Provenance-Awareness in R

Chris A. Silles
Andrew R. Runnalls

School of Computing, University of Kent, UK
Outline

1. A Brief History of R
2. Provenance-Aware R
3. Issues
4. Conclusion
What is R?

- A language and interactive environment for statistical computing and graphics
  - Language: Lazy Functional; Strongly-typed; C-like syntax
  - Environment: Read *expressions* from a prompt and evaluates them; package based for extensibility
  - Snippet of R code:
    ```r
    > x <- 1:5
    > y <- x * 2
    > y
    [1]  2  4  6  8 10
    ```

- Is used extensively in many industries, and enjoys an extremely active userbase

- Most interestingly: R is an open-source implementation of S
What is R?

- A language and interactive environment for statistical computing and graphics
  - **Language**: Lazy Functional; Strongly-typed; C-like syntax
  - **Environment**: Read expressions from a prompt and evaluates them; package based for extensibility
  - Snippet of R code:
    ```
    > x <- 1:5
    > y <- x * 2
    > y
    [1]  2  4  6  8 10
    ```
- Is used extensively in many industries, and enjoys an extremely active userbase
- Most interestingly: R is an open-source implementation of S
What is R?

- A language and interactive environment for statistical computing and graphics
  - **Language**: Lazy Functional; Strongly-typed; C-like syntax
  - **Environment**: Read *expressions* from a prompt and evaluates them; package based for extensibility
  - Snippet of R code:
    ```r
    > x <- 1:5
    > y <- x * 2
    > y
    [1]  2  4  6  8 10
    ```
- Is used extensively in many industries, and enjoys an extremely active userbase
- Most interestingly: R is an open-source implementation of S

---

Chris A. Silles  (University of Kent)  Provenance-Awareness in R  15 June 2010  3 / 14
What is R?

- A language and interactive environment for statistical computing and graphics
  - **Language**: Lazy Functional; Strongly-typed; C-like syntax
  - **Environment**: Read *expressions* from a prompt and evaluates them; package based for extensibility
  - Snippet of R code:
    ```r
    > x <- 1:5
    > y <- x * 2
    > y
    [1] 2 4 6 8 10
    ```

- Is used extensively in many industries, and enjoys an extremely active userbase

- Most interestingly: R is an open-source implementation of S
What is R?

- A language and interactive environment for statistical computing and graphics
  - **Language**: Lazy Functional; Strongly-typed; C-like syntax
  - **Environment**: Read *expressions* from a prompt and evaluates them; package based for extensibility
  - Snippet of R code:
    ```r
    > x <- 1:5
    > y <- x * 2
    > y
    [1]  2  4  6  8 10
    ```
- Is used extensively in many industries, and enjoys an extremely active userbase
- Most interestingly: R is an open-source implementation of S
What is R?

- A language and interactive environment for statistical computing and graphics
  - **Language**: Lazy Functional; Strongly-typed; C-like syntax
  - **Environment**: Read *expressions* from a prompt and evaluates them; package based for extensibility
  - Snippet of R code:
    ```
    > x <- 1:5
    > y <- x * 2
    > y
    [1] 2 4 6 8 10
    ```
- Is used extensively in many industries, and enjoys an extremely active userbase
  - Most interestingly: R is an open-source implementation of S
What is R?

- A language and interactive environment for statistical computing and graphics
  - **Language**: Lazy Functional; Strongly-typed; C-like syntax
  - **Environment**: Read *expressions* from a prompt and evaluates them; package based for extensibility
  - Snippet of R code:
    ```
    > x <- 1:5
    > y <- x * 2
    > y
    [1]  2  4  6  8 10
    ```
- Is used extensively in many industries, and enjoys an extremely active userbase
- Most interestingly: R is an open-source implementation of S
Early Provenance-Aware Computing: New S

In 1988 New-S succeeded S, and it became one of – if not – the first provenance-aware software applications with its novel S AUDIT facility. Its authors, Becker and Chambers describe it in their paper Auditing of Data Analyses¹.

An audit file was maintained by New-S which recorded each top-level command issued in this and previous sessions within the workspace, and identified those objects read from and written to.

