Towards abductive functional programming

Koko Muroya
Steven Cheung & Dan R. Ghica
(University of Birmingham)

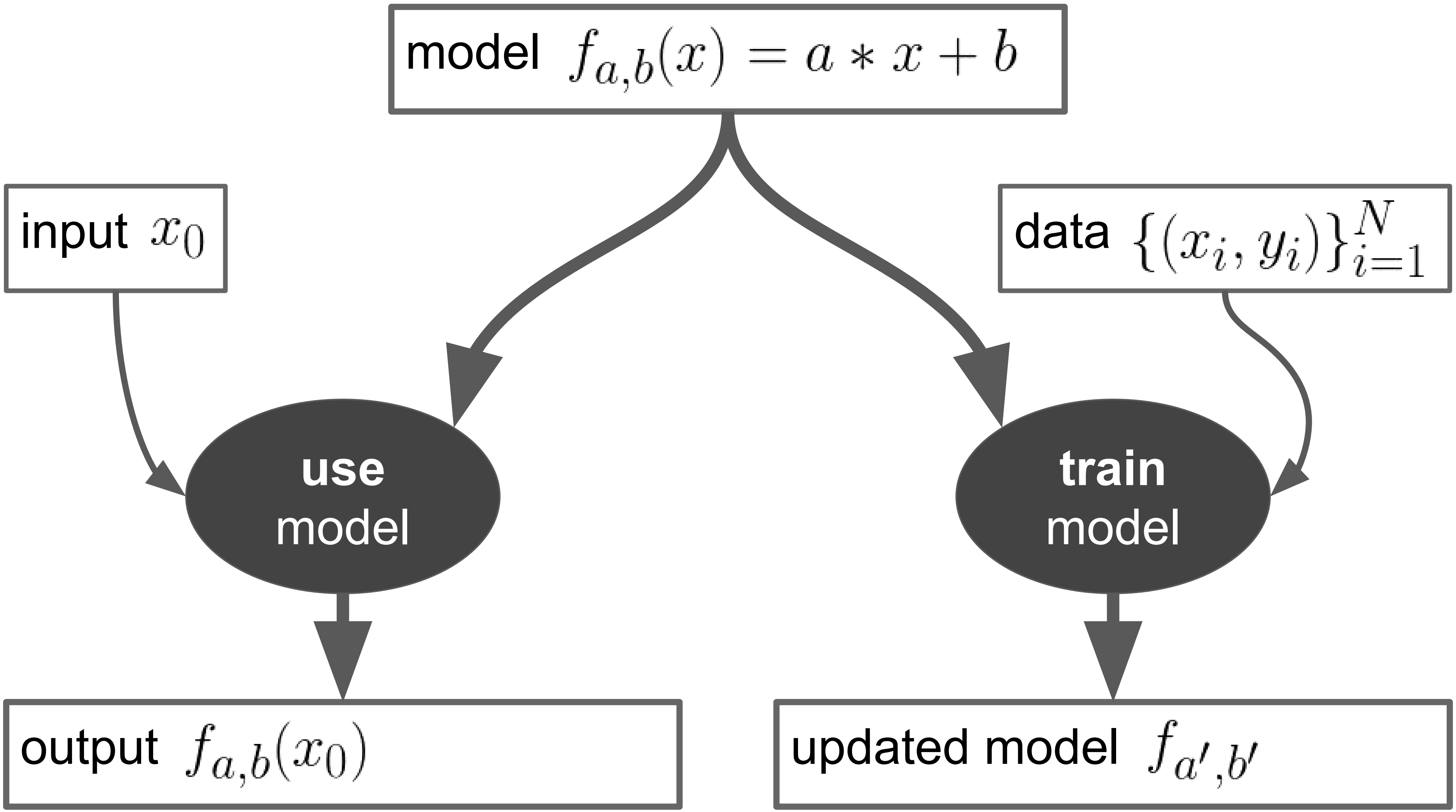
ML Family workshop (Oxford), 7 Sep. 2017

Parameter tuning via targeted abduction

Koko Muroya
Steven Cheung & Dan R. Ghica
(University of Birmingham)

