Analytica V: Towards the Mordell-Weil Theorem

Edmund M. Clarke, Avi S. Gavlovski, Klaus Sutner, Wolfgang Windsteiger

Dept. of Computer Science, Carnegie Mellon University, USA
RISC Institute, JKU Linz, Austria

Overview

- History of Analytica
- System Design
  - Language
  - Inference Rules
  - Proof Search
- New Developments in Analytica *V*
  - Efficient Side-Condition Test
  - Handling "Definition by Cases"
  - Pruning Search Space by Using Symmetries
- Examples towards the Mordell-Weil Theorem

Analytica History

- early 1990's: Analytica
  - Developed by E. Clarke and Z. Zhao.
  - First theorem prover based on a computer algebra system (*Mathematica* 1.2).
  - Fully automated proofs of e.g. Ramanujan-identities, properties of continuous functions, etc.
  - Main "method": transform "proof problem" to a "symbolic computation problem" and use available symbolic computation algorithms then.
  - CADE-11 (1992), *Mathematica* Journal (1993), JAR (1998).
- 2003: ANALYTICA 2
- 2006: Analytica *V*

Analytica History

- early 1990's: Analytica
- 2003: ANALYTICA 2
  - Adapt to *Mathematica* 5.0.
  - Use notebook FrontEnd for code documentation.
  - Use *Mathematica*'s XML capabilities to connect to external knowledge repositories such as OmDoc.
  - Calculemus (2003).
- 2006: Analytica *V*

Analytica History

- early 1990's: Analytica
- 2003: ANALYTICA 2
- 2006: Analytica *V*
  - Clear separation between *Mathematica* and Analytica expressions (Theorema).
  - Proof search instead of simple rewriting based on pattern matching.
  - Specific techniques (e.g. exploiting symmetry properties) integrated into generic proof search.

System Design: Language

- First-order predicate logic

System Design: Language

- First-order predicate logic
- + Currying: For pragmatic reasons, e.g.

  ϕ, ψ homomorphisms. Product homomorphism $(\phi * \psi)[x]$

System Design: Language

- First-order predicate logic
- + Currying: For pragmatic reasons, e.g.

  ϕ, ψ homomorphisms. Product homomorphism $(\phi * \psi)[x]$

- + Overloading: Encode domain information in terms and atomic propositions, e.g.

$$(\phi{*}\psi)[x] := \phi[x]{*}\psi[x]$$

  written in Analytica as

$$\texttt{times}[\texttt{Hom}[\texttt{A\_}, \texttt{B\_}], \phi\_, \psi\_][\texttt{x\_}] :> \texttt{times}[\texttt{B}, \phi[x], \psi[x]]$$

System Design: Language

- First-order predicate logic
- + Currying: For pragmatic reasons, e.g.

  ϕ, ψ homomorphisms. Product homomorphism $(\phi * \psi)[x]$

- + Overloading: Encode domain information in terms and atomic propositions, e.g.

$$(\phi * \psi)[x] := \phi[x] * \psi[x]$$

written in Analytica as

$$\texttt{times}[\texttt{Hom}[\texttt{A\_}, \texttt{B\_}], \phi\_, \psi\_][\texttt{x\_}] :> \texttt{times}[\texttt{B}, \phi[x], \psi[x]]$$

System Design: Inference Rules

- Proving based on standard sequent calculus

System Design: Inference Rules

- Proving based on standard sequent calculus
- Proof state: list of sequents

System Design: Inference Rules

- Proving based on standard sequent calculus
- Proof state: list of sequents
- Inference rule: (T, R)

System Design: Inference Rules

- Proving based on standard sequent calculus
- Proof state: list of sequents
- Inference rule: (T, R) where
  - T ... the rule's applicability test

System Design: Inference Rules

- Proving based on standard sequent calculus
- Proof state: list of sequents
- Inference rule: (T, R) where
  - T ... the rule's applicability test
  - R ... the rule's transformation on the proof state

