JOHANNES KEPLER
UNIVERSITY LINZ

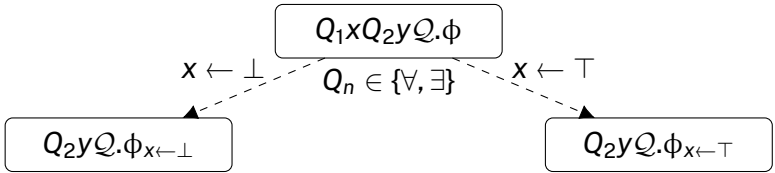
Search-Space Pruning with Int-Splits

Supported by the LIT AI Lab funded by the State of Upper Austria.

Maximilian Heisinger, Irfansha Shaik, Jaco van de Pol
Institute for Symbolic AI, JKU Linz
Department of Computer Science, University of Aarhus

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Austria
jku.at

Divide and Conquer in QBF



Divide and Conquer as a strong fit to QBF

QBF answers some of Divide and Conquer's biggest problems elegantly:

Which variable to split on?

variable order is already given by the prefix $\mathcal{Q}$

Divide and Conquer as a strong fit to QBF

QBF answers some of Divide and Conquer's biggest problems elegantly:

Which variable to split on?

variable order is already given by the prefix $\mathcal{Q}$

How deep to split?

$\log_2$ CPUs or along structures in the formula (e.g. *int-splits*)

Related: ParaQooba as Distributed QBF Solver

Builds a *guiding path* along the quantifier prefix $\mathcal{Q}$ (similar to solvers like MPIDepQBF)

Related: ParaQooba as Distributed QBF Solver

Builds a *guiding path* along the quantifier prefix $\mathcal{Q}$ (similar to solvers like MPIDepQBF)

Implements *existential* and *universal* assessment functions.

Related: ParaQooba as Distributed QBF Solver

Builds a *guiding path* along the quantifier prefix $\mathcal{Q}$ (similar to solvers like MPIDepQBF)

Implements *existential* and *universal* assessment functions.

Solves the formulas in the leaves of the solve tree using a portfolio of solvers, powered by QuAPI.

Related: ParaQooba as Distributed QBF Solver

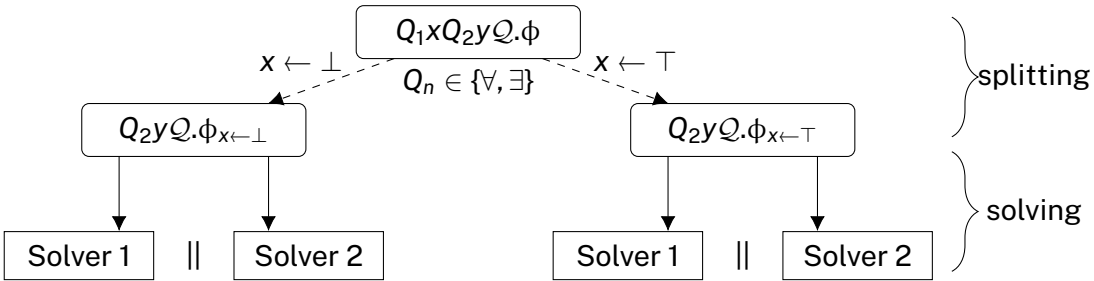
Builds a *guiding path* along the quantifier prefix $\mathcal{Q}$ (similar to solvers like MPIDepQBF)

Implements *existential* and *universal* assessment functions.

Solves the formulas in the leaves of the solve tree using a portfolio of solvers, powered by QuAPI.

Monolithic solver architecture, the solver is one process (with sub-processes) that finishes and gives a SAT/UNSAT answer.

ParaQooba Solver Structure



Search-Space Pruning: Int-Splits

Instead of the solve-tree resembling a binary tree, have arbitrary many children per node covering sets of variables interpreted as bit-vectors.

Search-Space Pruning: Int-Splits

Instead of the solve-tree resembling a binary tree, have arbitrary many children per node covering sets of variables interpreted as bit-vectors.

Can reduce the number of sub-formulas in half for each layer, dampening the exponential growth of leaf formulas to solve with growing depth.

Search-Space Pruning: Int-Splits

Instead of the solve-tree resembling a binary tree, have arbitrary many children per node covering sets of variables interpreted as bit-vectors.

