

Towards Behavioral Theory of Algorithms

Dean Rosenzweig

Microsoft Research,
on leave from University of Zagreb

Algorithms

Algorithms

- arithmetical

Algorithms

- arithmetical
- algebraic

Algorithms

- arithmetical
- algebraic
- geometric

Algorithms

- arithmetical
- algebraic
- geometric
- of calculus

Algorithms

- arithmetical
- algebraic
- geometric
- of calculus
- paper-and-pencil

Algorithms

- arithmetical
- algebraic
- geometric
- of calculus
- paper-and-pencil
- mouse-clicking

Algorithms

- arithmetical
- algebraic
- geometric
- of calculus
- paper-and-pencil
- mouse-clicking
- patient handling

Steps

Steps

- long-division algorithm

Steps

- long-division algorithm – a step in
- cryptographic algorithm

Steps

- long-division algorithm – a step in
- cryptographic algorithm – a step in
- cryptographic protocol

Steps

- long-division algorithm – a step in
- cryptographic algorithm – a step in
- cryptographic protocol – a step in
- connection establishment

Steps

- long-division algorithm – a step in
- cryptographic algorithm – a step in
- cryptographic protocol – a step in
- connection establishment – a step in
- client-server application

Church-Turing Thesis

Church-Turing Thesis

Every computable function $\mathbb{N}^k \rightarrow \mathbb{N}$ is computable by a Turing machine.

Church-Turing Thesis

Every computable function $\mathbb{N}^k \rightarrow \mathbb{N}$ is computable by a Turing machine.

Every arithmetical algorithm can be simulated by a Turing machine.

Church-Turing Thesis

Every computable function $\mathbb{N}^k \rightarrow \mathbb{N}$ is computable by a Turing machine.

Every arithmetical algorithm can be simulated by a Turing machine.

If this is essentially all about algorithms, then why semantics?

Church-Turing Thesis

Every computable function $\mathbb{N}^k \rightarrow \mathbb{N}$ is computable by a Turing machine.

Every arithmetical algorithm can be simulated by a Turing machine.

If this is essentially all about algorithms, then why semantics?

The abstraction level of the simulation is fixed.

Church-Turing Thesis

Every computable function $\mathbb{N}^k \rightarrow \mathbb{N}$ is computable by a Turing machine.

Every arithmetical algorithm can be simulated by a Turing machine.

If this is essentially all about algorithms, then why semantics?

The abstraction level of the simulation is fixed. Intent of the algorithm buried under layers of representation.

Abstract Algorithms

Algorithms over ADT's (lists, trees, . . .)

Abstract Algorithms

Algorithms over ADT's (lists, trees, . . .)

Euclidean ring: structure where Euclidean algorithm works (integers, Gaussian integers, polynomials, . . .)

Abstract Algorithms

Algorithms over ADT's (lists, trees, . . .)

Euclidean ring: structure where Euclidean algorithm works (integers, Gaussian integers, polynomials, . . .)

Complete metric space: structure where Banach fixpoint algorithm works

Abstract Algorithms

Algorithms over ADT's (lists, trees, . . .)

Euclidean ring: structure where Euclidean algorithm works (integers, Gaussian integers, polynomials, . . .)

Complete metric space: structure where Banach fixpoint algorithm works

CPO: structure where Knaster-Tarski least fixpoint algorithm works

Are Abstract Algorithms Algorithms?

Are Abstract Algorithms Algorithms?

or just schemata,

Are Abstract Algorithms Algorithms?

or just schemata, which instantiate to algorithms only upon “implementation”,
via “encoding”?

Are Abstract Algorithms Algorithms?

or just schemata, which instantiate to algorithms only upon “implementation”, via “encoding”?

By behavioral theory, **yes**,

Are Abstract Algorithms Algorithms?

or just schemata, which instantiate to algorithms only upon “implementation”, via “encoding”?

By behavioral theory, **yes**, they are algorithms and need to be captured.

Are Abstract Algorithms Algorithms?

or just schemata, which instantiate to algorithms only upon “implementation”, via “encoding”?

By behavioral theory, **yes**, they are algorithms and need to be captured.

What do they work on, what are their states?

Abstract States

Abstract States

FO structures (vocabulary Υ , carrier $|X|$, interpretation of function and relation symbols)

Abstract States

FO structures (vocabulary Υ , carrier $|X|$, interpretation of function and relation symbols)

all structures of logic are FO

Abstract States

FO structures (vocabulary Υ , carrier $|X|$, interpretation of function and relation symbols)

all structures of logic are FO (although not all logic is FO)

Abstract States

FO structures (vocabulary Υ , carrier $|X|$, interpretation of function and relation symbols)

all structures of logic are FO (although not all logic is FO)

inessential technical choices:

Abstract States

FO structures (vocabulary Υ , carrier $|X|$, interpretation of function and relation symbols)

all structures of logic are FO (although not all logic is FO)

inessential technical choices:

