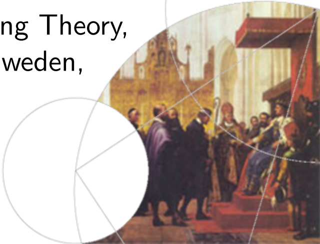
A Functional Language for Specifying Business Reports

Patrick Bahr

University of Copenhagen, Department of Computer Science
paba@diku.dk

23rd Nordic Workshop on Programming Theory,
Mälardalen University, Västerås, Sweden,
October 26 - 28, 2011



Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions

Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions

What are Enterprise Resource Planning Systems?

ERP systems integrate several software components that are essential for managing a business.

What are Enterprise Resource Planning Systems?

ERP systems integrate several software components that are essential for managing a business.

ERP systems integrate

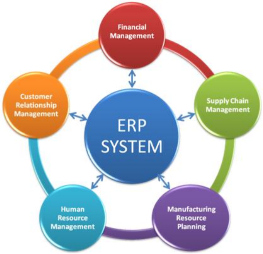
- Financial Management
- Supply Chain Management
- Manufacturing Resource Planning
- Human Resource Management
- Customer Relationship Management
- . . .

What are Enterprise Resource Planning Systems?

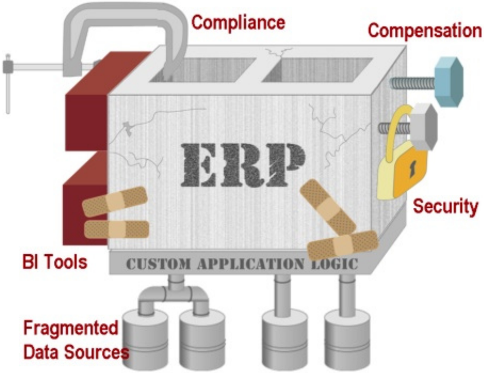
ERP systems integrate several software components that are essential for managing a business.

ERP systems integrate

- Financial Management
- Supply Chain Management
- Manufacturing Resource Planning
- Human Resource Management
- Customer Relationship Management
- . . .


Financial Management
Supply Chain Management
Manufacturing Resource Planning
Human Resource Management
Customer Relationship Management
ERP SYSTEM

What do ERP Systems Look Like?



Issues of Many ERP Implementations

Complexity

- processes are specified in general purpose language
- gap between specification and implementation
- large monolithic system

Issues of Many ERP Implementations

Complexity

- processes are specified in general purpose language
- gap between specification and implementation
- large monolithic system

Inflexibility

- code is duplicated in order to avoid unexpected side effects
- the use of general purpose languages makes customisation expensive
- the (relational) database determines the way data is stored and accessed

Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions

Entering POETS

Process-oriented event-driven transaction systems

compact core system

Entering POETS

Process-oriented event-driven transaction systems

compact core system • customisable via DSLs

Entering POETS

Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model

Entering POETS

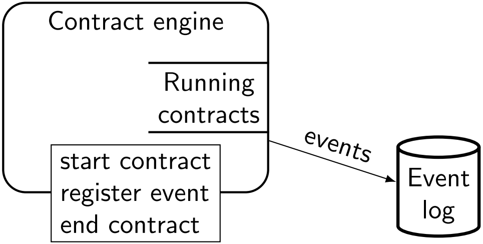
Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model

