Exotic Semiring Constraints

Michael Codish[1] Yoav Fekete[1] Carsten Fuhs[2]
Jürgen Giesl[3] Johannes Waldmann[4]

SMT12

[1]Dept. of CS, Ben-Gurion University of the Negev, Israel
[2]Dept. of CS, University College London, United Kingdom
[3]LuFG Informatik 2, RWTH Aachen University, Germany
[4]Fakultät IMN, HTWK Leipzig, Germany

Example: Arctic Semiring Constraints

arctic semiring: domain $A = \{-\infty\} \cup \mathbb{N}$,
$x \oplus y = \max(x, y)$, $x \otimes y = x + y$.

example constraint system:

$$(a_{11} \geq 0) \wedge (b_{11} \geq 0) \wedge \bigwedge_{\substack{i \in \{1,2\} \\ j \in \{1,2\}}} \left\{ \begin{array}{l} \quad (c_{ij} = (a_{i1} \otimes b_{1j}) \oplus (a_{i2} \otimes b_{2j})) \\ \wedge (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) > (c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j})) \\ \quad \vee (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) = -\infty) \\ \quad\quad \wedge ((c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j}) = -\infty))) \\ \wedge (b_{ij} \geq b_{i1} \otimes b_{1j} \oplus b_{i2} \otimes b_{2j}) \end{array} \right.$$

imagine this with 100 … 1000 unknowns

hard for DPLL(T) because $\oplus$ introduces disjunctions, $-\infty$ introduces case distinctions (in $\oplus$ and $\otimes$)

Our solution: unary bit-blasting.

Example: Arctic Semiring Constraints

arctic semiring: domain $A = \{-\infty\} \cup \mathbb{N}$,
$x \oplus y = \max(x, y)$, $x \otimes y = x + y$.

example constraint system:

$$(a_{11} \geq 0) \wedge (b_{11} \geq 0) \wedge$$

$$\bigwedge_{\substack{i \in \{1,2\} \\ j \in \{1,2\}}} \begin{cases} \quad (c_{ij} = (a_{i1} \otimes b_{1j}) \oplus (a_{i2} \otimes b_{2j})) \\ \wedge \, (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) > (c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j})) \\ \quad \vee \, (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) = -\infty) \\ \quad\quad \wedge ((c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j}) = -\infty))) \\ \wedge \, (b_{ij} \geq b_{i1} \otimes b_{1j} \oplus b_{i2} \otimes b_{2j}) \end{cases}$$

imagine this with 100 … 1000 unknowns

hard for DPLL(T) because $\oplus$ introduces disjunctions, $-\infty$ introduces case distinctions (in $\oplus$ and $\otimes$)

Our solution: unary bit-blasting.

Example: Arctic Semiring Constraints

arctic semiring: domain $A = \{-\infty\} \cup \mathbb{N}$, $x \oplus y = \max(x, y)$, $x \otimes y = x + y$.

example constraint system:

$$(a_{11} \geq 0) \wedge (b_{11} \geq 0) \wedge \bigwedge_{\substack{i \in \{1,2\} \\ j \in \{1,2\}}} \left\{ \begin{array}{l} \quad (c_{ij} = (a_{i1} \otimes b_{1j}) \oplus (a_{i2} \otimes b_{2j})) \\ \wedge\ (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) > (c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j})) \\ \quad \vee\ (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) = -\infty) \\ \qquad \wedge((c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j}) = -\infty))) \\ \wedge\ (b_{ij} \geq b_{i1} \otimes b_{1j} \oplus b_{i2} \otimes b_{2j}) \end{array} \right.$$

imagine this with 100 … 1000 unknowns

hard for DPLL(T) because $\oplus$ introduces disjunctions, $-\infty$ introduces case distinctions (in $\oplus$ and $\otimes$)

Our solution: unary bit-blasting.

