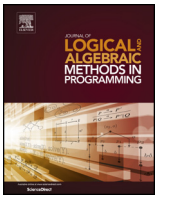
Jurnal Metode Logika dan Aljabar dalam Pemrograman 93 (2017) 68-91

Daftar isi tersedia di ScienceDirect

Jurnal Metode Logika dan Aljabar di Pemrograman

www.elsevier.com/locate/jlamp



Menjalankan dan memverifikasi program-program fungsional-imperatif tingkat tinggi di Maude

Vlad Rusu Sebuah , * , Andrei Arusoaie b

Sebuah *Inria Lille Nord Eropa, Prancis*

b *Universitas "Alexandru Ioan Cuza" di Iasi, Rumania*

artikel info

Sejarah artikel:

Diterima 16 November 2016

Diterima dalam bentuk revisi 24 Juli 2017 Diterima 12 September 2017

Tersedia online 21 September 2017

Kata kunci:

Maude

Fungsi tingkat tinggi

Monad negara bagian

Logika Reachability

abstrak

Kami menggabungkan fungsi tingkat tinggi dan monad status di Maude, sehingga menyematkan bahasa fungsional tingkat tinggi dengan fitur-fitur penting dalam kerangka kerja Maude. Kami mengilustrasikan, melalui program sederhana dalam bahasa yang dihasilkan: eksekusi program yang konkret dan simbolis; verifikasinya berkenaan dengan properti yang diekspresikan dalam Reachability Logic, generalisasi bahasa-parametrik dari Hoare Logic; dan verifikasi properti kesetaraan program. Pendekatan kami terbukti bagus dan diimplementasikan di Full Maude dengan memanfaatkan fitur reflektif dan sistem modulnya.

1. Perkenalan

Maude [1] adalah kerangka kerja logis dan semantik berkinerja tinggi berdasarkan logika persamaan dan penulisan ulang. Pada intinya, Maude bukanlah bahasa tingkat tinggi, dalam arti tidak memiliki tipe fungsional; oleh karena itu, fungsi tidak dapat diberikan sebagai argumen atau dikembalikan sebagai nilai fungsi lainnya. Batasan dalam ekspresi ini, bagaimanapun, sebagian besar dibuat oleh fitur lain dari bahasa: the *sistem modul* untuk pemrograman berparameter, dan *refleksi* untuk memanipulasi konstruksi Maude sebagai istilah meta-level.

Kontribusi Dalam makalah ini kami menggunakan sistem modul dan meta-level untuk menggabungkan fungsi tingkat tinggi dan monad negara di Maude. Ini berarti embedding dangkal di Maude dari bahasa pemrograman fungsional tingkat tinggi dengan fitur-fitur penting. Dalam bahasa yang dihasilkan, seseorang dapat memprogram dalam semangat, misalnya, Haskell [2] , sambil tetap memiliki akses ke fitur bahasa dan sistem Maude yang sudah dikenal (untuk pengguna Maude).

Kami mengilustrasikan bahasa yang dihasilkan dengan program fungsional-imperatif sederhana. Kami menunjukkan bagaimana program dapat: dijalankan dengan data konkret dan simbolik; diverifikasi secara resmi sehubungan dengan properti yang dinyatakan dalam Reachability Logic [3–6] , generalisasi bahasa-independen logika Hoare; dan secara resmi terbukti setara. Kelayakan eksekusi simbolik, verifikasi program dan verifikasi program-ekuivalen secara resmi dibuktikan.

Pekerjaan yang berhubungan Fungsi tingkat tinggi adalah umum di sebagian besar bahasa fungsional. Di Maude, kemungkinan untuk mendefinisikan fungsi-fungsi tingkat tinggi diketahui [1] tetapi fitur ini (termasuk definisi dan evaluasi fungsi lambda) belum, untuk

* Penulis yang sesuai.

Alamat email: vlad.rusu@inria.fr (V. Rusu), andrei.arusoaie@uaic.ro (A. Arusoaie).

[1] Lihat, misalnya, jawaban atas pertanyaan tentang hal ini pada daftar bantuan Maude di https://lists.cs.illinois.edu/lists/arc/maude-help/2006-01/msg00005.html).

https://doi.org/10.1016/j.jlamp.2017.09.002

2352-2208 / © 2017 Elsevier Inc. Semua hak dilindungi undang-undang.

pengetahuan terbaik kami, dieksploitasi sepenuhnya sebelumnya. Mungkin salah satu alasannya adalah bahwa untuk mengimplementasikan konstruksi fungsional tingkat tinggi, seseorang memerlukan fitur-fitur canggih yang saat ini tidak tersedia di (inti) Maude tetapi hanya di *Maude penuh,* ekstensi Maude juga diterapkan di Maude melalui refleksi.

