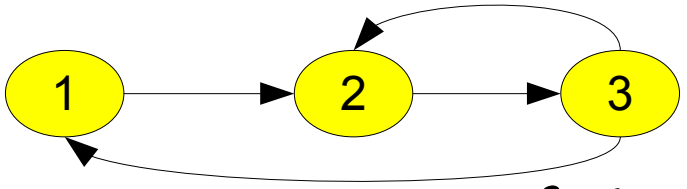


Example

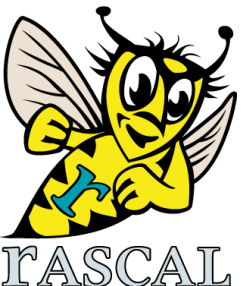
Analyzing the call structure of an application

A Graph is a Relation

- We have already seen that the graph

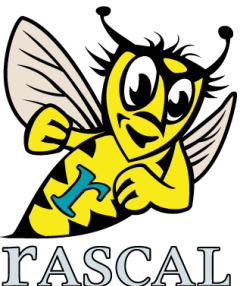


- can be represented as: {<1,2>,<2,3>,<3,2>,<3,1>}
- Each pair represents an edge in the graph.
- It is convenient to introduce an abbreviation:
  - alias graph[&T] = rel[&T,&T];



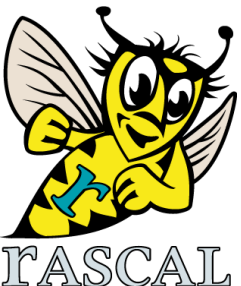
Graph library

- predecessors
- successors
- order
- bottom
- top
- reach
- reachR
- reachX
- shortestPathPair



Examples

- G = {<1,2>, <1,3>, <2,4>, <3,4>}
- predecessors(G, 4) = {3,2}
- successors(G, 2) = {4}
- order(G) = [1, 3, 2, 4]
- bottom(G) = {4}
- top(G) = {1}
- reach(G, {2}) = {4}
- reachR(G, {1}, {1, 2, 3}) = {3,2}
- reachX(G, {1}, {2}) = {4,3}



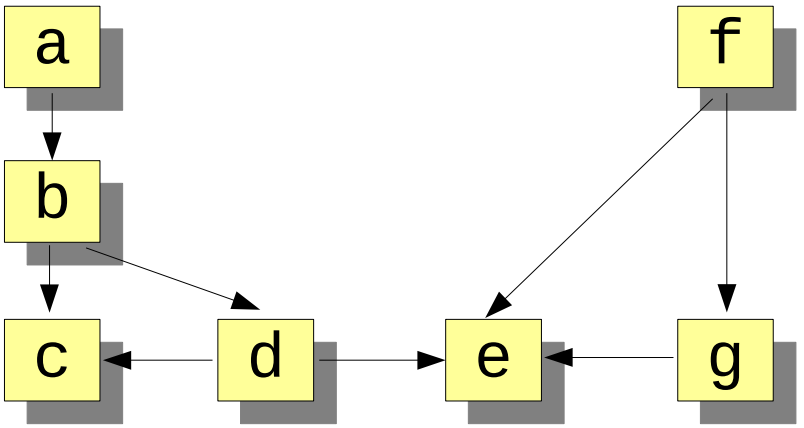
Defining: top

- The roots of a relation viewed as a graph
- top({<1,2>,<1,3>,<2,4>,<3,4>}) yields {1}
- Consists of all elements that occur on the lhs but not on the rhs of a tuple
-
```
set[&T] top(graph[&T] R) {
    return domain(R) - range(R);
}
```

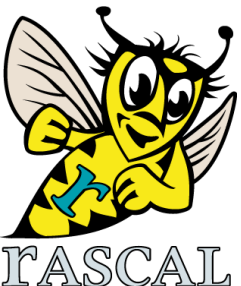
Defining bottom

- The leaves of a relation viewed as a graph
- bottom({<1,2>,<1,3>,<2,4>,<3,4>}) yields {4}
- Consists of all elements that occur on the rhs but not on the lhs of a tuple
- set[&T] bottom(graph[&T] R) {
  return range(R) – domain(R);
  }