Example: Arctic Semiring Constraints

arctic semiring: domain $A = \{-\infty\} \cup \mathbb{N}$,
$x \oplus y = \max(x, y)$, $x \otimes y = x + y$.

example constraint system:

$$(a_{11} \geq 0) \wedge (b_{11} \geq 0) \wedge \bigwedge_{\substack{i \in \{1,2\} \\ j \in \{1,2\}}} \left\{ \begin{array}{l} \quad (c_{ij} = (a_{i1} \otimes b_{1j}) \oplus (a_{i2} \otimes b_{2j})) \\ \wedge\ (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) > (c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j})) \\ \quad \vee\ (((a_{i1} \otimes a_{1j}) \oplus (a_{i2} \otimes a_{2j}) = -\infty) \\ \qquad \wedge((c_{i1} \otimes a_{1j}) \oplus (c_{i2} \otimes a_{2j}) = -\infty))) \\ \wedge\ (b_{ij} \geq b_{i1} \otimes b_{1j} \oplus b_{i2} \otimes b_{2j}) \end{array} \right.$$

imagine this with 100 … 1000 unknowns

hard for DPLL(T) because $\oplus$ introduces disjunctions,
$-\infty$ introduces case distinctions (in $\oplus$ and $\otimes$)

Our solution: unary bit-blasting.

Where do these constraints occur?

the framework is *exotic semirings*, examples:

- arctic $\{-\infty\} \cup \mathbb{Z}, \max, +$
- tropical $\mathbb{N} \cup \{+\infty\}, \min, +$
- fuzzy $\{-\infty\} \cup \mathbb{N} \cup \{+\infty\}, \min, \max$

applications:

- formal languages (star height problem) (Imre Simon 1988)
- idempotent analysis
- disjunctive invariants in static analysis
- automated analysis of termination of programs (modelled as rewriting systems)

Arctic Matrices for Termination

$\mathcal{R} = \{ \mathsf{a\,a} \to \mathsf{a\,b\,a} \}$ and $\mathcal{S} = \{ \mathsf{b} \to \mathsf{b\,b} \}$
to show relative termination of $\mathcal{R}$ w.r.t. $\mathcal{S}$
(no $\mathcal{R} \cup \mathcal{S}$-derivation with infinitely many $\mathcal{R}$ steps)

interpret symbols by matrices $\mathsf{a} \mapsto A, \mathsf{b} \mapsto B$
with $A_{1,1} \geq 0 \wedge B_{1,1} \geq 0 \wedge (A^2 >_0 ABA) \wedge (B \geq B^2)$.
where $(x >_0 y)$ is $x > y \vee (x = -\infty = y)$

matrix dimension 2 gives constraints from intro slide
where c_{ij} is contents of $C = AB$,

one solution is $A = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & -\infty \\ 0 & -\infty \end{pmatrix}$.

Arctic Matrices for Termination

$\mathcal{R} = \{ \text{a a} \to \text{a b a} \}$ and $\mathcal{S} = \{ \text{b} \to \text{b b} \}$
to show relative termination of $\mathcal{R}$ w.r.t. $\mathcal{S}$
(no $\mathcal{R} \cup \mathcal{S}$-derivation with infinitely many $\mathcal{R}$ steps)

interpret symbols by matrices $\text{a} \mapsto A, \text{b} \mapsto B$
with $A_{1,1} \geq 0 \wedge B_{1,1} \geq 0 \wedge (A^2 >_0 ABA) \wedge (B \geq B^2)$.
where $(x >_0 y)$ is $x > y \vee (x = -\infty = y)$

matrix dimension 2 gives constraints from intro slide
where c_{ij} is contents of $C = AB$,

one solution is $A = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & -\infty \\ 0 & -\infty \end{pmatrix}$.