System Design: Inference Rules

- Proving based on standard sequent calculus
- Proof state: list of sequents
- Inference rule: (T, R) where
  - T ... the rule's applicability test
  - R ... the rule's transformation on the proof state
- "Event": formulae appearing new in the current sequent

System Design: Inference Rules

- Proving based on standard sequent calculus
- Proof state: list of sequents
- Inference rule: (T, R) where
  - T ... the rule's applicability test
  - R ... the rule's transformation on the proof state
- "Event": formulae appearing new in the current sequent
- Inference rules listen for events

System Design: Inference Rules

- Proving based on standard sequent calculus
- Proof state: list of sequents
- Inference rule: (T, R) where
  - T ... the rule's applicability test
  - R ... the rule's transformation on the proof state
- "Event": formulae appearing new in the current sequent
- Inference rules listen for events
- Event is automatically maintained by the proof search machinery

System Design: Proof Search

```
While[ ProofState != EMPTY,
   EnqueueRules[
      QueryRulebase[CurrentSequent[ ],CurrentEvent[ ]]];
   If[ EmptyRuleQueue[ ],
      DetectOutOfRules[ ],
      ApplyProofRule[DequeueRule[ ]]
   ]]
```

1. As long as the proof is not finished . . .

System Design: Proof Search

```
While[ ProofState != EMPTY,
   EnqueueRules[
      QueryRulebase[CurrentSequent[ ],CurrentEvent[ ]]];
   If[ EmptyRuleQueue[ ],
      DetectOutOfRules[ ],
      ApplyProofRule[DequeueRule[ ]]
   ]]
```

2. Apply the applicability test of all rules to the current sequent and the current event (i.e. the new formulae) . . .

System Design: Proof Search

```
While[ ProofState != EMPTY,
   EnqueueRules[
      QueryRulebase[CurrentSequent[ ],CurrentEvent[ ]]];
   If[ EmptyRuleQueue[ ],
      DetectOutOfRules[ ],
      ApplyProofRule[DequeueRule[ ]]
   ]]
```

3. ...and put all applicable rules into a queue.

System Design: Proof Search

```
While[ ProofState != EMPTY,
   EnqueueRules[
      QueryRulebase[CurrentSequent[ ],CurrentEvent[ ]]];
   If[ EmptyRuleQueue[ ],
      DetectOutOfRules[ ],
      ApplyProofRule[DequeueRule[ ]]
   ]]
```

4. If there are no more inference rules to apply . . .

System Design: Proof Search

```
While[ ProofState != EMPTY,
   EnqueueRules[
      QueryRulebase[CurrentSequent[ ],CurrentEvent[ ]]];
   If[ EmptyRuleQueue[ ],
      DetectOutOfRules[ ],
      ApplyProofRule[DequeueRule[ ]]
   ]]
```

4. If there are no more inference rules to apply . . .
   - then "adapt strategy to this fact"

System Design: Proof Search

```
While[ ProofState != EMPTY,
   EnqueueRules[
      QueryRulebase[CurrentSequent[ ],CurrentEvent[ ]]];
   If[ EmptyRuleQueue[ ],
      DetectOutOfRules[ ],
      ApplyProofRule[DequeueRule[ ]]
   ]]
```

4. If there are no more inference rules to apply . . .
   - then “adapt strategy to this fact”
   - else apply the first rule from the queue to the current sequent.

System Design: Proof Search

We have several "hooks", where the proof search can be fine-tuned:

- List of inference rules to be applied before a queued rule is applied.

System Design: Proof Search

We have several "hooks", where the proof search can be fine-tuned:

- List of inference rules to be applied before a queued rule is applied.
- List of inference rules to be applied after a queued rule has been applied.

System Design: Proof Search

We have several "hooks", where the proof search can be fine-tuned:

- List of inference rules to be applied before a queued rule is applied.
- List of inference rules to be applied after a queued rule has been applied.
- List of custom simplification rules to be applied after a queued rule is applied (in addition to standard simplifications that are always applied, e.g. boolean simplifications!).