Monad adalah bahan penting dalam beberapa bahasa fungsional seperti Haskell. Secara khusus, monad negara digunakan untuk memperkenalkan fitur-fitur penting tanpa mengorbankan kemurnian bahasa pemrograman fungsional. Monad negara bagian juga telah diperkenalkan di asisten bukti Coq [7] untuk membuktikan program yang ditulis dengan gaya imperatif [8] . Integrasi monad negara bagian di Maude adalah pengetahuan terbaik kami yang baru dalam kerangka ini.

Pekerjaan terkait lainnya adalah verifikasi program, khususnya, sehubungan dengan Reachability-Logic (rl) rumus. rl adalah formalisme untuk mendefinisikan semantik bahasa pemrograman dan logika spesifikasi program yang dapat dilihat sebagai generalisasi bahasa-parametrik dari logika Hoare. Untuk memverifikasi program dalam suatu bahasa, katakan, *L,* seseorang mendefinisikan semantik operasional *L,* kemudian menentukan properti program yang diharapkan *P.* bunga; ketepatan program didefinisikan sebagai konsekuensi semantik antara rl rumus yang mendefinisikan semantik dari *L* dan mereka yang mendefinisikan properti dari *P.* Pendekatan ini [9] telah digunakan pada bahasa / program yang didefinisikan di K kerangka [10] . Salah satu kontribusi dari makalah ini menunjukkan bahwa pendekatan sebelumnya juga dapat digunakan pada bahasa yang didefinisikan oleh *embedding dangkal,* sedangkan karya yang ada digunakan *embeddings dalam.* Keuntungan dari embedding yang dangkal adalah seseorang dapat menggunakan konstruksi bahasa tuan rumah saat menulis program dalam bahasa tamu. Sebaliknya, dalam program embedding yang dalam harus tetap berada dalam batasan sintaks dan semantik bahasa tamu. Dikotomi yang sama (embeddings dangkal vs. dalam) membedakan makalah ini dengan karya kami sebelumnya tentang eksekusi simbolik parametrik bahasa, verifikasi program, dan kesetaraan program [11–15] . Tentu saja ada banyak karya lain di bidang ini, yang, berbeda dengan kami, sebagian besar didedikasikan untuk bahasa tertentu. Eksekusi simbolik — menjalankan program dengan nilai-nilai simbolik daripada nilai konkret — telah diperkenalkan pada tahun tujuh puluhan [16] dan telah digunakan untuk men-debug, menguji dan menganalisis program. Alat termasuk eksekusi simbolik yang dikombinasikan dengan bahan lain termasuk Java PathFinder [17] , DART [18] , IMUT [19] , EXE [20] , PEX [21] . Alat yang dibangun di atas eksekusi simbolik untuk melakukan analisis dan verifikasi program termasuk [22–25] . Mengenai kesetaraan program, orang dapat menyebutkan bekerja pada ketepatan kompilator C. [26,27] , dan pendekatan yang menargetkan kelas bahasa tertentu: fungsional [28] , kode mikro [29] , CLP [30] . Beberapa pendekatan menargetkan jenis program tertentu: versi berurutan dari potongan kode tertentu [31] , prosedur rekursif [32] .

Di bidang umum penulisan ulang, teknik pemeriksaan model berdasarkan penyempitan telah dikembangkan [33] . Yang paling perbedaan yang signifikan dengan pendekatan kami adalah bahwa teknik berbasis penyempitan hanya memungkinkan aturan penulisan ulang tanpa syarat, yang kami izinkan (dan digunakan secara intensif) dalam makalah ini; di sisi lain, mereka berurusan dengan logika linier-temporal, yang lebih ekspresif daripada logika jangkauan. Terakhir tapi bukan yang akhir, *rewritingmodulo* smt [34,35] adalah teori dan implementasi eksekusi simbolik di Maude, pada dasarnya isomorfik dengan yang kami kembangkan K [36] —Sebuah contoh penemuan dan perwujudan sekaligus konsep yang "di udara" pada saat tertentu dalam waktu.

Akhirnya kami membandingkan makalah ini dengan versi lokakarya [37] . Perbedaan pertama adalah di sini kita menerapkan simbolik eksekusi, verifikasi program dan program ekuivalen dengan bahasa yang tertanam dangkal di Maude, dengan fitur-fitur fungsional-imperatif tingkat tinggi, sedangkan di versi sebelumnya kami hanya menerapkan verifikasi formal, ke program dalam bahasa imperatif sederhana yang tertanam dalam di Maude. Perbedaan lainnya adalah bahwa di versi awal kami bersikeras *inkrementitas* dalam buktinya, sedangkan di sini fitur ini tidak dominan (namun tetap ada). Perbedaan ketiga dan terakhir adalah dimasukkannya semua bukti dalam makalah ini untuk membuatnya mandiri.

Kode sumber Maude untuk pekerjaan yang dilaporkan dalam makalah ini tersedia di: https://profs.info.uaic.ro/~arusoaie.andrei/ maude-monad.zip .