Example S AUDIT File

```r
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- Top-level commands
- Data objects read
- Data objects written
Example S AUDIT File

```r
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- **Top-level commands**
- **Data objects read**
- **Data objects written**
Example S AUDIT File

```r
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- Top-level commands
- **Data objects read**
- Data objects written
Example S AUDIT File

```r
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- Top-level commands
- Data objects read
- Data objects written
Environments and Bindings

R is **dynamically typed**. This is handled by **Bindings**.

During the evaluation of:

```r
x <- 5
```

- `x` is a symbol
- `5` is a vector (of one element)
- A binding associates a value with a symbol
- This binding is stored in the **global environment**

There are multiple environments; each containing their own bindings. Environments may also enclose other environments.
Environments and Bindings

R is dynamically typed. This is handled by **Bindings**.

During the evaluation of:

```
x <- 5
```

- `x` is a symbol
- 5 is a vector (of one element)
- A binding associates a value with a symbol
- This binding is stored in the global environment

There are multiple environments; each containing their own bindings. Environments may also enclose other environments.
R is dynamically typed. This is handled by Bindings.

During the evaluation of:

\[ x \leftarrow 5 \]

- \( x \) is a symbol
- \( 5 \) is a vector (of one element)
- A binding associates a value with a symbol
- This binding is stored in the global environment

There are multiple environments; each containing their own bindings. Environments may also enclose other environments.
Environments and Bindings

R is dynamically typed. This is handled by Bindings.

During the evaluation of:

```r
x <- 5
```

- *x* is a symbol
- *5* is a vector (of one element)
- A binding associates a value with a symbol
  - This binding is stored in the global environment

There are multiple environments; each containing their own bindings. Environments may also enclose other environments.
R is dynamically typed. This is handled by Bindings.

During the evaluation of:

\[ x \leftarrow 5 \]

- \( x \) is a symbol
- \( 5 \) is a vector (of one element)
- A binding associates a value with a symbol
- This binding is stored in the global environment

There are multiple environments; each containing their own bindings. Environments may also enclose other environments.
Environments and Bindings

R is dynamically typed. This is handled by Bindings.
During the evaluation of:
\[ x \leftarrow 5 \]
- \( x \) is a symbol
- \( 5 \) is a vector (of one element)
- A binding associates a value with a symbol
- This binding is stored in the global environment

There are multiple environments; each containing their own bindings. Environments may also enclose other environments.
What Provenance?

- **Bindings** connect **Symbols** with **Values**

So when talking about Provenance of data items, we need to know about **Bindings**, rather than then **Value** objects.

The principal objectives of a Provenance-Aware R are to identify the following of a given binding:

- **Pedigree**: The full sequence of commands responsible
- **Parents**: Bindings that have been read during its creation
- **Children**: Bindings that have read it during their creation
What Provenance?

- **Bindings** connect **Symbols** with **Values**

- So when talking about Provenance of data items, we need to know about **Bindings**, rather than then **Value** objects.

The principal objectives of a Provenance-Aware R are to identify the following of a given binding:

- Pedigree: The full sequence of commands responsible
- Parents: Bindings that have been read during its creation
- Children: Bindings that have read it during their creation
What Provenance?

- **Bindings** connect **Symbols** with **Values**

- So when talking about Provenance of data items, we need to know about **Bindings**, rather than then **Value** objects.

The principal objectives of a Provenance-Aware R are to identify the following of a given binding:

- Pedigree: The full sequence of commands responsible
- Parents: Bindings that have been read during its creation
- Children: Bindings that have read it during their creation
What Provenance?

- **Bindings** connect **Symbols** with **Values**

- So when talking about Provenance of data items, we need to know about **Bindings**, rather than then **Value** objects.

The principal objectives of a Provenance-Aware R are to identify the following of a given binding:

- **Pedigree**: The full sequence of commands responsible
- **Parents**: Bindings that have been read during its creation
- **Children**: Bindings that have read it during their creation
What Provenance?

- **Bindings** connect **Symbols** with **Values**

- So when talking about Provenance of data items, we need to know about **Bindings**, rather than then **Value** objects.