System Design: Prover Configuration

Prover configuration = a collection of global values, which determine the behavior of the prover.

System Design: Prover Configuration

Prover configuration = a collection of global values, which determine the behavior of the prover.

Examples:

- "Hooks" described on the previous slide

System Design: Prover Configuration

Prover configuration = a collection of global values, which determine the behavior of the prover.

Examples:

- "Hooks" described on the previous slide
- Auxiliary definitions/theorems usable for look-up (see later)

System Design: Prover Configuration

Prover configuration = a collection of global values, which determine the behavior of the prover.

Examples:

- "Hooks" described on the previous slide
- Auxiliary definitions/theorems usable for look-up (see later)
- Auxiliary lemmata (for goal reduction)

System Design: Prover Configuration

Prover configuration = a collection of global values, which determine the behavior of the prover.

Examples:

- "Hooks" described on the previous slide
- Auxiliary definitions/theorems usable for look-up (see later)
- Auxiliary lemmata (for goal reduction)
- Inference rules available for queueing

System Design: Prover Configuration

Prover configuration = a collection of global values, which determine the behavior of the prover.

Examples:

- "Hooks" described on the previous slide
- Auxiliary definitions/theorems usable for look-up (see later)
- Auxiliary lemmata (for goal reduction)
- Inference rules available for queueing

> *Understand:* The theory, with respect to which a proof is produced in Analytica V, is "described" by the prover configuration!

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

Applications:

- Checking side-conditions during simplification (interplay reasoning – computing)
- Checking side-conditions during conversion to *Mathematica*

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

Applications:

- Checking side-conditions during simplification (interplay reasoning – computing)
- Checking side-conditions during conversion to *Mathematica*
- Simplification of proof state (eliminate "trivial" goals)

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

Typical example:

- Is G closed w.r.t. $*$?

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

Typical example:

- Is G closed w.r.t. $*$?
- We know that G is an abelian multiplicative group with identity 1 and inverse function $(.)^{-1}$.
- We know the hierachical build-up of algebraic structures.

Analytica V: Efficient Side-Condition Test

Problem: Devise a “small”, “efficient” mechanism to test whether a formula ϕ “follows easily” from Γ.

Typical example:

- Is G closed w.r.t. $*$?
- We know that G is an abelian multiplicative group with identity 1 and inverse function $(.)^{-1}$.
- We know the hierachical build-up of algebraic structures.

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

Typical example:

- Is G closed w.r.t. $*$?
- We know that G is an abelian multiplicative group with identity 1 and inverse function $(.)^{-1}$.
- We know the hierachical build-up of algebraic structures.

Possible Solutions:

- Use *Mathematica*'s standard conditional rewriting: too weak!

Analytica V: Efficient Side-Condition Test

Problem: Devise a “small”, “efficient” mechanism to test whether a formula ϕ “follows easily” from Γ.

Typical example:

- Is G closed w.r.t. $*$?
- We know that G is an abelian multiplicative group with identity 1 and inverse function $(.)^{-1}$.
- We know the hierachical build-up of algebraic structures.

Possible Solutions:

- Use *Mathematica*'s standard conditional rewriting: too weak!
- Expand definitions in Γ: blows up Γ! Search space!

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

What does it mean: ϕ "follows easily" from Γ?

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

What does it mean: ϕ "follows easily" from Γ?

- $\phi \in \Gamma$.

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

What does it mean: ϕ "follows easily" from Γ?

- $\phi \in \Gamma$.
- $\phi \in \mathcal{B}$, where $\mathcal{B}$ is built-in knowledge ($\rightsquigarrow$ prover configuration).

Analytica V: Efficient Side-Condition Test

Problem: Devise a "small", "efficient" mechanism to test whether a formula ϕ "follows easily" from Γ.

What does it mean: ϕ "follows easily" from Γ?