Arctic Matrices for Termination

$\mathcal{R} = \{ \text{a a} \to \text{a b a} \}$ and $\mathcal{S} = \{ \text{b} \to \text{b b} \}$
to show relative termination of $\mathcal{R}$ w.r.t. $\mathcal{S}$
(no $\mathcal{R} \cup \mathcal{S}$-derivation with infinitely many $\mathcal{R}$ steps)

interpret symbols by matrices $\text{a} \mapsto A, \text{b} \mapsto B$
with $A_{1,1} \geq 0 \wedge B_{1,1} \geq 0 \wedge (A^2 >_0 ABA) \wedge (B \geq B^2)$.
where $(x >_0 y)$ is $x > y \vee (x = -\infty = y)$

matrix dimension 2 gives constraints from intro slide
where c_{ij} is contents of $C = AB$,

one solution is $A = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & -\infty \\ 0 & -\infty \end{pmatrix}$.

Arctic Matrices for Termination

$\mathcal{R} = \{ \text{a a} \to \text{a b a} \}$ and $\mathcal{S} = \{ \text{b} \to \text{b b} \}$
to show relative termination of $\mathcal{R}$ w.r.t. $\mathcal{S}$
(no $\mathcal{R} \cup \mathcal{S}$-derivation with infinitely many $\mathcal{R}$ steps)

interpret symbols by matrices $\text{a} \mapsto A, \text{b} \mapsto B$
with $A_{1,1} \geq 0 \wedge B_{1,1} \geq 0 \wedge (A^2 >_0 ABA) \wedge (B \geq B^2)$.
where $(x >_0 y)$ is $x > y \vee (x = -\infty = y)$

matrix dimension 2 gives constraints from intro slide
where c_{ij} is contents of $C = AB$,

one solution is $A = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & -\infty \\ 0 & -\infty \end{pmatrix}$.

Topics of this talk

- definition and motivation of exotic semiring constraints
- solving by translation (QF_LIA, QF_IDL)
- solving by unary bitblasting:
  - naturals, integers, exotic numbers
- implementation, empirical evaluation
  - the “killer” example
  - comparison of different approaches

Translation to QF_LIA, QF_IDL

Represent exotic number as pair of "(minus/plus) infinity" (a boolean) and "contents" (a number).

arctic multiplication (plus): $(m, c) = \bigotimes_k (m_k, c_k)$ iff $(m = \bigvee_k m_k) \wedge (\neg m \rightarrow (c = \sum_k c_k))$.

arctic addition (max): $(m, c) = \bigoplus_k (m_k, c_k)$ iff $(m = \bigwedge_k m_k) \wedge (\neg m \rightarrow \bigwedge_k (\neg m_k \rightarrow c \geq c_k))$ $\wedge (\neg m \rightarrow \bigvee_k (\neg m_k \wedge c = c_k))$.

For fuzzy semiring constraints, operations are min and max (and no $+$), so encoding goes to QF_IDL.

We submitted a bunch of these to SMT-COMP'12. arctic/tropical in QF_LIA demonstration, QF_IDL?

Translation to QF_LIA, QF_IDL

Represent exotic number as pair of "(minus/plus) infinity" (a boolean) and "contents" (a number).

arctic multiplication (plus): $(m, c) = \bigotimes_k (m_k, c_k)$ iff $(m = \bigvee_k m_k) \wedge (\neg m \to (c = \sum_k c_k))$.

arctic addition (max): $(m, c) = \bigoplus_k (m_k, c_k)$ iff $(m = \bigwedge_k m_k) \wedge (\neg m \to \bigwedge_k (\neg m_k \to c \geq c_k))$ $\wedge (\neg m \to \bigvee_k (\neg m_k \wedge c = c_k))$.

For fuzzy semiring constraints, operations are min and max (and no $+$), so encoding goes to QF_IDL.

We submitted a bunch of these to SMT-COMP'12. arctic/tropical in QF_LIA demonstration, QF_IDL?

Translation to QF_LIA, QF_IDL

Represent exotic number as pair of "(minus/plus) infinity" (a boolean) and "contents" (a number).

arctic multiplication (plus): $(m, c) = \bigotimes_k (m_k, c_k)$ iff $(m = \bigvee_k m_k) \wedge (\neg m \to (c = \sum_k c_k))$.

arctic addition (max): $(m, c) = \bigoplus_k (m_k, c_k)$ iff $(m = \bigwedge_k m_k) \wedge (\neg m \to \bigwedge_k (\neg m_k \to c \geq c_k))$ $\wedge (\neg m \to \bigvee_k (\neg m_k \wedge c = c_k))$.