Entering POETS

Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model



Contract engine

Running contracts

start contract
register event
end contract

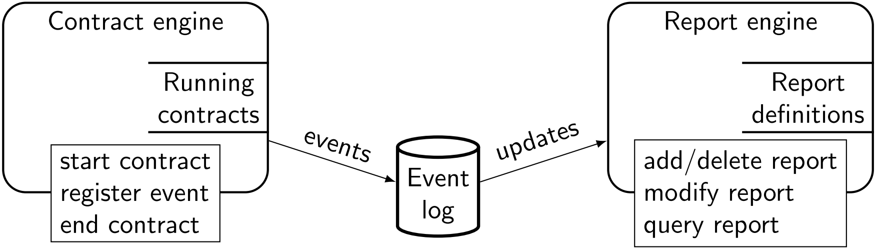
events

Event log

Entering POETS

Process-oriented event-driven transaction systems

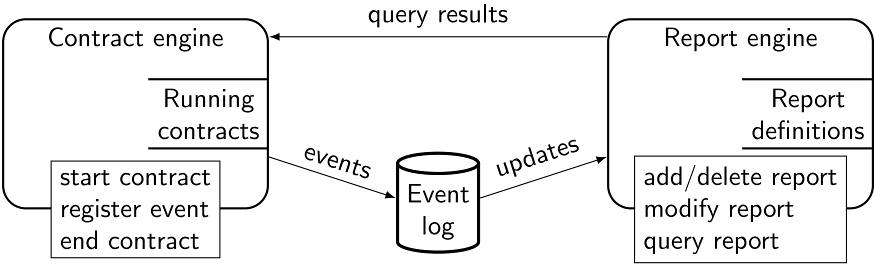
compact core system • customisable via DSLs • simple data model



Entering POETS

Process-oriented event-driven transaction systems

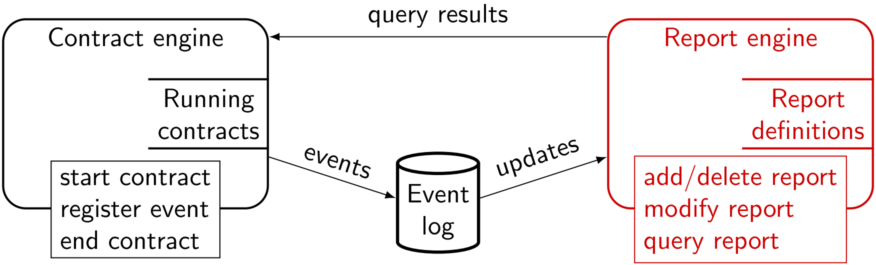
compact core system • customisable via DSLs • simple data model



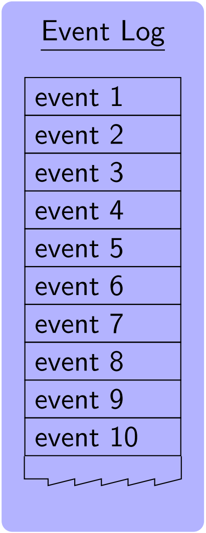
Entering POETS

Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model



What is a Report?



Event Log

event 1

event 2

event 3

event 4

event 5

event 6

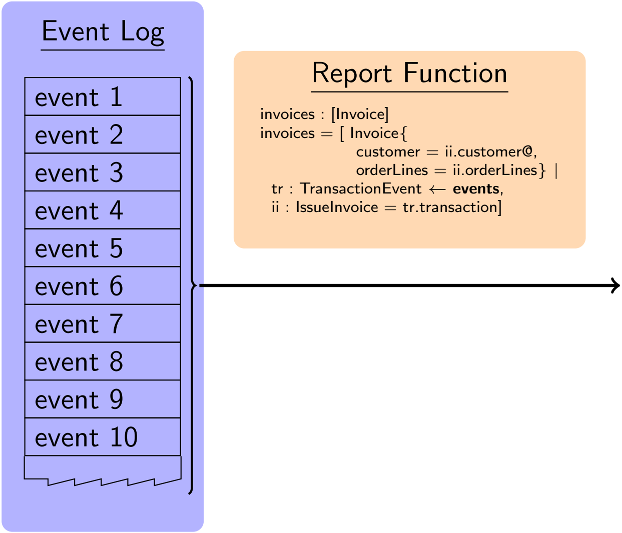
event 7

event 8

event 9

event 10

What is a Report?