Booleans and Boolean operations in vocabulary, purely functional structures

Abstract States

FO structures (vocabulary Υ , carrier $|X|$, interpretation of function and relation symbols)

all structures of logic are FO (although not all logic is FO)

inessential technical choices:

Booleans and Boolean operations in vocabulary, purely functional structures

undef in vocabulary, to model partial functions

Abstract States are Memories

Abstract States are Memories

Structure as memory: location $\langle f, \langle a_1, \dots, a_n \rangle \rangle$ ($f \in \Upsilon, a_i \in X$)

Abstract States are Memories

Structure as memory: location $\langle f, \langle a_1, \dots, a_n \rangle \rangle$ ($f \in \Upsilon, a_i \in X$)

structure = assignment of values to locations

Abstract States are Memories

Structure as memory: location $\langle f, \langle a_1, \dots, a_n \rangle \rangle$ ($f \in \Upsilon, a_i \in X$)

structure = assignment of values to locations

given $|X|, \Upsilon, X \leftrightarrow (\dots, f_X, \dots) \leftrightarrow (\dots, l_X, \dots)$

Abstract States 2

An algorithm over Υ has a class of *states*: isomorphism-closed class of Υ -structures S

Abstract States 2

An algorithm over Υ has a class of *states*: isomorphism-closed class of Υ -structures S

ex. Euclidean rings, CPOs, ADTs, . . .

Abstract States 2

An algorithm over Υ has a class of *states*: isomorphism-closed class of Υ -structures S

ex. Euclidean rings, CPOs, ADTs, . . .

isomorphism closure means: everything relevant must be shown in the vocabulary

Abstract States 2

An algorithm over Υ has a class of *states*: isomorphism-closed class of Υ -structures S

ex. Euclidean rings, CPOs, ADTs, . . .

isomorphism closure means: everything relevant must be shown in the vocabulary

initial states $\emptyset \neq I \subseteq S$, also isomorphism closed

Algorithms Transform States

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$,

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$, but how?

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$, but how?

if $Y = \tau(X)$, let $\Delta(X) = Y - X = \{\langle l, l_Y \rangle : l_Y \neq l_X\}$

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$, but how?

if $Y = \tau(X)$, let $\Delta(X) = Y - X = \{\langle l, l_Y \rangle : l_Y \neq l_X\}$

assume $|Y| = |X|$, algorithm doesn't change carrier

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$, but how?

if $Y = \tau(X)$, let $\Delta(X) = Y - X = \{\langle l, l_Y \rangle : l_Y \neq l_X\}$

assume $|Y| = |X|$, algorithm doesn't change carrier (to be discussed later)

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$, but how?

if $Y = \tau(X)$, let $\Delta(X) = Y - X = \{\langle l, l_Y \rangle : l_Y \neq l_X\}$

assume $|Y| = |X|$, algorithm doesn't change carrier (to be discussed later)

$\Delta(X) = \{\langle \langle f, \vec{a} \rangle, b \rangle : f_Y(\vec{a}) = b \neq f_X(\vec{a}), \vec{a}, b \in |X| = |Y| \}$

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$, but how?

if $Y = \tau(X)$, let $\Delta(X) = Y - X = \{\langle l, l_Y \rangle : l_Y \neq l_X\}$

assume $|Y| = |X|$, algorithm doesn't change carrier (to be discussed later)

$\Delta(X) = \{\langle \langle f, \vec{a} \rangle, b \rangle : f_Y(\vec{a}) = b \neq f_X(\vec{a}), \vec{a}, b \in |X| = |Y| \}$

structures as memories: Y arises from X by executing a set of assignments of form $\langle \langle f, \vec{a} \rangle, b \rangle$,

Algorithms Transform States

algorithm defines one-step state transformation $\tau : S \rightarrow S$, but how?

if $Y = \tau(X)$, let $\Delta(X) = Y - X = \{\langle l, l_Y \rangle : l_Y \neq l_X\}$

assume $|Y| = |X|$, algorithm doesn't change carrier (to be discussed later)

$\Delta(X) = \{\langle \langle f, \vec{a} \rangle, b \rangle : f_Y(\vec{a}) = b \neq f_X(\vec{a}), \vec{a}, b \in |X| = |Y| \}$

structures as memories: Y arises from X by executing a set of assignments of form $\langle \langle f, \vec{a} \rangle, b \rangle$, could also be read as $f(\vec{a}) := b$, **updates**

Algorithms Do in Finite Steps

Algorithms Do in Finite Steps

what else?