- $\phi \in \Gamma$.
- $\phi \in \mathcal{B}$, where $\mathcal{B}$ is built-in knowledge ($\rightsquigarrow$ prover configuration).
- ϕ can be derived from $\Gamma \cup \mathcal{B}$ using only (universally quantified) Modus Ponens ($\rightsquigarrow$ prover configuration).

Analytica V: Efficient Side-Condition Test

Solution in Analytica V:

1. Build implication graph a priori.

Analytica V: Efficient Side-Condition Test

Solution in Analytica V:

1. Build implication graph a priori.
2. Check whether ϕ follows from Γ by traversing the implication graph backwards starting from ϕ.

Analytica V: Efficient Side-Condition Test

Solution in Analytica V:

1. Build implication graph a priori.
2. Check whether ϕ follows from Γ by traversing the implication graph backwards starting from ϕ.
3. If this process finds a formula in Γ or in $\mathcal{B}$ then we are done, otherwise we fail.

Analytica V: Building the Implication Graph

Process formulae recursively in order to enable the backward traversal:

- $\forall x : p[x] \Leftrightarrow Q \quad \rightsquigarrow \quad \forall x : p[x] \Rightarrow Q$

Analytica V: Building the Implication Graph

Process formulae recursively in order to enable the backward traversal:

- $\forall x : p[x] \Leftrightarrow Q \quad \rightsquigarrow \quad \forall x : p[x] \Rightarrow Q$
- $\forall x : P \Rightarrow Q_1 \wedge \ldots \wedge Q_n \;\rightsquigarrow\; \forall x : P \Rightarrow Q_1, \ldots, \forall x : P \Rightarrow Q_n$

Analytica V: Building the Implication Graph

Process formulae recursively in order to enable the backward traversal:

- $\forall x : p[x] \Leftrightarrow Q \;\; \leadsto \;\; \forall x : p[x] \Rightarrow Q$
- $\forall x : P \Rightarrow Q_1 \wedge \ldots \wedge Q_n \;\; \leadsto \;\; \forall x : P \Rightarrow Q_1, \ldots, \forall x : P \Rightarrow Q_n$
- $\forall x : P \Rightarrow Q$, where Q is atomic *and* P and Q have the same free variables: allow traversal from $Q[x]$ to $P[x]$.

Analytica V: Building the Implication Graph

Process formulae recursively in order to enable the backward traversal:

- $\forall x : p[x] \Leftrightarrow Q \;\rightsquigarrow\; \forall x : p[x] \Rightarrow Q$
- $\forall x : P \Rightarrow Q_1 \wedge \ldots \wedge Q_n \;\rightsquigarrow\; \forall x : P \Rightarrow Q_1, \ldots, \forall x : P \Rightarrow Q_n$
- $\forall x : P \Rightarrow Q$, where Q is atomic *and* P and Q have the same free variables: allow traversal from $Q[x]$ to $P[x]$.
- $\forall x : P \Rightarrow Q$, where variables y are free in P but not in Q: observe $\forall x, y : P \Rightarrow Q$ is equivalent to $\forall x : (\exists y : P) \Rightarrow Q$. Allow traversal from $Q[x, y]$ to $P[x, \text{forsome}[y]]$, where forsome$[y]$ is a marker to ignore the arguments y.

Analytica V: Efficient Side-Condition Test – Example

$$\begin{array}{lll} \text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] & \rightarrow & \text{abelian}[A, *] \\ & \searrow & \text{mult.gr}[A, \text{i}, \text{reciprocal}] \end{array}$$

Analytica V: Efficient Side-Condition Test – Example

$$\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] \quad \longrightarrow \quad \text{abelian}[A, *]$$

$$\searrow \quad \text{mult.gr}[A, \text{i}, \text{reciprocal}]$$

$$\text{mult.gr}[A, \text{i}, \text{reciprocal}] \quad \longrightarrow \quad \text{gr}[A, *, \text{i}, \text{reciprocal}]$$