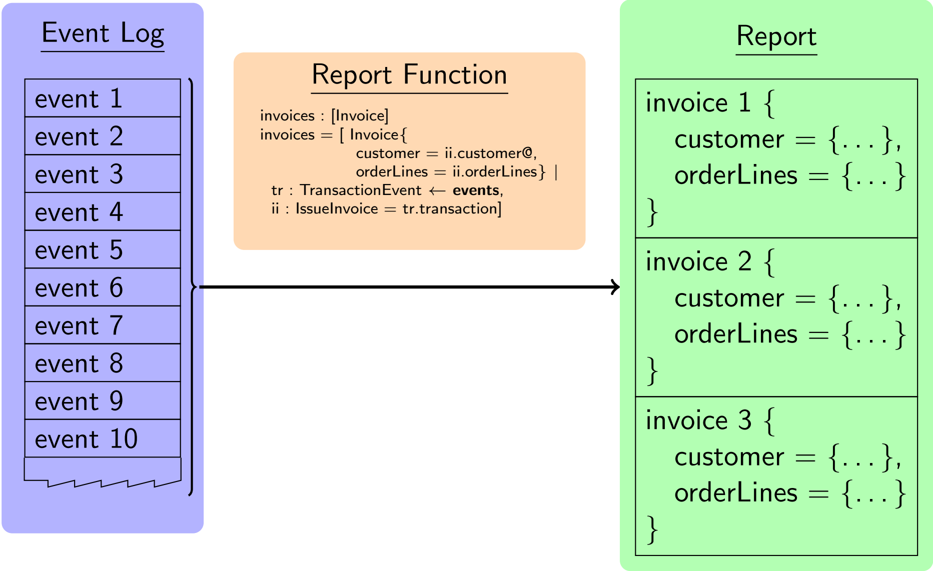
Event Log

event 1
event 2
event 3
event 4
event 5
event 6
event 7
event 8
event 9
event 10

Report Function

```
invoices : [Invoice]
invoices = [ Invoice{
                customer = ii.customer@,
                orderLines = ii.orderLines} |
  tr : TransactionEvent ← events,
  ii : IssueInvoice = tr.transaction]
```

What is a Report?



Event Log

event 1
event 2
event 3
event 4
event 5
event 6
event 7
event 8
event 9
event 10

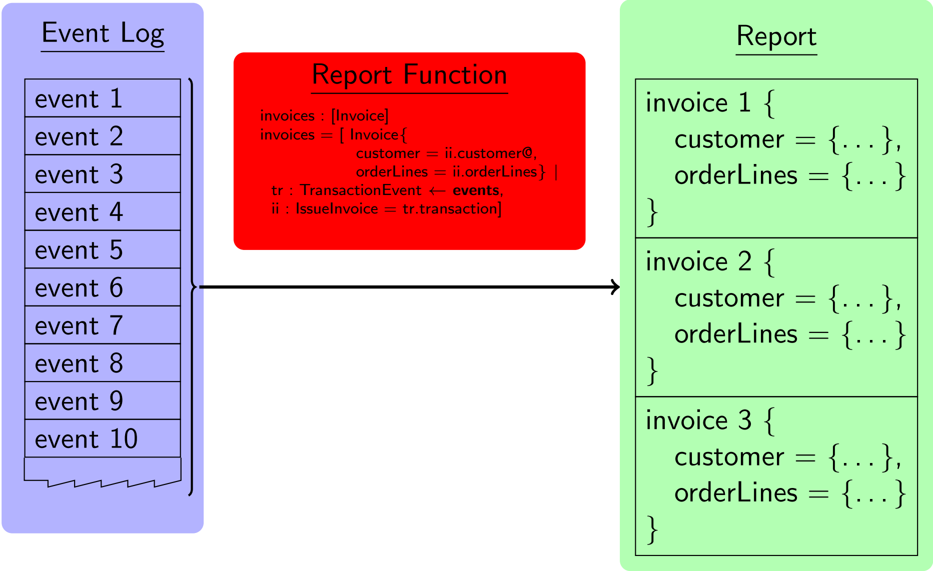
Report Function

```
invoices : [Invoice]
invoices = [ Invoice{
                 customer = ii.customer@,
                 orderLines = ii.orderLines} |
  tr : TransactionEvent ← events,
  ii : IssueInvoice = tr.transaction]
```

Report

```
invoice 1 {
  customer = {...},
  orderLines = {...}
}
invoice 2 {
  customer = {...},
  orderLines = {...}
}
invoice 3 {
  customer = {...},
  orderLines = {...}
}
```

What is a Report?