For fuzzy semiring constraints, operations are min and max (and no $+$), so encoding goes to QF_IDL.

We submitted a bunch of these to SMT-COMP'12. arctic/tropical in QF_LIA demonstration, QF_IDL?

Translation to QF_LIA, QF_IDL

Represent exotic number as pair of "(minus/plus) infinity" (a boolean) and "contents" (a number).

arctic multiplication (plus): $(m, c) = \bigotimes_k (m_k, c_k)$ iff $(m = \bigvee_k m_k) \wedge (\neg m \to (c = \sum_k c_k))$.

arctic addition (max): $(m, c) = \bigoplus_k (m_k, c_k)$ iff $(m = \bigwedge_k m_k) \wedge (\neg m \to \bigwedge_k (\neg m_k \to c \geq c_k))$ $\wedge (\neg m \to \bigvee_k (\neg m_k \wedge c = c_k))$.

For fuzzy semiring constraints, operations are min and max (and no $+$), so encoding goes to QF_IDL.

We submitted a bunch of these to SMT-COMP'12. arctic/tropical in QF_LIA demonstration, QF_IDL?

Translation to QF_LIA, QF_IDL

Represent exotic number as pair of "(minus/plus) infinity" (a boolean) and "contents" (a number).

arctic multiplication (plus): $(m, c) = \bigotimes_k (m_k, c_k)$ iff $(m = \bigvee_k m_k) \wedge (\neg m \rightarrow (c = \sum_k c_k))$.

arctic addition (max): $(m, c) = \bigoplus_k (m_k, c_k)$ iff $(m = \bigwedge_k m_k) \wedge (\neg m \rightarrow \bigwedge_k (\neg m_k \rightarrow c \geq c_k))$ $\wedge (\neg m \rightarrow \bigvee_k (\neg m_k \wedge c = c_k))$.

For fuzzy semiring constraints, operations are min and max (and no $+$), so encoding goes to QF_IDL.

We submitted a bunch of these to SMT-COMP'12. arctic/tropical in QF_LIA demonstration, QF_IDL?

Unary Bit-Blasting: Order Encoding

translation QF_LIA to SAT (sound, incomplete):

- restrict to finite domain $\{0, 1, \ldots, B\}$
- number $x \Rightarrow$ monotone list of booleans $[x_1, \ldots, x_B]$ where $x_i \leftrightarrow (x \geq i)$, e.g., $3 = [1, 1, 1, 0, 0, 0, 0, 0]$
- arithmetical operations $\Rightarrow$ boolean functions

unary encoding is highly redundant,
but allows for propagations,
and thus SAT solvers perform well

Unary Bit-Blasting: Order Encoding

translation QF_LIA to SAT (sound, incomplete):

- restrict to finite domain $\{0, 1, \ldots, B\}$
- number $x \Rightarrow$ monotone list of booleans $[x_1, \ldots, x_B]$ where $x_i \leftrightarrow (x \geq i)$, e.g., $3 = [1, 1, 1, 0, 0, 0, 0, 0]$
- arithmetical operations $\Rightarrow$ boolean functions

unary encoding is highly redundant,
but allows for propagations,
and thus SAT solvers perform well

Unary Bit-Blasting: Order Encoding

translation QF_LIA to SAT (sound, incomplete):

- restrict to finite domain $\{0, 1, \ldots, B\}$
- number $x \Rightarrow$ monotone list of booleans $[x_1, \ldots, x_B]$ where $x_i \leftrightarrow (x \geq i)$, e.g., $3 = [1, 1, 1, 0, 0, 0, 0, 0]$
- arithmetical operations $\Rightarrow$ boolean functions

unary encoding is highly redundant,
but allows for propagations,
and thus SAT solvers perform well

