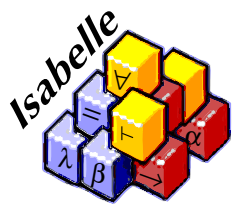
Formalizing the Logic-Automaton Connection

Stefan Berghofer Markus Reiter

Institut für Informatik
Technische Universität München



TPHOLs, 20.8.2009

Quiz

Which of these formulae are true (for natural numbers)?

$$\forall x{\geq}7.\ \exists y\ z.\ 3 * y + 5 * z = x$$

$$\forall x{\geq}8.\ \exists y\ z.\ 3 * y + 5 * z = x$$

$$\forall x{\geq}8.\ \exists y\ z.\ 4 * y + 5 * z = x$$

Quiz

Which of these formulae are true (for natural numbers)?

$$\forall x{\geq}7.\ \exists y\ z.\ 3 * y + 5 * z = x$$

$$\forall x{\geq}8.\ \exists y\ z.\ 3 * y + 5 * z = x$$

$$\forall x{\geq}8.\ \exists y\ z.\ 4 * y + 5 * z = x$$

Quiz

Which of these formulae are true (for natural numbers)?

$$\forall x{\geq}7.\ \exists y\ z.\ 3 * y + 5 * z = x$$

$$\forall x{\geq}8.\ \exists y\ z.\ 3 * y + 5 * z = x$$

$$\forall x{\geq}8.\ \exists y\ z.\ 4 * y + 5 * z = x$$

Stamp problem

Any postage of 8 cents or more can be made up using stamps of the denominations 3 cents and 5 cents.

Outline

Outline

Quantifier elimination method

Decision Procedures for Presburger Arithmetic

Quantifier elimination method

Algebraic e.g. Cooper's algorithm

Decision Procedures for Presburger Arithmetic

Quantifier elimination method

Algebraic e.g. Cooper's algorithm

Semantic e.g. using automata on bitstrings (this talk)
(also works for WS1S, see MONA)

Decision Procedures for Presburger Arithmetic

Quantifier elimination method

Algebraic e.g. Cooper's algorithm

Semantic e.g. using automata on bitstrings (this talk)
(also works for WS1S, see MONA)

Implementation in a theorem prover

Decision Procedures for Presburger Arithmetic

Quantifier elimination method

Algebraic e.g. Cooper's algorithm

Semantic e.g. using automata on bitstrings (this talk)
(also works for WS1S, see MONA)

Implementation in a theorem prover

Oracle-based Use an external tool such as MONA, and simply trust the answer of the tool.

Decision Procedures for Presburger Arithmetic

Quantifier elimination method

Algebraic e.g. Cooper's algorithm

Semantic e.g. using automata on bitstrings (this talk)
(also works for WS1S, see MONA)

Implementation in a theorem prover

Oracle-based Use an external tool such as MONA, and simply trust the answer of the tool.

Certificate-based Use an external tool, but try to reconstruct a proof inside the theorem prover from a certificate (or trace) returned by the tool, rather than just trusting it.

Decision Procedures for Presburger Arithmetic

Quantifier elimination method

Algebraic e.g. Cooper's algorithm

Semantic e.g. using automata on bitstrings (this talk)
(also works for WS1S, see MONA)

Implementation in a theorem prover

Oracle-based Use an external tool such as MONA, and simply trust the answer of the tool.

Certificate-based Use an external tool, but try to reconstruct a proof inside the theorem prover from a certificate (or trace) returned by the tool, rather than just trusting it.

Derived rule Write a decision procedure in the implementation language of the theorem prover (e.g. ML or OCaml) that constructs a proof by applying primitive inference rules.

Decision Procedures for Presburger Arithmetic

Quantifier elimination method

Algebraic e.g. Cooper's algorithm

Semantic e.g. using automata on bitstrings (this talk)
(also works for WS1S, see MONA)

Implementation in a theorem prover

Oracle-based Use an external tool such as MONA, and simply trust the answer of the tool.

Certificate-based Use an external tool, but try to reconstruct a proof inside the theorem prover from a certificate (or trace) returned by the tool, rather than just trusting it.

Derived rule Write a decision procedure in the implementation language of the theorem prover (e.g. ML or OCaml) that constructs a proof by applying primitive inference rules.

Reflection Write and verify the decision procedure as a recursive function in HOL itself (this talk).

Related Work

[Boutin, TACS 1997] Decision procedure for abelian rings in Coq (reflection)

Related Work

[Boutin, TACS 1997] Decision procedure for abelian rings in Coq (reflection)

[Norrish, TPHOLs 2003] Cooper's algorithm in HOL (derived rule)

Related Work

[Boutin, TACS 1997] Decision procedure for abelian rings in Coq (reflection)

[Norrish, TPHOLs 2003] Cooper's algorithm in HOL (derived rule)

[Chaieb and Nipkow, JAR 2008] Cooper's / Ferrante and Rackoff's algorithm in Isabelle (reflection / derived rule)

Related Work

[Boutin, TACS 1997] Decision procedure for abelian rings in Coq (reflection)

[Norrish, TPHOLs 2003] Cooper's algorithm in HOL (derived rule)

[Chaieb and Nipkow, JAR 2008] Cooper's / Ferrante and Rackoff's algorithm in Isabelle (reflection / derived rule)

[Nipkow, IJCAR 2008] Quantifier elimination for discrete linear orders, linear real, and Presburger arithmetic (reflection)

Related Work

[Boutin, TACS 1997] Decision procedure for abelian rings in Coq (reflection)

[Norrish, TPHOLs 2003] Cooper's algorithm in HOL (derived rule)

[Chaieb and Nipkow, JAR 2008] Cooper's / Ferrante and Rackoff's algorithm in Isabelle (reflection / derived rule)

[Nipkow, IJCAR 2008] Quantifier elimination for discrete linear orders, linear real, and Presburger arithmetic (reflection)

[Basin and Friedrich, FroCoS 2000] Combining WS1S and HOL (oracle)

Does Reflection work?

Hooking an 'oracle' to a theorem prover is risky business. The oracle could be buggy [...]. The only way to avoid a buggy oracle is to reconstruct a proof in the theorem prover based on output from the oracle, or perhaps verify the oracle itself. For a semantics based decision procedure, proof reconstruction is not a realistic option: one would have to formalize the entire automata-theoretic machinery within HOL [...].

[Basin and Friedrich, FroCoS 2000]

Does Reflection work?

Hooking an 'oracle' to a theorem prover is risky business. The oracle could be buggy [...]. The only way to avoid a buggy oracle is to reconstruct a proof in the theorem prover based on output from the oracle, or perhaps verify the oracle itself. For a semantics based decision procedure, proof reconstruction is not a realistic option: one would have to formalize the entire automata-theoretic machinery within HOL [...].

[Basin and Friedrich, FroCoS 2000]

Our Claim

Verifying automata-based decision procedures in HOL is not as unrealistic as it may seem!