Event Log

event 1

event 2

event 3

event 4

event 5

event 6

event 7

event 8

event 9

event 10

Report Function

```
invoices : [Invoice]
invoices = [ Invoice{
                customer = ii.customer@,
                orderLines = ii.orderLines} |
  tr : TransactionEvent ← events,
  ii : IssueInvoice = tr.transaction]
```

Report

```
invoice 1 {
   customer = {. . .},
   orderLines = {. . .}
}
invoice 2 {
   customer = {. . .},
   orderLines = {. . .}
}
invoice 3 {
   customer = {. . .},
   orderLines = {. . .}
}
```

The Report Language

The central data types

- records
- lists

The Report Language

The central data types

- records: events are records
- lists

The Report Language

The central data types

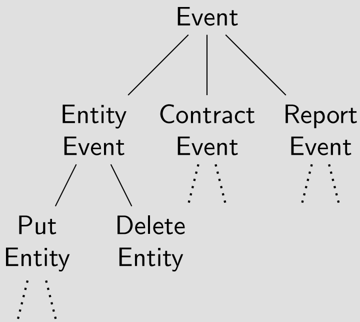
- records: events are records
- lists: the event log is a list of events

The Report Language

The central data types

- records: events are records
- lists: the event log is a list of events

Nominal subtyping

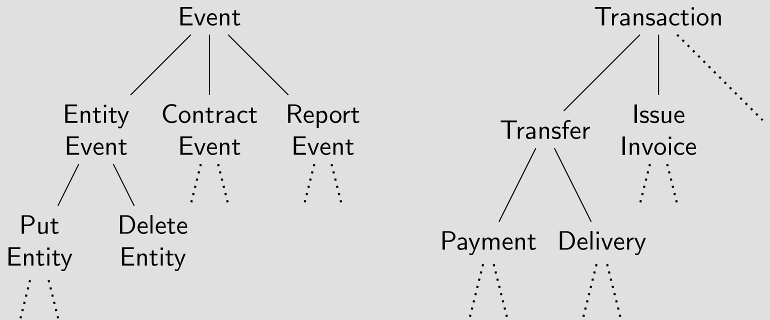


The Report Language

The central data types

- records: events are records
- lists: the event log is a list of events

Nominal subtyping



The Report Language – An Example Function

Example

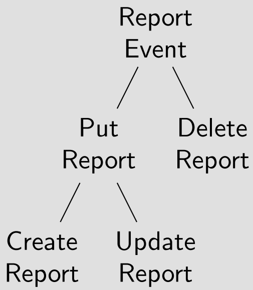
$$\begin{array}{l}
\mathit{reportNames} : [\textbf{String}] \\
\mathit{reportNames} = [\mathit{pr}.\mathit{name} \mid \\
\quad \mathit{cr} : \text{CreateReport} \leftarrow \textbf{events}, \\
\quad \mathit{pr} : \text{PutReport} = \mathit{head}\ [\mathit{ur} \mid \\
\qquad \mathit{ur} : \text{ReportEvent} \leftarrow \textbf{events}, \\
\qquad \mathit{ur}.\mathit{id} \equiv \mathit{cr}.\mathit{id}] \\
\quad]
\end{array}$$

The Report Language – An Example Function

Example

```
reportNames : [String]
reportNames = [pr.name |
    cr : CreateReport ← events,
    pr : PutReport = head [ur |
        ur : ReportEvent ← events,
        ur.id ≡ cr.id]
    ]
```

Report Event Hierarchy



Nominal Subtyping with Benefits

Nominal subtype relation $<:$

- User defined subtyping partial order on records
- Fixed subtyping relation on built-in types