Analytica V: Efficient Side-Condition Test – Example

$$
\begin{array}{rcl}
\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] & \longrightarrow & \text{abelian}[A, *] \\
 & \searrow & \text{mult.gr}[A, \text{i}, \text{reciprocal}] \\
\text{mult.gr}[A, \text{i}, \text{reciprocal}] & \longrightarrow & \text{gr}[A, *, \text{i}, \text{reciprocal}] \\
\text{gr}[A, *, \text{id}, \text{reciprocal}] & \longrightarrow & \text{monoid}[A, *, \text{i}] \\
 & \searrow & \text{inv}[A, *, \text{i}, \text{reciprocal}]
\end{array}
$$

Analytica V: Efficient Side-Condition Test – Example

$$\begin{array}{rcl}
\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] & \rightarrow & \text{abelian}[A, *] \\
 & \searrow & \text{mult.gr}[A, \text{i}, \text{reciprocal}] \\[1em]
\text{mult.gr}[A, \text{i}, \text{reciprocal}] & \rightarrow & \text{gr}[A, *, \text{i}, \text{reciprocal}] \\[1em]
\text{gr}[A, *, \text{id}, \text{reciprocal}] & \rightarrow & \text{monoid}[A, *, \text{i}] \\
 & \searrow & \text{inv}[A, *, \text{i}, \text{reciprocal}] \\[1em]
\text{monoid}[A, *, \text{i}] & \rightarrow & \text{semigroup}[A, *] \\
 & \searrow & \text{id}[\text{i}, *, A]
\end{array}$$

Analytica V: Efficient Side-Condition Test – Example

$$\begin{array}{rcl}
\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] & \rightarrow & \text{abelian}[A, *] \\
& \searrow & \text{mult.gr}[A, \text{i}, \text{reciprocal}] \\
\text{mult.gr}[A, \text{i}, \text{reciprocal}] & \rightarrow & \text{gr}[A, *, \text{i}, \text{reciprocal}] \\
\text{gr}[A, *, \text{id}, \text{reciprocal}] & \rightarrow & \text{monoid}[A, *, \text{i}] \\
& \searrow & \text{inv}[A, *, \text{i}, \text{reciprocal}] \\
\text{monoid}[A, *, \text{i}] & \rightarrow & \text{semigroup}[A, *] \\
& \searrow & \text{id}[\text{i}, *, A] \\
\text{semigroup}[A, *] & \rightarrow & \text{closed}[A, *] \\
& \searrow & \text{associative}[*, A]
\end{array}$$

Analytica V: Efficient Side-Condition Test – Example

$$\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] \;\rightarrow\; \text{abelian}[A, *]$$
$$\searrow\; \text{mult.gr}[A, \text{i}, \text{reciprocal}]$$

$$\text{mult.gr}[A, \text{i}, \text{reciprocal}] \;\rightarrow\; \text{gr}[A, *, \text{i}, \text{reciprocal}]$$

$$\text{gr}[A, *, \text{id}, \text{reciprocal}] \;\rightarrow\; \text{monoid}[A, *, \text{i}]$$
$$\searrow\; \text{inv}[A, *, \text{i}, \text{reciprocal}]$$

$$\text{monoid}[A, *, \text{i}] \;\rightarrow\; \text{semigroup}[A, *]$$
$$\searrow\; \text{id}[\text{i}, *, A]$$

$$\text{semigroup}[A, *] \;\rightarrow\; \text{closed}[G, *]$$
$$\searrow\; \text{associative}[*, A]$$