Algorithms Do in Finite Steps

what else? algorithms in each step do something **finite**:

Algorithms Do in Finite Steps

what else? algorithms in each step do something **finite**: $\Delta(X)$ is finite

Algorithms Do in Finite Steps

what else? algorithms in each step do something **finite**: $\Delta(X)$ is finite

finiteness will follow from other assumptions,

Algorithms Do in Finite Steps

what else? algorithms in each step do something **finite**: $\Delta(X)$ is finite

finiteness will follow from other assumptions, for different kinds of algorithms for different reasons

Abstractness Once More

Abstractness Once More

if $X \cong X'$, then $\tau(X) \cong \tau(X')$,

Abstractness Once More

if $X \cong X'$, then $\tau(X) \cong \tau(X')$, an algorithm cannot distinguish

Abstractness Once More

if $X \cong X'$, then $\tau(X) \cong \tau(X')$, an algorithm cannot distinguish

thus also $\Delta(X) \cong \Delta(X')$

Species of Algorithms

Species of Algorithms

- the simplest model: isolated, non-parallel

Species of Algorithms

- the simplest model: isolated, non-parallel small-step: Gurevich 2000.

Species of Algorithms

- the simplest model: isolated, non-parallel small-step: Gurevich 2000.
- parallel isolated algorithms

Species of Algorithms

- the simplest model: isolated, non-parallel small-step: Gurevich 2000.
- parallel isolated algorithms wide-step: Blass-Gurevich 2001.

Species of Algorithms

- the simplest model: isolated, non-parallel small-step: Gurevich 2000.
- parallel isolated algorithms wide-step: Blass-Gurevich 2001.
- interactive small-step algorithms (ordinary): Blass-Gurevich 2004.

Species of Algorithms

- the simplest model: isolated, non-parallel small-step: Gurevich 2000.
- parallel isolated algorithms wide-step: Blass-Gurevich 2001.
- interactive small-step algorithms (ordinary): Blass-Gurevich 2004.
- interactive small-step (general): Blass-Gurevich-Rossman 2005.

Species of Algorithms

- the simplest model: isolated, non-parallel small-step: Gurevich 2000.
- parallel isolated algorithms wide-step: Blass-Gurevich 2001.
- interactive small-step algorithms (ordinary): Blass-Gurevich 2004.
- interactive small-step (general): Blass-Gurevich-Rossman 2005.
- parallel interactive algorithms: Blass-Gurevich-Rosenzweig 2005.

Species of Algorithms

- the simplest model: isolated, non-parallel small-step: Gurevich 2000.
- parallel isolated algorithms wide-step: Blass-Gurevich 2001.
- interactive small-step algorithms (ordinary): Blass-Gurevich 2004.
- interactive small-step (general): Blass-Gurevich-Rossman 2005.
- parallel interactive algorithms: Blass-Gurevich-Rosenzweig 2005.
- distributed algorithms: Blass-Gurevich-Rosenzweig 2005.

- a case study: cryptographic algorithms, Rosenzweig-Runje-Slani 2004/5.

Q: Is This Just Philosophy?

Q: Is This Just Philosophy?

A: No, it enters the next Visual Studio.

Q: Is This Just Philosophy?

A: No, it enters the next Visual Studio.

Q: (a lady in ms-media) What does your group do?

Q: Is This Just Philosophy?

A: No, it enters the next Visual Studio.

Q: (a lady in ms-media) What does your group do?

A: (the author) Well, . . . , we investigate modelling methodology,. . . , what we are after is understanding . . .

Q: Is This Just Philosophy?

A: No, it enters the next Visual Studio.

Q: (a lady in ms-media) What does your group do?

A: (the author) Well, . . . , we investigate modelling methodology,. . . , what we are after is understanding . . .

Q: Yeah?

Q: Is This Just Philosophy?

A: No, it enters the next Visual Studio.

Q: (a lady in ms-media) What does your group do?

A: (the author) Well, . . . , we investigate modelling methodology,. . . , what we are after is understanding . . .

Q: Yeah?

A: Well, when you understand, you can build a good model. . .

Q: Yeah?

Q: Yeah?

A: And from a good model, you can automatically generate good tests for the implementation.

Q: Yeah?

A: And from a good model, you can automatically generate good tests for the implementation.

Q: Oh I see.

Whither Visual Studio?

Whither Visual Studio?

A large class of algorithms (parallel ordinary interactive) is provably encodable in the specification language of ASMs

Whither Visual Studio?

A large class of algorithms (parallel ordinary interactive) is provably encodable in the specification language of ASMs

15 years of modelling experience with ASM language (semantics of real programming languages, compilers, concurrent systems, hardware, . . .)

Whither Visual Studio?

A large class of algorithms (parallel ordinary interactive) is provably encodable in the specification language of ASMs

15 years of modelling experience with ASM language (semantics of real programming languages, compilers, concurrent systems, hardware, . . .)

ASM language implemented in AsmL (and others), recently Spec#,

Whither Visual Studio?

A large class of algorithms (parallel ordinary interactive) is provably encodable in the specification language of ASMs

15 years of modelling experience with ASM language (semantics of real programming languages, compilers, concurrent systems, hardware, . . .)

ASM language implemented in AsmL (and others), recently Spec#, direct extension of c#

Whither Visual Studio?

A large class of algorithms (parallel ordinary interactive) is provably encodable in the specification language of ASMs

15 years of modelling experience with ASM language (semantics of real programming languages, compilers, concurrent systems, hardware, . . .)