The principal objectives of a Provenance-Aware R are to identify the following of a given binding:

- **Pedigree**: The full sequence of commands responsible
- **Parents**: Bindings that have been read during its creation
- **Children**: Bindings that have read it during their creation
What Provenance?

- Bindings connect Symbols with Values
- So when talking about Provenance of data items, we need to know about Bindings, rather than then Value objects.

The principal objectives of a Provenance-Aware R are to identify the following of a given binding:

- **Pedigree**: The full sequence of commands responsible
- **Parents**: Bindings that have been read during its creation
- **Children**: Bindings that have read it during their creation
What Provenance?

- **Bindings** connect **Symbols** with **Values**

- So when talking about Provenance of data items, we need to know about **Bindings**, rather than then **Value** objects.

The principal objectives of a Provenance-Aware R are to identify the following of a given binding:

- **Pedigree**: The full sequence of commands responsible
- **Parents**: Bindings that have been read during its creation
- **Children**: Bindings that have read it during their creation
Strategy

What we need to go about this:

- A **mechanism** for trapping reads and writes in the **user workspace** (i.e. the **global environment**)
- **Containers** for storing provenance information
  - Associated with bindings
- **New R commands** for inspecting provenance
  - `provenance(x)`: Returns a list comprising: expression, symbol, timestamp, parents, children
  - `pedigree(x)`: Displays the sequence of commands that has led to binding x’s current state
Strategy

What we need to go about this:

- **A mechanism** for trapping reads and writes in the user workspace (i.e. the **global environment**)
- **Containers** for storing provenance information
  - Associated with bindings
- **New R commands** for inspecting provenance
  - `provenance(x)`: Returns a list comprising: expression, symbol, timestamp, parents, children
  - `pedigree(x)`: Displays the sequence of commands that has led to binding x’s current state
Strategy

What we need to go about this:

- **A mechanism** for trapping reads and writes in the user workspace (i.e. the global environment)
- **Containers** for storing provenance information
  - Associated with bindings
- **New R commands** for inspecting provenance
  - `provenance(x)`: Returns a list comprising: expression, symbol, timestamp, parents, children
  - `pedigree(x)`: Displays the sequence of commands that has led to binding x’s current state
Making it Easier: The CXXR Project

Rather than working with the original C version of R, we have introduced provenance-awareness to CXXR. Founded in 2007, CXXR\(^2\) aims to progressively reengineer the R interpreter from C into C++, with the intention that full functionality of the standard R distribution is preserved and so that behaviour of R code is unaffected.

CXXR allows Monitor functions to be set on bindings, which are triggered when a read or write occurs (in a given environment).

\(^2\)www.cs.kent.ac.uk/projects/cxxr
Making it Easier: The CXXR Project

Rather than working with the original C version of R, we have introduced provenance-awareness to CXXR. Founded in 2007, CXXR\(^2\) aims to progressively reengineer the R interpreter from C into C++, with the intention that Full functionality of the standard R distribution is preserved and so that behaviour of R code is unaffected.

CXXR allows Monitor functions to be set on bindings, which aretriggered when a read or write occurs (in a given environment).

\(^2\)www.cs.kent.ac.uk/projects/cxxr
Making it Easier: The CXXR Project

Rather than working with the original C version of R, we have introduced provenance-awareness to CXXR. Founded in 2007, CXXR\(^2\) aims to progressively reengineer the R interpreter from C into C++, with the intention that Full functionality of the standard R distribution is preserved and so that behaviour of R code is unaffected.

CXXR allows Monitor functions to be set on bindings, which are triggered when a read or write occurs (in a given environment).