Nominal Subtyping with Benefits

Nominal subtype relation $<:$

- User defined subtyping partial order on records
- Fixed subtyping relation on built-in types

Record Constraints

$$\tau_1.f : \tau_2$$

Nominal Subtyping with Benefits

Nominal subtype relation $<:$

- User defined subtyping partial order on records
- Fixed subtyping relation on built-in types

Record Constraints

$$\tau_1.f : \tau_2$$

E.g. field selector operator $\_.f$ has type

$$\alpha.f : \beta \Rightarrow \alpha \rightarrow \beta$$

Nominal Subtyping with Benefits

Nominal subtype relation $<:$

- User defined subtyping partial order on records
- Fixed subtyping relation on built-in types

Record Constraints

$$\tau_1.f : \tau_2$$

E.g. field selector operator $\_.f$ has type

$$\alpha.f : \beta \Rightarrow \alpha \to \beta$$

E.g. record modifier operator $\_\ \{f_1 = \_, \ldots, f_n = \_\}$ has type

$$\alpha.f_1 : \alpha_1, \ldots, \alpha.f_n : \alpha_n \Rightarrow \alpha \to \alpha_1 \to \ldots \to \alpha_n \to \alpha$$

Record Field Constraints

What do we gain?

- Field names can be used by different record types.
- Nominal subtyping feels like structural subtyping (unless you want to create a record).

Record Field Constraints

What do we gain?

- Field names can be used by different record types.
- Nominal subtyping feels like structural subtyping (unless you want to create a record).

Example

$$
\begin{aligned}
&\mathit{fullName} : (a.\mathit{firstName} : \textbf{String}, a.\mathit{lastName} : \textbf{String}) \Rightarrow a \rightarrow \textbf{String} \\
&\mathit{fullName}\ x = x.\mathit{firstName} +\!\!+\ \texttt{"\ \ "}\ +\!\!+\ x.\mathit{lastName}
\end{aligned}
$$

Record Field Constraints

What do we gain?

- Field names can be used by different record types.
- Nominal subtyping feels like structural subtyping (unless you want to create a record).

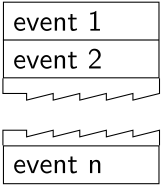
Example

$$\mathit{fullName} : (a.\mathit{firstName} : \textbf{String},\ a.\mathit{lastName} : \textbf{String}) \Rightarrow a \rightarrow \textbf{String}$$
$$\mathit{fullName}\ x = x.\mathit{firstName} +\!\!+ \texttt{" "} +\!\!+ x.\mathit{lastName}$$

$$\begin{aligned}\mathit{setFullName} : {} & (a.\mathit{firstName} : \textbf{String},\ a.\mathit{lastName} : \textbf{String}) \Rightarrow \\ & \textbf{String} \rightarrow a \rightarrow a\end{aligned}$$
$$\begin{aligned}\mathit{setFullName}\ \mathit{name}\ x = {} & \textbf{let}\ (\mathit{first}, \mathit{last}) = \mathit{decompose}\ \mathit{name} \\ & \textbf{in}\ x\ \{\mathit{firstName} = \mathit{first},\ \mathit{lastName} = \mathit{last}\}\end{aligned}$$

