An Abstract Semantics for Inference of Types and Effects in a Multi-Tier Web Language

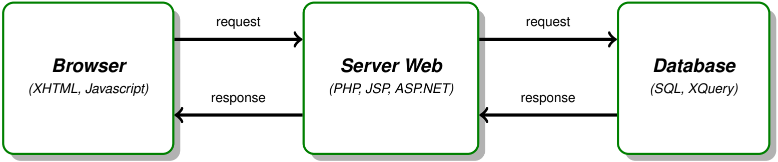
Letterio Galletta Giorgio Levi

Dipartimento di Informatica, Università di Pisa
{galletta, levi}@di.unipi.it

Workshop on Automated Specification and Verification of Web Systems, 2011

Multi-tier architecture

Standard web applications have a multi-tier architecture



Each tier runs on a different computational environment characterized by its language and its data representation (impedance mismatch problem)

LINKS: a Multi-Tier Web Language

Multi-tier web languages allow one to blend server, client and database code and provide automatic mechanisms for the partition of the application over tiers

LINKS: a Multi-Tier Web Language

Multi-tier web languages allow one to blend server, client and database code and provide automatic mechanisms for the partition of the application over tiers

LINKS

LINKS is a functional multi-tier web language

- from a single source code the compiler generates code for each tier
- support an unified cross-tier programming model by exploiting web continuations

LINKS: a Multi-Tier Web Language

Multi-tier web languages allow one to blend server, client and database code and provide automatic mechanisms for the partition of the application over tiers

LINKS

LINKS is a functional multi-tier web language

- from a single source code the compiler generates code for each tier
- support an unified cross-tier programming model by exploiting web continuations

Web continuations in LINKS

Closures (expression to be executed plus bindings of free variables) stored in HTML pages

LINKS: security

Baltopoulos and Gordon have shown that

- storing web continuation in HTML page is not secure
- an attacker can violate
  1. Secrecy
  2. Data Integrity
  3. Control Integrity

LINKS: security

Solution

To overtake the security issues they have proposed a secure implementation that includes

1. a compilation strategy based on authenticated encryption
2. a types-and-effects system to enable source level reasoning about security of web applications

LINKS: security

Solution

To overtake the security issues they have proposed a secure implementation that includes

1. a compilation strategy based on authenticated encryption
2. a types-and-effects system to enable source level reasoning about security of web applications

The secure implementation has been formalized for TINYLINKS, a λ-calculus augmented with

1. XML values for representing web pages
2. `event` e `assert` annotation for expressing safety properties

TinyLinks

Syntax

$$
\begin{array}{l}
\mathtt{f}, \mathtt{y}, \mathtt{x} \\
\mathtt{p} \\
\mathtt{c} ::= \mathtt{Unit} \mid \mathtt{Zero} \mid \mathtt{Succ} \mid \mathtt{String} \\
\quad \mid \mathtt{Nil} \mid \mathtt{Cons} \mid \mathtt{Tuple} \mid \mathtt{Elem} \mid \mathtt{Text} \\
\mathtt{g} ::= + \mid - \mid * \mid / \\
\mathtt{L} ::= \mathtt{p}\,(\mathtt{V}_1, \ldots, \mathtt{V}_n) \\
\mathtt{V}, \mathtt{U} ::= \mathtt{x} \mid \mathtt{c}\,(\mathtt{V}_1, \ldots, \mathtt{V}_n) \mid \mathtt{href}(\mathtt{E}) \\
\quad \mid \lambda \mathtt{x}_1. \ldots, \mathtt{x}_n. \mathtt{E} \mid \mathtt{form}([\,\mathtt{l}_1, \ldots, \mathtt{l}_n\,], \mathtt{E}) \\
\mathtt{E} ::= \mathtt{V} \mid \mathtt{var}\,\mathtt{x} = \mathtt{E}_1; \mathtt{E}_2 \mid \mathtt{g}\,(\mathtt{E}_1, \mathtt{E}_2) \\
\quad \mid \mathtt{V}\,(\mathtt{U}_1, \ldots, \mathtt{U}_n) \mid \mathtt{post}([\,\mathtt{l}_1 = \mathtt{V}_1, \ldots, \mathtt{l}_n = \mathtt{V}_n\,], \mathtt{U}) \\
\quad \mid \mathtt{get}(\mathtt{V}) \mid \mathtt{event}\,\mathtt{L} \mid \mathtt{assert}\,\mathtt{L} \\
\quad \mid \begin{array}{l} \mathtt{switch}(\mathtt{V})\{ \\ \quad \mathtt{case}\,\mathtt{c}\,(\mathtt{x}_1, \ldots, \mathtt{x}_n) \to \mathtt{E}_1 \\ \quad \_ \to \mathtt{E}_2 \\ \} \end{array}
\end{array}
$$