ASM language implemented in AsmL (and others), recently Spec#, direct extension of c#

Spec# used for assertional verification, model-checking and, under SpecExplorer tool, for model-based test generation, for any .net based code

testers want it, and now VS wants it

Postulates (Definition) for Small-Step

Postulates (Definition) for Small-Step

- states

Postulates (Definition) for Small-Step

- states
- isomorphism

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything
- **updates**

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything
- **updates** $\Delta(X)$

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything
- **updates** $\Delta(X)$ if non-contradictory, $\tau(X) = X + \Delta(X)$, otherwise algorithm fails,

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything
- **updates** $\Delta(X)$ if non-contradictory, $\tau(X) = X + \Delta(X)$, otherwise algorithm fails,
- **bounded work**

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything
- **updates** $\Delta(X)$ if non-contradictory, $\tau(X) = X + \Delta(X)$, otherwise algorithm fails,
- **bounded work** “algorithm is defined by a finite text”:

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything
- **updates** $\Delta(X)$ if non-contradictory, $\tau(X) = X + \Delta(X)$, otherwise algorithm fails,
- **bounded work** “algorithm is defined by a finite text”: there is a finite set of terms T s.t. $X =_T Y \Rightarrow \Delta(X) = \Delta(Y)$

Postulates (Definition) for Small-Step

- **states**
- **isomorphism** preserves everything
- **updates** $\Delta(X)$ if non-contradictory, $\tau(X) = X + \Delta(X)$, otherwise algorithm fails,
- **bounded work** “algorithm is defined by a finite text”: there is a finite set of terms T s.t. $X =_T Y \Rightarrow \Delta(X) = \Delta(Y)$

interaction is the source of all nondeterminism

Interactive Algorithms, Ordinary

Interactive Algorithms, Ordinary

intrastep interaction

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

queries: send m to p

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

queries: send m to p (labels—send, to, structure elements— m, p)

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

queries: send m to p (labels—send, to, structure elements— m, p)

answers:

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

queries: send m to p (labels—send, to, structure elements— m, p)

answers: structure elements

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

queries: send m to p (labels—send, to, structure elements— m, p)

answers: structure elements

ordinary: all answers needed,

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

queries: send m to p (labels—send, to, structure elements— m, p)

answers: structure elements

ordinary: all answers needed, order of answers doesn't matter

Interactive Algorithms, Ordinary

intrastep interaction (step is in the eye of beholder)

queries: send m to p (labels—send, to, structure elements— m, p)

answers: structure elements

ordinary: all answers needed, order of answers doesn't matter

environment behavior: answer *function*

Synchrony and Asynchrony

Synchrony and Asynchrony

queries with trivial arguments: *input*

Synchrony and Asynchrony

queries with trivial arguments: *input*

queries with trivial answers: *output*, asynchronous

Synchrony and Asynchrony

queries with trivial arguments: *input*

queries with trivial answers: *output*, asynchronous

output-input queries: synchronous,

Synchrony and Asynchrony

queries with trivial arguments: *input*

queries with trivial answers: *output*, asynchronous

output-input queries: synchronous, caller cannot complete the step without an answer

Synchrony and Asynchrony

queries with trivial arguments: *input*

queries with trivial answers: *output*, asynchronous

output-input queries: synchronous, caller cannot complete the step without an answer

output-input can be faked

Synchrony and Asynchrony

queries with trivial arguments: *input*

queries with trivial answers: *output*, asynchronous

output-input queries: synchronous, caller cannot complete the step without an answer

output-input can be faked by output + input

Synchrony and Asynchrony

queries with trivial arguments: *input*

queries with trivial answers: *output*, asynchronous

output-input queries: synchronous, caller cannot complete the step without an answer

output-input can be faked by output + input with a shared unique argument value - “callback”

cf. socket layer of tcp/ip, . . . , modelling synchrony in asynchronous π -calculi

Postulates for Ordinary Interactive Small-Step Algorithms

Postulates for Ordinary Interactive Small-Step Algorithms

- **states**

Postulates for Ordinary Interactive Small-Step Algorithms

- **states**
- **isomorphism**

Postulates for Ordinary Interactive Small-Step Algorithms

- **states**
- **isomorphism** preserves everything

Postulates for Ordinary Interactive Small-Step Algorithms

- **states**
- **isomorphism** preserves everything
- **interaction** algorithm determines, for X , a *causality* relation $\alpha \vdash_X q$

Postulates for Ordinary Interactive Small-Step Algorithms

- **states**
- **isomorphism** preserves everything
- **interaction** algorithm determines, for X , a *causality* relation $\alpha \vdash_X q$
- **updates**

Postulates for Ordinary Interactive Small-Step Algorithms

- **states**
- **isomorphism** preserves everything
- **interaction** algorithm determines, for X , a *causality* relation $\alpha \vdash_X q$
- **updates** if α is a context (minimal answer function closed under \vdash_X), either failure or $\Delta^+(X, \alpha)$

- **bounded work**

- **bounded work** there is a finite set of terms T s.t. $X =_{T,\alpha} Y$ entails equal behavior at X, Y under α , and a uniform bound on length and breadth of contexts

Backgrounds—Data Structures and Fresh Objects

Backgrounds—Data Structures and Fresh Objects

say an algorithm needs ordered pairs, or lists, or. . . , in all states

Backgrounds—Data Structures and Fresh Objects

say an algorithm needs ordered pairs, or lists, or . . . , in all states

the problem: how to “create” fresh objects, without having to “create” pairs, lists, . . . related to it?

