Constraint Normalization and Parameterized Caching for Quantitative Program Analysis, por

Tegan Brennan, Nestan Tsiskaridze, Nicolás Rosner, Abdulbaki Aydin, y Tevfik Bultan

LIA significa Linear Integer Arithmetic

Guillermo Mosse

Contexto y Motivación

Ejemplo de constraint: $\{(x, y) \in \mathbb{Z}^2 : y = 2 \wedge 4 \geqslant x \geqslant y\}$

Otro ejemplo:
$\{(a, b, e) : a = \text{"@"} \wedge b = \text{".com"} \wedge \mathit{suffix\_of}\,(b, e) \wedge \mathit{contains}(a, e)\}$

Pueden aparecer como path constraints al hacer symbolic execution de programas que manipulen strings, como por ejemplo:

- Generadores de queries para bases de datos
- Generadores dinámicos de código
- Validadores de inputs en aplicaciones web

Contexto y Motivación

Ejemplo de constraint: $\{(x, y) \in \mathbb{Z}^2 : y = 2 \wedge 4 \geqslant x \geqslant y\}$

Otro ejemplo:
$\{(a, b, e) : a = \text{"@"} \wedge b = \text{".com"} \wedge \mathit{suffix\_of}\,(b, e) \wedge \mathit{contains}(a, e)\}$

Pueden aparecer como path constraints al hacer symbolic execution de programas que manipulen strings, como por ejemplo:

- Generadores de queries para bases de datos
- Generadores dinámicos de código
- Validadores de inputs en aplicaciones web

¿Qué se puede querer al trabajar con constraints?

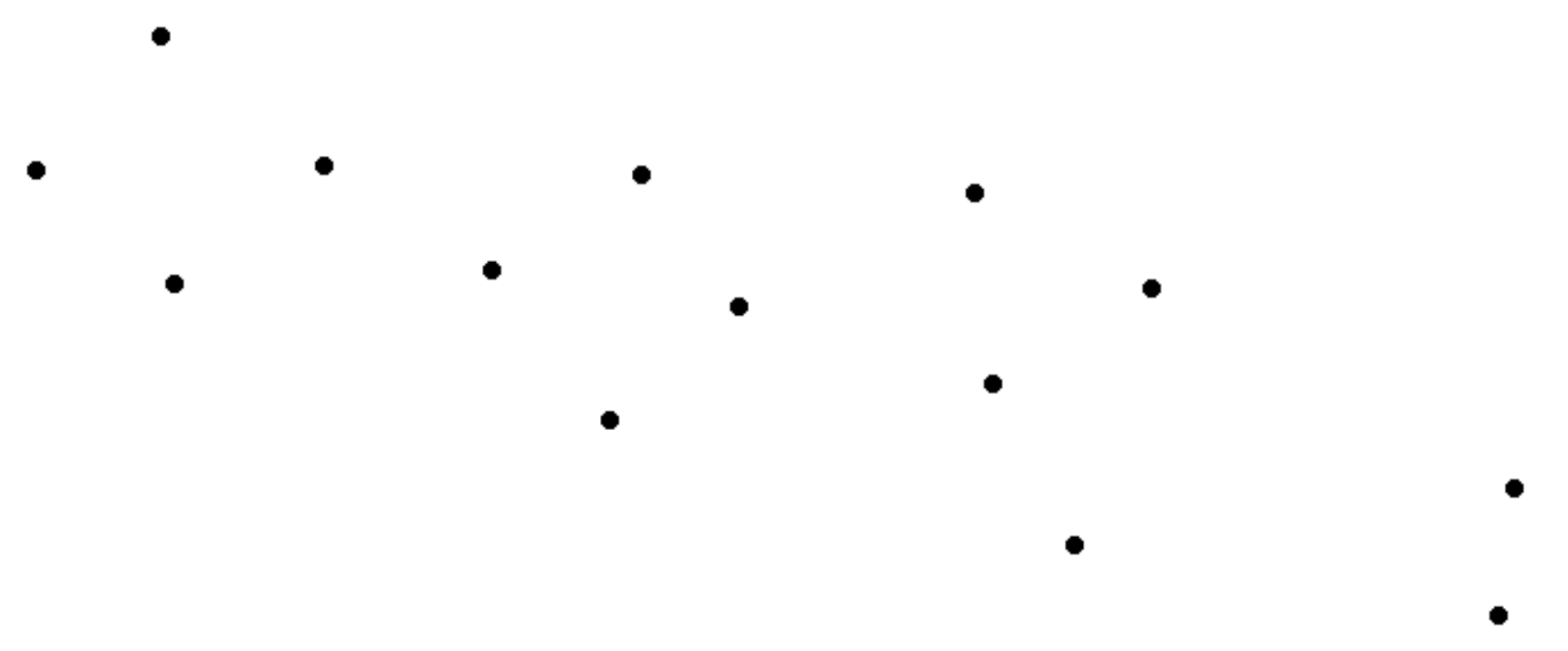
- Satisfiability.
- Solution set.
- Model counting (y bounded model counting).

¡Todas son muy costosas! (¿Satisfiability también?)