Types-and-effects system

A powelful extension of type systems which allows one to statically reason about program's execution

$$\frac{\Gamma \vdash M_1 : \tau_1 \,\&\, \phi_1 \qquad \ldots \qquad \Gamma \vdash M_n : \tau_n \,\&\, \phi_n}{\Gamma \vdash E(M_1, \ldots, M_n) : \tau \,\&\, \phi}$$

Types-and-effects system

A powelful extension of type systems which allows one to statically reason about program's execution

premise

$$\frac{\Gamma \vdash M_1 : \tau_1 \,\&\, \phi_1 \qquad \dots \qquad \Gamma \vdash M_n : \tau_n \,\&\, \phi_n}{\Gamma \vdash E(M_1, \dots, M_n) : \tau \,\&\, \phi}$$

conclusion **effects**

Types-and-effects system

A powelful extension of type systems which allows one to statically reason about program's execution

premise

$$\frac{\Gamma \vdash M_1 : \tau_1 \,\&\, \phi_1 \qquad \ldots \qquad \Gamma \vdash M_n : \tau_n \,\&\, \phi_n}{\Gamma \vdash E(M_1, \ldots, M_n) : \tau \,\&\, \phi}$$

conclusion

effects

Types-and-effects system

A powelful extension of type systems which allows one to statically reason about program's execution

premise $\Gamma \vdash M_1 : \tau_1 \,\&\, \phi_1 \quad \dots \quad \Gamma \vdash M_n : \tau_n \,\&\, \phi_n$

conclusion $\Gamma \vdash E(M_1, \dots, M_n) : \tau \,\&\, \phi$

effects

It compute for each program phrase its type augmented with a semantic property

- if $\Gamma \vdash M_i : \tau_i \,\&\, \phi_i \Rightarrow \tau_i$ is the type of the expression M_i and the semantic property ϕ_i holds
- then $\Gamma \vdash E(M_1, \dots, M_n) : \tau \,\&\, \phi \Rightarrow \tau$ is the type of the expression $E(M_1, \dots, M_n)$ and the semantic property ϕ holds

TinyLinks's types-and-effects system

Goal

Whenever an assertion `assert L` occurs in the execution there exists a previous occurrence of an event `event L`

$$\Gamma;\ \mathtt{F}\ \vdash \mathtt{E} \overset{exp}{\Rightarrow} <\_ : \mathtt{T} > \{\ \mathtt{F}_1\ \}$$

$$\Gamma;\ \mathtt{F}\ \vdash \mathtt{E} \overset{exp}{\Leftarrow} <\_ : \mathtt{T} > \{\ \mathtt{F}_1\ \}$$

TinyLinks's types-and-effects system

Goal

Whenever an assertion `assert L` occurs in the execution there exists a previous occurrence of an event `event L`

$$\Gamma;\ \mathtt{F} \vdash \mathtt{E} \overset{exp}{\Rightarrow} < \_ : \boxed{\mathtt{T}} > \{\ \mathtt{F}_1\ \}$$

$$\Gamma;\ \mathtt{F} \vdash \mathtt{E} \overset{exp}{\Leftarrow} < \_ : \boxed{\mathtt{T}} > \{\ \mathtt{F}_1\ \}$$

For each expression `E` compute

1. the type

TinyLinks's types-and-effects system

Goal

Whenever an assertion `assert L` occurs in the execution there exists a previous occurrence of an event `event L`

$$\Gamma; \boxed{\mathtt{F}} \vdash \mathtt{E} \overset{exp}{\Rightarrow} < \_ : \mathtt{T} > \{ \mathtt{F}_1 \}$$

$$\Gamma; \boxed{\mathtt{F}} \vdash \mathtt{E} \overset{exp}{\Leftarrow} < \_ : \mathtt{T} > \{ \mathtt{F}_1 \}$$