Backgrounds—Data Structures and Fresh Objects

say an algorithm needs ordered pairs, or lists, or . . . , in all states

the problem: how to “create” fresh objects, without having to “create” pairs, lists, . . . related to it?

background vocabulary (ex. pair, fst, snd, or cons, hd, tl, . . .)

Backgrounds—Data Structures and Fresh Objects

say an algorithm needs ordered pairs, or lists, or . . . , in all states

the problem: how to “create” fresh objects, without having to “create” pairs, lists, . . . related to it?

background vocabulary (ex. pair, fst, snd, or cons, hd, tl, . . .)

unary predicate Atomic also assumed

Background Classes

Background Classes

an isomorphism-closed class K of structures over a vocabulary is a *background class* (Blass-Gurevich 2000) if

Background Classes

an isomorphism-closed class K of structures over a vocabulary is a *background class* (Blass-Gurevich 2000) if

- $(\forall U)(\exists X \in K) \text{Atoms}(K) = U$

Background Classes

an isomorphism-closed class K of structures over a vocabulary is a *background class* (Blass-Gurevich 2000) if

- $(\forall U)(\exists X \in K) \text{Atoms}(K) = U$
- for $X, Y \in K$, any set-embedding $\xi : \text{Atoms}(X) \rightarrow \text{Atoms}(Y)$ uniquely extends to a structure-embedding

Background Classes

an isomorphism-closed class K of structures over a vocabulary is a *background class* (Blass-Gurevich 2000) if

- $(\forall U)(\exists X \in K) \text{Atoms}(K) = U$
- for $X, Y \in K$, any set-embedding $\xi : \text{Atoms}(X) \rightarrow \text{Atoms}(Y)$ uniquely extends to a structure-embedding
- for $X \in K$, every $x \in X$ has an *envelope*—smallest K -substructure containing x .

finitary background class: support of every singleton is finite (support—atoms in the envelope)

Backgrounds and Reserve of Algorithms

Backgrounds and Reserve of Algorithms

Fix K , of vocabulary Υ_0 . K is the background of an algorithm over $\Upsilon \supseteq \Upsilon_0$ if

Backgrounds and Reserve of Algorithms

Fix K , of vocabulary Υ_0 . K is the background of an algorithm over $\Upsilon \supseteq \Upsilon_0$ if

- no background function $f \in \Upsilon_0$ is ever updated, and

Backgrounds and Reserve of Algorithms

Fix K , of vocabulary Υ_0 . K is the background of an algorithm over $\Upsilon \supseteq \Upsilon_0$ if

- no background function $f \in \Upsilon_0$ is ever updated, and
- the reduct of every state to Υ_0 is in K .

Backgrounds and Reserve of Algorithms

Fix K , of vocabulary Υ_0 . K is the background of an algorithm over $\Upsilon \supseteq \Upsilon_0$ if

- no background function $f \in \Upsilon_0$ is ever updated, and
- the reduct of every state to Υ_0 is in K .

exposed elements: in domain or codomain of a foreground function

Backgrounds and Reserve of Algorithms

Fix K , of vocabulary Υ_0 . K is the background of an algorithm over $\Upsilon \supseteq \Upsilon_0$ if

- no background function $f \in \Upsilon_0$ is ever updated, and
- the reduct of every state to Υ_0 is in K .

exposed elements: in domain or codomain of a foreground function

active part of a state: the envelope of the set of exposed elements

reserve (“heap”) of a state: atoms not in the active part

reserve (“heap”) of a state: atoms not in the active part

Theorem Every permutation of the reserve extends to a unique isomorphism, which is the identity on the active part.

Parallel Ordinary Interactive Algorithms

Parallel Ordinary Interactive Algorithms

background of ordered pairs and hereditarily finite multisets assumed

Parallel Ordinary Interactive Algorithms

background of ordered pairs and hereditarily finite multisets assumed

there is a term `Proclet`, denoting a finite (multi)set in every state

Parallel Ordinary Interactive Algorithms

background of ordered pairs and hereditarily finite multisets assumed

there is a term `Proclet`, denoting a finite (multi)set in every state

every proclet executes the same algorithm, interpreting a 0-ary symbol `me` specially, as itself

