Equivalence in Computer Algebra

Relations, Canonical Forms, Normal Forms

Robert Smith

Symbolic Systems Engineer
Secure Outcomes, Inc.
Evergreen, CO

May 5, 2011

Motivation

- Naïve representation of data causes problems (e.g., finite subsets of $\mathbb{Z}$ as lists).

Motivation

- Naïve representation of data causes problems (e.g., finite subsets of $\mathbb{Z}$ as lists).
- How do we check if two representations of an object are equal?

Motivation

- Naïve representation of data causes problems (e.g., finite subsets of $\mathbb{Z}$ as lists).
- How do we check if two representations of an object are equal?
- We need a notion of standardizing the representation so we can algorithmically compare them.

Motivation

- Naïve representation of data causes problems (e.g., finite subsets of $\mathbb{Z}$ as lists).
- How do we check if two representations of an object are equal?
- We need a notion of standardizing the representation so we can algorithmically compare them.
- The goal of this talk is to give a semi-formal idea of things one needs to consider when writing computer algebra software from a mathematical standpoint.

Motivation

- Naïve representation of data causes problems (e.g., finite subsets of $\mathbb{Z}$ as lists).
- How do we check if two representations of an object are equal?
- We need a notion of standardizing the representation so we can algorithmically compare them.
- The goal of this talk is to give a semi-formal idea of things one needs to consider when writing computer algebra software from a mathematical standpoint.
- Specifically about equivalence.

Relations

- A relation is a property able to be shared by two elements.

Relations

- A relation is a property able to be shared by two elements.
- A relation is defined as a subset of the cartesian product of a set and itself.

Relations

- A relation is a property able to be shared by two elements.
- A relation is defined as a subset of the cartesian product of a set and itself.
- Those that are in this subset are *related*.

Relations

- A relation is a property able to be shared by two elements.
- A relation is defined as a subset of the cartesian product of a set and itself.
- Those that are in this subset are *related*.

Relations

- A relation is a property able to be shared by two elements.
- A relation is defined as a subset of the cartesian product of a set and itself.
- Those that are in this subset are *related*.

Definition

A **binary relation** ρ on a set S is defined by the set $R \subseteq S \times S$ such that for each $x, y \in S$, $(x, y) \in R$ iff $x \, \rho \, y$ is a tautology.

Equivalence

The most important relation is one stating equivalence.

Equivalence

The most important relation is one stating equivalence.

Definition

A binary relation $\sim$ is an **equivalence relation** on S if for all $a, b, c \in S$, the following hold:

The most important relation is one stating equivalence.

Definition

A binary relation $\sim$ is an **equivalence relation** on S if for all $a, b, c \in S$, the following hold:

Reflexivity $a \sim a$,

Equivalence

The most important relation is one stating equivalence.

Definition

A binary relation $\sim$ is an **equivalence relation** on S if for all $a, b, c \in S$, the following hold:

Reflexivity $a \sim a$,

Symmetry $a \sim b \iff b \sim a$, and

Equivalence

The most important relation is one stating equivalence.

Definition

A binary relation $\sim$ is an **equivalence relation** on S if for all $a, b, c \in S$, the following hold:

Reflexivity $a \sim a$,

Symmetry $a \sim b \iff b \sim a$, and

Transitivity $a \sim b \wedge b \sim c \implies a \sim c$.

Equivalence

The most important relation is one stating equivalence.

Definition

A binary relation $\sim$ is an **equivalence relation** on S if for all $a, b, c \in S$, the following hold:

Reflexivity $a \sim a$,

Symmetry $a \sim b \iff b \sim a$, and

Transitivity $a \sim b \wedge b \sim c \implies a \sim c$.

Equivalence

The most important relation is one stating equivalence.

Definition

A binary relation $\sim$ is an **equivalence relation** on S if for all $a, b, c \in S$, the following hold:

Reflexivity $a \sim a$,

Symmetry $a \sim b \iff b \sim a$, and

Transitivity $a \sim b \wedge b \sim c \implies a \sim c$.

Given $a \in S$, the set $\{x \in S \mid x \sim a\}$ is called the **equivalence class** of a. This is denoted $[a]$.