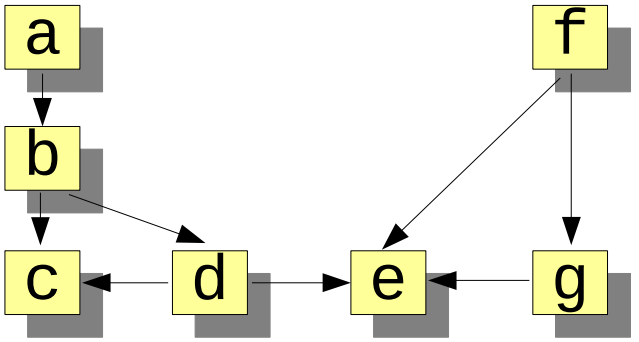
Analyzing the call structure of an application




```
alias graph[&T] = rel[&T, &T];
graph[str] calls = {<"a", "b">, <"b", "c">, <"b", "d">, <"d", "c">,
                    <"d","e">, <"f", "e">, <"f", "g">, <"g", "e">};
```



Some questions

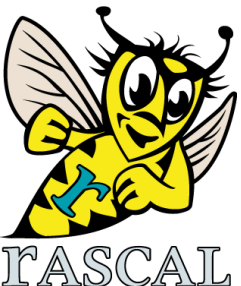


- How many calls are there?
  - int ncalls = size(calls);
  - 8

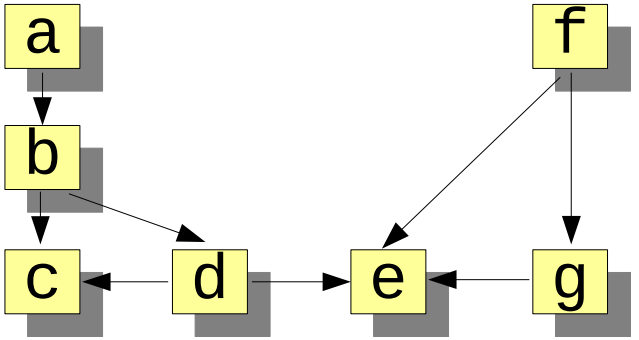
Number of elements

- How many procedures are there?
  - int nprocs = size(carrier(calls));
  - 7

All elements in domain or range of a relations



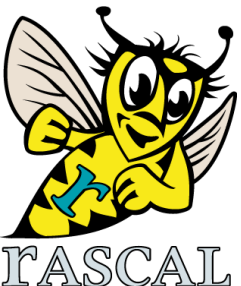
Some questions



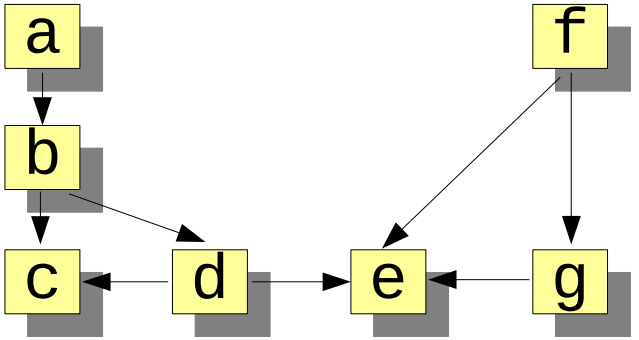
- What are the entry points?
  - set[str] entryPoints = top(calls)
  - {"a", "f"}
- What are the leaves?
  - set[str] bottomCalls = bottom(calls)
  - {"c", "e"}

The *roots* of a relation (viewed as a graph)

The *leaves* of a relation (viewed as a graph)