Parallel Ordinary Interactive Algorithms

background of ordered pairs and hereditarily finite multisets assumed

there is a term `Proclet`, denoting a finite (multi)set in every state

every `proclet` executes the same algorithm, interpreting a 0-ary symbol `me` specially, as itself

they communicate by special queries, pushing (many-to-one) or pulling (one-to-many)

they may also issue external queries, communicating with the environment proper

Pushing and Pulling

Pushing and Pulling

proctet algorithm is a small-step ordinary interactive algorithm

Pushing and Pulling

proctet algorithm is a small-step ordinary interactive algorithm

proctet p pushes to q by issueing a query push m to q

Pushing and Pulling

proclet algorithm is a small-step ordinary interactive algorithm

proclet p pushes to q by issueing a query push m to q

proclet q , as answer to query myMail, obtains the multiset of all m 's pushed to it "so far"

proclet p displays m at position i by issueing a query of form `display m at i`
(it may do this once in a step)

proclet p displays m at position i by issueing a query of form `display m at i` (it may do this once in a step)

proclet q sees it by issueing a query `pullFrom p at i` , obtaining the value displayed “so far”, or `undef` if none

So Far ?

So Far ?

take a quantifier-proclet, computing $(\forall x \in r)\varphi(x)$, where r denotes a finite (multi)set

So Far ?

take a quantifier-proclet, computing $(\forall x \in r)\varphi(x)$, where r denotes a finite (multi)set

it first displays a signal telling its children to go

So Far ?

take a quantifier-proclet, computing $(\forall x \in r)\varphi(x)$, where r denotes a finite (multi)set

it first displays a signal telling its children to go

the children, one per each $c \in r$, upon seeing the signal, compute the truth value of $\varphi(c)$ and push it to parent

upon receiving all the mail, the quantifier computes $\text{AsSet}(\text{myMail}) = \{\text{true}\}$,
where AsSet is a background function

upon receiving all the mail, the quantifier computes $\text{AsSet}(\text{myMail}) = \{\text{true}\}$, where AsSet is a background function

there is a tradeoff in intelligence between proclats and “scheduler”—by “proclats do everything” principle we opt for stupid scheduler and clever proclats,

upon receiving all the mail, the quantifier computes $\text{AsSet}(\text{myMail}) = \{\text{true}\}$, where AsSet is a background function

there is a tradeoff in intelligence between proclets and “scheduler”—by “proclets do everything” principle we opt for stupid scheduler and clever proclets, the quantifier must know the cardinality of r , and busy-wait till it gets enough mail—expressible in its causality relation

Computational Cryptography Model

Computational Cryptography Model

Example: symmetric encryption scheme is a triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$

$$\mathcal{K} : \text{Parameter} \times \text{Coins} \longrightarrow \text{Key}$$

$$\mathcal{E} : \text{Key} \times \text{String} \times \text{Coins} \longrightarrow \text{Ciphertext}$$

$$\mathcal{D} : \text{Key} \times \text{String} \longrightarrow \text{Plaintext}$$

such that

$$\Pr[\mathcal{D}(k, \mathcal{E}(k, m, c)) = m] = 1$$

Computational Cryptography Model

Example: symmetric encryption scheme is a triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$

$$\mathcal{K} : \text{Parameter} \times \text{Coins} \longrightarrow \text{Key}$$

$$\mathcal{E} : \text{Key} \times \text{String} \times \text{Coins} \longrightarrow \text{Ciphertext}$$

$$\mathcal{D} : \text{Key} \times \text{String} \longrightarrow \text{Plaintext}$$

such that

$$\Pr[\mathcal{D}(k, \mathcal{E}(k, m, c)) = m] = 1$$

Security is expressed in terms of probabilistic PTIME algorithms

Ensembles and Indistinguishability

Ensembles and Indistinguishability

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ and pairing $\langle \cdot, \cdot \rangle$ induce ensembles on strings

$$\langle \mathcal{E}(\mathcal{K}(\eta, \cdot), 1, \cdot), 0 \rangle$$

Ensembles and Indistinguishability

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ and pairing $\langle \cdot, \cdot \rangle$ induce ensembles on strings

$$\langle \mathcal{E}(\mathcal{K}(\eta, \cdot), 1, \cdot), 0 \rangle$$

Ensembles and Indistinguishability

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ and pairing $\langle \cdot, \cdot \rangle$ induce ensembles on strings

$$\langle \mathcal{E}(\mathcal{K}(\eta, \cdot), 1, \cdot), 0 \rangle$$

Indistinguishability by PPTIME algorithms (as a function of η)

$$\mathcal{E}(\mathcal{K}(\eta, \cdot), 1, \cdot) \approx \mathcal{E}(\mathcal{K}(\eta, \cdot), \langle 0, 1 \rangle, \cdot)$$