Contribuciones del paper

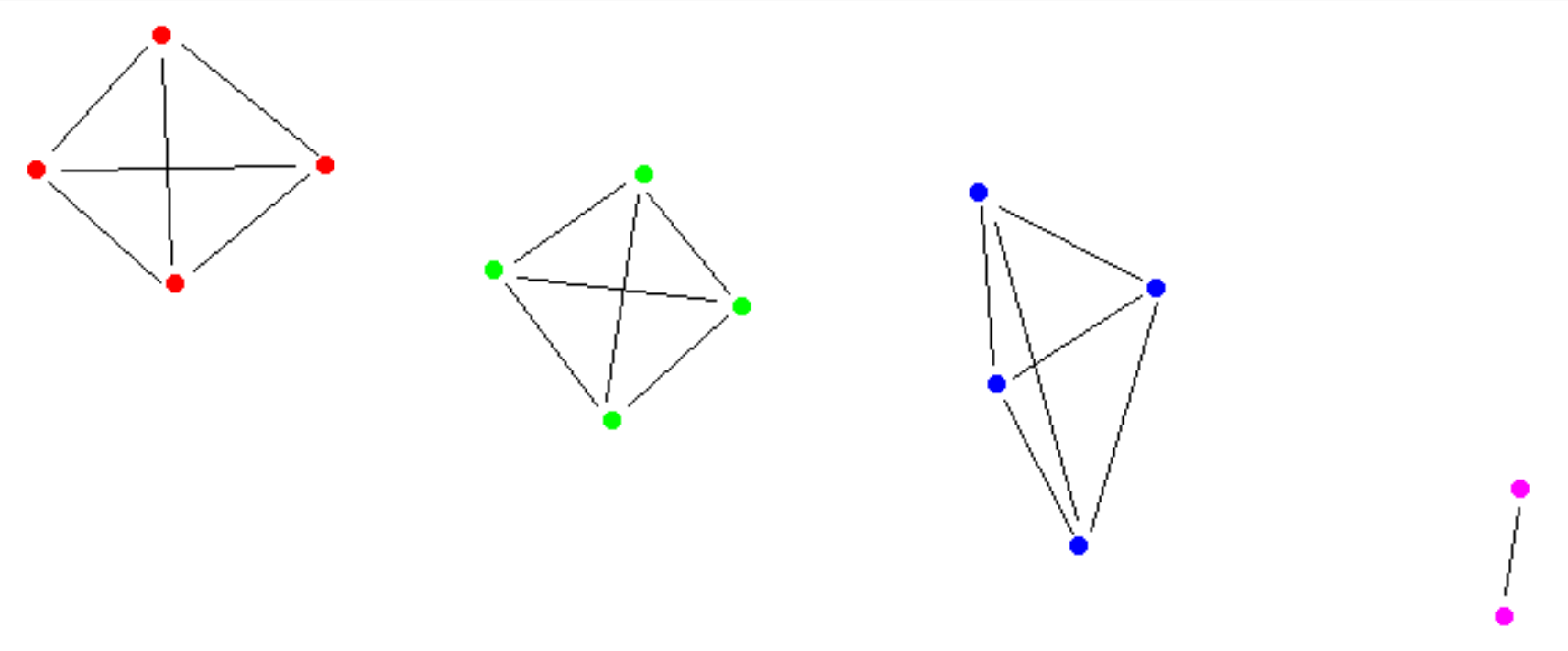
- Se dieron cuenta de que en model counting dos constraints escritas de manera distinta pueden ser equivalentes, i.e., expresan lo mismo.
- Diseñaron e implementaron una manera poco costosa de ver cuándo dos constraints son equivalentes: ver si sus formas normales son iguales
- Implementaron el algoritmo que computa la forma normal
- Hicieron un experimento cacheando soluciones y mostraron que mejora los tiempos.

La idea del paper: hecho con KolourPaint



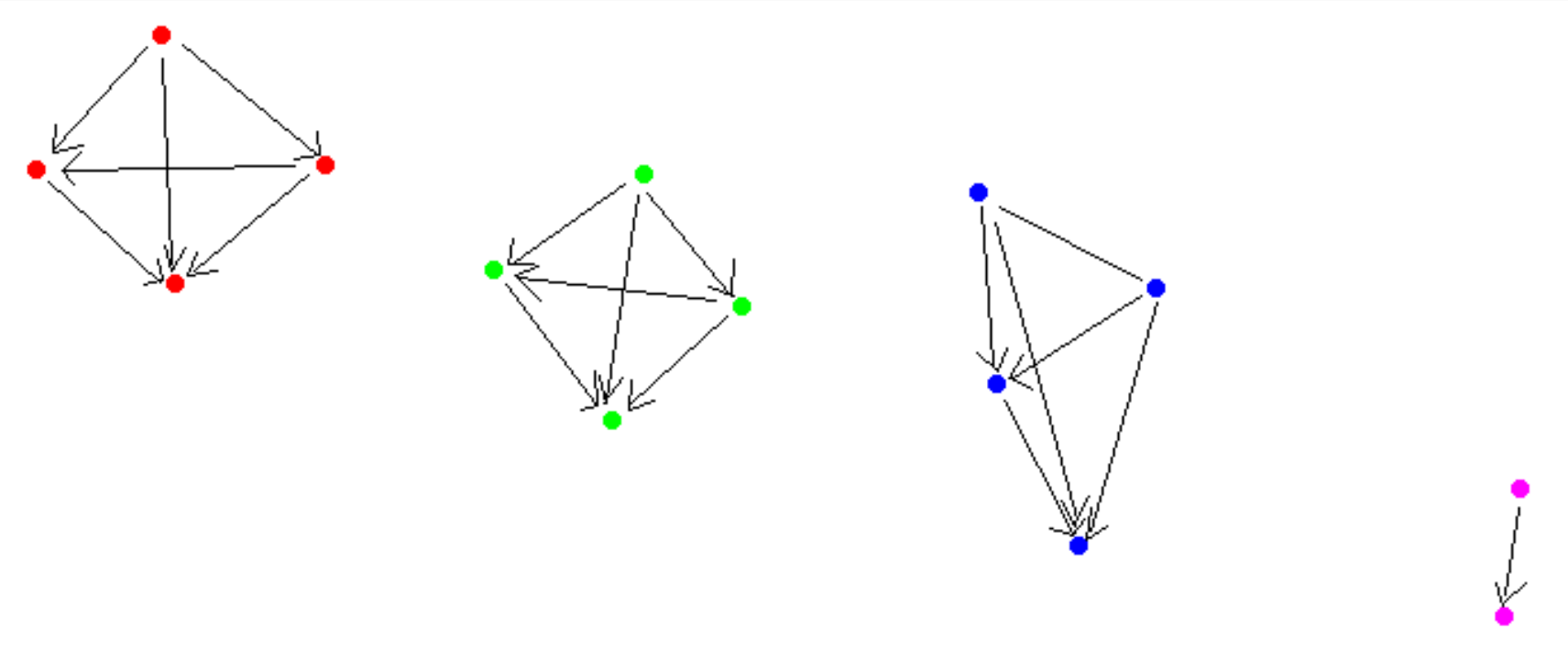
Mi universo de constraints
(tengan imaginación)