Outline

Presburger Arithmetic

Syntax (using de Bruijn indices)

datatype $pf = Eq\ (int\ list)\ int \mid Le\ (int\ list)\ int \mid And\ pf\ pf$
$\mid Or\ pf\ pf \mid Imp\ pf\ pf \mid Forall\ pf \mid Exist\ pf \mid Neg\ pf$

Presburger Arithmetic

Syntax (using de Bruijn indices)

datatype $pf = Eq\ (int\ list)\ int \mid Le\ (int\ list)\ int \mid And\ pf\ pf$
$\mid Or\ pf\ pf \mid Imp\ pf\ pf \mid Forall\ pf \mid Exist\ pf \mid Neg\ pf$

Example: Stamp problem

$\forall\, x{\geq}8.\ \exists\, y\ z.\ 3 * y + 5 * z = x$

Presburger Arithmetic

Syntax (using de Bruijn indices)

datatype $pf = Eq\ (int\ list)\ int \mid Le\ (int\ list)\ int \mid And\ pf\ pf$
$\mid Or\ pf\ pf \mid Imp\ pf\ pf \mid Forall\ pf \mid Exist\ pf \mid Neg\ pf$

Example: Stamp problem

$\forall\, x{\geq}8.\ \exists\, y\ z.\ 3 * y + 5 * z = x$

Encoding

$Forall\ (Imp\ (Le\ [-1]\ -8)\ (Exist\ (Exist\ (Eq\ [5,\ 3,\ -1]\ 0))))$

Evaluation

Diophantine (In)Equations

$$\textit{eval-dioph} :: \textit{int list} \Rightarrow \textit{nat list} \Rightarrow \textit{int}$$
$$\textit{eval-dioph}\ (k \cdot ks)\ (x \cdot xs) = k * \textit{int}\ x + \textit{eval-dioph}\ ks\ xs$$
$$\textit{eval-dioph}\ []\ xs = 0$$
$$\textit{eval-dioph}\ ks\ [] = 0$$

Evaluation

Diophantine (In)Equations

$$\textit{eval-dioph} :: \textit{int list} \Rightarrow \textit{nat list} \Rightarrow \textit{int}$$
$$\textit{eval-dioph}\ (k \cdot ks)\ (x \cdot xs) = k * \textit{int}\ x + \textit{eval-dioph}\ ks\ xs$$
$$\textit{eval-dioph}\ []\ xs = 0$$
$$\textit{eval-dioph}\ ks\ [] = 0$$

Formulae

$$\textit{eval-pf} :: \textit{pf} \Rightarrow \textit{nat list} \Rightarrow \textit{bool}$$
$$\textit{eval-pf}\ (\textit{Eq}\ ks\ l)\ xs = (\textit{eval-dioph}\ ks\ xs = l)$$
$$\textit{eval-pf}\ (\textit{Le}\ ks\ l)\ xs = (\textit{eval-dioph}\ ks\ xs \leq l)$$
$$\textit{eval-pf}\ (\textit{Neg}\ p)\ xs = (\neg\ \textit{eval-pf}\ p\ xs)$$
$$\textit{eval-pf}\ (\textit{And}\ p\ q)\ xs = (\textit{eval-pf}\ p\ xs \wedge \textit{eval-pf}\ q\ xs)$$
$$\textit{eval-pf}\ (\textit{Or}\ p\ q)\ xs = (\textit{eval-pf}\ p\ xs \vee \textit{eval-pf}\ q\ xs)$$
$$\textit{eval-pf}\ (\textit{Forall}\ p)\ xs = (\forall x.\ \textit{eval-pf}\ p\ (x \cdot xs))$$
$$\textit{eval-pf}\ (\textit{Exist}\ p)\ xs = (\exists x.\ \textit{eval-pf}\ p\ (x \cdot xs))$$

Automata on Bit Vectors

Input symbols of an automaton corresponding to a formula with n free variables $x_0, \ldots, x_{n-1}$ are bit lists of length n:

$$\begin{matrix} x_0 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \end{matrix} \left[\left[\begin{matrix} b_{0,0} \\ \vdots \\ b_{i,0} \\ \vdots \\ b_{n-1,0} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{n-1,j} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,m-1} \\ \vdots \\ b_{i,m-1} \\ \vdots \\ b_{n-1,m-1} \end{matrix}\right]\right]$$

Automata on Bit Vectors

Input symbols of an automaton corresponding to a formula with n free variables $x_0, \ldots, x_{n-1}$ are bit lists of length n:

$$\begin{matrix} x_0 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \end{matrix} \left[\left[\begin{matrix} b_{0,0} \\ \vdots \\ b_{i,0} \\ \vdots \\ b_{n-1,0} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{n-1,j} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,m-1} \\ \vdots \\ b_{i,m-1} \\ \vdots \\ b_{n-1,m-1} \end{matrix}\right]\right]$$

- i-th row: value of i-th variable (natural number)

Automata on Bit Vectors

Input symbols of an automaton corresponding to a formula with n free variables $x_0, \ldots, x_{n-1}$ are bit lists of length n:

$$\begin{matrix} x_0 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \end{matrix} \left[\left[\begin{matrix} b_{0,0} \\ \vdots \\ b_{i,0} \\ \vdots \\ b_{n-1,0} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{n-1,j} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,m-1} \\ \vdots \\ b_{i,m-1} \\ \vdots \\ b_{n-1,m-1} \end{matrix}\right]\right]$$

- i-th row: value of i-th variable (natural number)
- j-th column: j-th bit of variables

Automata on Bit Vectors

Input symbols of an automaton corresponding to a formula with n free variables $x_0, \ldots, x_{n-1}$ are bit lists of length n:

$$\begin{matrix} x_0 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \end{matrix} \left[\left[\begin{matrix} b_{0,0} \\ \vdots \\ b_{i,0} \\ \vdots \\ b_{n-1,0} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{n-1,j} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,m-1} \\ \vdots \\ b_{i,m-1} \\ \vdots \\ b_{n-1,m-1} \end{matrix}\right]\right]$$

- i-th row: value of i-th variable (natural number)
- j-th column: j-th bit of variables
- column 0: least significant bit

Automata on Bit Vectors

Input symbols of an automaton corresponding to a formula with n free variables $x_0, \ldots, x_{n-1}$ are bit lists of length n:

$$\begin{matrix} x_0 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \end{matrix} \left[\left[\begin{matrix} b_{0,0} \\ \vdots \\ b_{i,0} \\ \vdots \\ b_{n-1,0} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{n-1,j} \end{matrix}\right] \cdots \left[\begin{matrix} b_{0,m-1} \\ \vdots \\ b_{i,m-1} \\ \vdots \\ b_{n-1,m-1} \end{matrix}\right]\right]$$

- i-th row: value of i-th variable (natural number)
- j-th column: j-th bit of variables
- column 0: least significant bit

[Boudet and Comon, CAAP 1996]

Values of Variables

List of variables (encoded as list of column vectors)

$$
\begin{aligned}
&\mathit{nats\text{-}of\text{-}boolss} :: \mathit{nat} \Rightarrow \mathit{bool\ list\ list} \Rightarrow \mathit{nat\ list} \\
&\mathit{nats\text{-}of\text{-}boolss}\ n\ [] = \mathit{replicate}\ n\ 0 \\
&\mathit{nats\text{-}of\text{-}boolss}\ n\ (bs \cdot bss) = \\
&\quad \mathit{map}\ (\lambda(b, x).\ \mathit{nat\text{-}of\text{-}bool}\ b + 2 * x) \\
&\quad\ (\mathit{zip}\ bs\ (\mathit{nats\text{-}of\text{-}boolss}\ n\ bss))
\end{aligned}
$$