Ensembles and Indistinguishability

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ and pairing $\langle \cdot, \cdot \rangle$ induce ensembles on strings

$$\langle \mathcal{E}(\mathcal{K}(\eta, \cdot), 1, \cdot), 0 \rangle$$

Indistinguishability by PPTIME algorithms (as a function of η)

$$\mathcal{E}(\mathcal{K}(\eta, \cdot), 1, \cdot) \approx \mathcal{E}(\mathcal{K}(\eta, \cdot), \langle 0, 1 \rangle, \cdot)$$

$$\text{let } K = \mathcal{K}(\eta, \cdot) \text{ in } \langle \mathcal{E}(K, 1, \cdot), K \rangle$$

$$\not\approx$$
$$\text{let } K = \mathcal{K}(\eta, \cdot) \text{ in } \langle \mathcal{E}(K, \langle 0, 1 \rangle, \cdot), K \rangle$$

Abstract Cryptography Model

- Abstract representation of computational cryptography model
 - We abstract from security parameter η
 - Unlikely events become impossible
 - Strings are represented with elements of abstract base set

Abstract Cryptography Model

- Abstract representation of computational cryptography model
 - We abstract from security parameter η
 - Unlikely events become impossible
 - Strings are represented with elements of abstract base set
- Use of background structures
 - Abstraction: identification of elements with desired properties in a background structure
 - Necessary for creation in public key case

Example: Symmetric Encryption Scheme

Example: Symmetric Encryption Scheme

Background vocabulary: key, encrypt, decrypt (pair, left, right)

Example: Symmetric Encryption Scheme

Background vocabulary: key, encrypt, decrypt (pair, left, right)

Support is infinite collection Coins

key : Coins \longrightarrow Key

encrypt : Key \times Message \times Coins \longrightarrow Ciphertext

decrypt : Key \times Ciphertext \longrightarrow Message \cup {undef}

Example: Symmetric Encryption Scheme

Background vocabulary: key, encrypt, decrypt (pair, left, right)

Support is infinite collection Coins

key : Coins \longrightarrow Key
encrypt : Key \times Message \times Coins \longrightarrow Ciphertext
decrypt : Key \times Ciphertext \longrightarrow Message \cup {undef}

$$\text{decrypt}(k_1, \text{encrypt}(k_2, m, c)) = \begin{cases} m & \text{if } k_1 = k_2 \\ \text{undef} & \text{otherwise} \end{cases}$$

How to Represent Computational Indistinguishability?

How to Represent Computational Indistinguishability?

identity of structures?

How to Represent Computational Indistinguishability?

identity of structures?

isomorphism?

How to Represent Computational Indistinguishability?

identity of structures?

isomorphism?

something else?

Some Definitions

Some Definitions

X and Y are distinguished by algorithm A if $\text{output}_{\tau_A}(X) = \text{true}$ and $\text{output}_{\tau_A}(Y) = \text{false}$

Some Definitions

X and Y are distinguished by algorithm A if $\text{output}_{\tau_A}(X) = \text{true}$ and $\text{output}_{\tau_A}(Y) = \text{false}$

X and Y are indistinguishable by small-step algorithms if there is no small-step algorithm distinguishing them

Some Definitions

X and Y are distinguished by algorithm A if $\text{output}_{\tau_A(X)} = \text{true}$ and $\text{output}_{\tau_A(Y)} = \text{false}$

X and Y are indistinguishable by small-step algorithms if there is no small-step algorithm distinguishing them

X and Y are similar ($X \sim Y$) if $\text{Val}(t_1, X) = \text{Val}(t_2, X)$ iff $\text{Val}(t_1, Y) = \text{Val}(t_2, Y)$ for all terms t_1, t_2

Some Definitions

X and Y are distinguished by algorithm A if $\text{output}_{\tau_A(X)} = \text{true}$ and $\text{output}_{\tau_A(Y)} = \text{false}$

X and Y are indistinguishable by small-step algorithms if there is no small-step algorithm distinguishing them

X and Y are similar ($X \sim Y$) if $\text{Val}(t_1, X) = \text{Val}(t_2, X)$ iff $\text{Val}(t_1, Y) = \text{Val}(t_2, Y)$ for all terms t_1, t_2

a is accessible in X if $a = \text{Val}(t, X)$ for some t

Some Properties

Some Properties

indistinguishability = similarity

Some Properties

indistinguishability = similarity

No learning by own actions

Some Properties

indistinguishability = similarity

No learning by own actions

- $X \sim Y \Rightarrow \tau_A(X) \sim \tau_A(Y)$

Some Properties

indistinguishability = similarity

No learning by own actions

- $X \sim Y \Rightarrow \tau_A(X) \sim \tau_A(Y)$
- a inaccessible in $X \Rightarrow a$ inaccessible in $\tau_A(X)$

Environment actions only possible source of learning

Environment actions only possible source of learning

Holds also with importing over background structures

Example: Attempt at Abstraction by Isomorphism

Example: Attempt at Abstraction by Isomorphism

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Example: Attempt at Abstraction by Isomorphism

