First class reflection for shorter proofs

Mathieu Boespflug

☙

McGill University

☙

20 January 2012

Plan

- Part 1: Palindromes
- Part 2: higher-order programming
- Part 3: contextual programming

Formal systems

Example

- The language of *formulae* — words
- The set of *axioms* (or *assumptions*) — a-z,ϵ
- The language of *proofs*

$$(\text{ax})\,\frac{P\ \text{ is an axiom}}{P\ \text{ is a palindrome}}$$

$$(\text{ext})\,\frac{P\ \text{ is a palindrome}}{xPx\ \text{ is a palindrome}}$$

- *Theorems* are formulae that have proofs.

Palindromes: example

$$
\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ax)}\,\dfrac{\text{d is an axiom}}{\text{t is a palindrome}}}{\text{rtr is a palindrome}}}{\text{artra is a palindrome}}}{\text{tartrat is a palindrome}}}{\text{etartrate is a palindrome}}}{\text{detartrated is a palindrome}}
$$

Palindromes: example

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t \in \Gamma}{\Gamma \vdash t}\,(\mathrm{ax})}{\Gamma \vdash \mathrm{rtr}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{artra}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{tartrat}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{etartrate}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{detartrated}}\,(\mathrm{ext})
$$

Palindromes: another example

$$
\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ext)}\,\dfrac{\text{(ax)}\,\dfrac{t \in \Gamma}{\Gamma \vdash t}}{\Gamma \vdash rtr}}{\Gamma \vdash artra}}{\Gamma \vdash tartrat}}{\Gamma \vdash etartrate}}{\Gamma \vdash detartrated}}{\Gamma \vdash rdetartratedr}}{\Gamma \vdash ardetartratedra}}{\Gamma \vdash dardetartratedrad}}{\Gamma \vdash adardetartratedrada}}{\Gamma \vdash radardetartratedradar}
$$

A new inference rule

$$(\text{concat})\,\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash Q\,P\,Q}$$

Palindromes: another example, revisited

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{
          \dfrac{\mathrm{t} \in \Gamma}{\Gamma \vdash \mathrm{t}}\,(\mathrm{ax})
        }{\Gamma \vdash \mathrm{rtr}}\,(\mathrm{ext})
      }{\Gamma \vdash \mathrm{artra}}\,(\mathrm{ext})
    }{\Gamma \vdash \mathrm{tartrat}}\,(\mathrm{ext})
  }{\Gamma \vdash \mathrm{etartrate}}\,(\mathrm{ext})
}{\Gamma \vdash \mathrm{detartrated}}\,(\mathrm{ext})
\qquad
\dfrac{
  \dfrac{
    \dfrac{\mathrm{d} \in \Gamma}{\Gamma \vdash \mathrm{d}}\,(\mathrm{ax})
  }{\Gamma \vdash \mathrm{ada}}\,(\mathrm{ext})
}{\Gamma \vdash \mathrm{radar}}\,(\mathrm{ext})
$$

$$
(\mathrm{concat})\ \dfrac{\Gamma \vdash \mathrm{detartrated} \qquad \Gamma \vdash \mathrm{radar}}{\Gamma \vdash \mathrm{radardetartratedradar}}
$$

Recap...

- Identified alternative pattern to show palindromes.
- Introduced new inference rule to capture this pattern.
- Can prove (some) palindromes with derivation of shorter size.
- Have to convince ourselves that new inference rule does not allow new "palindromes".
- Want to prove:

$$\forall \Gamma .\ \forall P\ . \quad (\text{concat})\,\frac{\vdots \qquad \vdots}{\Gamma \vdash\ \ \mathrm{P}} \;\Rightarrow\; (\text{ext})\,\frac{\vdots}{\Gamma \vdash\ \ \mathrm{P}}$$

Proof equivalence

$$
\text{(concat)}\,\dfrac{\dfrac{}{\Gamma \vdash P} \qquad \text{(ext)}\,\dfrac{\dfrac{}{\Gamma \vdash Q}}{\Gamma \vdash xQx}}{\Gamma \vdash xQxP\,xQx}
\quad \Leftrightarrow \quad
\text{(ext)}\,\dfrac{\text{(concat)}\,\dfrac{\text{(ext)}\,\dfrac{\dfrac{}{\Gamma \vdash P}}{\Gamma \vdash xP\,x} \qquad \dfrac{}{\Gamma \vdash Q}}{\Gamma \vdash QxP\,xQ}}{\Gamma \vdash xQxP\,xQx}
$$