Unary Bit-Blasting: Order Encoding

translation QF_LIA to SAT (sound, incomplete):

- restrict to finite domain $\{0, 1, \ldots, B\}$
- number $x \Rightarrow$ monotone list of booleans $[x_1, \ldots, x_B]$ where $x_i \leftrightarrow (x \geq i)$, e.g., $3 = [1, 1, 1, 0, 0, 0, 0, 0]$
- arithmetical operations $\Rightarrow$ boolean functions

unary encoding is highly redundant,
but allows for propagations,
and thus SAT solvers perform well

Unary Bit-Blasting: Order Encoding

translation QF_LIA to SAT (sound, incomplete):

- restrict to finite domain $\{0, 1, \ldots, B\}$
- number $x \Rightarrow$ monotone list of booleans $[x_1, \ldots, x_B]$ where $x_i \leftrightarrow (x \geq i)$, e.g., $3 = [1, 1, 1, 0, 0, 0, 0, 0]$
- arithmetical operations $\Rightarrow$ boolean functions

unary encoding is highly redundant,
but allows for propagations,
and thus SAT solvers perform well

Order-Encoded Operations

should prefer conjunctive encodings.
Example: comparison of k-bit unary numbers,
e.g., $a = \langle 1, 1, 1, 0 \rangle, b = \langle 1, 1, 0, 0 \rangle$.

- this is easy: $a \geq b \iff \bigwedge_k \{a_k \leftarrow b_k\}$
- by negation: $a > b \iff \bigvee_k \{a_k \wedge \neg b_k\}$
- this is equivalent (by monotonicity) and we think it is better, since it is a conjunction: $a > b \iff a_1 \wedge \neg b_k \wedge \bigwedge_k \{a_{k+1} \leftarrow b_k\}$

no hard evidence.
also depends on whether we want to assert $a > b$, or have it as a subformula, e.g., assert $((a > b) \vee \dots)$

Order-Encoded Operations

should prefer conjunctive encodings.
Example: comparison of k-bit unary numbers,
e.g., $a = \langle 1, 1, 1, 0 \rangle, b = \langle 1, 1, 0, 0 \rangle$.

- this is easy: $a \geq b \iff \bigwedge_k \{a_k \leftarrow b_k\}$
- by negation: $a > b \iff \bigvee_k \{a_k \wedge \neg b_k\}$
- this is equivalent (by monotonicity) and we think it is better, since it is a conjunction: $a > b \iff a_1 \wedge \neg b_k \wedge \bigwedge_k \{a_{k+1} \leftarrow b_k\}$

no hard evidence.
also depends on whether we want to assert $a > b$, or have it as a subformula, e.g., assert $((a > b) \vee \dots)$

Order-Encoded Operations

should prefer conjunctive encodings.
Example: comparison of k-bit unary numbers,
e.g., $a = \langle 1, 1, 1, 0\rangle, b = \langle 1, 1, 0, 0\rangle$.

- this is easy: $a \geq b \iff \bigwedge_k \{a_k \leftarrow b_k\}$
- by negation: $a > b \iff \bigvee_k \{a_k \wedge \neg b_k\}$
- this is equivalent (by monotonicity)
  and we think it is better, since it is a conjunction:
  $a > b \iff a_1 \wedge \neg b_k \wedge \bigwedge_k \{a_{k+1} \leftarrow b_k\}$

no hard evidence.
also depends on whether we want to assert $a > b$,
or have it as a subformula, e.g., assert $((a > b) \vee \dots)$

Order-Encoded Operations

should prefer conjunctive encodings.
Example: comparison of k-bit unary numbers,
e.g., $a = \langle 1, 1, 1, 0 \rangle, b = \langle 1, 1, 0, 0 \rangle$.