For each expression `E` compute

1. the type
2. the preconditions

TinyLinks's types-and-effects system

Goal

Whenever an assertion `assert L` occurs in the execution there exists a previous occurrence of an event `event L`

$$\Gamma;\ \mathtt{F}\ \vdash \mathtt{E} \overset{exp}{\Rightarrow} <\_ : \mathtt{T}> \{\mathtt{F}_1\}$$

$$\Gamma;\ \mathtt{F}\ \vdash \mathtt{E} \overset{exp}{\Leftarrow} <\_ : \mathtt{T}> \{\mathtt{F}_1\}$$

For each expression `E` compute

1. the type
2. the preconditions
3. the post-conditions

TinyLinks's types-and-effects system

Some example of rules

$$\text{(T-Get)}\ \frac{\Gamma; \mathtt{F} \vdash \mathtt{V} \overset{val}{\Leftarrow} \mathtt{xml}}{\Gamma; \mathtt{F} \vdash \mathtt{get}(\mathtt{V}) \overset{exp}{\Rightarrow} \langle \_ : \mathtt{xml} \rangle \{\ \}}$$

TINYLINKS's types-and-effects system

Some example of rules

$$\text{(T-Post)}\ \frac{\Gamma;\mathtt{F} \vdash \mathtt{V_i} \overset{val}{\Leftarrow} \mathtt{string} \quad \forall i \in \{\,1,\,\dots\,,n\,\} \quad \Gamma;\mathtt{F} \vdash \mathtt{U} \overset{val}{\Leftarrow} \mathtt{xml}}{\Gamma;\mathtt{F} \vdash \mathtt{post}([\,(\mathtt{l_1} = \mathtt{V_1},\,\dots\,,\mathtt{l_n} = \mathtt{V_n})\,]\,,\mathtt{U}) \overset{exp}{\Rightarrow} \langle \_ : \mathtt{xml}\rangle\,\{\ \}}$$

TinyLinks's types-and-effects system

Some example of rules

$$
\text{(T-Assert)}\ \frac{\Gamma \vdash \diamond \qquad fv(\mathtt{F},\mathtt{L}) \subseteq dom(\Gamma) \qquad \mathtt{L} \in \mathtt{F} \qquad \mathtt{L} = \mathtt{p}\,(\mathtt{V_1},\,\ldots\,,\mathtt{V_n}) \qquad \Gamma;\mathtt{F} \vdash \mathtt{V_i} \overset{val}{\Rightarrow} \mathtt{T_i} \qquad \forall i \in \{\,1,\,\ldots\,,n\,\}}{\Gamma;\mathtt{F} \vdash \mathtt{assert}\,\mathtt{L} \overset{exp}{\Rightarrow} \langle \_ : \mathtt{unit} \rangle \,\{\,\mathtt{L}\,\}}
$$

TinyLinks's types-and-effects system

Some example of rules

$$\text{(T-App)}\ \frac{\Gamma; \mathtt{F} \vdash \mathtt{U} \stackrel{val}{\Rightarrow} \mathtt{T} \quad \mathtt{T} = \langle \mathtt{x_1} : \mathtt{T_1},\ \ldots,\ \mathtt{x_n} : \mathtt{T_n} \rangle \{\ \mathtt{F_1}\ \} \rightarrow \mathtt{W} \quad fv(\mathtt{T}) = \emptyset \qquad \Gamma; \mathtt{F} \vdash \mathtt{V_i} \stackrel{val}{\Leftarrow} \mathtt{T_i} \quad \forall i \in \{\ 1,\ \ldots\ , n\ \} \quad \mathtt{F_1}\,[\mathtt{V_1}/\mathtt{x_1}]\ \ldots\ [\mathtt{V_n}/\mathtt{x_n}] \subseteq \mathtt{F}}{\Gamma; \mathtt{F} \vdash \mathtt{U}\,(\mathtt{V_1},\ \ldots\ , \mathtt{V_n}) \stackrel{exp}{\Rightarrow} \mathtt{W}\,[\mathtt{V_1}/\mathtt{x_1}]\ \ldots\ [\mathtt{V_n}/\mathtt{x_n}]}$$

TINYLINKS's types-and-effects system

Safe Web Application

A web application $\mathtt{E}$ is safe if and only if there exists a proof within the types-and-effects system of the judgment

$$\emptyset; \emptyset \vdash \mathtt{E} \overset{exp}{\Leftarrow} \langle \_ : \mathtt{xml} \rangle \, \{ \ \}$$