La idea del paper



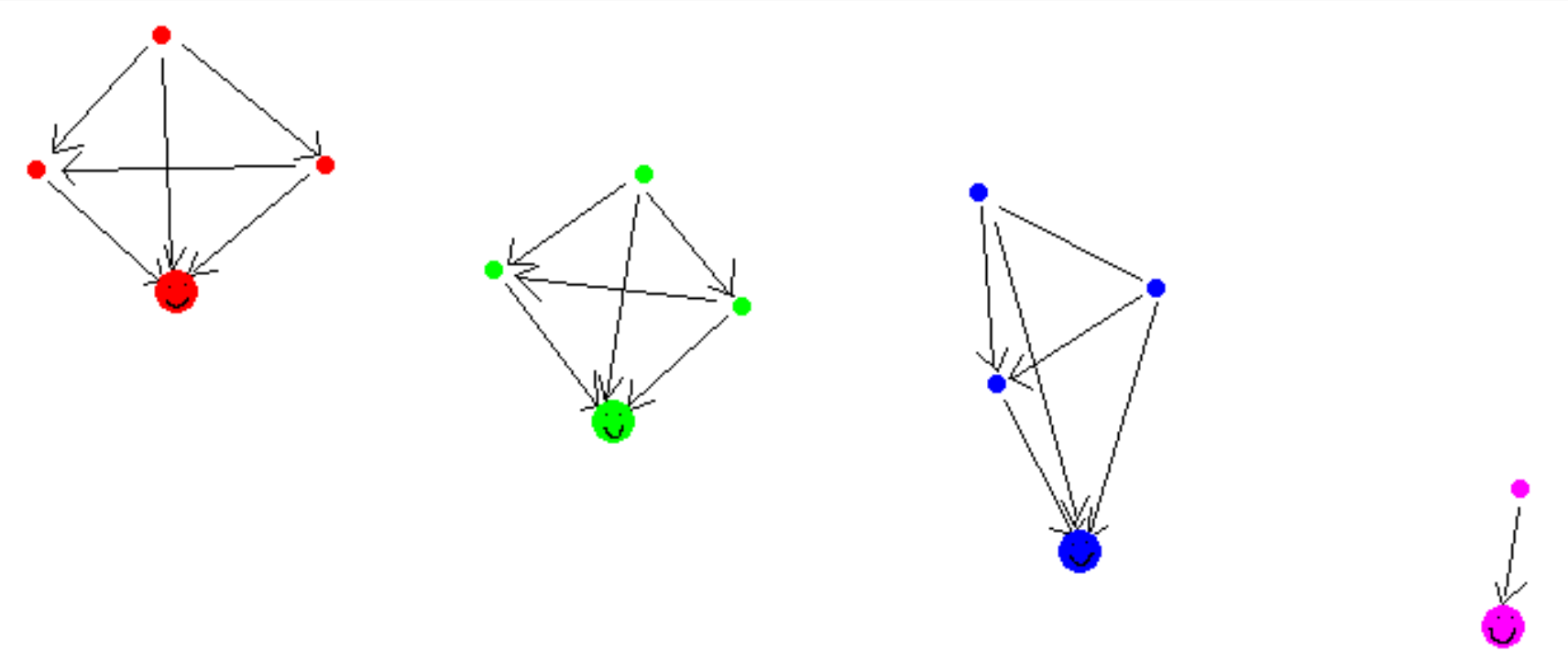
Defino cuándo son equivalentes (pero es muy costoso responder cuando una constraint es equivalente a otra)

La idea del paper



Es más fácil trabajar con una relación de orden
(¡puedo navegar por el grafo más fácil!)

La idea del paper



La forma normal es bajar hasta no poder más, y es única
Para comparar dos constraints computo su forma normal y me fijo si son iguales

¿Cómo son las constraints?

$$
\begin{aligned}
\mathcal{T}_S &:= c \mid v_S \mid \mathcal{T}_S \cdot \mathcal{T}_S \mid \text{char\_at}(\mathcal{T}_S, \mathcal{T}_A) \mid \text{int\_to\_str}(\mathcal{T}_A) \mid \\
&\quad \text{replace}(\mathcal{T}_S, \mathcal{T}_S, \mathcal{T}_S) \mid \text{substr}(\mathcal{T}_S, \mathcal{T}_A, \mathcal{T}_A) \\
\mathcal{T}_R &:= \epsilon \mid s \mid (\mathcal{T}_R) \mid \mathcal{T}_R \cdot \mathcal{T}_R \mid \mathcal{T}_R | \mathcal{T}_R \mid \mathcal{T}_R^* \\
\mathcal{T}_A &:= n \mid v_A \mid -\mathcal{T}_A \mid (\mathcal{T}_A) \mid \mathcal{T}_A + \mathcal{T}_A \mid \mathcal{T}_A - \mathcal{T}_A \mid \mathcal{T}_A \times n \mid \\
&\quad n \times \mathcal{T}_A \mid |\mathcal{T}_S| \mid \text{index\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{str\_to\_int}(\mathcal{T}_S)
\end{aligned}
$$

Definimos los términos de tipo String, Reg Exp y Aritmético

¿Cómo son las constraints?

$$
\begin{aligned}
L &:= \top \mid \bot \mid S \mid R \mid A \\
S &:= \mathcal{T}_S = \mathcal{T}_S \mid \mathcal{T}_S \neq \mathcal{T}_S \mid \text{contains}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{prefix\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \\
&\quad \text{suffix\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{not\_contains}(\mathcal{T}_S, \mathcal{T}_S) \mid \\
&\quad \text{not\_prefix\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{not\_suffix\_of}(\mathcal{T}_S, \mathcal{T}_S) \\
R &:= \mathcal{T}_S \in \mathcal{T}_R \mid \mathcal{T}_S \notin \mathcal{T}_R \\
A &:= \mathcal{T}_A = \mathcal{T}_A \mid \mathcal{T}_A < \mathcal{T}_A \mid \mathcal{T}_A \leq \mathcal{T}_A \mid \mathcal{T}_A \neq \mathcal{T}_A
\end{aligned}
$$

El lenguaje L tiene conjuncts de tipo String, Reg Exp, y Aritméticos.

¿Cómo son las constraints?

$$
\begin{aligned}
\mathcal{T}_S &:= c \mid v_S \mid \mathcal{T}_S \cdot \mathcal{T}_S \mid \text{char\_at}(\mathcal{T}_S, \mathcal{T}_A) \mid \text{int\_to\_str}(\mathcal{T}_A) \mid \\
&\quad \text{replace}(\mathcal{T}_S, \mathcal{T}_S, \mathcal{T}_S) \mid \text{substr}(\mathcal{T}_S, \mathcal{T}_A, \mathcal{T}_A) \\
\mathcal{T}_R &:= \epsilon \mid s \mid (\mathcal{T}_R) \mid \mathcal{T}_R \cdot \mathcal{T}_R \mid \mathcal{T}_R | \mathcal{T}_R \mid \mathcal{T}_R^* \\
\mathcal{T}_A &:= n \mid v_A \mid -\mathcal{T}_A \mid (\mathcal{T}_A) \mid \mathcal{T}_A + \mathcal{T}_A \mid \mathcal{T}_A - \mathcal{T}_A \mid \mathcal{T}_A \times n \mid \\
&\quad n \times \mathcal{T}_A \mid |\mathcal{T}_S| \mid \text{index\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{str\_to\_int}(\mathcal{T}_S)
\end{aligned}
$$

$$
\begin{aligned}
L &:= \top \mid \bot \mid S \mid R \mid A \\
S &:= \mathcal{T}_S = \mathcal{T}_S \mid \mathcal{T}_S \neq \mathcal{T}_S \mid \text{contains}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{prefix\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \\
&\quad \text{suffix\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{not\_contains}(\mathcal{T}_S, \mathcal{T}_S) \mid \\
&\quad \text{not\_prefix\_of}(\mathcal{T}_S, \mathcal{T}_S) \mid \text{not\_suffix\_of}(\mathcal{T}_S, \mathcal{T}_S) \\
R &:= \mathcal{T}_S \in \mathcal{T}_R \mid \mathcal{T}_S \notin \mathcal{T}_R \\
A &:= \mathcal{T}_A = \mathcal{T}_A \mid \mathcal{T}_A < \mathcal{T}_A \mid \mathcal{T}_A \leq \mathcal{T}_A \mid \mathcal{T}_A \neq \mathcal{T}_A
\end{aligned}
$$

Bounded model counting

Una aclaración: algo que no incluye lo anterior es que podemos pedir que las constraints vengan con un bound b que fije un máximo para el tamaño de las soluciones (sean ints o strings). Por ahora ignorémoslo.