Values of Variables

List of variables (encoded as list of column vectors)

$$
\begin{aligned}
&\mathit{nats\text{-}of\text{-}boolss} :: \mathit{nat} \Rightarrow \mathit{bool\ list\ list} \Rightarrow \mathit{nat\ list} \\
&\mathit{nats\text{-}of\text{-}boolss}\ n\ [] = \mathit{replicate}\ n\ 0 \\
&\mathit{nats\text{-}of\text{-}boolss}\ n\ (bs \cdot bss) = \\
&\quad \mathit{map}\ (\lambda(b, x).\ \mathit{nat\text{-}of\text{-}bool}\ b + 2 * x) \\
&\quad (\mathit{zip}\ bs\ (\mathit{nats\text{-}of\text{-}boolss}\ n\ bss))
\end{aligned}
$$

Single variable (encoded as row vector)

$$
\begin{aligned}
&\mathit{nat\text{-}of\text{-}bools} :: \mathit{bool\ list} \Rightarrow \mathit{nat} \\
&\mathit{nat\text{-}of\text{-}bools}\ [] = 0 \\
&\mathit{nat\text{-}of\text{-}bools}\ (b \cdot bs) = \mathit{nat\text{-}of\text{-}bool}\ b + 2 * \mathit{nat\text{-}of\text{-}bools}\ bs
\end{aligned}
$$

Deterministic Automata

Represented by type

$$dfa = \underbrace{nat\ bdd\ list}_{\text{transition table}} \times \underbrace{bool\ list}_{\text{accepting states}}$$

Deterministic Automata

Represented by type

$$dfa = \underbrace{nat\ bdd\ list}_{\text{transition table}} \times \underbrace{bool\ list}_{\text{accepting states}}$$

Note: start state = 0

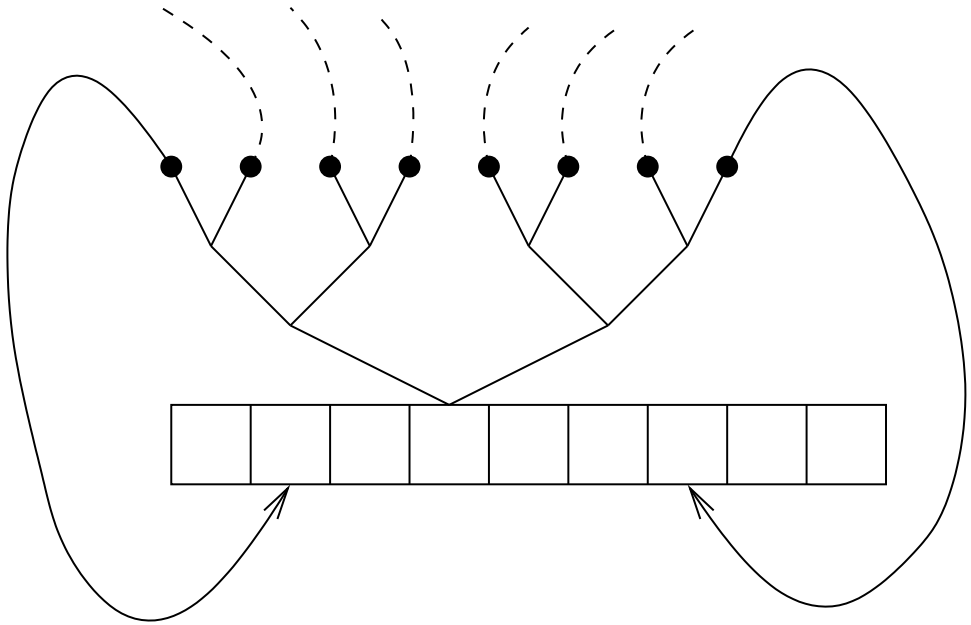
Deterministic Automata

Represented by type

$$dfa = \underbrace{nat\ bdd\ list}_{\text{transition table}} \times \underbrace{bool\ list}_{\text{accepting states}}$$

Note: start state = 0

Transition table



Nondeterministic Automata

Represented by type

$$nfa = \underbrace{bool\ list\ bdd\ list}_{\text{transition table}} \times \underbrace{bool\ list}_{\text{accepting states}}$$

Nondeterministic Automata

Represented by type

$$nfa = \underbrace{bool\ list\ bdd\ list}_{\text{transition table}} \times \underbrace{bool\ list}_{\text{accepting states}}$$

Note: (finite) sets of states represented as bitstrings

Outline

Automata Construction

- Complement: for negation

Automata Construction

- Complement: for negation
- Product automaton: for binary operators, i.e. $\vee$, $\wedge$, and $\longrightarrow$

Automata Construction

- Complement: for negation
- Product automaton: for binary operators, i.e. $\vee$, $\wedge$, and $\longrightarrow$
- Projection: for existential quantifiers

Automata Construction

- Complement: for negation
- Product automaton: for binary operators, i.e. $\vee$, $\wedge$, and $\longrightarrow$
- Projection: for existential quantifiers
- Atomic formulae: Diophantine (in)equations

Automata Construction

- Complement: for negation
- Product automaton: for binary operators, i.e. $\vee$, $\wedge$, and $\longrightarrow$
- Projection: for existential quantifiers
- Atomic formulae: Diophantine (in)equations

Naive implementation

- Product automaton: $m \cdot n$ states

Automata Construction

- Complement: for negation
- Product automaton: for binary operators, i.e. $\vee$, $\wedge$, and $\longrightarrow$
- Projection: for existential quantifiers
- Atomic formulae: Diophantine (in)equations

Naive implementation

- Product automaton: $m \cdot n$ states
- DFA from NFA: 2^n states

Automata Construction

- Complement: for negation
- Product automaton: for binary operators, i.e. $\vee$, $\wedge$, and $\longrightarrow$
- Projection: for existential quantifiers
- Atomic formulae: Diophantine (in)equations

Naive implementation

- Product automaton: $m \cdot n$ states
- DFA from NFA: 2^n states

Better: only generate reachable states (using DFS)

Quantifiers

Existential quantifiers

- Convert DFA to NFA (trivial)

Quantifiers

Existential quantifiers

- Convert DFA to NFA (trivial)
- Project away variable(s) to be quantified (yields NFA)

Quantifiers

Existential quantifiers

- Convert DFA to NFA (trivial)
- Project away variable(s) to be quantified (yields NFA)
- Convert NFA to DFA (subset construction)