Organisasi Setelah pengantar ini di Bagian 2 kami memperkenalkan pendekatan kami untuk menggabungkan fungsi tingkat tinggi dan monad negara ke dalam Maude. Di bagian 3 kami mempersembahkan Reachability Logic (rl) dan eksekusi simbolik berdasarkan bahasa yang semantik operasionalnya didefinisikan menggunakan rl. Hubungan antara eksekusi konkret dan simbolis dinyatakan. Di bagian 4 kami memperkenalkan prosedur konstruksi pohon berdasarkan eksekusi simbolik, bersama dengan hasil yang mengatakan bahwa, jika prosedur berhasil dihentikan pada satu set rl rumus menentukan program, maka program tersebut memenuhi rumus. Di bagian 5 kami menunjukkan bagaimana verifikasi program dapat membantu dalam membangun kesetaraan program. Semua konsep diilustrasikan pada program fungsional-imperatif tingkat tinggi yang sederhana. Kesimpulan dan arah kerja di masa depan digambar di Bagian 6 . Lampiran berisi bukti rinci untuk semua hasil.

2. Program-program imperatif fungsional tingkat tinggi di Maude

Pada bagian ini kami memperkenalkan Maude dan menunjukkan bagaimana fungsi tingkat tinggi dan monad status dapat digabungkan di dalamnya. Kami mengilustrasikan pendekatan dengan program sederhana, yang dapat dilihat sebagai milik bahasa imperatif-fungsional tingkat tinggi yang tertanam dangkal di Maude. Bahasa Maude diperkenalkan secara bertahap melalui konsep yang disajikan di seluruh bagian ini.

2.1. Menambahkan fungsi tingkat tinggi ke Maude

Maude adalah bahasa spesifikasi yang dapat dieksekusi. Spesifikasi Maude (yang terkadang kita sebut *program)* disusun menjadi beberapa unit yang disebut *modul* dan *teori,* yang dapat mengimpor satu sama lain, dapat diparameterisasi, dan dapat dibuat instance-nya

```
TRIV kelima adalah
    semacam Elt. Sial, ini adalah akhir komentar
```

Gambar 1. Teori TRIV.

```
fmod ARROW {X :: TRIV, Y :: TRIV} adalah *** parameter formal yang melindungi SUBST. *** modul yang
diimpor
urutkan Panah {X, Y}. *** semacam deklarasi
op __: Panah {X, Y} X $ Elt -> Y $ Elt. *** deklarasi operasi op lambda _: _: X $ Elt Y $ Elt -> Panah {X, Y} [ctor].

var f: Panah {X, Y}. *** deklarasi variabel vars xz: X $ Elt.

var y: Y $ Elt.
op ERR: -> [Y $ Elt]. *** deklarasi konstan
persamaan (lambda x: y) (z) = ***
    downTerm (subst (upTerm (x), upTerm (z), upTerm (y)), ERR).
endfm
```

Gambar 2. Modul PANAH.

dengan menghubungkan parameter aktual ke parameter formal dari unit berparameter. Terdapat sejumlah unit yang telah ditentukan sebelumnya untuk struktur data umum (Boolean, integer, string, ...) yang dikumpulkan ke dalam file Maude yang disebut the *pendahuluan.* Mungkin unit paling sederhana dari pendahuluan adalah teori TRIV, yang hanya menyatakan satu *menyortir* Elt. Teori ini ditunjukkan di Gambar 1 .

Gambar 2 menunjukkan contoh a *modul fungsional,* salah satu dari beberapa jenis modul yang tersedia di Maude. Ini memiliki dua parameter formal, X, Y itu adalah contoh teori TRIV. (Perbedaan antara modul dan teori tidak relevan di sini). Modul ini mengimpor modul lain SUBST yang akan kita bahas di bawah ini. Itu melindungi Kata kunci menunjukkan bahwa pengimporan tidak dimaksudkan untuk mengubah semantik modul yang diimpor. Kemudian, semacam parameterised Panah {X, Y} dinyatakan, yang, dalam niat kami, adalah jenis fungsi dari X untuk Y. Lebih tepatnya, sejak X, Y tidak diurutkan tetapi parameter formal untuk teori yang telah ditentukan sebelumnya TRIV, jenis berparameter Panah {X, Y} sesuai dengan fungsi dari X $ Elt

dan Y $ Elt dimana Elt adalah jenis yang didefinisikan dalam parameter TRIV.

Tentu saja, hanya menyatakan semacam itu Panah {X, Y} tidak menjadikannya semacam fungsi dari X $ Elt dan Y $ Elt: lebih banyak informasi diperlukan. Deklarasi op lambda _: _: X $ Elt Y $ Elt -> Panah {X, Y} [ctor] adalah bagian dari informasi ini. Ia mengatakan bahwa elemen di sortir Panah {X, Y} dapat *dibangun ([* ctor]) dengan *operasi* lambda _: _

yang kami gunakan untuk menunjukkan fungsi anonim. Operasi tersebut membutuhkan X $ Elt parameter yang dimaksudkan sebagai argumen fungsi, dan a Y $ Elt yang dimaksudkan untuk menjadi fungsi tubuh. Jadi, istilah bentuknya lambda x: y, dengan x, y
dideklarasikan dalam modul sebagai *variabel* jenis yang sesuai, memiliki jenis Panah {X, Y}.