Types-and-effects system

Usually the definition of a types-and-effects analysis requires

1. Definition of rules
2. State and prove the soundness of analysis
3. Definition of inference algorithm
4. Prove that the algorithm is correct (soundness/completeness)

Types-and-effects system

Usually the definition of a types-and-effects analysis requires

1. Definition of rules
2. State and prove the soundness of analysis
3. Definition of inference algorithm
4. Prove that the algorithm is correct (soundness/completeness)

The approach used for the TinyLinks's types-and-effects system is different

- each expression is translated in an expression of F7
  - this translation hides the property of the analysis

Our goal

Reconstruct the TinyLinks's types-and-effects system by abstract interpretation

Our goal

Reconstruct the TinyLinks's types-and-effects system by abstract interpretation

Benefits

1. precise definition of relation between analysis and semantics
2. analysis is correct by construction

Our goal

Reconstruct the TinyLinks's types-and-effects system by abstract interpretation

Benefits

1. precise definition of relation between analysis and semantics
2. analysis is correct by construction

We follow Cousot's methodology for type systems

1. we define a denotational semantics for TinyLinks (concrete semantics)
2. we define a abstract semantics that computes types augmented by effects

Concrete semantics

A denotational semantics that considers

Concrete semantics

A denotational semantics that considers

- TinyLinks as an untyped λ-calculus

$$Eval = (\ldots + \underbrace{EEnv \rightarrow (Eval \times EEnv)}_{Href} + \ldots)_{\perp}$$

Concrete semantics

A denotational semantics that considers

- TinyLinks as an untyped λ-calculus

$$Eval = (\ldots + \underbrace{EEnv \rightarrow (Eval \times EEnv)}_{Href} + \ldots)_{\perp}$$

Domain of values

Injection (Text, Form, Fun, . . .)

Concrete semantics

A denotational semantics that considers

- TinyLinks as an untyped λ-calculus

$$Eval = (\ldots + \underbrace{EEnv \to (Eval \times EEnv)}_{Href} + \ldots)_{\perp}$$

Domain of values

Injection (Text, Form, Fun, ...)

- the occurrence and assertion of the events

$$EEnv = Pred \to (Dval \times Mark)$$

Concrete semantics

A denotational semantics that considers

- TinyLinks as an untyped λ-calculus

$$Eval = (\ldots + \underbrace{EEnv \rightarrow (Eval \times EEnv)}_{Href} + \ldots)_\perp$$

Domain of values

Injection (Text, Form, Fun, . . .)

- the occurrence and assertion of the events

$$EEnv = Pred \rightarrow (Dval \times Mark)$$

Events environment

$\{ E, EA, A \}$

values in the events are integers only

Concrete semantics

Two semantic functions

1. for values

$$\mathcal{V}[\![-]\!] : \texttt{VAL} \rightarrow Env \rightarrow EEnv \rightarrow Eval$$

2. for expressions

$$[\![-]\!] : \texttt{EXP} \rightarrow Env \rightarrow EEnv \rightarrow (Eval \times EEnv)$$

Concrete semantics

Two semantic functions

1. for values

$$\mathcal{V}[\![-]\!] : \texttt{VAL} \rightarrow Env \rightarrow EEnv \rightarrow Eval$$

2. for expressions

$$[\![-]\!] : \texttt{EXP} \rightarrow Env \rightarrow EEnv \rightarrow (Eval \times EEnv)$$

Examples of semantic equation

$$\begin{aligned}
[\![\texttt{get(V)}]\!]\rho\,\phi \;= &\; let^{\star}\; v' \;= \mathcal{V}[\![\texttt{V}]\!]\rho\,\phi\; in \\
& case\; v'\; of \\
& \quad Href(f) \rightarrow f\,\phi \\
& \quad _ \rightarrow (\lfloor WrongValue() \rfloor,\, \iota)
\end{aligned}$$