Typically three levels of equality in computer algebra. Given elements A, B in S, they may be equivalent on many different levels.

Typically three levels of equality in computer algebra. Given elements A, B in S, they may be equivalent on many different levels.

Object-Level If $B \in [A]$ (and therefore $A \in [B]$). They are "mathematically equal".

Typically three levels of equality in computer algebra. Given elements A, B in S, they may be equivalent on many different levels.

Object-Level
: If $B \in [A]$ (and therefore $A \in [B]$). They are "mathematically equal".

Form-Level
: If A and B are structurally and syntactically the same (i.e., if their representation in memory is identical, but reside in different areas of memory).

Typically three levels of equality in computer algebra. Given elements A, B in S, they may be equivalent on many different levels.

Object-Level If $B \in [A]$ (and therefore $A \in [B]$). They are "mathematically equal".

Form-Level If A and B are structurally and syntactically the same (i.e., if their representation in memory is identical, but reside in different areas of memory).

Data-Level If A and B are coinciding objects in computer memory ("pointer equality").

Typically three levels of equality in computer algebra. Given elements A, B in S, they may be equivalent on many different levels.

Object-Level If $B \in [A]$ (and therefore $A \in [B]$). They are "mathematically equal".

Form-Level If A and B are structurally and syntactically the same (i.e., if their representation in memory is identical, but reside in different areas of memory).

Data-Level If A and B are coinciding objects in computer memory ("pointer equality").

Equivalence in Computer Algebra

Typically three levels of equality in computer algebra. Given elements A, B in S, they may be equivalent on many different levels.

Object-Level
: If $B \in [A]$ (and therefore $A \in [B]$). They are "mathematically equal".

Form-Level
: If A and B are structurally and syntactically the same (i.e., if their representation in memory is identical, but reside in different areas of memory).

Data-Level
: If A and B are coinciding objects in computer memory ("pointer equality").

If we have object-level equality defined mathematically for a set, how do we obtain form-level equality?

Equality Example

Suppose we have $A := x^2 + 2x + 1$ and $B := (x + 1)^2$.

Equality Example

Suppose we have $A := x^2 + 2x + 1$ and $B := (x+1)^2$.

- $A = B$ at the object-level, clearly.

Equality Example

Suppose we have $A := x^2 + 2x + 1$ and $B := (x+1)^2$.

- $A = B$ at the object-level, clearly.
- $A \neq B$ at the form-level. Supposing we represented A and B as ASTs, A would have five leaves, while B would have only 3. Therefore, they can't even construct a bijective map between leaves.

Equality Example

Suppose we have $A := x^2 + 2x + 1$ and $B := (x+1)^2$.

- $A = B$ at the object-level, clearly.
- $A \neq B$ at the form-level. Supposing we represented A and B as ASTs, A would have five leaves, while B would have only 3. Therefore, they can't even construct a bijective map between leaves.
- It follows that A and B don't have data-level equivalence.

Equality Example

Suppose we have $A := x^2 + 2x + 1$ and $B := (x+1)^2$.

- $A = B$ at the object-level, clearly.
- $A \neq B$ at the form-level. Supposing we represented A and B as ASTs, A would have five leaves, while B would have only 3. Therefore, they can't even construct a bijective map between leaves.
- It follows that A and B don't have data-level equivalence.

If we had a procedure expand, then we could say $A = \text{expand}\, B$ at the form-level.

One way to do this is to choose a “standard” or “representative” element from each equivalence class.

One way to do this is to choose a “standard” or “representative” element from each equivalence class.

Definition

Let S be a set under the equivalence relation $\sim$. The **canonical form** of an element $x \in S$, denoted $\kappa(x)$, is an element of $[x]$ such that for all $y \in [x]$, $\kappa(y) = \kappa(x)$. The function $\kappa : S \to S$ is called the **canonizing function**.

One way to do this is to choose a “standard” or “representative” element from each equivalence class.

Definition

Let S be a set under the equivalence relation $\sim$. The **canonical form** of an element $x \in S$, denoted $\kappa(x)$, is an element of $[x]$ such that for all $y \in [x]$, $\kappa(y) = \kappa(x)$. The function $\kappa : S \to S$ is called the **canonizing function**.