Sekarang kita dapat secara sintaks membangun fungsi lambda. Semantik mereka (dengan evaluasi) dideklarasikan dalam modul sebagai file *penjajaran* operasi _ _ : Panah {X, Y} X $ Elt -> Y $ Elt, yaitu, penjajaran suatu fungsi (semacam Panah {X, Y}) dan argumen (semacam X $ Elt) yang, dalam niat kami, menghasilkan hasil (semacam Y $ Elt) diperoleh dengan "menerapkan" fungsi pada argumen. Jadi, kita dapat membangun istilah (lambda x: y) (z) menunjukkan penerapan fungsi (lambda x: y) ke argumen z.

Tidak mengherankan, evaluasi dicapai dengan mengganti parameter formal x dengan parameter sebenarnya z dalam fungsi tubuh y. Kami mencapai ini menggunakan modul yang diimpor SUBST dimana operasi yang memadai subst () sudah ditentukan. Untuk bekerja untuk term apapun (dan bukan hanya jenis yang kita deklarasikan di modul kita) operasi substitusi didefinisikan pada *metarepresentation* dari x, y, dan z. Ini dicapai dengan *pemeliharaan* mereka ke istilah jenis yang telah ditentukan sebelumnya Istilah

tersedia dalam modul yang telah ditentukan META-LEVEL, menggunakan operasi upTerm () juga didefinisikan dalam modul itu. Substitusi juga menghasilkan suatu hasil Istilah, yang kemudian harus menjadi *diturunkan* kembali ke (semoga) istilah semacam Y $ Elt. Ini dicapai dengan operasi yang telah ditentukan sebelumnya downTerm () di META-LEVEL, yang diberi istilah semacam Istilah dan mencoba menurunkannya ke jenis tertentu. Jenis yang dimaksud harus ditentukan oleh pengguna (karena sistem Maude tidak memiliki cara untuk mengetahui niatnya). Oleh karena itu deklarasi tersebut ERR: -> [Y $ Elt] dari sebuah konstanta BERBUAT SALAH dari *jenis [* Y $ Elt] —Kind adalah jenis "superset" yang memungkinkan pengguna untuk mendeklarasikan istilah "error" selain istilah "yang sesuai" dalam jenis tersebut. Jadi, dalam kasus kami,

downTerm () mencoba untuk mengubah istilah tersebut subst (upTerm (x), upTerm (z), upTerm (y)) semacam itu Istilah untuk istilah
menyortir Y $ Elt, dan jika tidak berhasil mengembalikan konstanta BERBUAT SALAH. Ini adalah arti keseluruhan dari pernyataan terakhir di kami
modul, itu *persamaan (* lambda x: y) (z) = downTerm (subst (upTerm (x), upTerm (z), upTerm (y)), ERR).

Jadi, sebagian besar pekerjaan sebenarnya dilakukan dalam modul SUBST, yang hanya akan kami jelaskan secara singkat di sini. Impor modul META-LEVEL untuk memiliki akses ke semacam itu Istilah dan untuk itu *subsort* Variabel dan Konstan. Operasi substitusi (dari variabel dengan istilah dalam istilah) adalah standar, kecuali untuk situasi ketika lambda _: _ operasi terlibat: itu tidak menggantikan variabel yang diikat oleh lambda _: _, dan dalam kasus di mana argumen file

lambda _: _ terjadi dalam istilah yang akan diganti, argumen pertama-tama diganti namanya sebelum substitusi diterapkan. Ini diekspresikan oleh persamaan dalam Gambar 3 , yang kedua adalah *bersyarat.* Ekspresi seperti 'lambda _: _ [x, t'] dalam gambar tersebut adalah meta-representasi Maude (semacam Istilah) untuk istilah lambda x: t '(semacam itu Y $ Elt).

Kami mencatat bahwa implementasi saat ini memungkinkan pengguna untuk menulis istilah lambda yang tidak masuk akal seperti lambda 1: 1.
Namun, ketika seseorang mencoba untuk mengevaluasi istilah-istilah tersebut pada argumen, hasilnya adalah konstanta yang telah ditentukan sebelumnya BERBUAT SALAH. Ini cukup

```
fmod SUBST adalah
melindungi META-LEVEL.
. . .
eq subst (x, t, 'lambda _: _ [x, t']) = 'lambda _: _ [x, t']. ceq subst (x, t, 'lambda _: _ [x', t ']) =

    'lambda _: _ [ganti nama (x'), subst (x, t, subst (x ', rename (x'), t '))] jika muncul (x', t).

. . .
endfm
```

Gambar 3. Modul SUBST (kutipan).

```
tampilan Int dari TRIV ke INT adalah
    urutkan Elt ke Int
endv
```

Gambar 4. Melihat Int.

mirip dengan apa yang terjadi dalam bahasa pemrograman umum: banyak program yang tidak masuk akal diterima oleh parser; mereka hanya menghasilkan kesalahan pada waktu proses.