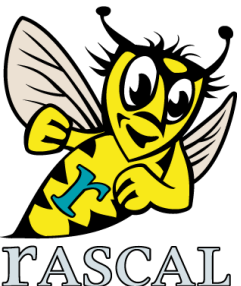


Some questions

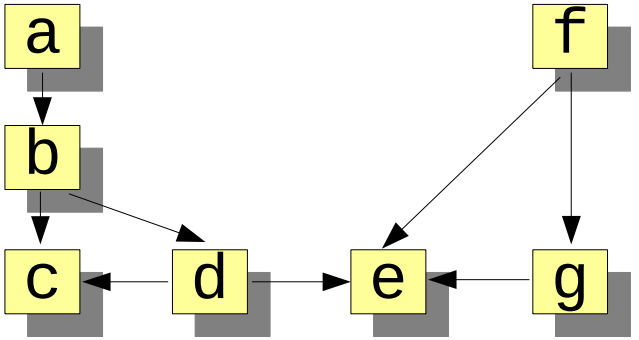


- What are the indirect calls between procedures?
  - graph[str] closureCalls = calls+
  - {<"a", "b">, <"b", "c">, <"b", "d">, <"d", "c">, <"d","e">, <"f", "e">, <"f", "g">, <"g", "e">, <"a", "c">, <"a", "d">, <"b", "e">, <"a", "e">}
- What are the calls from entry point a?
  - set[str] calledFromA = closureCalls["a"]
  - {"b", "c", "d", "e"}

The image of domain value "a"

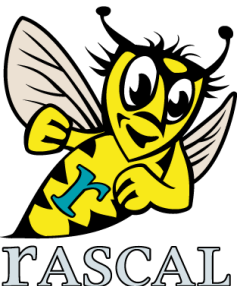


Some questions



- What are the calls from entry point f?
  - set[str] calledFromF = closureCalls["f"];
  - {"e", "g"}
- What are the common procedures?
  - set[str] commonProcs = calledFromA & calledFromF
  - {"e"}

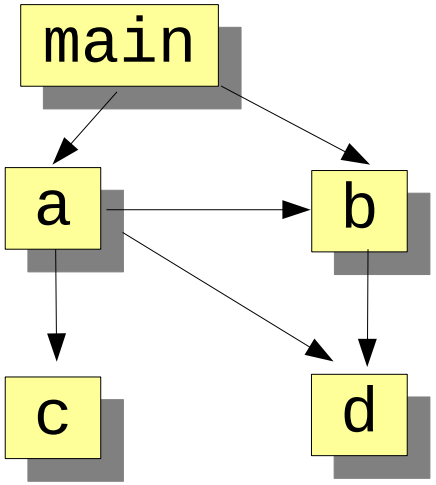
Intersection

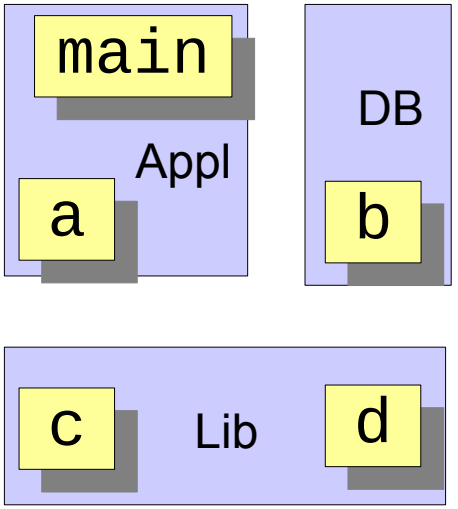


Example

Component structure of an application

PartOf



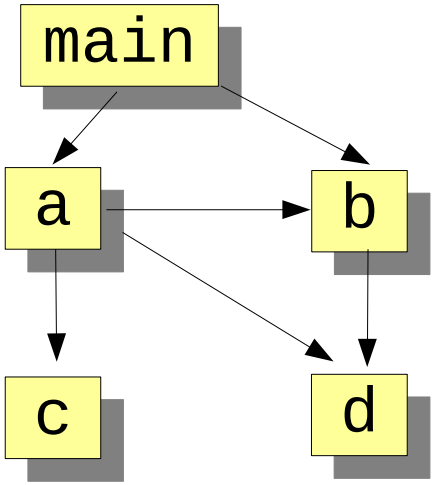



```
set[comp] Components = {"Appl", "DB", "Lib"};

rel[proc, comp] PartOf =
    {<"main", "Appl">, <"a", "Appl">, <"b", "DB">,
                                      <"c", "Lib">, <"d", "Lib">};
```