Analytica V: Efficient Side-Condition Test – Example

ab.mult.gr$[A, \text{i}, \text{reciprocal}]$ $\rightarrow$ abelian$[A, *]$
$\searrow$ mult.gr$[A, \text{i}, \text{reciprocal}]$

mult.gr$[A, \text{i}, \text{reciprocal}]$ $\rightarrow$ gr$[A, *, \text{i}, \text{reciprocal}]$

gr$[A, *, \text{id}, \text{reciprocal}]$ $\rightarrow$ monoid$[A, *, \text{i}]$
$\searrow$ inv$[A, *, \text{i}, \text{reciprocal}]$

monoid$[A, *, \text{i}]$ $\rightarrow$ semigroup$[G, *]$
$\searrow$ id$[\text{i}, *, A]$

semigroup$[G, *]$ $\rightarrow$ closed$[G, *]$
$\searrow$ associative$[*, A]$

Analytica V: Efficient Side-Condition Test – Example

$$
\begin{array}{rcl}
\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] & \longrightarrow & \text{abelian}[A, *] \\
& \searrow & \text{mult.gr}[A, \text{i}, \text{reciprocal}] \\
\text{mult.gr}[A, \text{i}, \text{reciprocal}] & \longrightarrow & \text{gr}[A, *, \text{i}, \text{reciprocal}] \\
\text{gr}[A, *, \text{id}, \text{reciprocal}] & \longrightarrow & \text{monoid}[G, *, \text{forsome}[\text{i}]] \\
& \searrow & \text{inv}[A, *, \text{i}, \text{reciprocal}] \\
\text{monoid}[G, *, \text{forsome}[\text{i}]] & \longrightarrow & \text{semigroup}[G, *] \\
& \searrow & \text{id}[\text{i}, *, A] \\
\text{semigroup}[G, *] & \longrightarrow & \text{closed}[G, *] \\
& \searrow & \text{associative}[*, A]
\end{array}
$$

Analytica V: Efficient Side-Condition Test – Example

$$\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] \;\longrightarrow\; \text{abelian}[A, *]$$
$$\searrow\; \text{mult.gr}[A, \text{i}, \text{reciprocal}]$$

$$\text{mult.gr}[A, \text{i}, \text{reciprocal}] \;\longrightarrow\; \text{gr}[G, *, \text{forsome}[\text{i}], \text{forsome}[\text{r}]]$$

$$\text{gr}[G, *, \text{forsome}[\text{i}], \text{forsome}[\text{r}]] \;\longrightarrow\; \text{monoid}[G, *, \text{forsome}[\text{i}]]$$
$$\searrow\; \text{inv}[A, *, \text{i}, \text{reciprocal}]$$

$$\text{monoid}[G, *, \text{forsome}[\text{i}]] \;\longrightarrow\; \text{semigroup}[G, *]$$
$$\searrow\; \text{id}[\text{i}, *, A]$$

$$\text{semigroup}[G, *] \;\longrightarrow\; \text{closed}[G, *]$$
$$\searrow\; \text{associative}[*, A]$$

Analytica V: Efficient Side-Condition Test – Example

$$
\begin{array}{rcl}
\text{ab.mult.gr}[A, \text{i}, \text{reciprocal}] & \rightarrow & \text{abelian}[A, *] \\
 & \searrow & \text{mult.gr}[G, \text{forsome}[\text{i}], \text{forsome}[\text{r}]] \\
\text{mult.gr}[G, \text{forsome}[\text{i}], \text{forsome}[\text{r}]] & \rightarrow & \text{gr}[G, *, \text{forsome}[\text{i}], \text{forsome}[\text{r}]] \\
\text{gr}[G, *, \text{forsome}[\text{i}], \text{forsome}[\text{r}]] & \rightarrow & \text{monoid}[G, *, \text{forsome}[\text{i}]] \\
 & \searrow & \text{inv}[A, *, \text{i}, \text{reciprocal}] \\
\text{monoid}[G, *, \text{forsome}[\text{i}]] & \rightarrow & \text{semigroup}[G, *] \\
 & \searrow & \text{id}[\text{i}, *, A] \\
\text{semigroup}[G, *] & \rightarrow & \text{closed}[G, *] \\
 & \searrow & \text{associative}[*, A]
\end{array}
$$