Kami memilih untuk mendefinisikan lambda beroperasi dengan cara ini karena alternatifnya — mengekspresikan segala sesuatu pada metalevel Maude — akan membuat kode fungsi tingkat tinggi tidak dapat dibaca, dan terlebih lagi untuk konstruksi monadik negara yang dibangun di atas fungsi tingkat tinggi.

Instantiating PANAH modul Kami sekarang membuat instance PANAH modul untuk membuat fungsi dari, katakanlah, Int untuk Int, di mana jenis yang telah ditentukan sebelumnya Int didefinisikan dalam pendahuluan Maude di modul INT. Secara intuitif, seseorang harus "menghubungkan" parameter formal X dan Y ke modul INT. Di Maude, hal ini dilakukan dengan konstruksi yang disebut a *melihat*. Sejak X dan Y adalah parameter teori TRIV di Gambar 1 , tampilan berikut (sudah ditentukan sebelumnya) menghubungkan sort Elt dari TRIV untuk Int dari INT (Gambar 4).

Nama tampilan itu sewenang-wenang, tetapi memilihnya menjadi nama jenis yang diinginkan sebagai parameter aktual (yang Elt dipetakan dalam tampilan) memungkinkan kita untuk membangun modul ARROW {Int, Int}, yang memberikan "ilusi" bahwa parameter formal X, Y adalah (di sini) dipetakan ke jenis Int. Modul ARROW {Int, Int} seperti PANAH {X, Y}

di Gambar 2 kecuali bahwa itu mendeklarasikan sort ARROW {Int, Int} dan sejenisnya X $ Elt, Y $ Elt sekarang keduanya Int. Setelah memuat modul ARROW {Int, Int} di Maude, mari kita minta Maude untuk mengevaluasi istilah tersebut lambda x: Int: x: Int + 1 dan
lalu istilah (lambda x: Int: x: Int + 1) 3:

```
Maude> (kurangi dalam ARROW {Int, Int}: (lambda x: Int: x: Int + 1).) Hasil Panah {Int, Int}:

lambda x: Int: x: Int + 1
Maude> (kurangi dalam ARROW {Int, Int}: (lambda x: Int: x: Int + 1) 3.) Hasil NzNat:

4
```

Artinya, kami telah meminta Maude untuk tampil *reduksi persamaan* untuk persyaratan di atas. Jenis hasil yang diharapkan, serta hasil itu sendiri, adalah benar.

Fungsi dari beberapa argumen Apa yang kita miliki sejauh ini adalah tipe untuk fungsi lambda dari satu argumen. Jenis fungsi dari beberapa argumen memanfaatkan gagasan *kari,* yang dalam kasus kami berarti memiliki semacam Panah {X, Panah {Y, Z}}
untuk fungsi tipe X × Y untuk Z. Untuk itu dibutuhkan sebuah modul PANAH {X, Panah {Y, Z}}.

Ingat, bagaimanapun, bahwa parameter modul formal ditautkan ke parameter aktual melalui *dilihat,* dan menurut konvensi, tampilan memiliki nama yang sama dengan tipe yang ingin diteruskan sebagai parameter aktual. Oleh karena itu, kami membutuhkan file *berparameterisasi* melihat:

```
(lihat Panah {X :: TRIV, Y :: TRIV} dari TRIV ke ARROW {X, Y} adalah
        urutkan Elt ke Panah {X, Y}. endv)
```

Ternyata tampilan berparameter seperti itu tidak tersedia di (Core) Maude, tetapi tersedia di Full Maude, implementasi reflektif dari Maude di Maude dengan beberapa ekstensi berguna (dan beberapa perbedaan kecil). Karenanya selanjutnya kita akan bekerja di Full Maude. Fitur yang paling terlihat dari Full Maude adalah unit kode dan perintah diapit tanda kurung, seperti beberapa yang terakhir ditunjukkan di atas. Full Maude juga dapat memuat unit kode Core Maude karenanya, yang kemudian ditulis tanpa tanda kurung seperti yang ditunjukkan sebelumnya di Gambar. 1–3 .

Ini menyimpulkan presentasi fungsi tingkat tinggi di Maude. Untuk mengilustrasikannya lebih lanjut, berikut adalah perintah yang mendemonstrasikan sederhana *evaluasi parsial:*

```
    (kurangi di ARROW {Int, Arrow {Int, Int}}:
                        (lambda x: Int: (lambda y: Int: x: Int + y: Int)) 1.)
    hasil Panah {Int, Int}:
        lambda y: Int: y: Int + 1
```

Hasilnya adalah yang diharapkan (modulo komutatifitas penjumlahan di Maude).