¿Cuándo dos constraints son equivalentes?

Ejemplo:

$$c_1 := \{(x, y) \in \mathbb{Z}^2 : y = 2 \wedge y \leqslant x \leqslant 4\}$$

Soluciones: $\{(2, 2); (2, 3); (2, 4)\}$

es equivalente a:

$$c_1 := \{(x, y) \in \mathbb{Z}^2 : y = 3 \wedge y \leqslant x \leqslant 5\}$$

Soluciones: $\{(3, 3); (3, 4); (3, 5)\}$

porque ambas tienen 3 soluciones...?

¿Cuándo dos constraints son equivalentes?

Ejemplo:

$$c_1 := \{(x, y) \in \mathbb{Z}^2 : y = 2 \wedge y \leqslant x \ \leqslant 4\}$$

Soluciones: $\{(2, 2); (2, 3); (2, 4)\}$

es equivalente a:

$$c_1 := \{(x, y) \in \mathbb{Z}^2 : y = 3 \wedge y \leqslant x \leqslant 5\}$$

Soluciones: $\{(3, 3); (3, 4); (3, 5)\}$

porque ambas tienen 3 soluciones...?

Y qué pasa con:

$c_3 := \{(x, y) \in \mathbb{Z}^2 : 0 <= x <= 1 \wedge y = 3\}$
$c_4 := \{(x) \in \mathbb{Z}^2 : 0 <= x <= 1\}$

¿Cuándo dos constraints son equivalentes?

Ejemplo:

$$c_1 := \{(x, y) \in \mathbb{Z}^2 : y = 2 \wedge y \leqslant x \ \leqslant 4\}$$

Soluciones: $\{(2, 2); (2, 3); (2, 4)\}$

es equivalente a:

$$c_1 := \{(x, y) \in \mathbb{Z}^2 : y = 3 \wedge y \leqslant x \leqslant 5\}$$

Soluciones: $\{(3, 3); (3, 4); (3, 5)\}$

porque ambas tienen 3 soluciones...?

Y qué pasa con:

$c_3 := \{(x, y) \in \mathbb{Z}^2 : 0 <= x <= 1 \wedge y = 3\}$
$c_4 := \{(x) \in \mathbb{Z}^2 : 0 <= x <= 1\}$

Definir así la relación de equivalencia es muy poco manejable. ¿Por qué?

¿Cuándo dos constraints son equivalentes?

Dos constraints van a ser equivalentes cuando pueda llegar de una a la otra.

¿De qué manera?

Por medio de unas "transformaciones de constraints" que **siempre** mantienen la cardinalidad de las soluciones, **tengo definidas explícitamente**, y representan una transformación **sintáctica**.

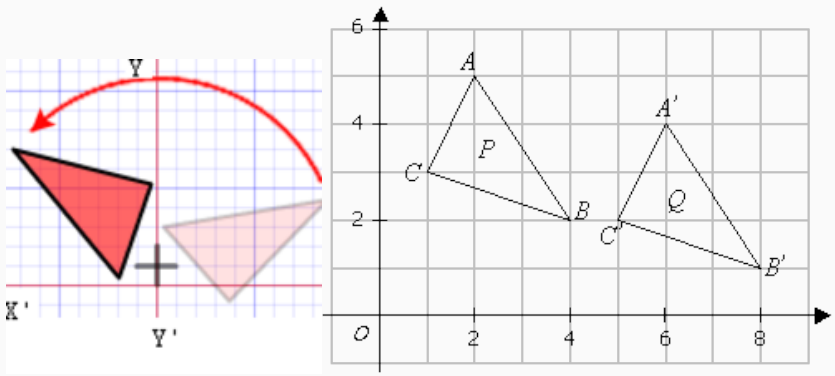
¿Qué transformaciones de constraints siempre mantienen la cardinalidad?

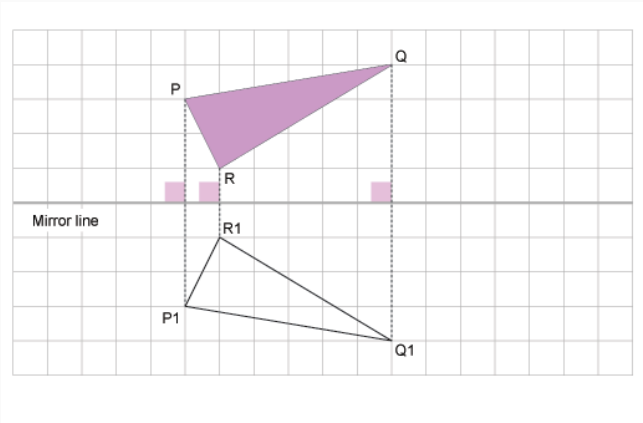
Transformaciones de constraints: construyamos $\mathbb{G}_{card}$

- Puedo cambiar el nombre de las variables para strings e ints.
- Puedo cambiar el orden de las condiciones separadas por ANDs.
- Puedo permutar los elementos del alfabeto Σ que se usa para los strings.

Transformaciones de constraints: construyamos $\mathbb{G}_{card}$

- Puedo cambiar el nombre de las variables para strings e ints.
- Puedo cambiar el orden de las condiciones separadas por ANDs.
- Puedo permutar los elementos del alfabeto Σ que se usa para los strings.
- Si veo a una constraint como un **conjunto de puntos en el espacio**...
  ¿rotaciones? ¿traslaciones? ¿simetrías?