ML Family workshop (Oxford), 7 Sep. 2017

A programming idiom for optimisation & ML



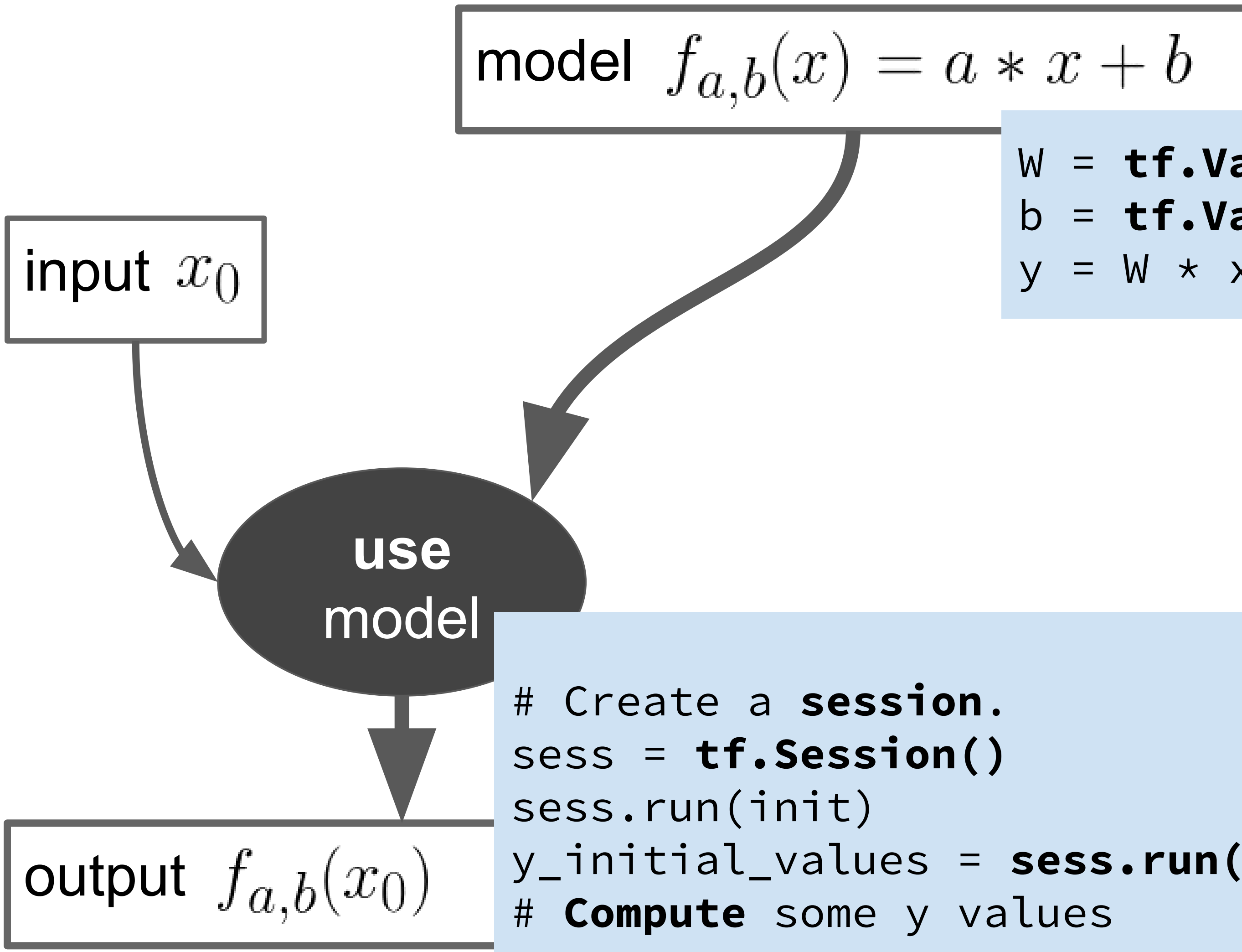
Example: parameter optimisation in TensorFlow

model $f_{a,b}(x) = a * x + b$

```
# Build inference graph.
# Create and initialise variables W and b.
W = tf.Variable(...)
b = tf.Variable(...)
y = W * x_data + b #NOTE: Nothing actually computed here!
```

https://www.tensorflow.org/
https://github.com/sherrym/tf-tutorial

Example: parameter optimisation in TensorFlow




```
W = tf.Variable(...)
b = tf.Variable(...)
y = W * x_data + b
```

```
# Create a session.
sess = tf.Session()
sess.run(init)
y_initial_values = sess.run(y)
# Compute some y values
```