Quantifiers

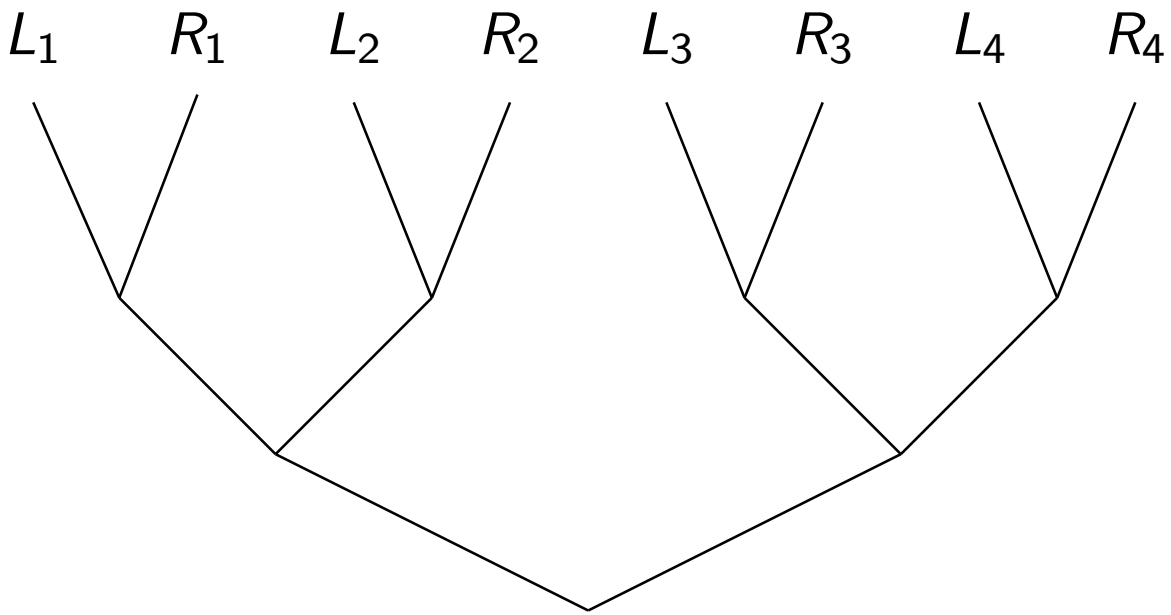
Existential quantifiers

- Convert DFA to NFA (trivial)
- Project away variable(s) to be quantified (yields NFA)
- Convert NFA to DFA (subset construction)

Universal quantifiers

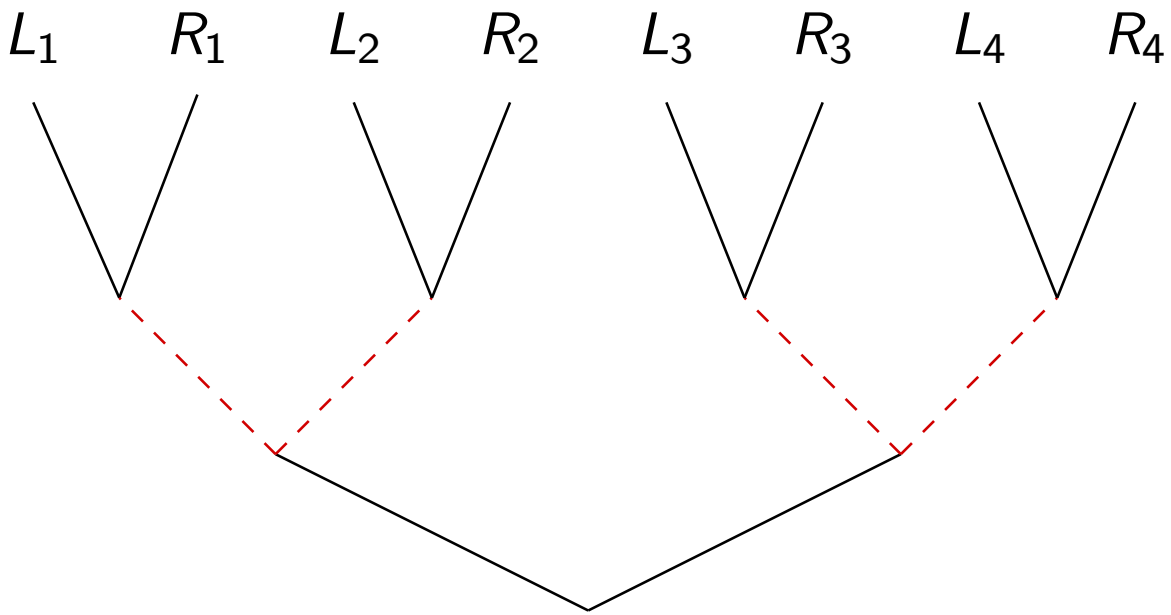
Note: $(\forall\, x.\ P\ x) = (\neg\ (\exists\, x.\ \neg\ P\ x))$

Projection



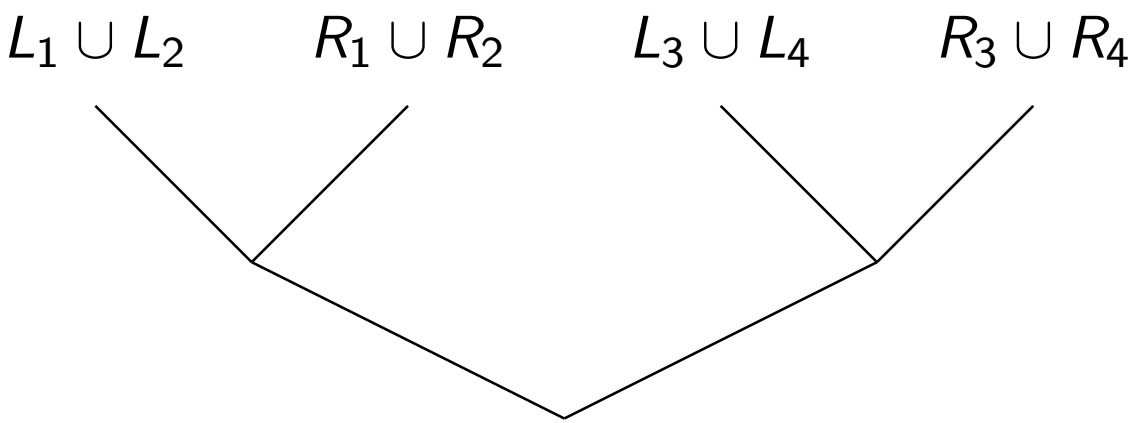
quantify-bdd :: *nat* ⇒ *bool list bdd* ⇒ *bool list bdd*
quantify-bdd i (*Leaf q*) = *Leaf q*
quantify-bdd 0 (*Branch l r*) = *bdd-binop bv-or l r*
quantify-bdd (*Suc i*) (*Branch l r*) =
  Branch (*quantify-bdd i l*) (*quantify-bdd i r*)

Projection



quantify-bdd :: *nat* ⇒ *bool list bdd* ⇒ *bool list bdd*
quantify-bdd i (*Leaf q*) = *Leaf q*
quantify-bdd 0 (*Branch l r*) = *bdd-binop bv-or l r*
quantify-bdd (*Suc i*) (*Branch l r*) =
  Branch (*quantify-bdd i l*) (*quantify-bdd i r*)

Projection



quantify-bdd :: *nat* $\Rightarrow$ *bool list bdd* $\Rightarrow$ *bool list bdd*
quantify-bdd i (*Leaf q*) = *Leaf q*
quantify-bdd 0 (*Branch l r*) = *bdd-binop bv-or l r*
quantify-bdd (*Suc i*) (*Branch l r*) =
  Branch (*quantify-bdd i l*) (*quantify-bdd i r*)

Diophantine Equations

Method by Boudet and Comon

$$
\begin{aligned}
&(\mathit{eval\text{-}dioph}\ \mathit{ks}\ \mathit{xs} = \mathit{l}) = \\
&(\mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ \mathit{xs})\ \mathit{mod}\ 2 = \mathit{l}\ \mathit{mod}\ 2 \wedge \\
&\ \mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{div}\ 2)\ \mathit{xs}) = \\
&\ (\mathit{l} - \mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ \mathit{xs}))\ \mathit{div}\ 2)
\end{aligned}
$$

Diophantine Equations

Method by Boudet and Comon

$$(\mathit{eval\text{-}dioph}\ ks\ xs = l) =$$
$$(\mathit{eval\text{-}dioph}\ ks\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ xs)\ \mathit{mod}\ 2 = l\ \mathit{mod}\ 2 \wedge$$
$$\mathit{eval\text{-}dioph}\ ks\ (\mathit{map}\ (\lambda x.\ x\ \mathit{div}\ 2)\ xs) =$$
$$(l - \mathit{eval\text{-}dioph}\ ks\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ xs))\ \mathit{div}\ 2)$$

xs is a solution iff...