Can reduce the number of sub-formulas in half for each layer, dampening the exponential growth of leaf formulas to solve with growing depth.

First introduced as a feature of ParaQooba *at TACAS'23, then described in another paper on Arxiv and by us here.*

Int-Splits Syntax

$Q_1(\vec{x}_1)_{c_1} \dots Q_n(\vec{x}_n)_{c_n}.\varphi$

Int-Splits Syntax

$Q_1(\vec{x}_1)_{c_1} \dots Q_n(\vec{x}_n)_{c_n}.\varphi$

$Q_i \in \{\forall, \exists\}$.

Int-Splits Syntax

$Q_1(\vec{x}_1)_{c_1} \dots Q_n(\vec{x}_n)_{c_n}.\varphi$

$Q_i \in \{\forall, \exists\}$.

$\vec{x}_i$ are bit-vector variables of size ≥ 1 such that there is no Boolean variable that occurs in two different bit-vector variables.

Int-Splits Syntax

$Q_1(\vec{x}_1)_{c_1} \dots Q_n(\vec{x}_n)_{c_n}.\varphi$

$Q_i \in \{\forall, \exists\}$.

$\vec{x}_i$ are bit-vector variables of size ≥ 1 such that there is no Boolean variable that occurs in two different bit-vector variables.

The constraints $c_i \in \{ <v_i,\ >v_i,\ =V,\ \top \}$ restrict the domain of the quantifier Q_i. In the definition of a constraint $v_i \in \mathbb{N}$ is a constant and V is a set of bit-vectors.

Int-Splits Syntax

$Q_1(\vec{x}_1)_{c_1} \dots Q_n(\vec{x}_n)_{c_n}.\varphi$

$Q_i \in \{\forall, \exists\}$.

$\vec{x}_i$ are bit-vector variables of size ≥ 1 such that there is no Boolean variable that occurs in two different bit-vector variables.

The constraints $c_i \in \{ <v_i,\ >v_i,\ =V,\ \top \}$ restrict the domain of the quantifier Q_i. In the definition of a constraint $v_i \in \mathbb{N}$ is a constant and V is a set of bit-vectors.

φ is a propositional formula in CNF over the Boolean variables of the $\vec{x}_i$.

Int-Split Semantics

$\phi = Q\vec{x}_c \mathcal{Q}.\varphi$

ϕ is true iff $\nu(\phi) = 1$, with the interpretation function ν defined as follows.

$\nu(\exists \vec{x}_\top \mathcal{Q}.\varphi) = 1$ iff $\sum_{\sigma \in \mathcal{S}(\vec{x})} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) \geq 1$

$\nu(\forall \vec{x}_\top \mathcal{Q}.\varphi) = 1$ iff $\prod_{\sigma \in \mathcal{S}(\vec{x})} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) = 1$

Int-Split Semantics

$\phi = Q\vec{x}_c \mathcal{Q}.\varphi$
ϕ is true iff $\nu(\phi) = 1$, with the interpretation function ν defined as follows.

$\nu(\exists \vec{x}_\top \mathcal{Q}.\varphi) = 1$ iff $\sum_{\sigma \in \mathcal{S}(\vec{x})} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) \geq 1$
$\nu(\forall \vec{x}_\top \mathcal{Q}.\varphi) = 1$ iff $\prod_{\sigma \in \mathcal{S}(\vec{x})} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) = 1$

$\nu(\exists \vec{x}_{\diamondsuit v} \mathcal{Q}.\varphi) = 1$ iff $\sum_{\sigma \in \mathcal{S}(\vec{x}), \langle \sigma(\vec{x}) \rangle \diamondsuit v} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) \geq 1$ with $\diamondsuit \in \{<, >\}$
$\nu(\forall \vec{x}_{\diamondsuit v} \mathcal{Q}.\varphi) = 1$ iff $\prod_{\sigma \in \mathcal{S}(\vec{x}), \langle \sigma(\vec{x}) \rangle \diamondsuit v} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) = 1$ with $\diamondsuit \in \{<, >\}$