\(^2\)www.cs.kent.ac.uk/projects/cxxr
Architecture of PA-CXXR

For each top-level expression:

- Maintain:
  - **Seen** set: Bindings read, or written
  - **Parentage** list: Bindings read (in order)

- On read of binding $x$:
  - If $x$ is not in **Seen**:
    - Add references to $x$’s Binding to Parentage and Seen

- On write of Binding $x$:
  - Create a **Provenance** object, which:
    - References the top level expression being evaluated
    - References the symbol with which it is associated
    - Records the current timestamp
    - References the Parentage
  - Register $x$ as a child of each of its parents
  - Associate the Provenance with the Binding $x$
  - Add reference to $x$ to Seen
Architecture of PA-CXXR

For each top-level expression:

- Maintain:
  - **Seen** set: Bindings read, or written
  - **Parentage** list: Bindings read (in order)

- On read of binding \( x \):
  - If \( x \) is not in **Seen**:
    - Add references to \( x \)'s Binding to Parentage and **Seen**

- On write of Binding \( x \):
  - Create a **Provenance** object, which:
    - References the top level expression being evaluated
    - References the symbol with which it is associated
    - Records the current timestamp
    - References the Parentage
  - Register \( x \) as a child of each of its parents
  - Associate the **Provenance** with the Binding \( x \)
  - Add reference to \( x \) to **Seen**
Architecture of PA-CXXR

For each top-level expression:

- **Maintain:**
  - **Seen** set: Bindings read, or written
  - **Parentage** list: Bindings read (in order)

- **On read of binding** \( x \):
  - If \( x \) is not in **Seen**:
    - Add references to \( x \)'s Binding to **Parentage** and **Seen**

- **On write of Binding** \( x \):
  - Create a **Provenance** object, which:
    - References the top level expression being evaluated
    - References the symbol with which it is associated
    - Records the current timestamp
    - References the **Parentage**
  - Register \( x \) as a child of each of its parents
  - Associate the **Provenance** with the Binding \( x \)
  - Add reference to \( x \) to **Seen**
Loops

Iterative loops cause bindings to be read from and written to multiple times. It is not necessary to record this information in order to establish accurate parentage (and offspring); however, it may be useful to do so (e.g. How many loop iterations were there?). A binding should appear as a parent only once. This behaviour is accurately modelled by the Seen set in the algorithm.
Loops

Iterative loops cause bindings to be read from and written to multiple times. It is not necessary to record this information in order to establish accurate parentage (and offspring); however, it may be useful to do so (e.g. How many loop iterations were there?). A binding should appear as a parent only once. This behaviour is accurately modelled by the Seen set in the algorithm.
Iterative loops cause bindings to be read from and written to multiple times. It is not necessary to record this information in order to establish accurate parentage (and offspring); however, it may be useful to do so (e.g. How many loop iterations were there?). A binding should appear as a parent only once. This behaviour is accurately modelled by the Seen set in the algorithm.
Loops

Iterative loops cause bindings to be read from and written to multiple times.
It is not necessary to record this information in order to establish accurate parentage (and offspring); however, it may be useful to do so (e.g. How many loop iterations were there?).
A binding should appear as a parent only once.
This behaviour is accurately modelled by the Seen set in the algorithm.
Lazy Evaluation

At the heart of R’s lazy evaluation mechanism is a Promise object comprising:

- An **Expression** that is to be evaluated
- An **Environment** in which to evaluate it

Promises present a slight challenge

- Not evaluated until necessary
- Evaluation may result in creation of a new binding
- Originally, this new binding was excluded from the parentage because of the Seen set mechanism
- A special case exists for handling Promises
Lazy Evaluation

At the heart of R’s lazy evaluation mechanism is a Promise object comprising:

- An Expression that is to be evaluated
- An Environment in which to evaluate it

Promises present a slight challenge

- Not evaluated until necessary
- Evaluation may result in creation of a new binding
- Originally, this new binding was excluded from the parentage because of the Seen set mechanism
- A special case exists for handling Promises
Lazy Evaluation

At the heart of R’s lazy evaluation mechanism is a Promise object comprising:

- An Expression that is to be evaluated
- An Environment in which to evaluate it

Promises present a slight challenge

- Not evaluated until necessary
- Evaluation may result in creation of a new binding
- Originally, this new binding was excluded from the parentage because of the Seen set mechanism
- A special case exists for handling Promises
Lazy Evaluation

At the heart of R’s lazy evaluation mechanism is a Promise object comprising:

- An Expression that is to be evaluated
- An Environment in which to evaluate it

Promises present a slight challenge

- Not evaluated until necessary
- Evaluation may result in creation of a new binding