- ...it is a solution modulo *2*, and...

Diophantine Equations

Method by Boudet and Comon

$$(\mathit{eval\text{-}dioph}\ \mathit{ks}\ \mathit{xs} = \mathit{l}) =$$
$$(\mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ \mathit{xs})\ \mathit{mod}\ 2 = \mathit{l}\ \mathit{mod}\ 2 \wedge$$
$$\mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{div}\ 2)\ \mathit{xs}) =$$
$$(\mathit{l} - \mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ \mathit{xs}))\ \mathit{div}\ 2)$$

xs is a solution iff...

- ...it is a solution modulo *2*, and...
- ...quotient of *xs* and *2* is a solution of another equation with same coefficients, but different right-hand side.

Diophantine Equations

Method by Boudet and Comon

$(\mathit{eval\text{-}dioph}\ \mathit{ks}\ \mathit{xs} = \mathit{l}) =$
$(\mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ \mathit{xs})\ \mathit{mod}\ 2 = \mathit{l}\ \mathit{mod}\ 2 \wedge$
$\mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{div}\ 2)\ \mathit{xs}) =$
$(\mathit{l} - \mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ \mathit{xs}))\ \mathit{div}\ 2)$

xs is a solution iff...

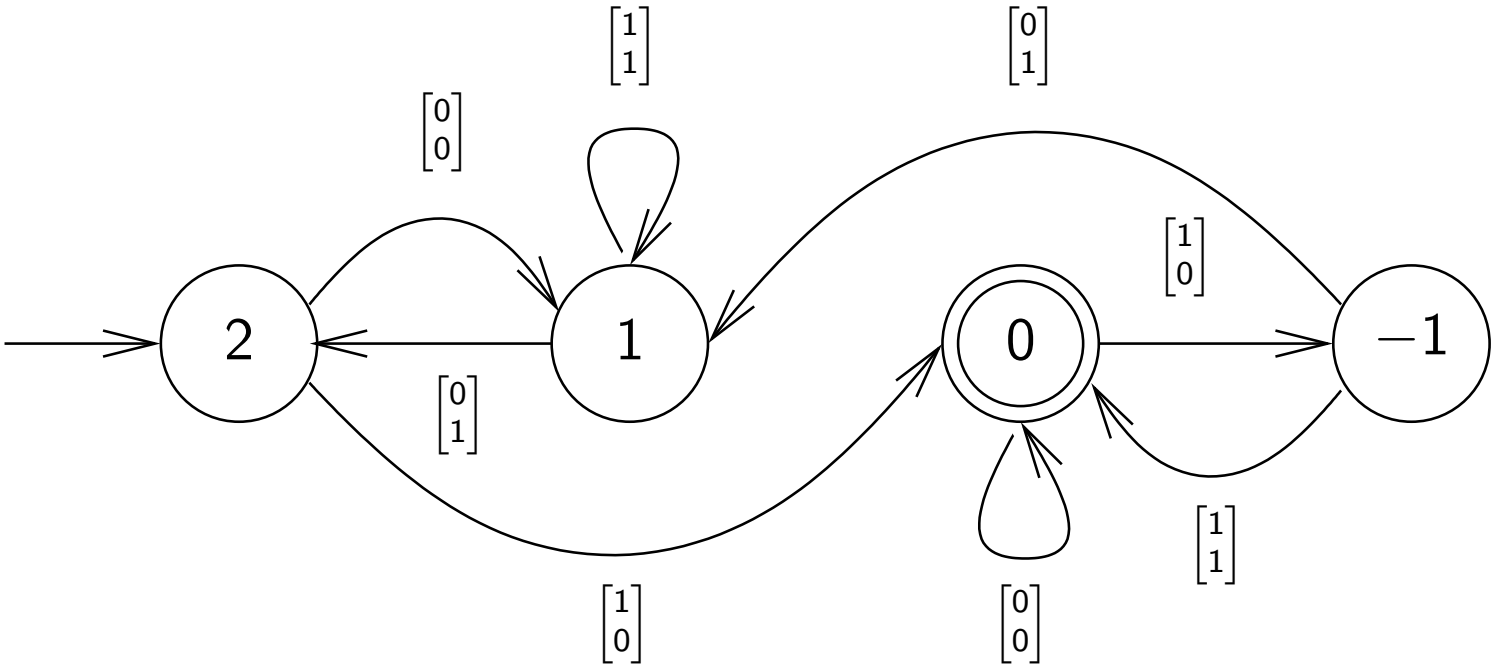
- ...it is a solution modulo *2*, and...
- ...quotient of *xs* and *2* is a solution of another equation with same coefficients, but different right-hand side.

Reachable right-hand sides are bounded

$\mathit{If}\ |m| \leq \mathit{max}\ |l|\ (\sum k{\leftarrow}\mathit{ks}.\ |k|)\ \ \mathit{then}$
$|(m - \mathit{eval\text{-}dioph}\ \mathit{ks}\ (\mathit{map}\ (\lambda x.\ x\ \mathit{mod}\ 2)\ \mathit{xs}))\ \mathit{div}\ 2|$
$\leq \mathit{max}\ |l|\ (\sum k{\leftarrow}\mathit{ks}.\ |k|).$