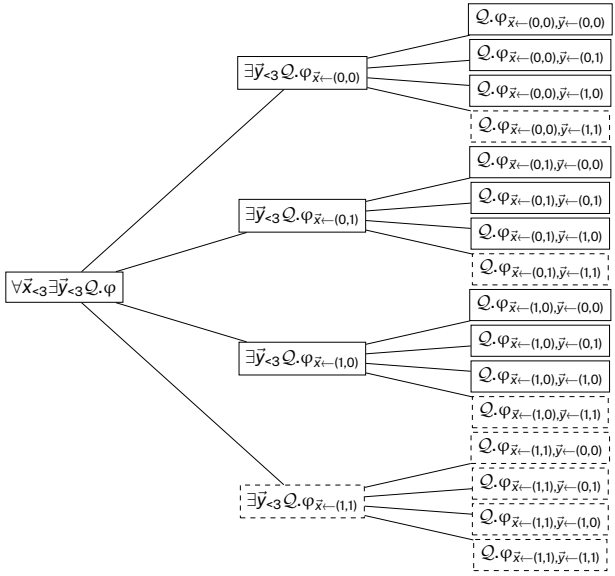
Int-Split Semantics

$\phi = Q\vec{x}_c \mathcal{Q}.\varphi$
ϕ is true iff $\nu(\phi) = 1$, with the interpretation function ν defined as follows.

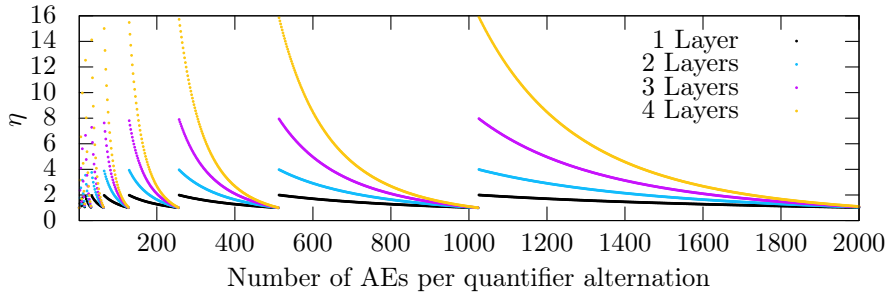
$\nu(\exists \vec{x}_\top \mathcal{Q}.\varphi) = 1$ iff $\sum_{\sigma \in \mathcal{S}(\vec{x})} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) \geq 1$
$\nu(\forall \vec{x}_\top \mathcal{Q}.\varphi) = 1$ iff $\prod_{\sigma \in \mathcal{S}(\vec{x})} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) = 1$

$\nu(\exists \vec{x}_{\diamondsuit V} \mathcal{Q}.\varphi) = 1$ iff $\sum_{\sigma \in \mathcal{S}(\vec{x}), \langle \sigma(\vec{x}) \rangle \diamondsuit V} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) \geq 1$ with $\diamondsuit \in \{<, >\}$
$\nu(\forall \vec{x}_{\diamondsuit V} \mathcal{Q}.\varphi) = 1$ iff $\prod_{\sigma \in \mathcal{S}(\vec{x}), \langle \sigma(\vec{x}) \rangle \diamondsuit V} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) = 1$ with $\diamondsuit \in \{<, >\}$

$\nu(\exists \vec{x}_{=V} \mathcal{Q}.\varphi) = 1$ iff $\sum_{\sigma \in \mathcal{S}(\vec{x}), \sigma(\vec{x}) \in V} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) \geq 1$
$\nu(\forall \vec{x}_{=V} \mathcal{Q}.\varphi) = 1$ iff $\prod_{\sigma \in \mathcal{S}(\vec{x}), \sigma(\vec{x}) \in V} \nu(\mathcal{Q}.\varphi_{\sigma(\vec{x})}) = 1$


$\forall \vec{x}_{<3} \exists \vec{y}_{<3} \mathcal{Q}.\varphi$
$\exists \vec{y}_{<3} \mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,0)}$
$\exists \vec{y}_{<3} \mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,1)}$
$\exists \vec{y}_{<3} \mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,0)}$
$\exists \vec{y}_{<3} \mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,0),\vec{y}\leftarrow(0,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,0),\vec{y}\leftarrow(0,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,0),\vec{y}\leftarrow(1,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,0),\vec{y}\leftarrow(1,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,1),\vec{y}\leftarrow(0,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,1),\vec{y}\leftarrow(0,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,1),\vec{y}\leftarrow(1,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(0,1),\vec{y}\leftarrow(1,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,0),\vec{y}\leftarrow(0,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,0),\vec{y}\leftarrow(0,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,0),\vec{y}\leftarrow(1,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,0),\vec{y}\leftarrow(1,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,1),\vec{y}\leftarrow(0,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,1),\vec{y}\leftarrow(0,1)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,1),\vec{y}\leftarrow(1,0)}$
$\mathcal{Q}.\varphi_{\vec{x}\leftarrow(1,1),\vec{y}\leftarrow(1,1)}$

Int-Split Potential



Int-split efficiency $\eta = \frac{\text{Unaccounted Expansions}}{\text{Accounted Expansions}}$ with varying number of AEs and quantifier alternations.