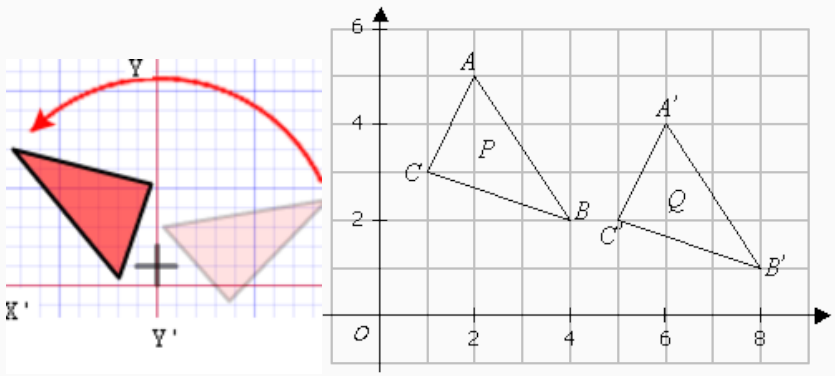


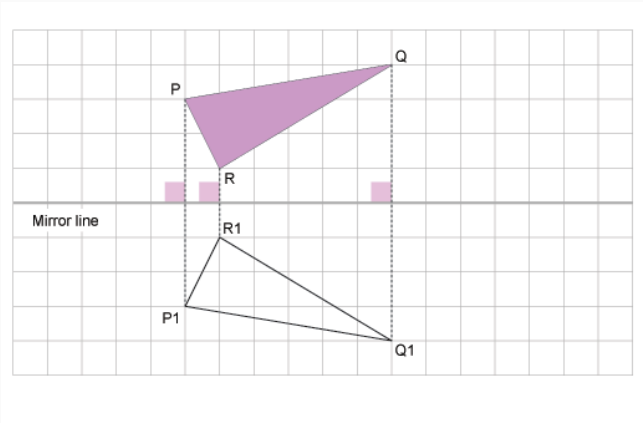


Transformaciones de constraints: construyamos $\mathbb{G}_{card}$

- Puedo cambiar el nombre de las variables para strings e ints.
- Puedo cambiar el orden de las condiciones separadas por ANDs.
- Puedo permutar los elementos del alfabeto Σ que se usa para los strings.
- Si veo a una constraint como un **conjunto de puntos en el espacio**...

  ¿rotaciones? ¿traslaciones? ¿simetrías?





¿Cuántas transformaciones hay?

¿Estas transformaciones definen la misma relación de equivalencia que la primera idea de "dos conjuntos son equivalentes si tienen la misma cantidad de soluciones"?

¡Es un grupo!

¡Las transformaciones forman un grupo!

O sea:

- Hay una operación que se le puede hacer a dos transformaciones: **componerlas**
- Hay un elemento que es "el neutro": la identidad (dejar las constraints tal como están)
- Toda transformación tiene un inverso: "deshacer el cambio".

Esperen, ¡aun hay más!

Más aun, es un producto cartesiano de 4 grupos más manejables todavía.

Dada $g \in \mathbb{G}_{card}$, ésta se puede identificar con una 4-upla formada por:

- una permutación de nombres de variables.
- una permutación de las condiciones separadas por ANDs.
- una permutación de los elementos del alfabeto Σ.
- una traslación de la constraint:(sumar $k \in \mathbb{Z}$ a todas las constantes a la vez).

Digresión sobre grupos

Si C es un conjunto y tengo un grupo G que actúa sobre ese conjunto (en el sentido de antes: dado $x \in C$ y $g \in G$ puedo hacer $g(x) = y \in C$), se puede estudiar la relación de equivalencia que queda definida.

Esto es algo muy hecho en matemática y al estudiar **cómo actúa un grupo sobre un espacio** uno puede obtener **información útil** sobre ese espacio. Es más: **así es como empezó la teoría de grupos**; surgieron con los grupos de permutación, usados por Lagrange en su trabajo sobre raíces de polinomios.

Digresión sobre relaciones de equivalencia

Dada una relación de equivalencia $\sim$ sobre un conjunto C que puede o no haber venido por un grupo, dados $x, y \in C$ es en general indecidible responder si $x \sim y$. (O muy costoso si la relación es finita). Este paper encuentra una manera de resolver el problema, ¡pero no siempre se puede hacer eso!

Digresión sobre relaciones de equivalencia

Dada una relación de equivalencia $\sim$ sobre un conjunto C que puede o no haber venido por un grupo, dados $x, y \in C$ es en general indecidible responder si $x \sim y$. (O muy costoso si la relación es finita). Este paper encuentra una manera de resolver el problema, ¡pero no siempre se puede hacer eso!

Un ejemplo indecidible "simple"

$$
\begin{array}{llll}
\langle & a,b,c,d,e,p,q,r,t,k & | & \\
& p^{10}a = ap, & pacqr = rpcaq, & ra = ar, \\
& p^{10}b = bp, & p^2adq^2r = rp^2daq^2, & rb = br, \\
& p^{10}c = cp, & p^3bcq^3r = rp^3cbq^3, & rc = cr, \\
& p^{10}d = dp, & p^4bdq^4r = rp^4dbq^4, & rd = dr, \\
& p^{10}e = ep, & p^5ceq^5r = rp^5ecaq^5, & re = er, \\
& aq^{10} = qa, & p^6deq^6r = rp^6edbq^6, & pt = tp, \\
& bq^{10} = qb, & p^7cdcq^7r = rp^7cdceq^7, & qt = tq, \\
& cq^{10} = qc, & p^8ca^3q^8r = rp^8a^3q^8, & \\
& dq^{10} = qd, & p^9da^3q^9r = rp^9a^3q^9, & \\
& eq^{10} = qe, & a^{-3}ta^3k = ka^{-3}ta^3 & \rangle
\end{array}
$$

(donde las igualdades "generan" la relación de equivalencia) el word problem es indecidible.

Ahora la relación de orden

¿Cómo transformamos la relación de equivalencia en una relación de orden?

Ahora la relación de orden

¿Cómo transformamos la relación de equivalencia en una relación de orden? Ordenando constraints de manera lexicográfica...o casi.

Ahora la relación de orden

¿Cómo transformamos la relación de equivalencia en una relación de orden? Ordenando constraints de manera lexicográfica...o casi.

Algorithm 2 C-LessThan(C_1, C_2): Conjunct Comparison

Input: Two conjuncts $C_1, C_2 \in \mathbb{L}$
Output: True if $C_1 \prec C_2$, otherwise False

```
for each f ∈ [Type, ‖ ‖, #Var, Op, VI, Int, ∀, Σ, Shift] do
    if f(C1) ≺ f(C2) then
        return True
    end if
end for
return False
```

Dos constraints son comparadas primero por la cantidad de conjuncts que tienen, y si el número es igual, por el primer conjunct de una que es menor que la otra.