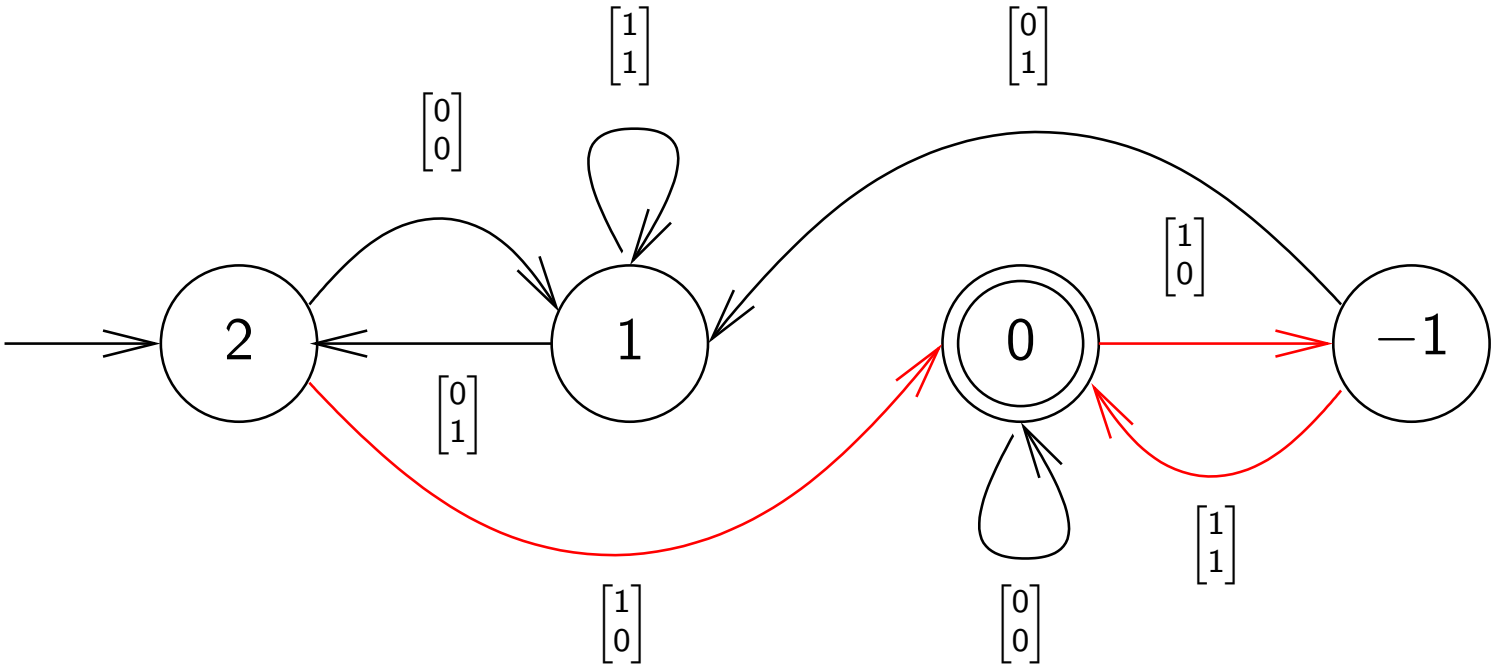
Diophantine Equations — Example

Formula: $2x - 3y = 2$



Diophantine Equations — Example

Formula: $2x - 3y = 2$

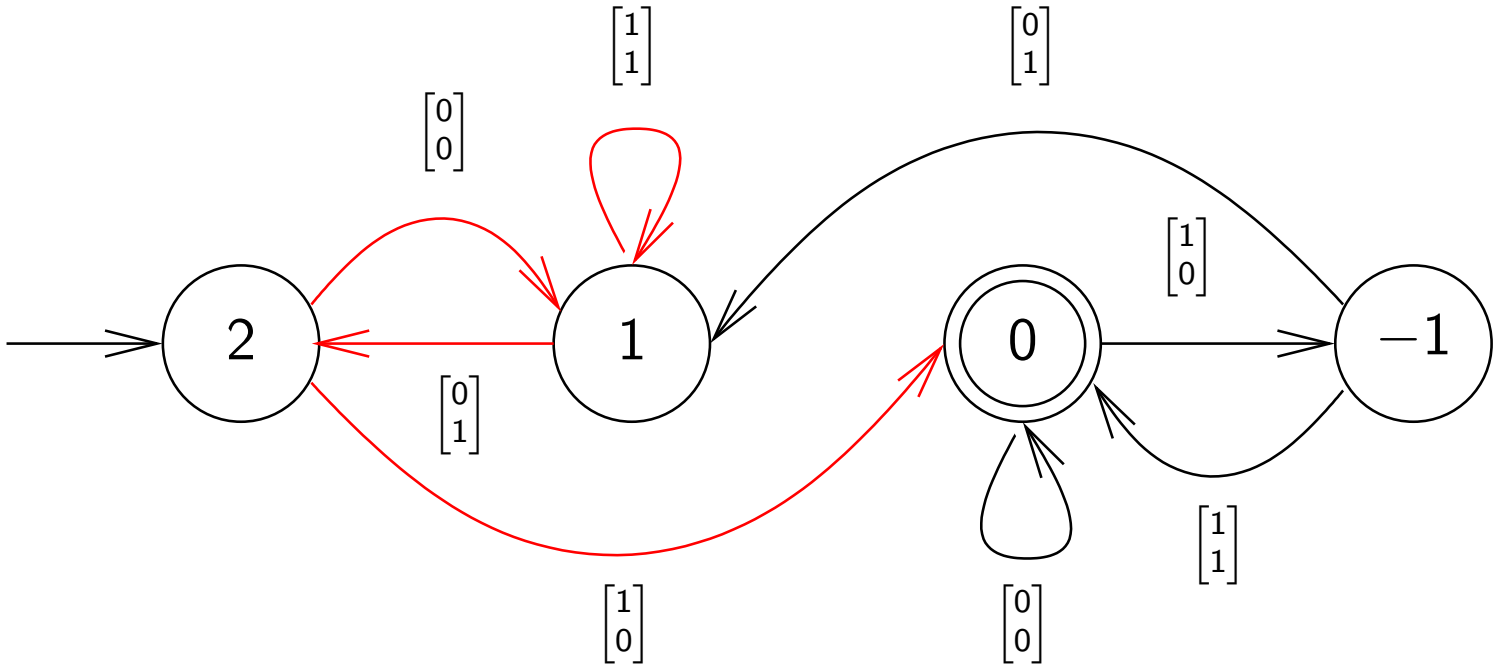


Some solutions

1	2	4	8	16	32	64	128	
1	1	1						$7 \cdot 2 = 14$
0	0	1						$4 \cdot 3 = 12$

Diophantine Equations — Example

Formula: $2x - 3y = 2$

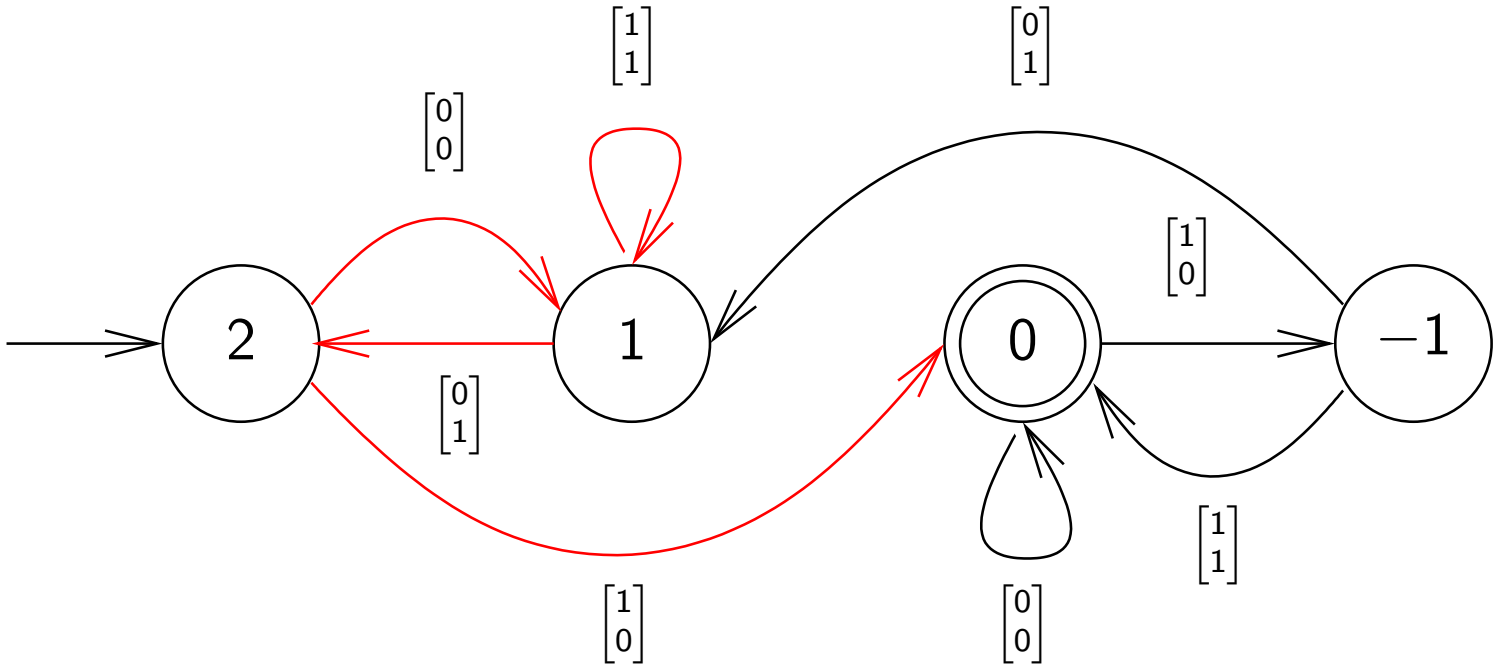


Some solutions

1	2	4	8	16	32	64	128	
0	1	1	0	1				$22 \cdot 2 = 44$
0	1	1	1	0				$14 \cdot 3 = 42$

Diophantine Equations — Example

Formula: $2x - 3y = 2$

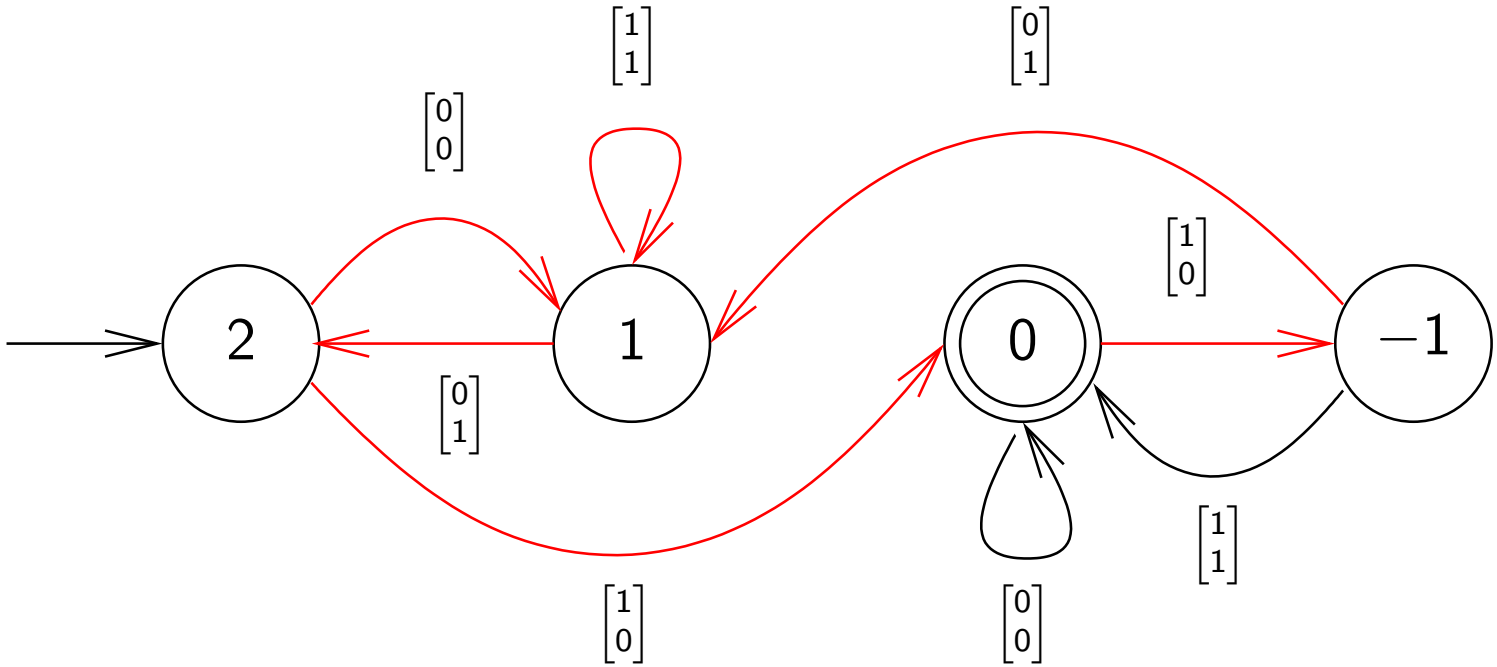


Some solutions

1	2	4	8	16	32	64	128	
0	1	0	1					$10 \cdot 2 = 20$
0	1	1	0					$6 \cdot 3 = 18$

Diophantine Equations — Example

Formula: $2x - 3y = 2$



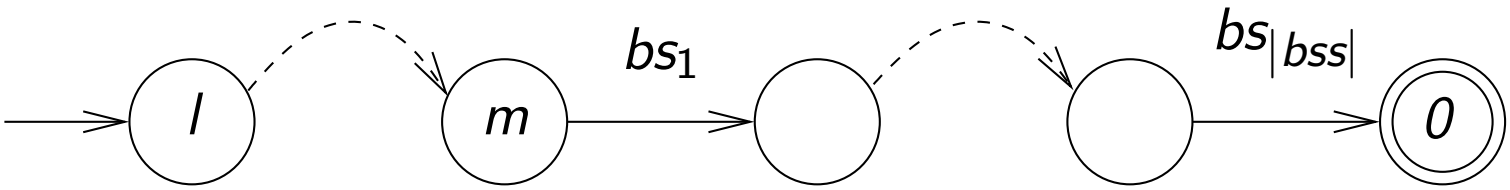
Some solutions

1	2	4	8	16	32	64	128	
0	1	0	1	1	0	0	1	$154 \cdot 2 = 308$
0	1	1	0	0	1	1	0	$102 \cdot 3 = 306$