- Originally, this new binding was excluded from the parentage because of the Seen set mechanism
- A special case exists for handling Promises
Lazy Evaluation

At the heart of R’s lazy evaluation mechanism is a Promise object comprising:

- An Expression that is to be evaluated
- An Environment in which to evaluate it

Promises present a slight challenge

- Not evaluated until necessary
- Evaluation may result in creation of a new binding
- Originally, this new binding was excluded from the parentage because of the Seen set mechanism

- A special case exists for handling Promises
Lazy Evaluation

At the heart of R’s lazy evaluation mechanism is a Promise object comprising:

- An Expression that is to be evaluated
- An Environment in which to evaluate it

Promises present a slight challenge

- Not evaluated until necessary
- Evaluation may result in creation of a new binding
- Originally, this new binding was excluded from the parentage because of the Seen set mechanism
- A special case exists for handling Promises
source(...)

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions evaluated
  - Accurate in terms of the user's session

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
source(...) 

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This occurs outside of R's usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`.
  - Cannot recall the precise expressions.
  - Accurate in terms of the user's session.

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated.
  - Appears as though the expressions were simply evaluated at the command line.
  - No record of the exact call to `source`—although `source` function is recorded as a parent!
source(...)  

R has a function \texttt{source(filepath)}. The file given as an argument is opened, parsed, and each line is turned into an \texttt{expression} that is then \texttt{evaluated}. This is occurs outside of R’s usual \textbf{Read-Evaluate-Print Loop} (REPL).

Two options for handling this:

- **Black box:**

- **White box:**
R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent
source(...) 

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box**:
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user’s session

- **White box**:
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
source(...) 

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`.
  - Cannot recall the precise expressions.
  - Accurate in terms of the user’s session.

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated.
  - Appears as though the expressions were simply evaluated at the command line.
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box**:
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user’s session

- **White box**:
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
source(....)

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user’s session

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
source(...)

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This is occurs outside of R's usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user's session

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
source(...)

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This is occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user’s session

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
**source(....)**

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This is occurs outside of R’s usual **Read-Evaluate-Print Loop (REPL)**.

Two options for handling this:

- **Black box:**
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user’s session

- **White box:**
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
source(...)

R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This is occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box**:
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user’s session

- **White box**:
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
R has a function `source(filepath)`. The file given as an argument is opened, parsed, and each line is turned into an expression that is then evaluated. This is occurs outside of R’s usual Read-Evaluate-Print Loop (REPL).

Two options for handling this:

- **Black box**:
  - Created bindings recorded as resulting only from the call to `source`
  - Cannot recall the precise expressions
  - Accurate in terms of the user’s session

- **White box**:
  - Created bindings recorded as resulting from the precise expression evaluated
  - Appears as though the expressions were simply evaluated at the command line
  - No record of the exact call to `source` — although `source` function is recorded as a parent!
Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an artifact’s pedigree, parents and children. We now need to look into the following:

- Reproducing data from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Recording provenance in other R environments
- Serializing provenance information
  - Serialization formats
  - OPM-compatibility
Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an artifact’s pedigree, parents and children. We now need to look into the following:

- Reproducing data from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Recording provenance in other R environments
- Serializing provenance information
  - Serialization formats
  - OPM-compatibility
Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an artifact’s pedigree, parents and children. We now need to look into the following:

- Reproducing data from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Recording provenance in other R environments
- Serializing provenance information
  - Serialization formats
  - OPM-compatibility
Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an artifact’s pedigree, parents and children. We now need to look into the following:

- Reproducing data from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Recording provenance in other R environments
- Serializing provenance information
  - Serialization formats
  - OPM-compatibility
Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an artifact’s pedigree, parents and children.

We now need to look into the following:

- Reproducing data from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Recording provenance in other R environments
  - Serializing provenance information
    - Serialization formats
    - OPM-compatibility
Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an artifact’s **pedigree, parents** and **children**. We now need to look into the following:

- Reproducing data from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Recording provenance in other R environments
- Serializing provenance information
  - Serialization formats
  - OPM-compatibility