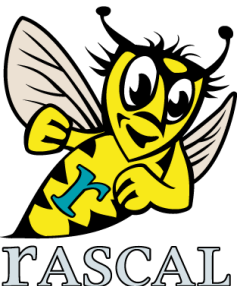
Component Structure of Application

- Suppose, we know:
  - the call relation between procedures (Calls)
  - the component of each procedure (PartOf)
- Question:
  - Can we lift the relation between procedures to a relation between components (ComponentCalls)?
- This is usefull for checking that real code conforms to architectural constraints

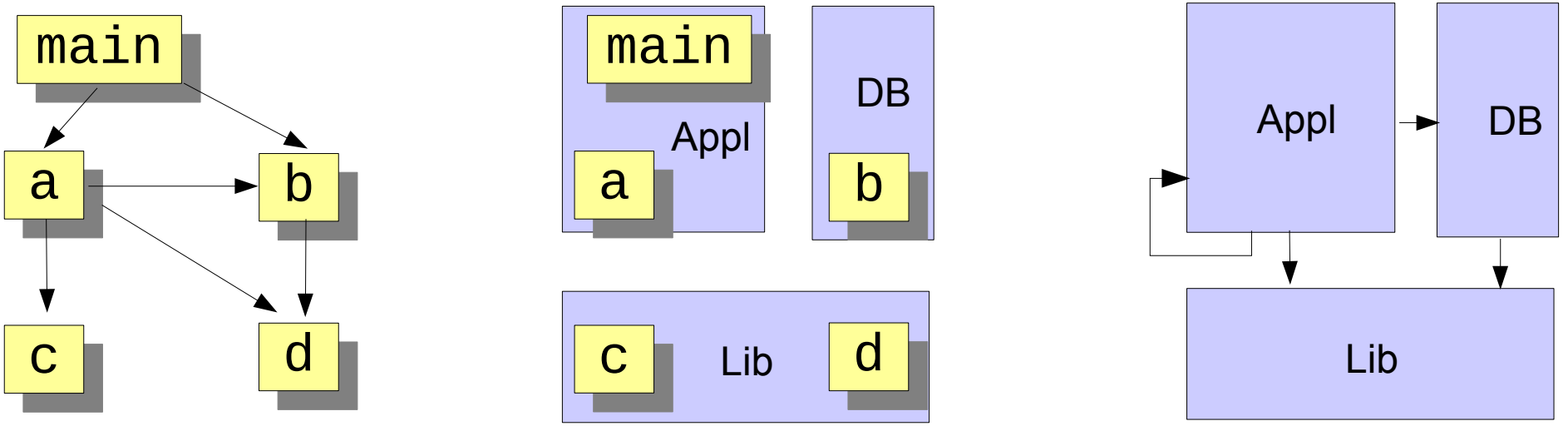
Calls




```
alias proc = str;
alias comp = str;
rel[proc,proc] Calls = {<"main", "a">, <"main", "b">, <"a", "b">,
                        <"a", "c">, <"a", "d">, <"b", "d">};
```

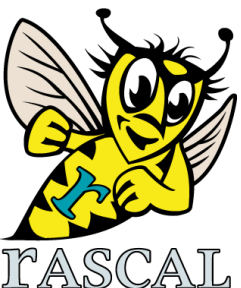


lift

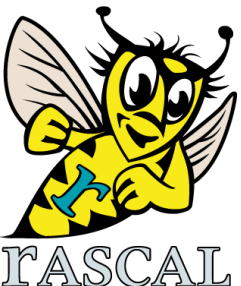



```
rel[comp,comp] lift(rel[proc,proc] aCalls,  rel[proc,comp] aPartOf) =
    { <C1, C2> | <proc P1, proc P2> <- aCalls,
                 <comp C1, comp C2>  <- aPartOf[P1] * aPartOf[P2] };
```

```
rascal>lift(Calls, PartOf);
rel[comp,comp]: {<"DB", "Lib">, <"Appl", "Lib">,
                 <"Appl", "DB">, <"Appl", "Appl">}
```