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{fst}, \text{snd}, \dots\}$

Example: Attempt at Abstraction by Isomorphism

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{fst}, \text{snd}, \dots\}$

Base set $|S_{\cong}| = |T_{\cong}| = \{0, 1, e, k, p, \dots\}$

Example: Attempt at Abstraction by Isomorphism

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{fst}, \text{snd}, \dots\}$

Base set $|S_{\cong}| = |T_{\cong}| = \{0, 1, e, k, p, \dots\}$

Interpretations in S_{\cong} and T_{\cong} : $f_{S_{\cong}} = f_{T_{\cong}} = e$

Interpretations in S_{\cong} and T_{\cong} : $f_{S_{\cong}} = f_{T_{\cong}} = e$

$$\Delta_{\cong}(I, S) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, p \rangle, \langle \text{fst}, p, 1 \rangle, \langle \text{snd}, p, 0 \rangle\}$$

$$\Delta_{\cong}(I, T) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, 1 \rangle\}$$

Interpretations in S_{\cong} and T_{\cong} : $f_{S_{\cong}} = f_{T_{\cong}} = e$

$$\Delta_{\cong}(I, S) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, p \rangle, \langle \text{fst}, p, 1 \rangle, \langle \text{snd}, p, 0 \rangle\}$$

$$\Delta_{\cong}(I, T) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, 1 \rangle\}$$

Interpretations in S_{\cong} and T_{\cong} : $f_{S_{\cong}} = f_{T_{\cong}} = e$

$$\Delta_{\cong}(I, S) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, p \rangle, \langle \text{fst}, p, 1 \rangle, \langle \text{snd}, p, 0 \rangle\}$$

$$\Delta_{\cong}(I, T) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, 1 \rangle\}$$

we had to *create* value for $\langle \text{decrypt}, \langle e, k \rangle \rangle$

Interpretations in S_{\cong} and T_{\cong} : $f_{S_{\cong}} = f_{T_{\cong}} = e$

$$\Delta_{\cong}(I, S) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, p \rangle, \langle \text{fst}, p, 1 \rangle, \langle \text{snd}, p, 0 \rangle\}$$

$$\Delta_{\cong}(I, T) = \{\langle g, k \rangle, \langle \text{decrypt}, \langle e, k \rangle, 1 \rangle\}$$

we had to *create* value for $\langle \text{decrypt}, \langle e, k \rangle \rangle$

its existence before interaction would have violated isomorphism!

Example Continued: Abstraction by Similarity

Example Continued: Abstraction by Similarity

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Example Continued: Abstraction by Similarity

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{left}, \text{right}, \dots\}$

Example Continued: Abstraction by Similarity

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{left}, \text{right}, \dots\}$

Base set $|S_\sim| = |T_\sim| = \{0, 1, e, k, p, \dots\}$

Example Continued: Abstraction by Similarity

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{left}, \text{right}, \dots\}$

Base set $|S_\sim| = |T_\sim| = \{0, 1, e, k, p, \dots\}$

Example Continued: Abstraction by Similarity

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{left}, \text{right}, \dots\}$

Base set $|S_\sim| = |T_\sim| = \{0, 1, e, k, p, \dots\}$

Interpretations in S_\sim : $f_{S_\sim} = e$,

Example Continued: Abstraction by Similarity

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{left}, \text{right}, \dots\}$

Base set $|S_\sim| = |T_\sim| = \{0, 1, e, k, p, \dots\}$

Interpretations in S_\sim : $f_{S_\sim} = e$, $\text{decrypt}_{S_\sim}(k, e) = p$, $\text{fst}_{S_\sim}(p) = 1$, $\text{snd}_{S_\sim}(p) = 0$

Example Continued: Abstraction by Similarity

S has $\{1, 0\}_K$, T has $\{1\}_K$, and they both learn K

Vocabulary $\Upsilon = \{f, g, \text{decrypt}, \text{left}, \text{right}, \dots\}$

Base set $|S_\sim| = |T_\sim| = \{0, 1, e, k, p, \dots\}$

Interpretations in S_\sim : $f_{S_\sim} = e$, $\text{decrypt}_{S_\sim}(k, e) = p$, $\text{fst}_{S_\sim}(p) = 1$, $\text{snd}_{S_\sim}(p) = 0$

Interpretations in T_\sim : $f_{S_\sim} = e$, $\text{decrypt}_{S_\sim}(k, e) = 1$

What we have learned is, in both cases,

$$\Delta_{\sim}(I, S) = \Delta_{\sim}(I, T) = \{(g, k)\}$$

What we have learned is, in both cases,

$$\Delta_{\sim}(I, S) = \Delta_{\sim}(I, T) = \{(g, k)\}$$

What we have learned is, in both cases,

$$\Delta_{\sim}(I, S) = \Delta_{\sim}(I, T) = \{(g, k)\}$$

We did not have to create differences, we *discovered* them.

Abstraction by Similarity

- Soundness follows directly adapting Abadi-Rogaway 2000.

$$R(S) \sim R(T) \implies S \approx T$$

- Completeness along Micciancio-Warinschi 2002.

$$S \approx T \implies R(S) \sim R(T)$$

- Proof-porting . . .