QDIMACS Syntax Extension

Preamble and Int-Splits Entry-Point

⟨*input*⟩ ::= ⟨*preamble*⟩ [⟨*prefix*⟩] ⟨*matrix*⟩ EOF

⟨*preamble*⟩ ::= ⟨*comment or split*⟩* ⟨*problem line*⟩

⟨*comment or split*⟩ ::= ‘c’ ⟨*text*⟩ EOL
| ‘s’ ⟨*split*⟩ EOL *explicit change of formula semantics*

⟨*problem line*⟩ ::= ‘p cnf’ ⟨*pnum*⟩ ⟨*pnum*⟩ EOL

QDIMACS Syntax Extension

Our Int-Splits Syntax Extension

⟨*split*⟩ ::= int [⟨*int-split-vars*⟩] ⟨*int-split-list*⟩

⟨*int-split-vars*⟩ ::= ‘[’ ⟨*pnum*⟩+ ‘]’

⟨*int-split-list*⟩ ::= ⟨*int-split*⟩ [‘;’ ⟨*int-split-list*⟩]

⟨*int-split*⟩ ::= ‘<’ ⟨*pnum*⟩
| ‘>’ ⟨*pnum*⟩
| ‘=’ ‘{’ ⟨*bit pattern*⟩+ ‘}’

Selected Benchmarks with Int-Splits as Divide-and-Conquer Guide (Sequential Solver: Caqe)

Formula	#v	#c	s	u	$\#_{w/}$	η	$t_{w/}$ [h]	$t_{w/o}$ [h]	$\frac{t_{w/}}{t_{w/o}}$	$\frac{t_{seq}}{t_{par}}$
Hex 10 5x5-13	1582	5531	19	10	361	0.68	4.51	8.49	0.53	47.7
Hex 02 5x5-13	1570	5437	19	10	361	0.68	8.33	14.7	0.57	32.9
Hex 16 5x5-13	1550	5437	18	10	324	0.78	7.17	10.7	0.67	39.1
GTTT Domino	943	3075	25	10	625	0.28	0.144	0.189	0.76	11.6
GTTT El	1180	3951	25	10	625	0.28	0.265	0.320	0.83	10.4
GTTT Elly	1465	5024	25	10	625	0.28	14.6	17.4	0.84	8.59
GTTT Tic	1155	3848	25	10	625	0.28	0.612	0.726	0.84	9.47
GTTT Tippy	1409	4799	25	10	625	0.28	69.2	77.4	0.89	7.72
Hex 05 6x6-13	1812	5437	31	10	961	0.032	73.4	78.6	0.93	2.47

Ongoing: Verifying Correctness

Can we verify that adding an int-split does not change the interpretation of a formula?

Ongoing: Verifying Correctness

Can we verify that adding an int-split does not change the interpretation of a formula?

Conflict between what formula authors want to express and extracting hints for solvers.

Ongoing: Verifying Correctness

Can we verify that adding an int-split does not change the interpretation of a formula?

Conflict between what formula authors want to express and extracting hints for solvers.

Better yet: Fully commit to changed formula semantics and extend quantifiers to support int-splits?

Ongoing: Automatic Extraction & Direct Solver Support

Can we extract int-splits from existing problems? And is that even required?

Ongoing: Automatic Extraction & Direct Solver Support

Can we extract int-splits from existing problems? And is that even required?

Int-splits reduce problem size! But also extend QBF...which may not be ideal.

Ongoing: Automatic Extraction & Direct Solver Support

Can we extract int-splits from existing problems? And is that even required?

Int-splits reduce problem size! But also extend QBF...which may not be ideal.

Produce solvers with int-split support and expand constraints for solvers without int-split support.

Reference Implementation of Int-Split Formula Splitter

MIT-Licensed at:

https://github.com/maximaximal/qdimacs-splitter

JOHANNES KEPLER
UNIVERSITY LINZ