Concrete semantics

Two semantic functions

1. for values

$$\mathcal{V}[\![-]\!] : \texttt{VAL} \to Env \to EEnv \to Eval$$

2. for expressions

$$[\![-]\!] : \texttt{EXP} \to Env \to EEnv \to (Eval \times EEnv)$$

Examples of semantic equation

$$\begin{aligned}
[\![\texttt{assert}\,\texttt{q}(\texttt{V})]\!]\rho\,\phi \;&= let^{\star}\ ev\ = evalToDval(\mathcal{V}[\![\texttt{V}]\!]\rho\,\phi)\ in\\
&\quad let\ (ev',\, m)\ = \phi\, q\\
&\quad if\ ev\ =\ ev'\ then\\
&\qquad (\lfloor Unit() \rfloor,\, \phi\left[(ev',\, EA)/q\right])\\
&\quad else\\
&\qquad (\lfloor WrongValue() \rfloor,\, \iota)
\end{aligned}$$

Unsoundness in TinyLinks analysis

Consider the expression

$$\texttt{get}(\texttt{Text}(\texttt{"Hello World!"}))$$

Unsoundness in TinyLinks analysis

Consider the expression

$$\texttt{get}(\texttt{Text}(\text{"}\texttt{Hello World!}\text{"}))$$

Semantics

$$[\![\texttt{get}(\texttt{Text}(\text{"}\texttt{Hello World!}\text{"}))]\!]\rho\,\phi \;=\; (\lfloor WrongValue(),\,\iota\rfloor)$$

Error: the denotation of `Text("Hello World!")` is not a link

Unsoundness in TinyLinks analysis

Consider the expression

$$\texttt{get}(\texttt{Text}(\texttt{"Hello World!"}))$$

Semantics

$$[\![\texttt{get}(\texttt{Text}(\texttt{"Hello World!"}))]\!]\rho\,\phi \;=\; (\lfloor\, WrongValue(),\, \iota\,\rfloor)$$

Error: the denotation of `Text("Hello World!")` is not a link

TinyLinks's types-and-effects system

$$\emptyset; \emptyset \vdash \texttt{get}(\texttt{Text}(\texttt{"Hello World!"})) \overset{exp}{\Leftarrow} \langle \_ : \texttt{xml} \rangle \; \{ \; \}$$

the expression is type checked and the computed type is `xml`

Abstract domain

Values

Trouble

- types have annotations

$$\texttt{integer}\,\{\ \} \rightarrow \texttt{integer}\,\{\,q : 5,\ p : 3\,\}$$

Abstract domain

Values

Trouble

- types have annotations

$$\texttt{integer}\,\{\ \} \rightarrow \texttt{integer}\,\{\,\mathrm{q}:5,\,\mathrm{p}:3\,\}$$

- annotated types are not a free algebra

$$\texttt{integer}\,\{\ \} \rightarrow \texttt{integer}\,\{\,\mathrm{q}:5,\,\mathrm{p}:3\,\}$$
$$\texttt{integer}\,\{\ \} \rightarrow \texttt{integer}\,\{\,\mathrm{p}:3,\,\mathrm{q}:5\,\}$$

Abstract domain

Values

Trouble

- types have annotations

$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\mathtt{q}:5,\ \mathtt{p}:3\,\}$$

- annotated types are not a free algebra

$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\mathtt{q}:5,\ \mathtt{p}:3\,\}$$
$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\mathtt{p}:3,\ \mathtt{q}:5\,\}$$

Solution: simple type and constraints

Abstract domain

Values

Trouble

- types have annotations

$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\mathrm{q}:5,\ \mathrm{p}:3\,\}$$

- annotated types are not a free algebra

$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\mathrm{q}:5,\ \mathrm{p}:3\,\}$$
$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\mathrm{p}:3,\ \mathrm{q}:5\,\}$$

Solution: simple type and constraints

- we substitute the annotations in the types with annotation variables

Abstract domain

Values

Trouble

- types have annotations

$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\texttt{q}:5,\ \texttt{p}:3\,\}$$

- annotated types are not a free algebra

$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\texttt{q}:5,\ \texttt{p}:3\,\}$$
$$\texttt{integer}\,\{\ \} \to \texttt{integer}\,\{\,\texttt{p}:3,\ \texttt{q}:5\,\}$$

Solution: simple type and constraints

- we substitute the annotations in the types with annotation variables
- we introduce constraints to restrict annotation variables

$$\texttt{integer}(\gamma_1) \to \texttt{integer}(\gamma_2) \qquad \gamma_1 \supseteq \emptyset \quad \gamma_2 \supseteq \{\,\texttt{p}:3,\ \texttt{q}:5\,\}$$

Abstract domain

Values

- the events depend on the concrete values ($Dval$)