Making It Scale

ain’t easy



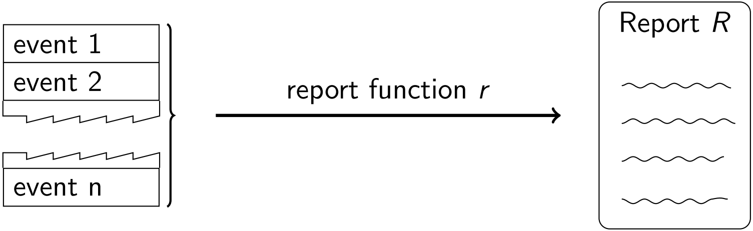
event 1

event 2

event n

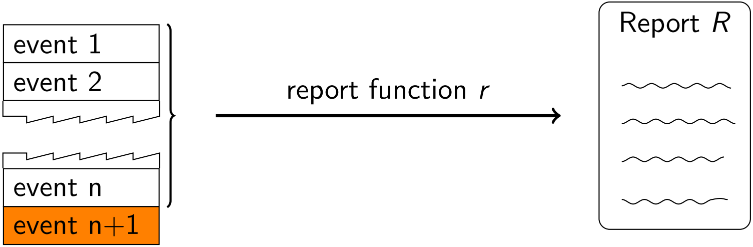
Making It Scale

ain't easy



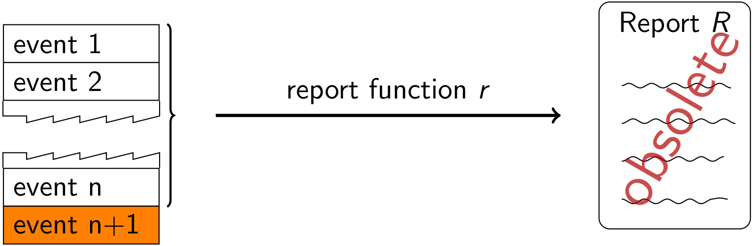
Making It Scale

ain't easy



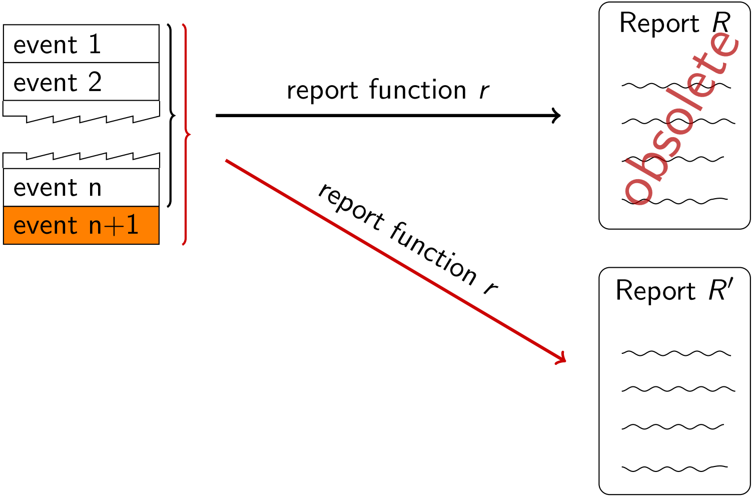
Making It Scale

ain't easy



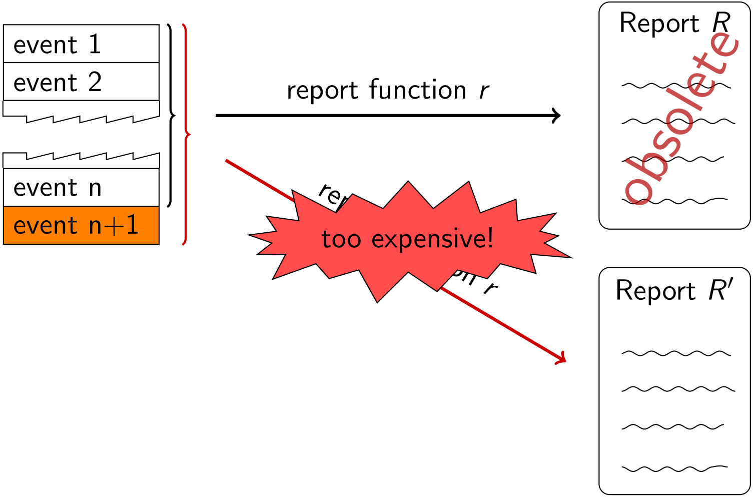
Making It Scale

ain't easy



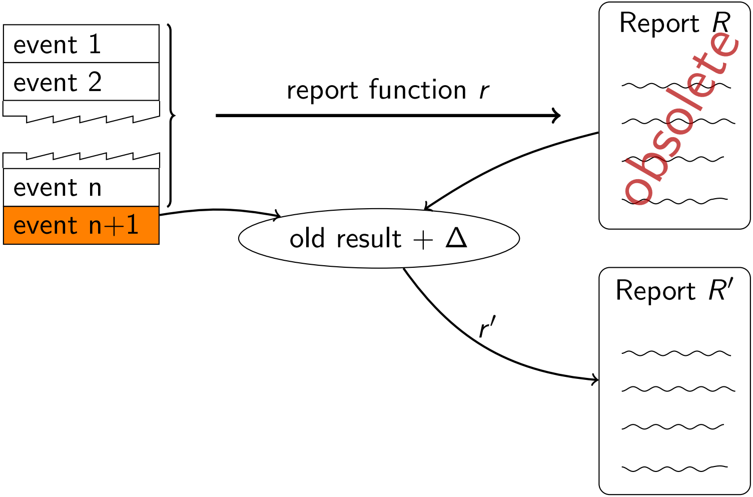
Making It Scale

ain’t easy



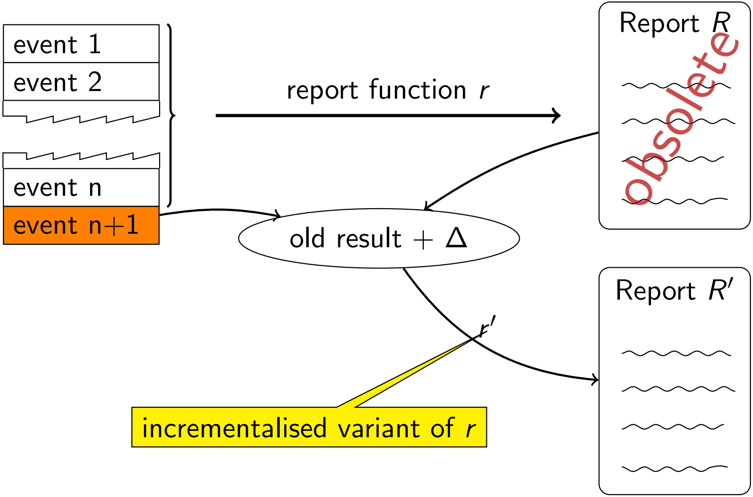
Making It Scale

ain't easy



Making It Scale

ain't easy



Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

$$\textbf{fold}\ f\ \ e\ (x \mathbin{\#} xs)$$

Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

$$\textbf{fold}\ f\ \ e\ (x \# xs) = f \quad \underbrace{\ \ x\ \ } \qquad \underbrace{(\textbf{fold}\ f\ \ e\ xs)}$$

Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

$$\mathbf{fold}\ f\ e\ (x \mathbin{\#} xs) = f\ \underbrace{x}_{\text{new element}}\ \underbrace{(\mathbf{fold}\ f\ e\ xs)}_{\text{old (intermediate) result}}$$

Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

$$\textbf{fold}\ f\ e\ (x \# xs) = f\ \underbrace{x}_{\text{new element}}\ \underbrace{(\textbf{fold}\ f\ e\ xs)}_{\text{old (intermediate) result}}$$

Limitations

- This works well with single folds.
- For nested folds more powerful equations are needed.

Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

$$\textbf{fold}\ f\ e\ (x \# xs) = f\ \underbrace{x}_{\text{new element}}\ \underbrace{(\textbf{fold}\ f\ e\ xs)}_{\text{old (intermediate) result}}$$

Limitations

- This works well with single folds.
- For nested folds more powerful equations are needed.
  - commutative operations
  - multisets instead of lists

Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions

Conclusions

The Last Slide

What do we have?

- Simple yet powerful data model for ERP
- Purely functional language for extracting & aggregating complex information
- Highly customisable & flexible
- Incrementalisation of report functions

Conclusions

The Last Slide

What do we have?

- Simple yet powerful data model for ERP
- Purely functional language for extracting & aggregating complex information
- Highly customisable & flexible
- Incrementalisation of report functions

What are we planning?

- More powerful incrementalisation transformations
- Possibly restricting the language further
- A better cost model