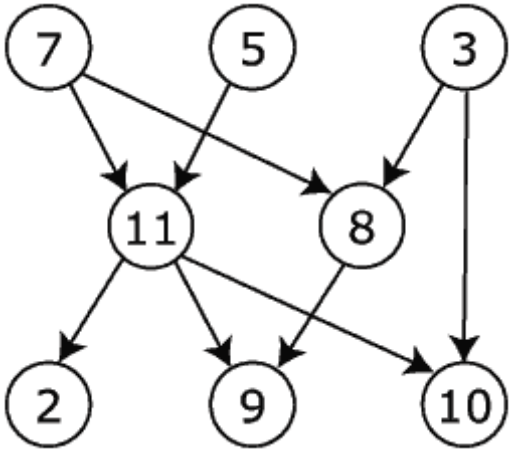
Visualizing relations



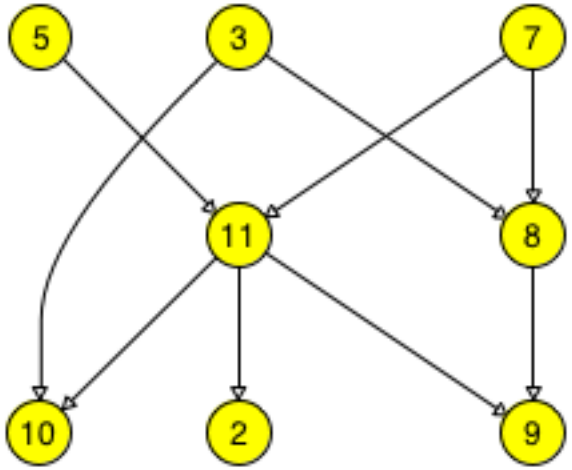
Visualizing Relations

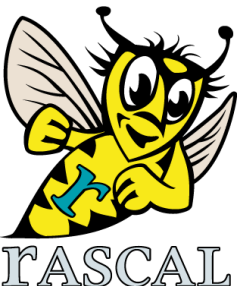
- Most common: as graph
- nodes = [...]; arbitrary figures, all with an id
- edges = [edge(id1, id2), ...]; the ids are used to define the edges, may have label, toArrow or fromArrow attribute
- graph(nodes, edges, hint(...), gap(...));
- hint("layered") or hint("spring")

Example: layered layout



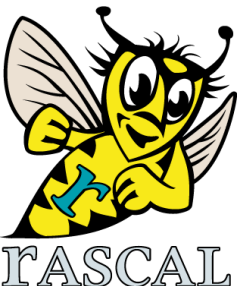

```
public Figure n(str nm) =
    ellipse(text(nm), id(nm), grow(1.2), fillColor("yellow"));

public Edge e(str i, str j,  FProperty props ...) =
    edge(i, j, props + toArrow(triangle(5)));

public Figure topsort() =
    graph([n("2"),  n("3"),  n("5"), n("7"), n("8"),
           n("9"),  n("10"), n("11")],
          [e("3", "8"), e("3", "10"), e("5", "11"),
           e("7", "11"), e("7", "8"), e("8", "9"),
           e("11", "2"), e("11", "9"), e("11","10")],
          hint("layered"), gap(50));
```