Example: parameter optimisation in TensorFlow

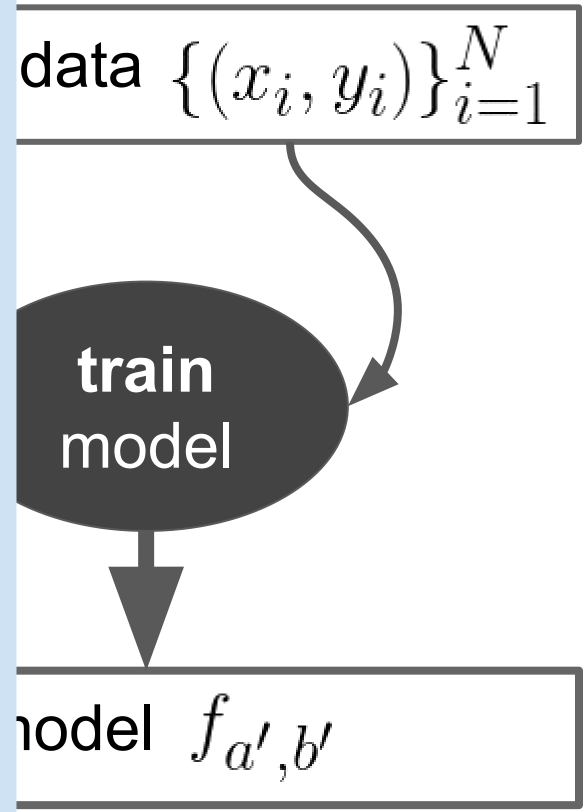
```
W = tf.Variable(...
b = tf.Variable(...
y = W * x_data + b
```

model $f_{a,b}(x) = a * x + b$

```
# Build training graph.
loss = tf.some_loss_function(y, y_data)
# Create an operation that calculates loss.
tf.train.some_optimiser.minimize(loss)
# Create an operation that minimizes loss.
init = tf.initialize_all_variables()
# Create an operation initializes variables.

sess = tf.Session()
sess.run(init)

# Perform training:
for step in range(201):
    sess.run(train)
```



Muroya (U. B'ham.)

TensorFlow

- shallow embedded DSL
  - lack of integration with host language
  - cannot use libraries in graphs
  - difficult to debug / type graphs
- imperative “variable” update

TensorFlow

- shallow embedded DSL
  - lack of integration with host language
  - cannot use libraries in graphs
  - difficult to debug / type graphs
- imperative parameter (“variable”) update

Proper *functional* language?

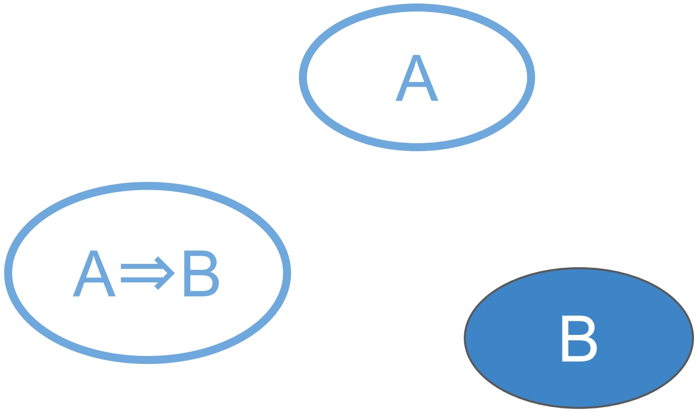
- simple & uniform programming language
  - full integration with base language
  - typed in ML-style
  - well-defined operational semantics
- funcional parameter update

Key idea:

Abductive reasoning

Abductive inference: background

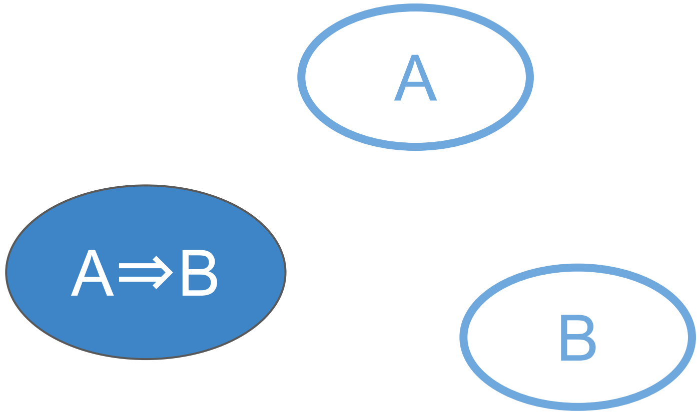
- logical inference
  - deduction (specialisation)
  - induction (generalisation)
  - **abduction (explanation)**



- previous applications
  - abductive logic programming
  - program verification (http://fbinfer.com/)

Abductive inference: background

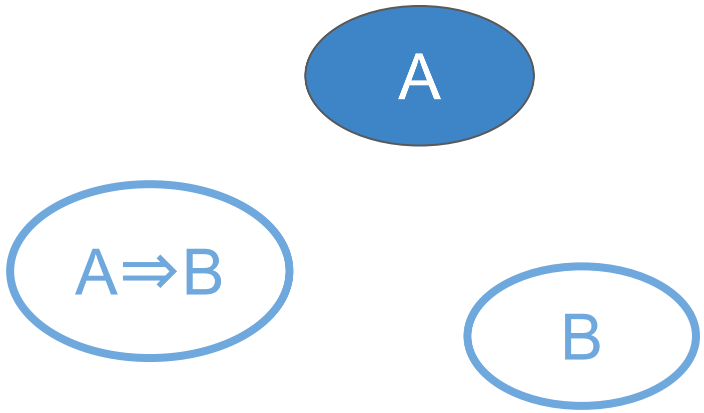
- logical inference
  - deduction (specialisation)
  - induction (generalisation)
  - **abduction (explanation)**



- previous applications
  - abductive logic programming
  - program verification (http://fbinfer.com/)

Abductive inference: background

- logical inference
  - deduction (specialisation)
  - induction (generalisation)
  - **abduction (explanation)**



- previous applications
  - abductive logic programming
  - program verification (http://fbinfer.com/)