Proof reduction: example

$$
\text{(concat)}\,\dfrac{
\text{(ext)}\,\dfrac{
\text{(ext)}\,\dfrac{
\text{(ext)}\,\dfrac{
\text{(ext)}\,\dfrac{
\text{(ext)}\,\dfrac{
\text{(ax)}\,\dfrac{\mathrm{t} \in \Gamma}{\Gamma \vdash \mathrm{t}}
}{\Gamma \vdash \mathrm{rtr}}
}{\Gamma \vdash \mathrm{artra}}
}{\Gamma \vdash \mathrm{tartrat}}
}{\Gamma \vdash \mathrm{etartrate}}
}{\Gamma \vdash \mathrm{detartrated}}
\qquad
\text{(ext)}\,\dfrac{
\text{(ext)}\,\dfrac{
\text{(ax)}\,\dfrac{\mathrm{d} \in \Gamma}{\Gamma \vdash \mathrm{d}}
}{\Gamma \vdash \mathrm{ada}}
}{\Gamma \vdash \mathrm{radar}}
}{\Gamma \vdash \mathrm{radardetartratedradar}}
$$

Proof reduction: example

$$
(\mathrm{ext})\dfrac{
(\mathrm{concat})\dfrac{
(\mathrm{ext})\dfrac{
(\mathrm{ext})\dfrac{
(\mathrm{ext})\dfrac{
(\mathrm{ext})\dfrac{
(\mathrm{ext})\dfrac{
(\mathrm{ext})\dfrac{
(\mathrm{ax})\dfrac{\mathrm{t} \in \Gamma}{\Gamma \vdash \mathrm{t}}
}{\Gamma \vdash \mathrm{rtr}}
}{\Gamma \vdash \mathrm{artra}}
}{\Gamma \vdash \mathrm{tartrat}}
}{\Gamma \vdash \mathrm{etartrate}}
}{\Gamma \vdash \mathrm{detartrated}}
}{\Gamma \vdash \mathrm{rdetartratedr}}
\qquad
(\mathrm{ext})\dfrac{
(\mathrm{ax})\dfrac{\mathrm{d} \in \Gamma}{\Gamma \vdash \mathrm{d}}
}{\Gamma \vdash \mathrm{ada}}
}{\Gamma \vdash \mathrm{adardetartratedrada}}
}{\Gamma \vdash \mathrm{radardetartratedradar}}
$$

Proof reduction: example

$$
\begin{prooftree}
\AxiomC{$\mathrm{t} \in \Gamma$}
\LeftLabel{(ax)}
\UnaryInfC{$\Gamma \vdash \mathrm{t}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{rtr}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{artra}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{tartrat}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{etartrate}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{detartrated}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{rdetartratedr}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{ardetartratedra}$}
\AxiomC{$\mathrm{d} \in \Gamma$}
\LeftLabel{(ax)}
\UnaryInfC{$\Gamma \vdash \mathrm{d}$}
\LeftLabel{(concat)}
\BinaryInfC{$\Gamma \vdash \mathrm{dardetartratedrad}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{adardetartratedrada}$}
\LeftLabel{(ext)}
\UnaryInfC{$\Gamma \vdash \mathrm{radardetartratedradar}$}
\end{prooftree}
$$

Proof reduction: example

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t \in \Gamma}{\Gamma \vdash t}\,(\mathrm{ax})}{\Gamma \vdash \mathrm{rtr}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{artra}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{tartrat}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{etartrate}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{detartrated}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{rdetartratedr}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{ardetartratedra}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{dardetartratedrad}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{adardetartratedrada}}\,(\mathrm{ext})}{\Gamma \vdash \mathrm{radardetartratedradar}}\,(\mathrm{ext})
$$

Proof reduction

$$(\mathrm{concat})\,\dfrac{\overline{\Gamma \vdash P} \qquad (\mathrm{ext})\,\dfrac{\overline{\Gamma \vdash Q}}{\Gamma \vdash xQx}}{\Gamma \vdash xQxP\,xQx} \quad \longrightarrow \quad (\mathrm{ext})\,\dfrac{(\mathrm{concat})\,\dfrac{(\mathrm{ext})\,\dfrac{\overline{\Gamma \vdash P}}{\Gamma \vdash xP\,x} \qquad \overline{\Gamma \vdash Q}}{\Gamma \vdash QxP\,xQ}}{\Gamma \vdash xQxP\,xQx}$$

- Orient equivalence rule: get reduction rule.
- Take reduction rule to *define* new inference rule.
- Inference rule + repeat reduction rule = program?