2.2. Monad negara bagian

Monad negara bagian dengan *negara* menyortir S dan *keluaran* menyortir SEBUAH adalah fungsi dari S untuk memasangkan (SEBAGAI). Asumsikan modul berparameter PASANGKAN {X :: TRIV, Y :: TRIV} mendefinisikan semacam Sandingkan {X, Y} dengan konstruktor (_, _) dan pengakses fst dan snd dihubungkan dengan persamaan yang sesuai. Kemudian kami mendefinisikan pengurutan berparameter Monad {A, S} dari monad dengan jenis negara bagian S dan keluaran menyortir SEBUAH untuk menjadi semacam itu Panah {S, Pair {A, S}}:

```
(fmod MONAD {A :: TRIV, S :: TRIV} melindungi ARROW {S, Pair {A,
S}}
                    * (urutkan Panah {S, Pasangkan {A, S}} ke Monad {A, S}).
endfm)
```

Itu *mengganti nama* konstruksi * di sini digunakan untuk mengganti nama jenis Panah {S, Pair {A, S}} ARROW {S, Pair {A, S}} ke nama yang lebih informatif dan ringkas Monad {A, S}.

Bersama dengan monad, dua operasi biasanya didefinisikan: membasahi, yang "mengembalikan" nilai tertentu dari tipe keluaran tanpa mengubah status, dan mengikat, yang "mengikat" hasil komputasi ke pengenal sehingga nilainya dapat digunakan kembali dalam perhitungan berikutnya. Kami mendefinisikannya dalam dua modul terpisah.

```
(fmod RET {A :: TRIV, S :: TRIV} melindungi ARROW {A, Monad
{A, S}}. var a: A $ Elt.

var s: S $ Elt.
  op ret: -> Panah {A, Monad {A, S}}. eq ret (a) (s) = (a, s).

endfm)
```

Dalam modul di atas, membasahi didefinisikan sebagai semacam konstanta Panah {A, Monad {A, S}}, yaitu fungsi dari keluaran SEBUAH ke monad Monad {A, S} (yang, ingat, juga merupakan fungsi). Persamaannya ret (a) (s) = (a, s) bisa jadi alternatif ditulis ret (a) = lambda s: (a, s); keduanya mengatakan itu ret (a) "Mengembalikan" output Sebuah dan tinggalkan negara bagian s tidak berubah, yang sesuai dengan definisi yang diharapkan.

Modul untuk operasi mengikat melibatkan monad dengan jenis keluaran yang berbeda:

```
(mod BIND {A :: TRIV, B :: TRIV, melindungi MONAD {A, S}.     S :: TRIV} adalah

melindungi MONAD {B, S}.
melindungi ARROW {A, Monad {B, S}}. var m: Monad {A, S}.

var m ': Monad {B, S}.
var f: Panah {A, Monad {B, S}}. vars a ': A $ Elt.

vars s ': S $ Elt.
  op bind: Monad {A, S} Panah {A, Monad {B, S}} -> Monad {B, S}. ceq bind (m, f) (s) = (f (a) (s ')) if (a, s'):
  = m (s).
  op do _: = _ in_: A $ Elt Monad {A, S} Monad {B, S} -> Monad {B, S}. persamaan (lakukan a: = m di m ') =
  mengikat (m, (lambda a: m')). op _ ;; _: Monad {A, S} Monad {B, S} -> Monad {B, S}. eq (m ;; m ') (s) = m' snd
  (m (s)). akhir)
```

Operasi utama mengikat mengambil monad m semacam itu Monad {A, S}, sebuah fungsi f dari SEBUAH untuk Monad {B, S} dan menghasilkan monad mengikat (m, f) semacam itu Monad {B, S}. Ini pertama kali berlaku m ke kondisi saat ini s, yang menghasilkan sepasang (sebagai'), kemudian berlaku f (a) untuk s '; cf. persamaan bind (m, f) (s) = (f (a) (s ')) if (a, s'): = m (s), yang kondisinya adalah a *persamaan yang cocok,* mirip dengan a *biarkan masuk* konstruksi dalam bahasa lain.

Sebuah gula sintaksis berguna yang terlibat mengikat adalah konstruksinya lakukan: = m dalam m ' dimana hasil perhitungannya m "disimpan" di "variabel" Sebuah; "nilai" dari variabel tersebut dapat digunakan dalam perhitungan selanjutnya m '. Ini

konstruksi memberikan tampilan kode penting menjadi kode fungsional murni. Konstruksi lain yang mirip imperatif adalah operasi pengurutan m ;; m ', yang berarti menyebarkan status (sambil membuang output) yang dihitung oleh m
untuk m '.

Jadi, fungsi tingkat tinggi dan monad status tersedia di Maude. Kami sekarang menunjukkan beberapa contoh sederhana dari program fungsional-imperatif tingkat tinggi.

2.3. Contoh: menyortir tumpukan

Contoh berjalan di sisa kertas adalah program untuk menyortir tumpukan.