Abductive inference: our use

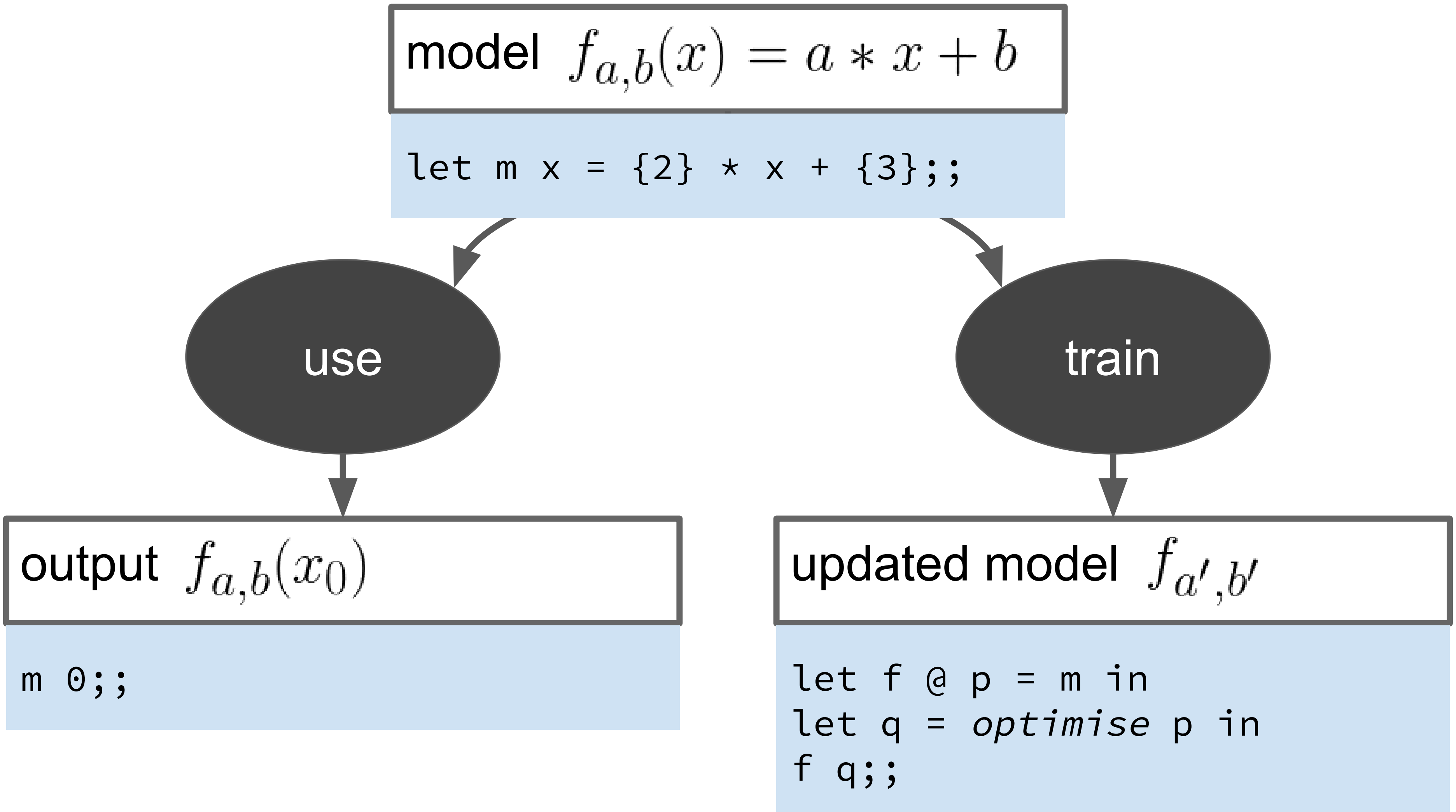
- possible deductive rule for abduction

$$\frac{\Gamma \vdash A}{\Gamma \vdash (P \Rightarrow A) \wedge P}$$

"abduct" explanation *P* of *A*

in "targeted" way

“Parameter tuning via targeted abduction”



"Parameter tuning via targeted abduction"

model $f_{a,b}(x) = a * x + b$

```
let m x = {2} * x + {3};;
```

provisional constants ("targets")