This implies that $x \sim y \iff \kappa(x) = \kappa(y)$.

Canonizing Function Examples

- Consider $\mathbb{Q} \cong \mathbb{Z}^2$. Denote an element $a/b \in \mathbb{Q}$ as $\langle a, b \rangle$ for clarity.

Canonizing Function Examples

- Consider $\mathbb{Q} \cong \mathbb{Z}^2$. Denote an element $a/b \in \mathbb{Q}$ as $\langle a, b\rangle$ for clarity. Then one such canonizing function is

$$\kappa(\langle a, b\rangle) = \left\langle \operatorname{sgn}(ab)\frac{|a|}{\gcd(a,b)}, \frac{|b|}{\gcd(a,b)} \right\rangle.$$

Canonizing Function Examples

- Consider $\mathbb{Q} \cong \mathbb{Z}^2$. Denote an element $a/b \in \mathbb{Q}$ as $\langle a, b \rangle$ for clarity. Then one such canonizing function is
$$\kappa(\langle a, b \rangle) = \left\langle \operatorname{sgn}(ab)\frac{|a|}{\gcd(a,b)}, \frac{|b|}{\gcd(a,b)} \right\rangle .$$
- Consider the symmetric group represented by an n-tuple of distinct natural numbers.

Canonizing Function Examples

- Consider $\mathbb{Q} \cong \mathbb{Z}^2$. Denote an element $a/b \in \mathbb{Q}$ as $\langle a, b \rangle$ for clarity. Then one such canonizing function is

$$\kappa(\langle a, b \rangle) = \left\langle \operatorname{sgn}(ab)\frac{|a|}{\gcd(a,b)}, \frac{|b|}{\gcd(a,b)} \right\rangle.$$

- Consider the symmetric group represented by an n-tuple of distinct natural numbers. A canonical form of these objects would be a composition of disjoint cycles ordered by each cycle's least element, e.g.,

$$\kappa[(x_1, x_2, \ldots, x_n)] = (x_{1,1}, \ldots, x_{1,p}) \circ \cdots \circ (x_{k,1}, \ldots, x_{k,q})$$

with

$$\forall k : \min_j(x_{k,j}) = x_{1,j} \quad \text{and} \quad x_{1,1} < x_{2,1} < \cdots < x_{k,1}.$$

What Do Canonical Forms Do?

- Aside from picking a representative element of each equivalence class, it has a more practical value in computer algebra.

What Do Canonical Forms Do?

- Aside from picking a representative element of each equivalence class, it has a more practical value in computer algebra.
- If all elements of a domain are in canonical form, then we can do one very important thing: test for equality. With the previous polynomial example, expansion (and ordering by degree) allows testing equality of coefficients pairwise.

What Do Canonical Forms Do?

- Aside from picking a representative element of each equivalence class, it has a more practical value in computer algebra.
- If all elements of a domain are in canonical form, then we can do one very important thing: test for equality. With the previous polynomial example, expansion (and ordering by degree) allows testing equality of coefficients pairwise.
- This is why your grade-school teacher required all fractions be put into “canonical form”, so he or she could compare easily.

Computing with Non-Equivalence Relations

- Often, the relation of interest is not an equivalence relation. Instead it might be, e.g., an ordering relation.

Computing with Non-Equivalence Relations

- Often, the relation of interest is not an equivalence relation. Instead it might be, e.g., an ordering relation.
- However, this relation might be difficult to analyze.

Computing with Non-Equivalence Relations

- Often, the relation of interest is not an equivalence relation. Instead it might be, e.g., an ordering relation.
- However, this relation might be difficult to analyze.
- In fact, it may be difficult to effectively compute in the computer algebra world.

Normalizing Functions

If the objects in question have a canonical form, then it might be possible to translate a more general relation into an equivalence relation.

Normalizing Functions

If the objects in question have a canonical form, then it might be possible to translate a more general relation into an equivalence relation.

Definition

Given a relation ρ on X, a function $\eta : X^2 \to Y$ is called the ρ**-normalizing function** if for all $x \in X$, $\eta(x, x) = \eta_0$ and for all $a, b \in X$,

$$\eta(a, b) = \eta_0 \iff a \, \rho \, b.$$

If the objects in question have a canonical form, then it might be possible to translate a more general relation into an equivalence relation.