Menerapkan tumpukan Modul Maude berikut membangun tumpukan menggunakan konstanta emptyStack dan sebuah unit asosiasi (AU) dalam operasi tetap _ :: _. Operasi hd dan tl untuk kepala dan ekor juga didefinisikan, bersama dengan persamaan yang menghubungkannya ke konstruktor. Predikat persamaan _ === _ untuk tumpukan diberikan.

```
(fmod STACK {X :: TOTAL-ORDER} adalah urutkan Stack {X}.

subsort X $ Elt <Stack {X}.
op emptyStack: -> Stack {X}.
op _ :: _: Stack {X} Stack {X} -> Stack {X} [ctor assoc id: emptyStack]. op hd: Tumpukan {X} ~> X $ Elt.

op tl: Stack {X} ~> Stack {X}.
op _ === _: Tumpukan {X} Tumpukan {X} ~> Bool. vars pqr: X $ Elt.

var stk: Stack {X}.
eq tl (p :: stk) = stk. persamaan hd (p :: stk)
= p.
eq stk === stk = true. endfm)
```

Parameter modul, X, bukan sebuah TRIV seperti sebelumnya tapi a TOTAL-ORDER, teori yang menyatakan semacam Elt dan total pesanan _ <= _ di atasnya. Itu *deklarasi subsort* subsort X $ Elt <Stack {X} mengatakan bahwa elemen X $ Elt adalah tumpukan juga; juga, simbol ~> menunjukkan bahwa operator hd dan tl adalah fungsi parsial.

Negara Kami mendefinisikan negara menjadi sepasang yang terdiri dari bilangan asli, yang akan digunakan untuk membuat variabel baru ketika variabel seperti itu dibutuhkan, dan tumpukan.

```
(fmod STATE {X :: TOTAL-ORDER} melindungi STACK adalah
{X}.
melindungi NAT.
urutkan Status {X}.
* * * Argumen pertama adalah untuk g op st: Nat Stack {X} menghidupkan variabel segar
  -> State {X
                                                          }.
endfm)
```

Operasi tumpukan monadik dasar Seperti biasa, kami mengasosiasikan dengan tumpukan operasi Dorong dan pop. Menjadi operasi monadik, mereka harus memiliki status sebagai parameter dan akan mengembalikan pasangan keluaran dan status. Itu Dorong operasi mendorong argumennya pada stack dan tidak mengeluarkan apa pun — di sini, dimodelkan dengan "mengeluarkan" tt, semacam konstan Satuan didefinisikan dalam modul yang diimpor SATUAN. Itu pop operasi menghapus bagian atas tumpukan dan mengeluarkannya menggunakan operasi tumpukan hd
dan tl.

```
(fmod PUSH {X :: TOTAL-ORDER} melindungi UNIT. ...

op push: X $ Elt -> Monad {Unit, Status {X}}.
eq push (n) st (p, stk) = (tt, st (p, n :: stk)). endfm)

(fmod POP {X :: TOTAL-ORDER} adalah ... op pop: -> Monad {X,
State {X}}.
eq pop (st (p, stk)) = (hd (stk), st (p, tl (stk))). endfm)
```

```
(fmod SORT {X :: TOTAL-ORDER} adalah ...
op min: Panah {X, Panah {X, Bool}} -> Monad {Satuan, Status {X}}. eq (min (leq) st (t, emptyStack)) = (tt, st (t,
emptyStack)). eq (min (leq) st (t, n)) = (tt, st (t, n)).

eq min (leq) st (t, n :: m :: stk) =
        (lakukan x: = masuk
            (min (leq) ;;
                do y: = masuk
                        jika (leq x) y                    kemudian
                          dorong (y) ;;
                          dorong (x)
                        lain
                          dorong (x) ;;
                          ambisius)
                        fi
        )) st (t + 2, n :: m :: stk)
jika x: = freshVar (t) / \ y: = freshVar (t + 1) ...
```

Gambar 5. Modul URUTKAN (kutipan): operasi min.

```
op sort: Panah {X, Panah {X, Bool}} -> Monad {Satuan, Status {X}}. eq sort (leq) st (t, emptyStack) = (tt, st (t,
emptyStack)). eq sort (leq) st (t, n) = (tt, st (t, n)).

eq sort (leq) st (t, n :: m :: stk) =
        (min (leq) ;;
          (do x: = pop in (sort (leq)                    ;; dorong (x)))
          )
          st (t + 1, n :: m :: stk) jika x: = freshVar (t).
endfm)
```

Gambar 6. Modul URUTKAN (kutipan): operasi menyortir.

Operasi min Operasi monadik ini mengambil tumpukan dan "memindahkan" elemen terkecil dari tumpukan (sesuai dengan hubungan keteraturan leq dibutuhkan sebagai parameter) ke bagian atas tumpukan. Ini didefinisikan sebagai berikut (Gambar 5): untuk tumpukan kosong dan tumpukan satu elemen output operasi tt dan membiarkan tumpukan dalam keadaan tidak berubah.