cf. definitive constants `0,1,2,...`

Parameter tuning via targeted abduction

model $f_{a,b}(x) = a * x + b$

```
let m x = {2} * x + {3};;
```

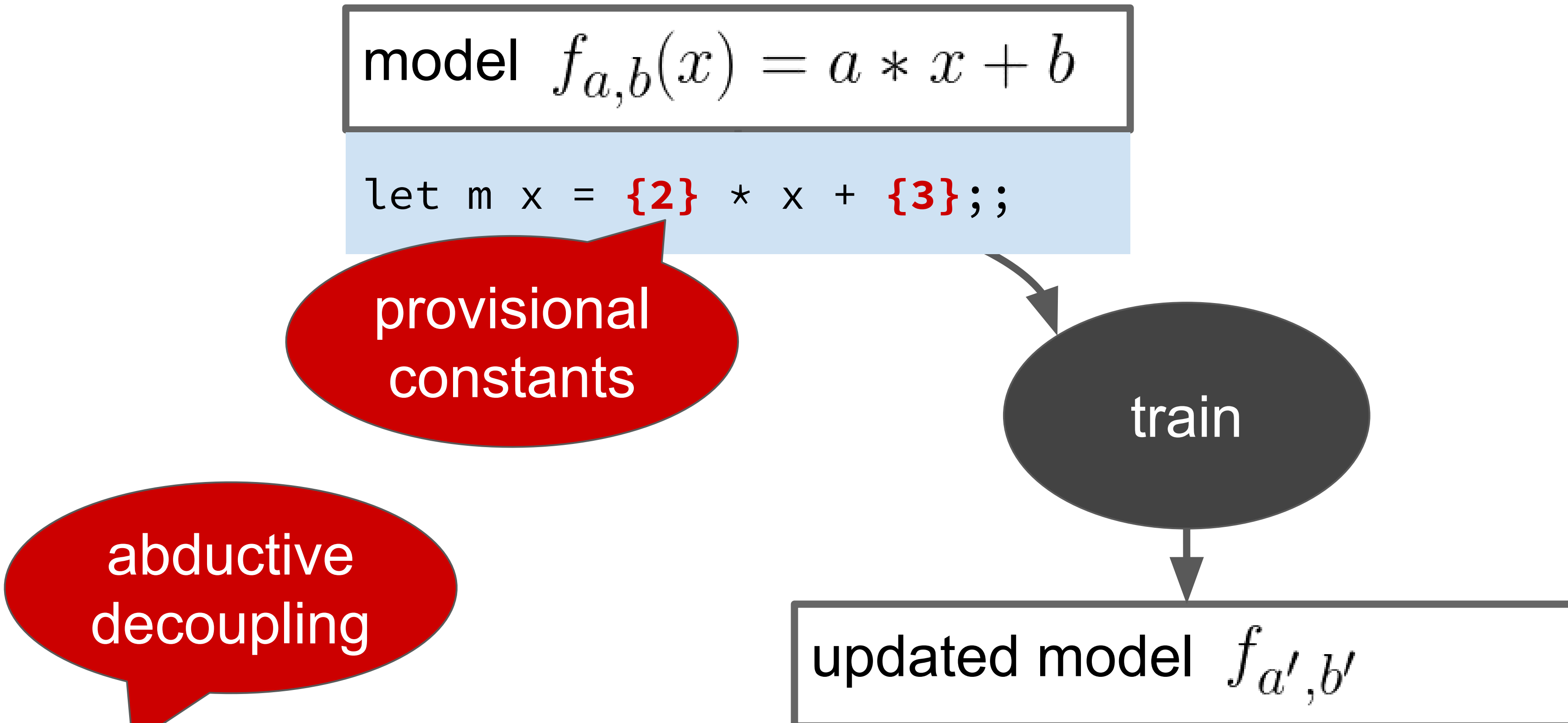
provisional constants

use

output $f_{a,b}(x_0)$

```
m 0;;
(* simply function application *)
```

"Parameter tuning via targeted abduction"




```
let f @ p = m in     (* "decouple" model f and parameters p *)
let q = optimise p in (* compute "better" parameter values *)
Let m' = f q in     (* "improve" model using new parameters *)
...
```

Abductive decoupling: informal semantics

```
let m x = {2} * x + {3};;

let f @ p = m in
let q = optimise p in
f q;;
```

```
val m = fun x -> {2} * x + {3}
```

model with provisional constants

```
-------------------------------------------------------------------
val f = fun (p1,p2) -> fun x -> p1 * x + p2
```

parameterised model

```
val p = (2,3)
```

parameter vector

Abductive decoupling: informal semantics

```
let m x = {2} * x + {3};;

let f @ p = m in
let q = optimise p in
f q;;
```

```
val m = fun x -> {2} * x + {3}
```

model with provisional constants

```
val f = fun (p1,p2) -> fun x -> p1 * x + p2
```

parameterised model

```
val p = (2,3)
```

parameter vector

$$\frac{\Gamma \vdash A}{\Gamma \vdash (P \Rightarrow A) \wedge P}$$

abduction rule

Promoting provisional to definitive constants

```
let m x = {2} * x + {3};;

let f @ p = m in
let q = p in
f q;;
```

```
val m = fun x -> {2} * x + {3}
```

model with provisional constants

```
------------------------------------------------------------------
val f = fun (p1,p2) -> fun x -> p1 * x + p2
```