Another recap...

- Introduced new inference rule.
- Wanted to prove it doesn't introduce new "palindromes".
- Introduced reduction rule — relates **new inference** to **existing inference rules**.
- Repeatedly applying reduction rules can be seen as a **meta-level program**.
- If we prove the meta-level program always works, then never need to reduce to old rules!
- Consequence: can use new inference rule...

  1. ... without having to trust it.
  2. ... without having to reconstruct long derivations as evidence.

Part 2: higher-order programming

(or when functions fly first class)

```
System.out.printLn(1 + 1);
```

$$1 + 1$$

```
'a' + 1
```

```
Character.getNumericValue('a')
+ 1
```

```
int_of_string 'a' + 1
```

```
let f = fun (x : char) -> 97
in
  f 'a' + 1
```

```
let f =
  fun (x : char list) -> 97
in f ['a'; 'b'; 'c'] + 1
```

```
let sum =
  ...
in sum ['a'; 'b'; 'c'] + 1
```

```
let map = ... in
let sum = ... in
  sum (map int_of_char
               ['a'; 'b';
'c']) + 1
```

Programming with functions: modularity

- map is a higher order function.
- Pattern: "apply the same transformation to each element of the input list"
- Need only write one map function.
- This one map function can be reused to apply any transformation to all elements of any input list.

Programs are...

... pieces of syntax strung together

... variables, bindings, function calls, ...

Programming with functions: representing programs

Principles:

- Variables represented as... variables.
- Binding structures represented as... functions.

Assumptions:

```
term : type.
tint : int -> term.
tplus : term * term -> term.
tfun : (term -> term) -> term.
tapp : term * term -> term.
tlet : term * (term -> term) -> term
```

Example:

```
plus (tint 1) (tint 1)
```

Programming with functions: representing programs

Principles:

- Variables represented as... variables.
- Binding structures represented as... functions.

Assumptions:

```
term : type.
tint : int -> term.
tplus : term * term -> term.
tfun : (term -> term) -> term.
tapp : term * term -> term.
tlet : term * (term -> term) -> term
```

Example:

```
tfun (fun x -> x)
```

Programming with functions: representing programs

Principles:

- Variables represented as... variables.
- Binding structures represented as... functions.

Assumptions:

```
term : type.
tint : int -> term.
tplus : term * term -> term.
tfun : (term -> term) -> term.
tapp : term * term -> term.
tlet : term * (term -> term) -> term
```

Example:

```
tapp (tfun (fun x -> x)) (tint 5)
```

Programming with functions: representing programs

Principles:

- Variables represented as... variables.
- Binding structures represented as... functions.

Assumptions:

```
term : type.
tint : int -> term.
tplus : term * term -> term.
tfun : (term -> term) -> term.
tapp : term * term -> term.
tlet : term * (term -> term) -> term
```

Example:

```
tlet (tfun (fun x -> x)) (fun f -> tapp f (tint 5))
```

Part 3: contextual programming

(or when reflection flies first class)

Meta-level programming language

- Want to manipulate programs as data.
- Programs represented using functions.
- Problem 1: can only apply functions!
- Problem 2: values are always closed.
- Solution: design a meta-level programming language.
- Expressions of base programming language are pieces of data in the meta-level programming language.