Abstract domain

Values

- the events depend on the concrete values ($Dval$)
- the abstract domain need to include the concrete values

Abstract domain

Values

- the events depend on the concrete values ($Dval$)
- the abstract domain need to include the concrete values

$$TipoS \times Dval \times Constr \times TPred$$

(type, substitution)∪ NoType

$Pred \rightarrow Dval$

concrete value

constraints

Abstract semantics

Example of semantic equation

$$
\begin{array}{l}
[\![\texttt{get(V)}]\!]^a \rho\, \phi \;= \\
\quad \gamma \in V_a \quad fresh \\
\quad let\ (ts,\, d,\, C,\, f)\ = \mathcal{V}[\![\texttt{V}]\!]^a \rho\, \phi\, in \\
\quad if\, ts \neq NoType\, then \\
\quad\quad case\ mgu(\{\ ts.t = link(\gamma)\ \} \cup ts.\theta)\ of \\
\quad\quad\quad S(\theta) \rightarrow let\ C'\ =\ \{\ (\theta(\gamma),\, q) \in \theta(C)\ \}\ in \\
\quad\quad\quad\quad if\, check(\theta(f \leftarrow C'),\, \phi)\ then \\
\quad\quad\quad\quad\quad (((\theta(xml(\gamma)),\, \theta), \\
\quad\quad\quad\quad\quad\quad\quad nodval,\, \theta(C) \setminus C',\, \theta(f \downarrow C)), \phi) \\
\quad\quad\quad\quad else \\
\quad\quad\quad\quad\quad (Error,\, \iota) \\
\quad\quad\quad \_ \rightarrow (Error,\, \iota) \\
\quad else \\
\quad\quad (Error,\, \iota)
\end{array}
$$

Analyzer

Both the concrete and abstract semantics have been implemented as OCaml programs

Analyzer

Both the concrete and abstract semantics have been implemented as OCaml programs

- TINYLINKS programs are represented in abstract syntax

Analyzer

Both the concrete and abstract semantics have been implemented as OCaml programs

- TinyLinks programs are represented in abstract syntax
- the implementation have an unique semantic function parametrized with respect to

Analyzer

Both the concrete and abstract semantics have been implemented as OCaml programs

- TINYLINKS programs are represented in abstract syntax
- the implementation have an unique semantic function parametrized with respect to
  - the primitive operations

Analyzer

Both the concrete and abstract semantics have been implemented as OCaml programs

- TinyLinks programs are represented in abstract syntax
- the implementation have an unique semantic function parametrized with respect to
  - the primitive operations
  - the semantic domain

Example 1

Expression

```
fun buy(value, dbpass) {
  var _ = assert PriceIs(value);
    Text("a")
}
```

Example 1

Expression

```
fun buy(value, dbpass) {
  var _ = assert PriceIs(value);
    Text("a")
}
```

Abstract semantics

```
(type - :
Function(_#value#var0_, Integer(), _annvar0_,
  Function(_#dbpass#var1_, _typevar1_, _annvar2_,
    Xml(_annvar4_), _annvar3_),
  _annvar1_)
No_dval [(_annvar2_,PriceIs)]
  {PriceIs -> _#value#var0_}, {})
```

Example 2

Expression

$$\texttt{buy 5}$$

Example 2

Expression

buy 5

Abstract semantics

```
(type - :
  Function(_#dbpass#var3_, _typevar3_, _annvar7_,
         Xml(_annvar9_), _annvar8_)
  Unknown [(_annvar7_,PriceIs)] {PriceIs -> 5}, {})
```

Example 3

Expression

`buy 5 "a"`

Example 3

Expression

$$\texttt{buy}\ 5\ \texttt{"a"}$$

Abstract semantics

```
Exception: No_type "apply_fun: no preconditions"
```

Conclusions

We have reconstructed the TinyLinks's types-and-effects system by abstract interpretation

- we have precisely defined relationship between semantics and analysis
- we have shown unsoundness of TinyLinks's types-and-effects system

Conclusions

We have reconstructed the TinyLinks's types-and-effects system by abstract interpretation

- we have precisely defined relationship between semantics and analysis
- we have shown unsoundness of TinyLinks's types-and-effects system

Future work

- Consider a type system with sub-types (`link` $<:$ `xml`, `form` $<:$ `xml`)
- Extend the class of value that can be used in the events
- Generalize the methodology and apply it to further analysis