parameterised model

```
val p = (2,3)
```

parameter vector

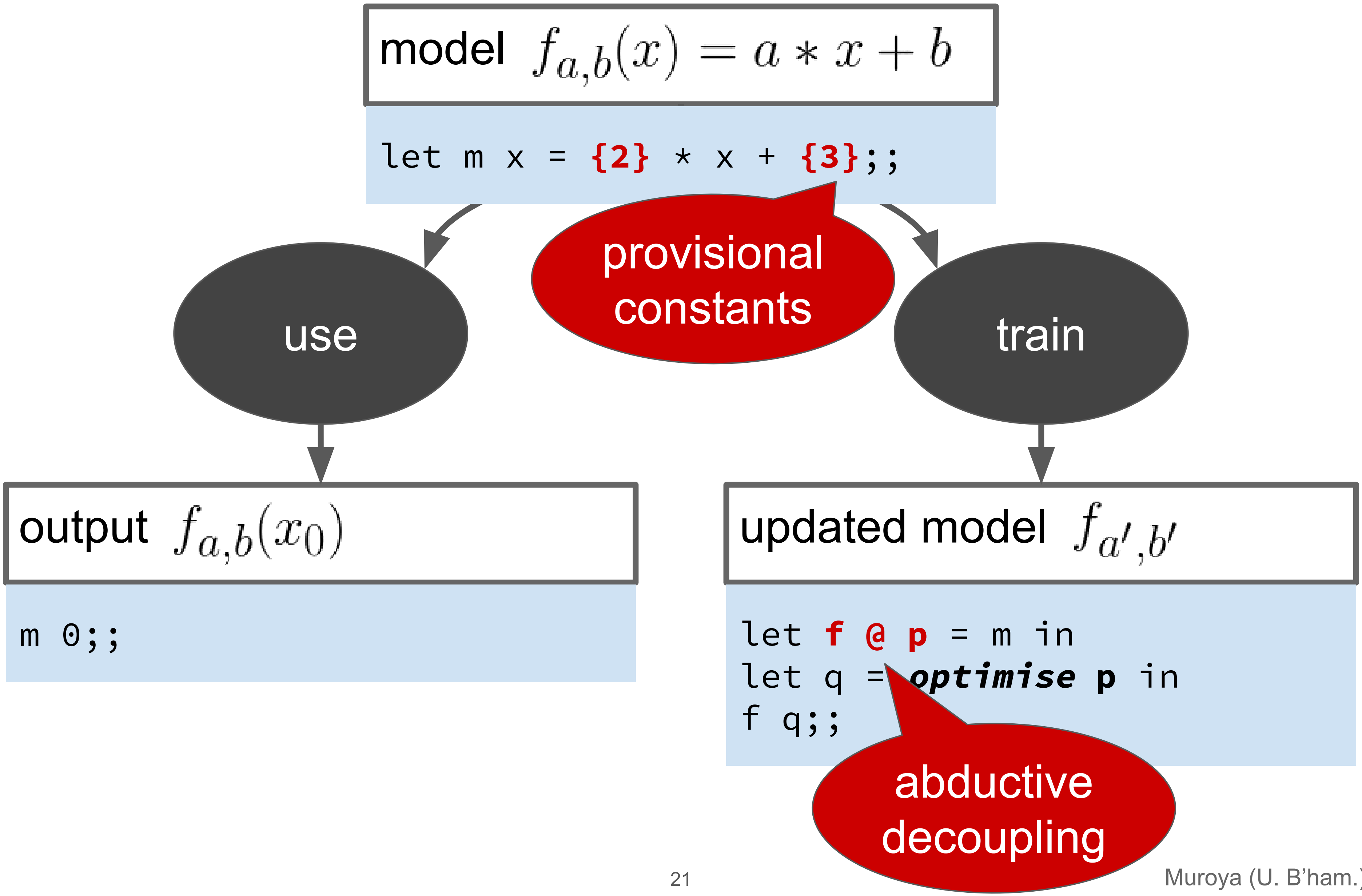
```
------------------------------------------------------------------
val q = (2,3)
```