Diophantine Equations — The Code

```
eq-dfa :: nat ⇒ int list ⇒ int ⇒ dfa
eq-dfa n ks l ≡
let (is, js) = dioph-dfs n ks l
in (map (λj. make-bdd
                (λxs. if eval-dioph ks xs mod 2 = j mod 2
                      then the is[int-to-nat-bij
                                    ((j − eval-dioph ks xs) div 2)]
                      else |js|)
                n [])
    js @
    [Leaf |js|],
    map (λj. j = 0) js @ [False])
```

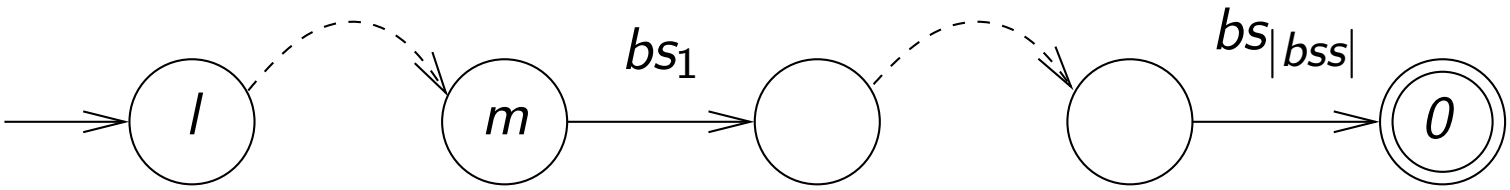
Diophantine Equations — Correctness



Strengthened statement

If $(l, m) \in (succsr\ (dioph\text{-}succs\ n\ ks))^*$ *and*
$\forall\, bs{\in}bss.\ is\text{-}alph\ n\ bs$ *then*
dfa-accepting $(eq\text{-}dfa\ n\ ks\ l)$
$(dfa\text{-}steps\ (eq\text{-}dfa\ n\ ks\ l)$
$(the\ (fst\ (dioph\text{-}dfs\ n\ ks\ l))[int\text{-}to\text{-}nat\text{-}bij\ m])\ bss) =$
$(eval\text{-}dioph\ ks\ (nats\text{-}of\text{-}boolss\ n\ bss) = m)$.