Analytica V: Efficient Side-Condition Test – Example

ab.mult.gr[G, forsome[i], forsome[r]] $\longrightarrow$ abelian[$A, *$]
$\searrow$ mult.gr[G, forsome[i], forsome[

mult.gr[G, forsome[i], forsome[r]] $\longrightarrow$ gr[$G, *$, forsome[i], forsome[r]]

gr[$G, *$, forsome[i], forsome[r]] $\longrightarrow$ monoid[$G, *$, forsome[i]]
$\searrow$ inv[$A, *$, i, reciprocal]

monoid[$G, *$, forsome[i]] $\longrightarrow$ semigroup[$G, *$]
$\searrow$ id[i, $*, A$]

semigroup[$G, *$] $\longrightarrow$ closed[$G, *$]
$\searrow$ associative[$*, A$]

Analytica V: Handling "Definition by Cases"

Quite straight-forward: Split proof into cases when expanding a definition by cases! (We provide a language construct to express cases.)

Analytica V: Pruning Search Space by Using Symmetries

We analyze conjunctions in the goal and disjunctions in the assumptions to detect symmetries.

Analytica V: Pruning Search Space by Using Symmetries

We analyze conjunctions in the goal and disjunctions in the assumptions to detect symmetries. E.g.

$$\texttt{seq}[\{a > 0, b > 0, x > a + b\}, \{x > a \wedge x > b\}]$$

Since the sequent is symmetric about a and b, we can reduce the conjunction in the goal:

$$\texttt{seq}[\{a > 0, b > 0, x > a + b\}, \{x > a\}]$$

Analytica V: Pruning Search Space by Using Symmetries

We analyze conjunctions in the goal and disjunctions in the assumptions to detect symmetries. E.g.

$$\mathsf{seq}[\{a > 0, b > 0, x > a + b\}, \{x > a \wedge x > b\}]$$

Since the sequent is symmetric about a and b, we can reduce the conjunction in the goal:

$$\mathsf{seq}[\{a > 0, b > 0, x > a + b\}, \{x > a\}]$$

More formally, in a sequent $seq[l, r]$, we reduce a conjunction $A \wedge B$ in the goal (or disjunction $A \vee B$ in the assumptions), if there exists a permutation f of the parameters in the sequent such that:

$$l[f] =_\alpha l, r[f] =_\alpha r, A[f] =_\alpha B$$

where $=_\alpha$ denotes equality up to α-conversion and some simple normalizations relating to commutative operators and predicates that are symmetric about their arguments.

Group Characters & Weak Mordell-Weil Theorem

Theorem (Dirichlet)

The arithmetic progression $a_k = \alpha + k \cdot \beta$ (with α, β coprime) contains infinitely many primes.

Proof based on group characters ...

Theorem

$GC[G]$ is an abelian multiplicative group with the identity character $\mathbf{1}(x) = 1$ and the inverse character reciprocal$(a)(x) = a(x)^{-1}$.

Theorem (Weak Mordell-Weil Theorem)

For any elliptic curve $E(\mathbb{Q})$, the quotient group $E(\mathbb{Q})/2E(\mathbb{Q})$ is finite.

Proof of Weak Mordell-Weil Theorem

We define a mapping
$\delta : E(\mathbb{Q}) \to \mathbb{Q}^*/(\mathbb{Q}^*)^2 \times \mathbb{Q}^*/(\mathbb{Q}^*)^2 \times \mathbb{Q}^*/(\mathbb{Q}^*)^2$ by:

$$\delta(P) = \begin{cases} (1,1,1) & \text{if } P = O, \\ ((a-b)(a-c), a-b, a-c) & \text{if } P = (a,0), \\ (b-a, (b-a)(b-c), b-c) & \text{if } P = (b,0), \\ (c-a, c-b, (c-a)(c-b)) & \text{if } P = (c,0), \\ (x-a, x-b, x-c) & \text{otherwise, } P = (x,y). \end{cases}$$

The proof of the theorem comprises three parts:

1. δ is a homomorphism.
2. The kernel of δ is $2E(\mathbb{Q})$.
3. The image of delta is contained in the finite subgroup generated by the prime factors of $a-b$, $b-c$, $c-a$, -1.