La relación de orden: algunos ejemplos

$[\textsc{Type}, \| \|, \#Var, Op, VI, Int, \mathbf{V}, \Sigma, Shift]$

$\{a : a < 1\}$ vs $\{a : a < 1 \wedge a > 2\}$

$\{a : a < 1\}$ vs $\{c : c < 1\}$

$\{a : a < 5\}$ vs $\{c : c < 1\}$

$\{(a, b) : a < 2 \wedge b < 4\}$ vs $\{(a, b) : a < 2 \wedge b \leqslant 1\}$

$\{(s, b) : s = \text{"rana"} \wedge b = 4\}$ vs $\{(s, b) : s = \text{"rama"} \wedge b = 1\}$

La relación de orden: algunos ejemplos

$[\textsc{Type}, \| \|, \#Var, Op, VI, Int, \mathbf{V}, \Sigma, Shift]$

$\{a : a < 1\} \quad < \{a : a < 1 \wedge a > 2\}$

$\{a : a < 1\}$ vs $\{c : c < 1\}$

$\{a : a < 5\}$ vs $\{c : c < 1\}$

$\{(a, b) : a < 2 \wedge b < 4\}$ vs $\{(a, b) : a < 2 \wedge b \leqslant 1\}$

$\{(s, b) : s = \text{"rana"} \wedge b = 4\}$ vs $\{(s, b) : s = \text{"rama"} \wedge b = 1\}$

La relación de orden: algunos ejemplos

$[\textsc{Type}, \| \|, \#Var, Op, VI, Int, \mathbb{V}, \Sigma, Shift]$

$\{a : a < 1\} \quad < \{a : a < 1 \wedge a > 2\}$

$\{a : a < 1\} \quad < \{c : c < 1\}$

$\{a : a < 5\}$ vs $\{c : c < 1\}$

$\{(a, b) : a < 2 \wedge b < 4\}$ vs $\{(a, b) : a < 2 \wedge b \leqslant 1\}$

$\{(s, b) : s = \text{"rana"} \wedge b = 4\}$ vs $\{(s, b) : s = \text{"rama"} \wedge b = 1\}$

La relación de orden: algunos ejemplos

$[\textsc{Type}, \| \|, \#Var, Op, VI, Int, \mathbb{V}, \Sigma, Shift]$

$\{a : a < 1\} \quad < \{a : a < 1 \wedge a > 2\}$

$\{a : a < 1\} \quad < \{c : c < 1\}$

$\{a : a < 5\} \quad < \{c : c < 1\}$

$\{(a, b) : a < 2 \wedge b < 4\}$ vs $\quad \{(a, b) : a < 2 \wedge b \leqslant 1\}$

$\{(s, b) : s = \text{"rana"} \wedge b = 4\}$ vs $\quad \{(s, b) : s = \text{"rama"} \wedge b = 1\}$

La relación de orden: algunos ejemplos

$[\textsc{Type}, \| \|, \#Var, Op, VI, Int, \mathbb{V}, \Sigma, Shift]$

$\{a : a < 1\} \quad < \{a : a < 1 \wedge a > 2\}$

$\{a : a < 1\} \quad < \{c : c < 1\}$

$\{a : a < 5\} \quad < \{c : c < 1\}$

$\{(a, b) : a < 2 \wedge b < 4\} \quad < \{(a, b) : a < 2 \wedge b \leqslant 1\}$

$\{(s, b) : s = \text{"rana"} \wedge b = 4\}$ vs $\{(s, b) : s = \text{"rama"} \wedge b = 1\}$

La relación de orden: algunos ejemplos

$$[\textsc{Type}, \| \|, \#Var, Op, VI, Int, \mathbf{V}, \Sigma, Shift]$$

$\{a : a < 1\} \quad < \{a : a < 1 \wedge a > 2\}$

$\{a : a < 1\} \quad < \{c : c < 1\}$

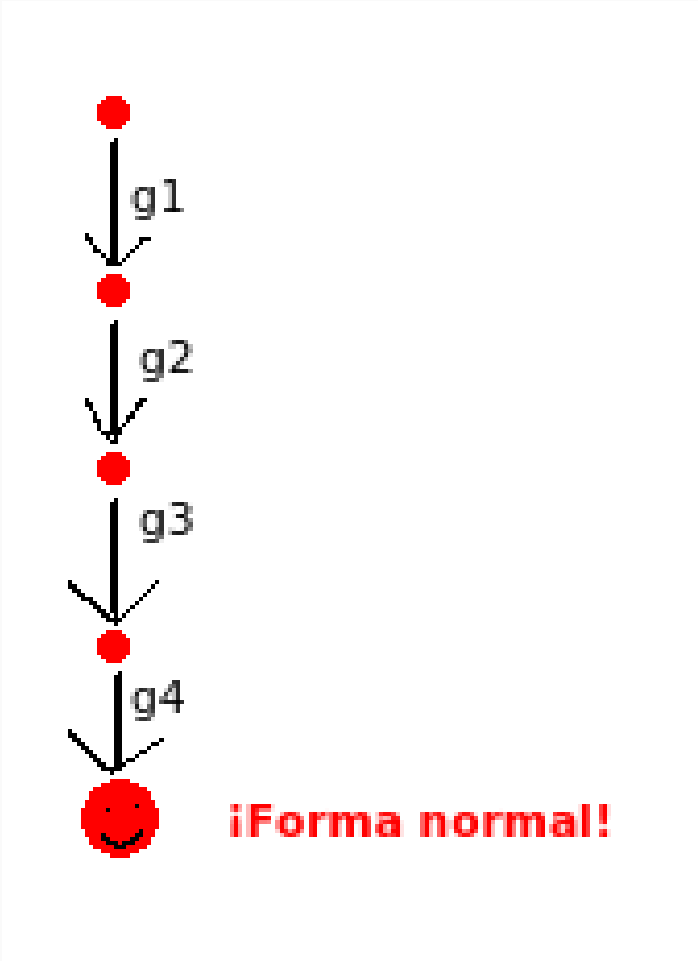
$\{a : a < 5\} \quad < \{c : c < 1\}$

$\{(a, b) : a < 2 \wedge b < 4\} \quad < \{(a, b) : a < 2 \wedge b \leqslant 1\}$

$\{(s, b) : s = \text{"rana"} \wedge b = 4\} \quad > \{(s, b) : s = \text{"rama"} \wedge b = 1\}$

El algoritmo

Una vez tenemos definida la relación de orden, podemos considerar solo las transformaciones g que a una constraint la "achican".



El algoritmo (II)

Algorithm 4 COMPLETE-NORMALIZATION(F)

Input: A constraint F
Output: The normalized form of F

1: $F_{min} := F$
2: **for each** permutation F' of conjuncts in F **do**
3: $\quad \sigma_{\mathbb{V}} := \text{MIN-}\sigma\text{-}\mathbb{V}(F')$
4: $\quad \sigma_{\Sigma} := \text{MIN-}\sigma\text{-}\Sigma(F')$
5: $\quad \sigma_{\mathcal{SH}} := \text{MIN-}\sigma\text{-}\mathcal{SH}(F')$
6: $\quad F' := \sigma_{\mathbb{V}} \circ \sigma_{\Sigma} \circ \sigma_{\mathcal{SH}}[F']$
7: $\quad$ **if** F-LESSTHAN(F', F_{min}) **then**
8: $\quad\quad F_{min} := F'$
9: $\quad$ **end if**
10: **end for**
11: **return** F_{min}

Observación: en la práctica, a partir de la constraint, computás el mínimo que podés obtener con las g de cada subgrupo discutido.