- this is easy: $a \geq b \iff \bigwedge_k \{a_k \leftarrow b_k\}$
- by negation: $a > b \iff \bigvee_k \{a_k \wedge \neg b_k\}$
- this is equivalent (by monotonicity) and we think it is better, since it is a conjunction: $a > b \iff a_1 \wedge \neg b_k \wedge \bigwedge_k \{a_{k+1} \leftarrow b_k\}$

no hard evidence.
also depends on whether we want to assert $a > b$, or have it as a subformula, e.g., assert $((a > b) \vee \dots)$

Order-Encoded Operations

should prefer conjunctive encodings.
Example: comparison of k-bit unary numbers,
e.g., $a = \langle 1, 1, 1, 0 \rangle, b = \langle 1, 1, 0, 0 \rangle$.

- this is easy: $a \geq b \iff \bigwedge_k \{a_k \leftarrow b_k\}$
- by negation: $a > b \iff \bigvee_k \{a_k \wedge \neg b_k\}$
- this is equivalent (by monotonicity)
  and we think it is better, since it is a conjunction:
  $a > b \iff a_1 \wedge \neg b_k \wedge \bigwedge_k \{a_{k+1} \leftarrow b_k\}$

no hard evidence.
also depends on whether we want to assert $a > b$,
or have it as a subformula, e.g., assert $((a > b) \vee \dots)$

Order Encoded Arithmetics

- max/min by point-wise $\vee/\wedge$ (linear formula size).
- addition: let $a = \langle a_1, \ldots, a_k \rangle, b = \langle b_1, \ldots, b_{k'} \rangle$.
  Then $a + b = c$ where $c = \langle c_1, \ldots, c_{k+k'} \rangle$ with

$$\bigwedge_{\substack{0 < i \leq k, \\ 0 < j \leq k'}} \left\{ \begin{array}{l} (a_i \rightarrow c_i) \ \wedge \ (\neg a_i \rightarrow \neg c_{k'+i}) \wedge \\ (b_j \rightarrow c_j) \ \wedge \ (\neg b_j \rightarrow \neg c_{k+j}) \wedge \\ (a_i \wedge b_j \rightarrow c_{i+j}) \ \wedge \ (\neg a_i \wedge \neg b_j \rightarrow \neg c_{i+j-1}) \end{array} \right\}$$

  quadratic size, but no extra variables.

  addition via sorting networks (less clauses, more vars) does not pay off (propagation is delayed?)

Order Encoded Arithmetics

- max/min by point-wise $\vee / \wedge$ (linear formula size).
- addition: let $a = \langle a_1, \ldots, a_k \rangle, b = \langle b_1, \ldots, b_{k'} \rangle$.
  Then $a + b = c$ where $c = \langle c_1, \ldots, c_{k+k'} \rangle$ with

$$\bigwedge_{\substack{0 < i \le k,\\ 0 < j \le k'}} \left\{ \begin{array}{l} (a_i \to c_i) \;\wedge\; (\neg a_i \to \neg c_{k'+i}) \wedge \\ (b_j \to c_j) \;\wedge\; (\neg b_j \to \neg c_{k+j}) \wedge \\ (a_i \wedge b_j \to c_{i+j}) \;\wedge\; (\neg a_i \wedge \neg b_j \to \neg c_{i+j-1}) \end{array} \right\}$$

  quadratic size, but no extra variables.

  addition via sorting networks (less clauses, more vars) does not pay off (propagation is delayed?)

Order Encoded Arithmetics

- max/min by point-wise $\vee/\wedge$ (linear formula size).
- addition: let $a = \langle a_1, \ldots, a_k \rangle, b = \langle b_1, \ldots, b_{k'} \rangle$. Then $a + b = c$ where $c = \langle c_1, \ldots, c_{k+k'} \rangle$ with

$$\bigwedge_{\substack{0 < i \leq k, \\ 0 < j \leq k'}} \left\{ \begin{array}{l} (a_i \to c_i) \ \wedge \ (\neg a_i \to \neg c_{k'+i}) \wedge \\ (b_j \to c_j) \ \wedge \ (\neg b_j \to \neg c_{k+j}) \wedge \\ (a_i \wedge b_j \to c_{i+j}) \ \wedge \ (\neg a_i \wedge \neg b_j \to \neg c_{i+j-1}) \end{array} \right\}$$

  quadratic size, but no extra variables.

  addition via sorting networks (less clauses, more vars) does not pay off (propagation is delayed?)