Definition

Given a relation ρ on X, a function $\eta : X^2 \to Y$ is called the ρ-**normalizing function** if for all $x \in X$, $\eta(x, x) = \eta_0$ and for all $a, b \in X$,

$$\eta(a, b) = \eta_0 \iff a \, \rho \, b.$$

The value of $\eta(a, b)$ is called then ρ-**normal form** of a and b.

Normalizing Function Examples

- Consider the typical floating-point representation $\sigma M \cdot 2^E$ for sign σ, mantissa M, and exponent E; and consider the relation '$\geq$'.

Normalizing Function Examples

- Consider the typical floating-point representation $\sigma M \cdot 2^E$ for sign σ, mantissa M, and exponent E; and consider the relation '$\geq$'. A possible normalizing function is $\eta(x, y) := \mathrm{sgn}(x - y)$ with $\eta_0 = 1$.

Normalizing Function Examples

- Consider the typical floating-point representation $\sigma M \cdot 2^E$ for sign σ, mantissa M, and exponent E; and consider the relation '$\geq$'. A possible normalizing function is $\eta(x, y) := \operatorname{sgn}(x - y)$ with $\eta_0 = 1$.
- Normalizing functions can be useful with equivalence relations when there is no clear canonizing function. Consider the problem of determining if $x = y$.

Normalizing Function Examples

- Consider the typical floating-point representation $\sigma M \cdot 2^E$ for sign σ, mantissa M, and exponent E; and consider the relation '$\geq$'. A possible normalizing function is $\eta(x, y) := \operatorname{sgn}(x - y)$ with $\eta_0 = 1$.
- Normalizing functions can be useful with equivalence relations when there is no clear canonizing function. Consider the problem of determining if $x = y$. If the domain supports it, $\eta(x, y) := x - y$ with $\eta_0 = 0$ is often helpful. This is called the *zero-equivalence problem*.

An Unfortunate Theorem

Computer algebra takes a pretty severe strike from Daniel Richardson in 1968. He tells us this.

An Unfortunate Theorem

Computer algebra takes a pretty severe strike from Daniel Richardson in 1968. He tells us this.

Theorem (Richardson)

Let R be the class of expressions generated by

1. *the rational numbers, π, and $\ln 2$,*

An Unfortunate Theorem

Computer algebra takes a pretty severe strike from Daniel Richardson in 1968. He tells us this.

Theorem (Richardson)

Let R be the class of expressions generated by

1. *the rational numbers, π, and $\ln 2$,*
2. *the variable x,*

An Unfortunate Theorem

Computer algebra takes a pretty severe strike from Daniel Richardson in 1968. He tells us this.

Theorem (Richardson)

Let R be the class of expressions generated by

1. *the rational numbers, π, and $\ln 2$,*
2. *the variable x,*
3. *the operations addition, multiplication, and function composition, and*

An Unfortunate Theorem

Computer algebra takes a pretty severe strike from Daniel Richardson in 1968. He tells us this.

Theorem (Richardson)

Let R be the class of expressions generated by

1. *the rational numbers, π, and $\ln 2$,*
2. *the variable x,*
3. *the operations addition, multiplication, and function composition, and*
4. *the sine, exponential, and absolute value functions.*

An Unfortunate Theorem

Computer algebra takes a pretty severe strike from Daniel Richardson in 1968. He tells us this.

Theorem (Richardson)

Let R be the class of expressions generated by

1. *the rational numbers, π, and $\ln 2$,*
2. *the variable x,*
3. *the operations addition, multiplication, and function composition, and*
4. *the sine, exponential, and absolute value functions.*

An Unfortunate Theorem

Computer algebra takes a pretty severe strike from Daniel Richardson in 1968. He tells us this.

Theorem (Richardson)

Let R be the class of expressions generated by

1. *the rational numbers, π, and $\ln 2$,*
2. *the variable x,*
3. *the operations addition, multiplication, and function composition, and*
4. *the sine, exponential, and absolute value functions.*

If $E \in R$, determining the truth of $E = 0$ is recursively undecidable.