Pregunta: ¿Por qué recorren todas las permutaciones de conjuncts?

Uno más rápido pero que no devuelve forma normal

Algorithm 5 NORMALIZATION(F)

Input: A constraint F
Output: A semi-normal form of F

1: $F' :=$ **Permute** conjuncts of F according to Algorithm 2 up until $\forall$
2: $\sigma_{\forall} :=$ MIN-σ-$\forall(F')$
3: $\sigma_{\Sigma} :=$ MIN-σ-$\Sigma(F')$
4: $\sigma_{\mathcal{SH}} :=$ MIN-σ-$\mathcal{SH}(F')$
5: $[\![F]\!] := \sigma_{\forall} \circ \sigma_{\Sigma} \circ \sigma_{\mathcal{SH}}[F']$
6: **return** $[\![F]\!]$

O sea, according to:

$[\text{TYPE}, \| \|, \#Var, Op, VI, Int, \forall, \Sigma, Shift]$

Dos preguntas:

- ¿Por qué el output dice semi-normal form?
- ¿Por qué es más rápido?

The price we pay

Ejemplo:

$$c_1 = \{(a, b) : b = 4 \wedge 1 < a \wedge a < 2\}$$

$$c_2 = \{(a, b) : 1 < a \wedge a < 2 \wedge b = 4\}$$

The price we pay

Ejemplo:

$$c_1 = \{(a, b) : b = 4 \wedge 1 < a \wedge a < 2\}$$

Forma semi normal: $\{(a, b) : a = 0 \wedge -3 < b \wedge b < -2\}$

$$c_2 = \{(a, b) : 1 < a \wedge a < 2 \wedge b = 4\}$$

Forma semi normal: $\{(a, b) : 0 < a \wedge a < 1 \wedge b = 3\}$

The price we pay

Ejemplo:

$$c_1 = \{(a, b) : b = 4 \wedge 1 < a \wedge a < 2\}$$

Forma semi normal: $\{(a, b) : a = 0 \wedge -3 < b \wedge b < -2\}$

$$c_2 = \{(a, b) : 1 < a \wedge a < 2 \wedge b = 4\}$$

Forma semi normal: $\{(a, b) : 0 < a \wedge a < 1 \wedge b = 3\}$

También funcionan:

$c_3 = \{(a, b) : b = 4 \wedge a = 2\}$ y $c_4 = \{(a, b) : a = 2 \wedge b = 4\}$

(Tampoco tienen la misma forma normal)

Bounded model counting (8.1), bound fijo

Algorithm 6 NORMALIZE-QUERY(F, V, b)

Input: A query (F, V, b)
Output: A normalized query $[\![F, V, b]\!]$

1: $[\![F]\!] :=$ NORMALIZATION(F)
2: $\sigma :=$ the transformation used to normalize F
3: $[\![V]\!] := \sigma[V]$
4: $[\![b]\!] := \sigma[b]$
5: **return** $([\![F]\!], [\![V]\!], [\![b]\!])$

El experimento

Algunos comentarios/preguntas:

- Eliminaron los duplicados para poder medir el cacheo no trivial
- ¿En qué orden conviene que vengan las constraints?
- ¿Con qué dataset ayuda más Cashew? SMC-Big o SMC-Small?
- "El tiempo de procesamiento de una constraint aparentemente puede llegar a ser más corto después de normalizar". ¿Por qué?

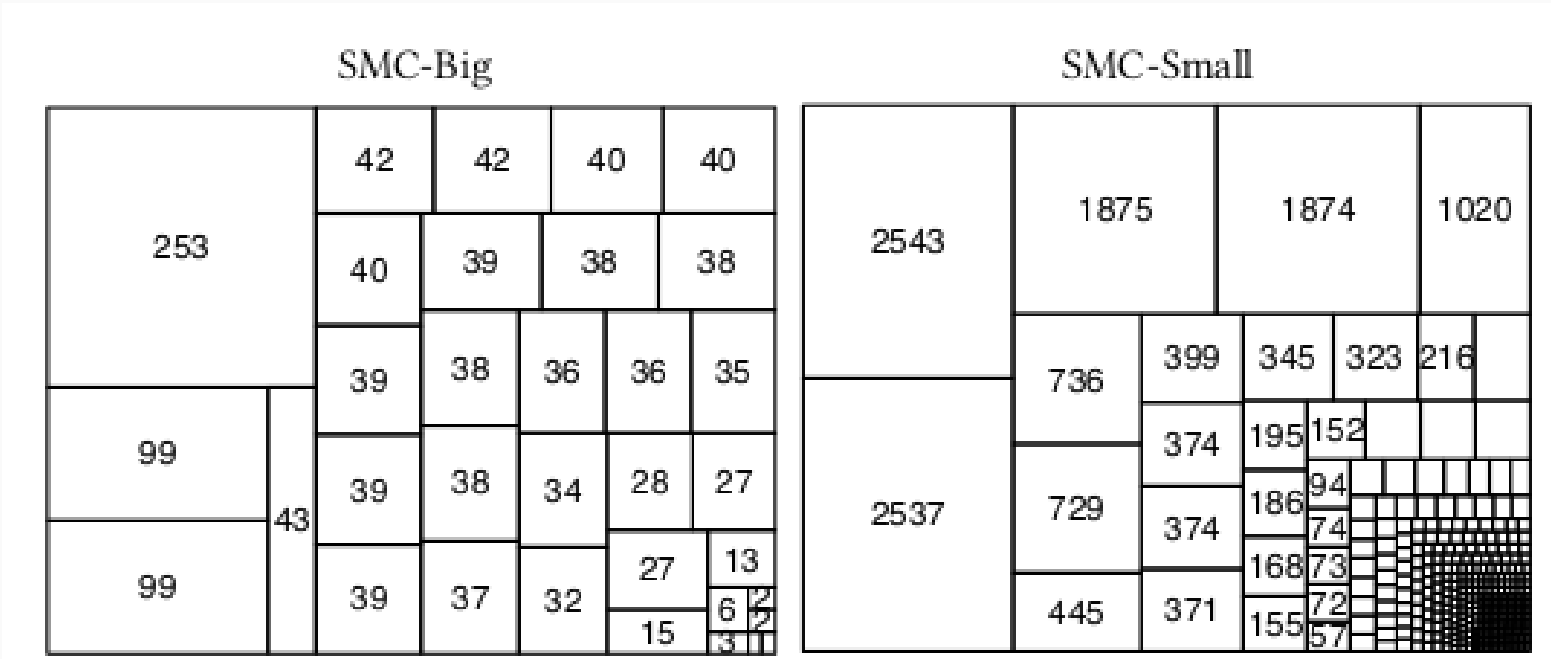


Figure 4: Orbit sizes for the original SMC datasets.