Order Encoded Arithmetics

- max/min by point-wise $\vee/\wedge$ (linear formula size).
- addition: let $a = \langle a_1, \ldots, a_k \rangle, b = \langle b_1, \ldots, b_{k'} \rangle$. Then $a + b = c$ where $c = \langle c_1, \ldots, c_{k+k'} \rangle$ with

$$\bigwedge_{\substack{0 < i \leq k, \\ 0 < j \leq k'}} \left\{ \begin{array}{l} (a_i \rightarrow c_i) \ \wedge \ (\neg a_i \rightarrow \neg c_{k'+i}) \wedge \\ (b_j \rightarrow c_j) \ \wedge \ (\neg b_j \rightarrow \neg c_{k+j}) \wedge \\ (a_i \wedge b_j \rightarrow c_{i+j}) \ \wedge \ (\neg a_i \wedge \neg b_j \rightarrow \neg c_{i+j-1}) \end{array} \right\}$$

  quadratic size, but no extra variables.

  addition via sorting networks (less clauses, more vars) does not pay off (propagation is delayed?)

Extensions

- integers: shift the encoding.
  transform $x \in \{-k+1, \dots, k\}$
  to $x + k$ and encode as natural.
  keep min and max, modify $+$ (shift back)
- exotic numbers: use one extra bit for $-\infty, +\infty$
  (either one for arctic and tropical, both for fuzzy)
  keep monotonicity, $\Rightarrow$ keep min and max
- Overflows: are not allowed (otherwise unsound)
  Either increase bit width as needed (in addition),
  or keep bit width and assert "$\neg$ overflow".

Extensions

- integers: shift the encoding.
  transform $x \in \{-k+1, \ldots, k\}$
  to $x + k$ and encode as natural.
  keep min and max, modify $+$ (shift back)
- exotic numbers: use one extra bit for $-\infty, +\infty$
  (either one for arctic and tropical, both for fuzzy)
  keep monotonicity, $\Rightarrow$ keep min and max
- Overflows: are not allowed (otherwise unsound)
  Either increase bit width as needed (in addition),
  or keep bit width and assert "$\neg$ overflow".

Extensions

- integers: shift the encoding.
  transform $x \in \{-k+1, \ldots, k\}$
  to $x + k$ and encode as natural.
  keep min and max, modify $+$ (shift back)
- exotic numbers: use one extra bit for $-\infty, +\infty$
  (either one for arctic and tropical, both for fuzzy)
  keep monotonicity, $\Rightarrow$ keep min and max
- Overflows: are not allowed (otherwise unsound)
  Either increase bit width as needed (in addition),
  or keep bit width and assert "$\neg$ overflow".

Improvements

- low level: boolean equipropagation [MCLS11] (Bee)
  (detect instances of $x \leftrightarrow y$ or $x \leftrightarrow \neg y$ and propagate)
- high level: algebraic simplification [EWZ08] (Matchbox)
  (extraction of common factors in matrix products)
  before emitting the QF_LIA system, or the CNF.

Improvements

- low level: boolean equipropagation [MCLS11] (Bee)
  (detect instances of $x \leftrightarrow y$ or $x \leftrightarrow \neg y$ and propagate)
- high level: algebraic simplification [EWZ08] (Matchbox)
  (extraction of common factors in matrix products)
  before emitting the QF_LIA system, or the CNF.

The "Killer" Example

Termination benchmark SRS/Gebhardt/19
$\{0000 \to 1011, 1001 \to 0010\}$ (open since 2006)
is terminating since tropical matrix constraint system

$$
\begin{aligned}
& 0_\# 0^3 \geq 1_\# 01^2 \wedge 1_\# 0^2 1 \geq 0_\# 010 \\
\wedge \;\; & 0^4 \geq \; 101^2 \wedge 10^2 1 \geq 0^2 10 \\
\wedge \;\; & (0^4 >_0 101^2 \;\vee\; 10^2 1 >_0 0^2 10).
\end{aligned}
$$

is solvable for 8×8, minisat needs one hour.