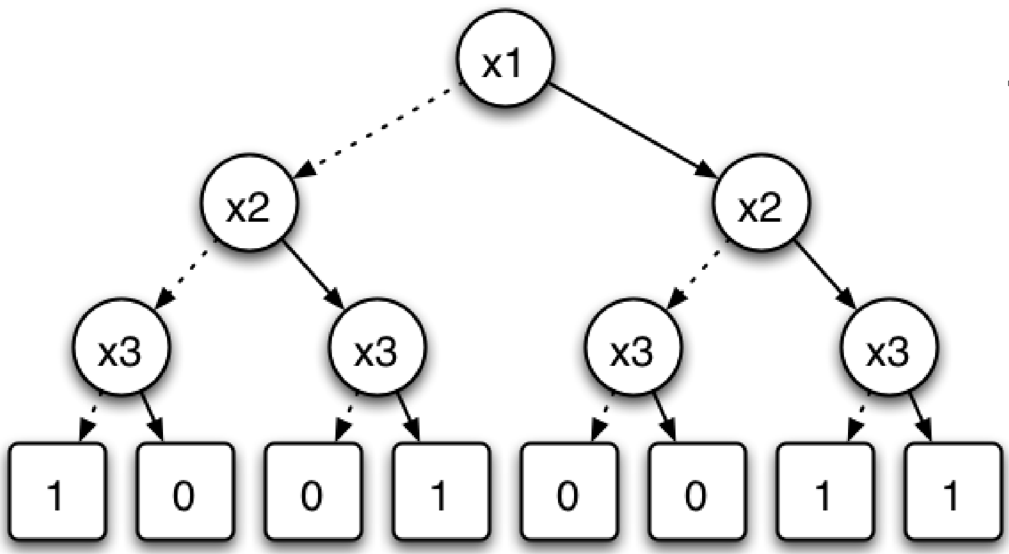
Contextual objects

- Introduce notion of context ψ.
- As we recurse over data, free variables appear.
- At the meta-level, pieces of data only make sense in some context ψ.
- Contextual modal type theory (Nanevski, Pfenning, Pientka): make types tell us in what context a piece of data makes sense.
- Type of meta-level functions mapping (open) data to (open) data:

$$\begin{array}{c}\texttt{Prop} : \texttt{type}. \\ \texttt{trans} : \forall\psi.\texttt{Prop}[\psi] \texttt{ -> } \texttt{Prop}[\psi].\end{array}$$

Example: binary decision diagrams

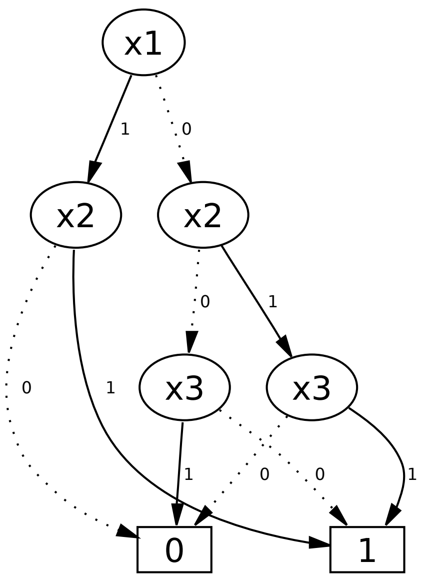
$$\forall x_1.\forall x_2.\forall x_3.\ (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$$



x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Example: binary decision diagrams

$$\forall x_1 . \forall x_2 . \forall x_3 . \ (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$$



- Equivalent formula have unique canonical graph.
- To prove $A \Rightarrow B$, can compare graph of A and graph of B.

Reflection and binary decision diagrams

Want to prove that $A \Rightarrow B$ for some given A and B.

1. `toBDD` A maps A to BDD.
2. `toBDD` B maps B to BDD.
3. Map each BDD to canonical graphs.
4. Test whether canonical graph is equal (using `test`).
5. Prove that
$$\forall\psi.\forall P_1{:}Prop[\psi].\forall P_2{:}Prop[\psi].$$
$$\texttt{test}\ (\texttt{toBDD}\ P_1)\ (\texttt{toBDD}\ P_2) = true \Rightarrow (P_1 \Rightarrow P_2)$$
6. If `test` returns $true$ then $A \Rightarrow B$ by property above.

Conclusion

- Using functions to encode formulas is very convenient.
- Introducing meta-level programming language to reason to encoded formulas gives us the formalism we need to express meta-level programs.
- If we prove that meta-level program is sound, then can use meta-level program to prove some property.
- No need to actually write out proof of property using inference rules.
- In effect the meta-level program is a new inference rule!
- Future work:

  1. Devise more reflective meta-level programs.
  2. Work out the details and properties of the meta-level language we need to achieve this.