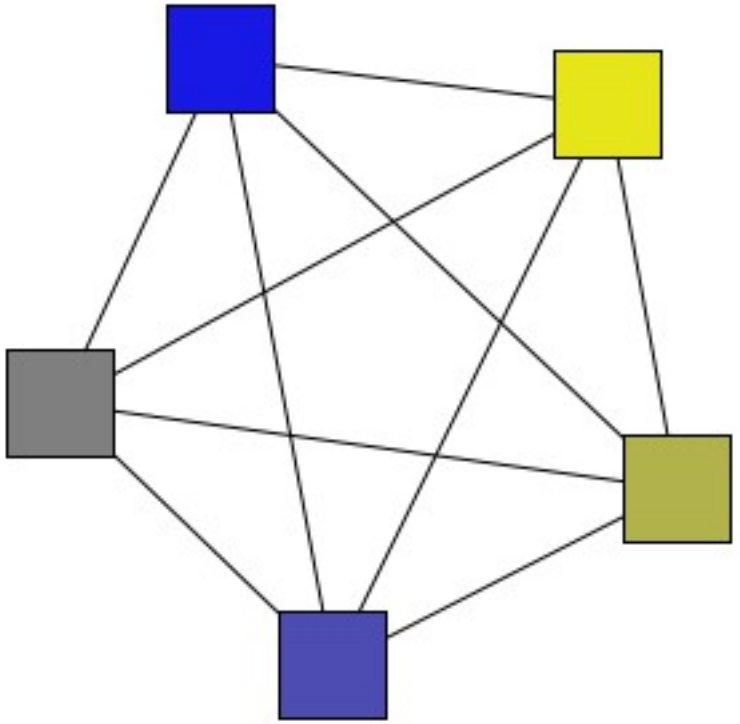


Example: spring layout

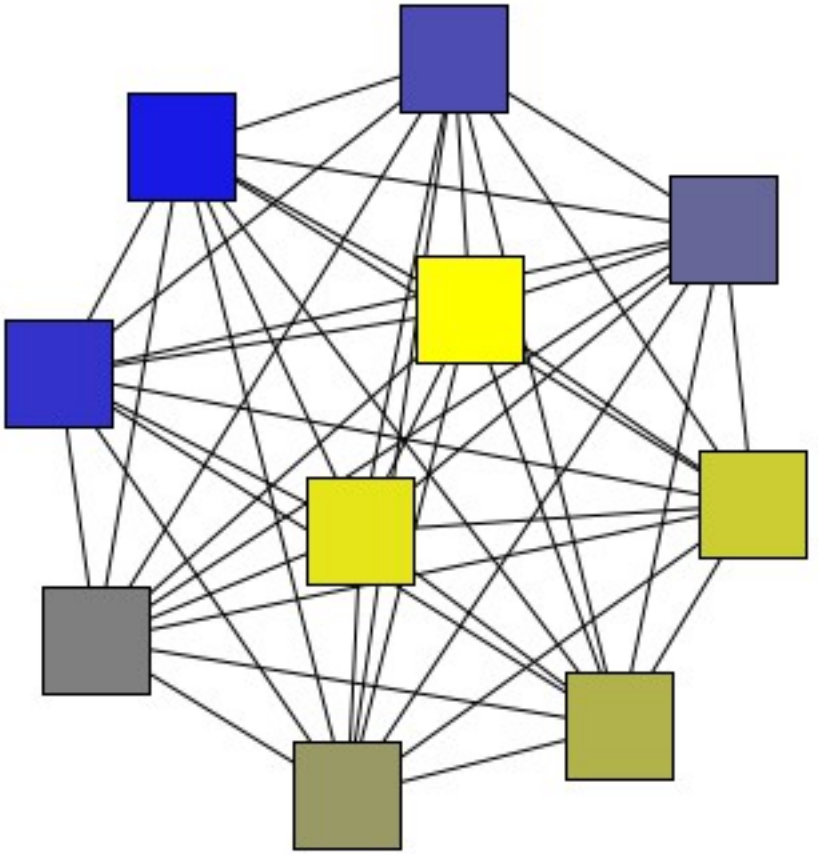
```
public void K(int n){
    colors = colorScale([0 .. n], color("yellow"), color("blue"));

    nodes = [box(id("<i>"), size(40), fillColor(colors(i))) | int i <- [1 .. n] ];

    edges = [ *[edge("<i>", "<j>") | int j <- [ i + 1 .. n]] | int i <- [ 1 .. n - 1 ] ];

    render(graph(nodes,  edges, size(1000), hint("spring")));
}
```

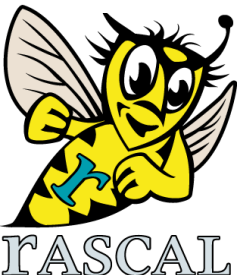


K(5)



K(10)





K(100)