The “Killer” Example

Termination benchmark SRS/Gebhardt/19
$\{0000 \to 1011, 1001 \to 0010\}$ (open since 2006)

is terminating since tropical matrix constraint system

$$
\begin{array}{ll}
 & 0_{\#}0^3 \geq 1_{\#}01^2 \wedge 1_{\#}0^21 \geq 0_{\#}010 \\
\wedge & 0^4 \geq \ 101^2 \wedge 10^21 \geq 0^210 \\
\wedge & (0^4 >_0 101^2 \ \vee \ 10^21 >_0 0^210).
\end{array}
$$

is solvable for 8×8, minisat needs one hour.

Experimental Results

using solvers satchmo-smt, Bee, Z3
on exotic constraints from termination problems

- 3 bit binary vs. 7 bit unary (equal range)
  outcome: unary is better
- Z3 (with DPLL(Simplex)?) vs. unary (with iterative deepening = increasing bit width)
  outcome: unary is better
- unary: straightforward (satchmo-smt) vs. preprocessed (Bee)
  outcome: not conclusive.
  (minisat preprocessor will run anyway)

Experimental Results

using solvers satchmo-smt, Bee, Z3
on exotic constraints from termination problems

- 3 bit binary vs. 7 bit unary (equal range)
  outcome: unary is better
- Z3 (with DPLL(Simplex)?) vs. unary (with iterative deepening $=$ increasing bit width)
  outcome: unary is better
- unary: straightforward (satchmo-smt) vs. preprocessed (Bee)
  outcome: not conclusive.
  (minisat preprocessor will run anyway)

Experimental Results

using solvers satchmo-smt, Bee, Z3
on exotic constraints from termination problems

- 3 bit binary vs. 7 bit unary (equal range)
  outcome: unary is better
- Z3 (with DPLL(Simplex)?) vs. unary (with iterative deepening $=$ increasing bit width)
  outcome: unary is better
- unary: straightforward (satchmo-smt) vs. preprocessed (Bee)
  outcome: not conclusive.
  (minisat preprocessor will run anyway)

Why and When does this Work?

Exotic termination constraint systems contain $>, \geq, \max, \min, +$, 0 (and no number > 0). So it seems quite likely that solvability equals solvability in small numbers.

Why does unary seem to be better than binary? Better propagation?

Why does DPLL(T) not work (fast enough)? Lots of disjunctions and booleans.

When does unary bitblasting not work? With “large” constants. (As most QF_LIA benchmarks have.)

Why and When does this Work?

Exotic termination constraint systems contain $>, \geq, \max, \min, +$, 0 (and no number > 0). So it seems quite likely that solvability equals solvability in small numbers.

Why does unary seem to be better than binary? Better propagation?

Why does DPLL(T) not work (fast enough)? Lots of disjunctions and booleans.

When does unary bitblasting not work? With "large" constants. (As most QF_LIA benchmarks have.)

Why and When does this Work?

Exotic termination constraint systems contain $>, \geq, \max, \min, +$, 0 (and no number > 0). So it seems quite likely that solvability equals solvability in small numbers.

Why does unary seem to be better than binary? Better propagation?

Why does DPLL(T) not work (fast enough)? Lots of disjunctions and booleans.

When does unary bitblasting not work? With "large" constants. (As most QF_LIA benchmarks have.)

Why and When does this Work?

Exotic termination constraint systems contain $>, \geq, \max, \min, +$, 0 (and no number > 0). So it seems quite likely that solvability equals solvability in small numbers.

Why does unary seem to be better than binary? Better propagation?

Why does DPLL(T) not work (fast enough)? Lots of disjunctions and booleans.

When does unary bitblasting not work? With “large” constants. (As most QF_LIA benchmarks have.)