(trivially updated) parameter vector

```
- = fun x -> 2 * x + 3
```

result: model with definitive constants

Parameter tuning via targeted abduction



Targeted abduction: syntax & types

(fixed) field

$$\frac{}{- \mid \Gamma \vdash \{c\} : \mathbb{F}}$$

provisional constant

opaque vector space, representing $\mathbb{F}^n$

$$\frac{\Delta, a \mid \Gamma, f : V_a \to T, x : V_a \vdash t : T'}{\Delta \mid \Gamma \vdash \texttt{abd}\ f\texttt{@}x\ \texttt{->}\ t : T \to T'}$$

abduction

```
let f@x = u in t ≡ (abd f@x -> t) u
```

Targeted abduction: syntax & types

(fixed) field

$$\frac{}{- \mid \Gamma \vdash \{c\} : \mathbb{F}}$$

provisional constant

opaque vector space, representing $\mathbb{F}^n$

$$\frac{\Delta, a \mid \Gamma, f : V_a \to T, x : V_a \vdash t : T'}{\Delta \mid \Gamma \vdash \texttt{abd}\ f\texttt{@}x\ \texttt{->}\ t : T \to T'}$$

```
(* abduction of open terms *)
let m x = {2} * x + n in
let f @ p = m in
...
```

Targeted abduction: *opaque vectors*

- size determined **dynamically**
- order of coordinates **unknown**
  - … yet we want deterministic programs
  - always point-free (no access to bases/coordinates)
  - only symmetric operations (invariant over permutation of bases/coordinates)

- possible in theory
  - symmetric tensors
- reasonable in practice
  - not all, *but most*, optimisation algorithms are symmetric

Targeted abduction: *symmetric vector operations*

standard vector operations

$$+_a : V_a \to V_a \to V_a \qquad \text{(vector addition)}$$

$$\times_a : \mathbb{F} \to V_a \to V_a \qquad \text{(scalar multiplication)}$$

$$\bullet_a : V_a \to V_a \to \mathbb{F}, \qquad \text{(dot product)}$$

iterated vector operations

$$+^L_a : (V_a \to V_a) \to V_a \to V_a \qquad \text{(left-iterative vector addition)}$$

$$+^R_a : V_a \to (V_a \to V_a) \to V_a \qquad \text{(right-iterative vector addition)}$$

$$\times^L_a : (V_a \to \mathbb{F}) \to V_a \to V_a \qquad \text{(left-iterative scalar multiplication)}$$

Targeted abduction: example use

numerical gradient descent

```
let m x = {2} * x + {3};;

let f @ p = m in
let q = grad_desc f p loss 0.001 in
f q;;
```

```
(* least square on some reference data *)
let loss f p = ...;;

(* numerical gradient descent *)
let grad_desc f p loss rate =
  let d = 0.001 in
  let g e =
    let old = loss f p in
    let new = loss f (p ⊞ (d ⊠ e)) in
    (((old - new) / d) * rate) ⊠ e in
  g |⊞ p;;
```

folding over standard basis

$$f +_a^L v_0 := \texttt{foldr}\ (\lambda e \lambda v. f(e) + v)\ E_a\ v_0$$

Targeted abduction: syntax & types

(fixed) field

opaque vector space, representing $\mathbb{F}^n$

only symmetric operations on vectors

$$\frac{}{- \mid \Gamma \vdash \{c\} : \mathbb{F}}$$

$$\frac{\Delta, a \mid \Gamma, f : V_a \to T, x : V_a \vdash t : T'}{\Delta \mid \Gamma \vdash \texttt{abd}\ f\texttt{@}x\ \texttt{->}\ t : T \to T'}$$

provisional constant

abduction

`let f@x = u in t` ≡ `(abd f@x -> t) u`

Targeted abduction: operational semantics

- provisional constants are linear!

```
let m x = {0} * x + {0};;
```

vs

```
let p = {0} in
let m x = p * x + p;;
```

- graph rewriting semantics
  - … based on Geometry of Interaction
  - http://www.cs.bham.ac.uk/~drg/goa/visualiser/
  - determinism
  - soundness of execution
  - safety of garbage-collection
  - call-by-value evaluation

Conclusions

- a fully-integrated language for parameter tuning
  - abductive decoupling “abd”
  - simply-typed + abduction rule
  - formal operational semantics
    - call-by-value
    - determinism
    - sound execution & safe garbage-collection

$$\frac{\Gamma \vdash A}{\Gamma \vdash (P \Rightarrow A) \wedge P}$$

http://www.cs.bham.ac.uk/~drg/goa/visualiser/

- open problems
  - actual ML compiler extension
    - abduction is dynamic & complex
    - … but not computationally dominant
  - stochastical machinary