Diophantine Equations — Correctness



Corollary ($l = m$)

If $\forall$ *bs*$\in$*bss. is-alph n bs then*
dfa-accepts (*eq-dfa n ks l*) *bss* =
(*eval-dioph ks* (*nats-of-boolss n bss*) = *l*).

Outline

The Decision Procedure

```
dfa-of-pf :: nat ⇒ pf ⇒ dfa
dfa-of-pf n (Eq ks l) = eq-dfa n ks l
dfa-of-pf n (Le ks l) = ineq-dfa n ks l
dfa-of-pf n (Neg p) = negate-dfa (dfa-of-pf n p)
dfa-of-pf n (And p q) =
  binop-dfa (∧) (dfa-of-pf n p) (dfa-of-pf n q)
dfa-of-pf n (Or p q) =
  binop-dfa (∨) (dfa-of-pf n p) (dfa-of-pf n q)
dfa-of-pf n (Exist p) =
  rquot (det-nfa (quantify-nfa 0 (nfa-of-dfa (dfa-of-pf (Suc n) p))))
   n
dfa-of-pf n (Forall p) = dfa-of-pf n (Neg (Exist (Neg p)))
```

The Decision Procedure

$$
\begin{aligned}
&\textit{dfa-of-pf} :: \textit{nat} \Rightarrow \textit{pf} \Rightarrow \textit{dfa}\\
&\textit{dfa-of-pf}\ n\ (\textit{Eq}\ \textit{ks}\ l) = \textit{eq-dfa}\ n\ \textit{ks}\ l\\
&\textit{dfa-of-pf}\ n\ (\textit{Le}\ \textit{ks}\ l) = \textit{ineq-dfa}\ n\ \textit{ks}\ l\\
&\textit{dfa-of-pf}\ n\ (\textit{Neg}\ p) = \textit{negate-dfa}\ (\textit{dfa-of-pf}\ n\ p)\\
&\textit{dfa-of-pf}\ n\ (\textit{And}\ p\ q) =\\
&\quad \textit{binop-dfa}\ (\wedge)\ (\textit{dfa-of-pf}\ n\ p)\ (\textit{dfa-of-pf}\ n\ q)\\
&\textit{dfa-of-pf}\ n\ (\textit{Or}\ p\ q) =\\
&\quad \textit{binop-dfa}\ (\vee)\ (\textit{dfa-of-pf}\ n\ p)\ (\textit{dfa-of-pf}\ n\ q)\\
&\textit{dfa-of-pf}\ n\ (\textit{Exist}\ p) =\\
&\quad \textit{rquot}\ (\textit{det-nfa}\ (\textit{quantify-nfa}\ 0\ (\textit{nfa-of-dfa}\ (\textit{dfa-of-pf}\ (\textit{Suc}\ n)\ p))))\\
&\quad\ n\\
&\textit{dfa-of-pf}\ n\ (\textit{Forall}\ p) = \textit{dfa-of-pf}\ n\ (\textit{Neg}\ (\textit{Exist}\ (\textit{Neg}\ p)))
\end{aligned}
$$

Theorem (Correctness)

$$
\begin{aligned}
&\textit{If}\ \forall\, \textit{bs} \in \textit{bss}.\ \textit{is-alph}\ n\ \textit{bs}\ \ \textit{then}\\
&\textit{dfa-accepts}\ (\textit{dfa-of-pf}\ n\ p)\ \textit{bss} = \textit{eval-pf}\ p\ (\textit{nats-of-boolss}\ n\ \textit{bss}).
\end{aligned}
$$

Outline

Performance

- Algorithm can compete quite well with standard decision procedure for Presburger arithmetic available in Isabelle.

Performance

- Algorithm can compete quite well with standard decision procedure for Presburger arithmetic available in Isabelle.
- Size of DFA for stamp problem (without minimization): 6 states

Performance

- Algorithm can compete quite well with standard decision procedure for Presburger arithmetic available in Isabelle.
- Size of DFA for stamp problem (without minimization): 6 states
- Size of DFAs for subformulae:

<table>
<tr><td rowspan="2">Forall
6</td><td rowspan="2">Imp
15</td><td>Exist
13</td><td>Exist
9</td><td>Eq [5, 3, −1] 0
9</td></tr>
<tr><td colspan="3">Le [−1] −8
5</td></tr>
</table>

Performance

- Algorithm can compete quite well with standard decision procedure for Presburger arithmetic available in Isabelle.
- Size of DFA for stamp problem (without minimization): 6 states
- Size of DFAs for subformulae:

<table>
<tr><td rowspan="2">$Forall$
6</td><td rowspan="2">Imp
15</td><td>$Exist$
13</td><td>$Exist$
9</td><td colspan="2">$Eq\ [5,\ 3,\ -1]\ 0$
9</td></tr>
<tr><td colspan="4">$Le\ [-1]\ -8$
5</td></tr>
</table>

- Use of DFS pays off!

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers
  - sign bit / most significant bit comes first

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers
  - sign bit / most significant bit comes first
- Other representations for BDDs

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers
  - sign bit / most significant bit comes first
- Other representations for BDDs
  - ROBDDs with sharing [Verma, Asian 2000]

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers
  - sign bit / most significant bit comes first
- Other representations for BDDs
  - ROBDDs with sharing [Verma, Asian 2000]
  - Requires memory for storing BDDs

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers
  - sign bit / most significant bit comes first
- Other representations for BDDs
  - ROBDDs with sharing [Verma, Asian 2000]
  - Requires memory for storing BDDs
  - Algorithms no longer purely functional (more challenging to reason about)

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers
  - sign bit / most significant bit comes first
- Other representations for BDDs
  - ROBDDs with sharing [Verma, Asian 2000]
  - Requires memory for storing BDDs
  - Algorithms no longer purely functional (more challenging to reason about)
- Presburger Arithmetic on reals (using Büchi automata)

Future Work

- Other methods for constructing DFAs for Diophantine equations, e.g. [Wolper and Boigelot, TACAS 2000]
  - can directly deal with variables over integers
  - sign bit / most significant bit comes first
- Other representations for BDDs
  - ROBDDs with sharing [Verma, Asian 2000]
  - Requires memory for storing BDDs
  - Algorithms no longer purely functional (more challenging to reason about)
- Presburger Arithmetic on reals (using Büchi automata)
- Extend to WS1S, and apply it to circuit verification problems described in [Basin and Friedrich, FROCOS 2000].