Parameterized Caching (8.3, el bound se mueve)

Algorithm 1 CONSTRAINT-CACHING(F, V, b):

Input: A query (F, V, b).
Output: The number of satisfying solutions of V in F under the length bound b.

1: $[\![F, V, b]\!]$ = NORMALIZE-QUERY(F, V, b)
2: **if** Hit on $[\![F, V, b]\!]$ **then**
3: **return** #(F,V,b)
4: **end if**
5: **if** Hit on $[\![F, V]\!]$ **then**
6: **Evaluate** the model-counter object for bound $[\![b]\!]$ using the model-counter evaluator;
7: **Store** the result under $[\![F, V, b]\!]$
8: **return** #(F,v,b)
9: **end if**
10: **Translate** $[\![F, V, b]\!]$ and **send** it to the selected model-counting solver
11: **Store** the returned model-counter object under $[\![F, V]\!]$
12: **Store** #(F,V,b) under $[\![F, V, b]\!]$
13: **return** #(F,V,b)

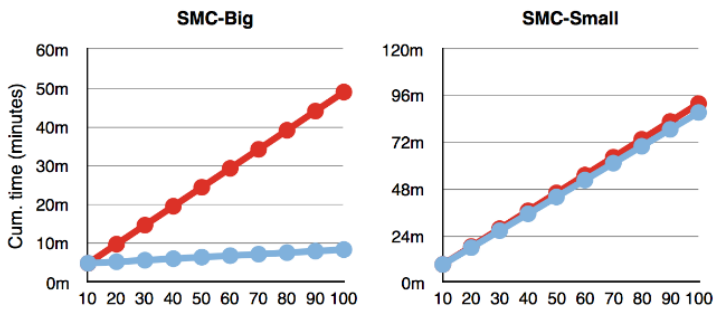


Figure 5: SMC datasets for increasing bounds. Cashew: non parameterized (red) and parameterized (blue) caching.

¿Y si queremos hacer otra cosa?

- ¿Qué pasa si necesitamos devolver el conjunto solución?
- ¿Qué pasa si queremos devolver solamente si existe solución?

Sólo satisfiability

Que la relación de equivalencia sea "mismo valor de verdad" es poco práctico

¿Qué transformaciones sintácticas mantienen el valor de verdad?

Sólo satisfiability

Que la relación de equivalencia sea "mismo valor de verdad" es poco práctico

¿Qué transformaciones sintácticas mantienen el valor de verdad?

Si $c = c_1 \wedge c_2$ es SAT, $c \rightarrow c_1$ (proyectar) mantiene la satisfacibilidad.

Si $\bar{c} = \bar{c_1} \vee \bar{c_2}$ es UNSAT, $\bar{c} \rightarrow \bar{c_1}$ (proyectar) mantiene la insatisfacibilidad.

...pero hay que tener en cuenta el valor de verdad de la fórmula y no queda claro cómo definir una forma normal.

Sólo satisfiability

Que la relación de equivalencia sea "mismo valor de verdad" es poco práctico

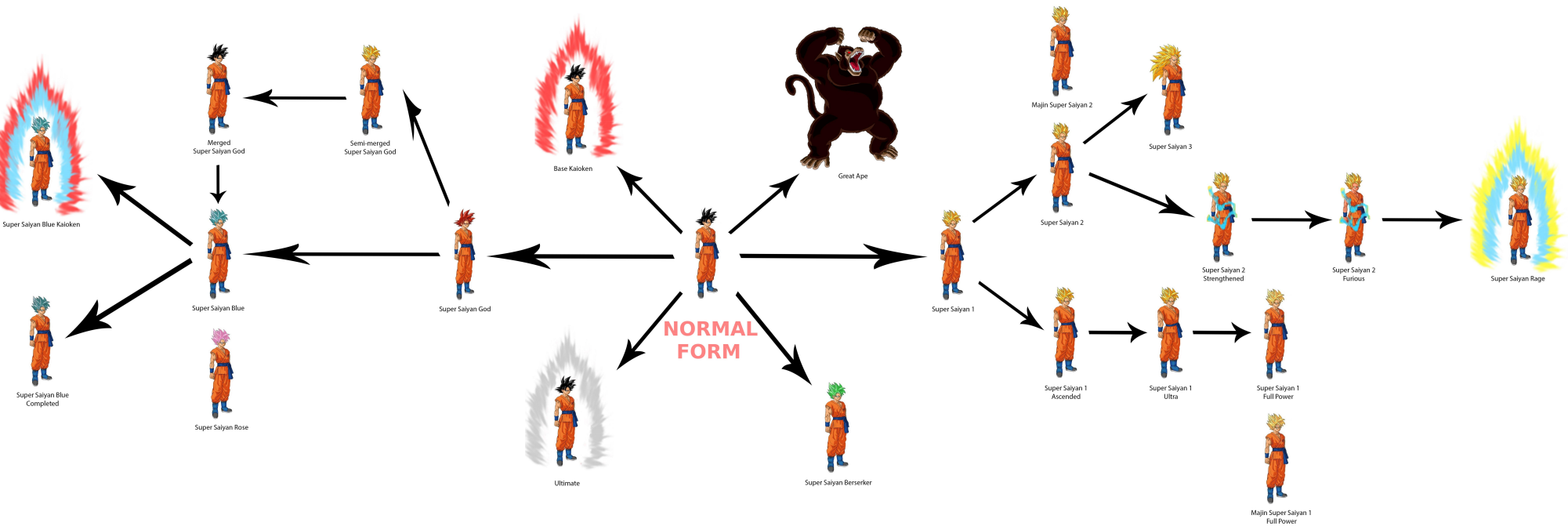
¿Qué transformaciones sintácticas mantienen el valor de verdad?

Si $c = c_1 \wedge c_2$ es SAT, $c \rightarrow c_1$ (proyectar) mantiene la satisfacibilidad.

Si $\bar{c} = \bar{c_1} \vee \bar{c_2}$ es UNSAT, $\bar{c} \rightarrow \bar{c_1}$ (proyectar) mantiene la insatisfacibilidad.

...pero hay que tener en cuenta el valor de verdad de la fórmula y no queda claro cómo definir una forma normal.

¿Preguntas?



¿Preguntas?