Namun, untuk tumpukan yang terdiri dari setidaknya dua elemen, operasi yang pertama memunculkan tumpukan dan menyimpan hasilnya ke dalam variabel x, lalu secara rekursif memanggil dirinya sendiri di tumpukan yang muncul. Selanjutnya, ini memunculkan hasil dan menyimpan dalam variabel y. Akhirnya, x dan y didorong kembali ke tumpukan sehingga yang terkecil masuk ke atas.

Kita dapat melihat, secara intuitif, bahwa pada akhirnya seseorang memang mendapatkan tumpukan di mana elemen terkecil berada di atas dan yang memiliki elemen yang sama dengan tumpukan yang diberikan sebagai parameter. Nanti di koran kami akan membuktikan properti ini secara resmi.

Perhatikan *kondisi yang cocok* pada persamaan ketiga: x dan y dibuat on-the- fl y oleh suatu fungsi freshVar () (tidak ditampilkan di sini) yang mengambil parameter bilangan asli yang kami tambahkan ke status untuk tujuan ini. Persamaan yang cocok ini digunakan untuk membuat persamaan ketiga *dapat dieksekusi* oleh Maude; tanpa mereka, x dan y akan menjadi variabel di sisi kanan persamaan yang tidak muncul di sisi kirinya. Persamaan semacam itu ditolak oleh Maude karena tidak dapat dieksekusi.

Perhatikan juga penggunaan builtin Maude jika-maka-lain-fi konstruksi. Di sini kami melakukan penyematan bahasa yang dangkal (dengan instruksi push, pop, do _: = _ in _, _ ;; _, min dll) ke Maude; yaitu, kami menggunakan sintaks dan semantik Maude untuk mendefinisikan instruksi dari bahasa kami yang disematkan, dan kami diizinkan untuk menggunakan semua konstruksi Maude, termasuk jika-maka-lain-fi, dalam program bahasa kita. Hal ini berbeda dengan deep embeddings, yang sama saja dengan menggunakan Maude untuk mendefinisikan sintaks dan semantik dari bahasa tertentu; kemudian, program dalam bahasa tersebut hanya akan diizinkan untuk menggunakan sintaks yang telah ditentukan, tidak termasuk konstruksi Maude lainnya (jika tidak, mereka bahkan tidak akan mengurai).

Operasi menyortir Operasi monadik ini (Gambar 6) adalah orang yang melakukan penyortiran tumpukan menurut pesanan leq itu penerima sebagai parameter. Dua persamaan pertama menangani kasus tumpukan kosong dan tumpukan satu elemen. Kasus yang menarik adalah tumpukan dengan setidaknya dua elemen, yang ditangani oleh persamaan ketiga. Operasi yang ditentukan sebelumnya min pertama kali dipanggil untuk mengeluarkan elemen terkecil dan meletakkannya di atas tumpukan. Elemen ini kemudian muncul dan disimpan dalam sebuah variabel x. Selanjutnya, operasi menyortir dipanggil secara rekursif pada tumpukan yang muncul, dan akhirnya disimpan sebelumnya x didorong kembali ke tumpukan.

Di sini sekali lagi orang dapat melihat, secara intuitif, bahwa pada akhirnya seseorang mendapatkan tumpukan yang diurutkan yang memiliki elemen yang sama dengan tumpukan yang dimulai. Nanti di makalah kita akan melihat bagaimana membuktikan ini secara resmi dengan menjalankan perintah Maude tertentu. Kami juga membuktikan di makalah bahwa perintah memang melakukan verifikasi yang diharapkan.

Menjalankan program Untuk menjalankan program, kami membuat instance pada modul Maude yang telah ditentukan sebelumnya Nat <= yang menyediakan jenis yang telah ditentukan sebelumnya Nat dan urutan <=. Sekarang, <= ditentukan sebelumnya sebagai operasi: _ <= _: Nat Nat -> Bool. Sebagai

seperti itu tidak dapat diteruskan sebagai parameter ke operasi monadik tingkat tinggi kita (yang membutuhkannya). Dengan demikian, kami mendefinisikan sebuah konstanta ord sebagai relasi Nat dan mendefinisikannya secara ekstensional sama dengan <=. Konstan ord kemudian dapat dilewatkan sebagai parameter.

```
(mod INSTANTIATE-SORT adalah
melindungi SORT {Nat <=}.
op ord: -> Panah {Nat <=, Panah {Nat <=, Bool}}. vars xy: Nat.

eq ord xy = x <= y.
op terbalik: Panah {Nat <=, Panah {Nat <=, Bool}} ->
                    Panah {Nat <=, Panah {Nat <=, Bool}}.
var leq: Panah {Nat <=, Panah {Nat <=, Bool}}. eq (kebalikan (leq) (x)) (y) = (leq y) x.
akhir)
```

Akhirnya kami mendefinisikan sebuah fungsi balik yang "membalikkan" perintah tertentu, dan memanggil urutkan (ord) dan urutkan (mundur (ord)) pada masukan untuk menguji kebenarannya:

```
Maude> (red sort (ord) st (0, 1 :: 4 :: 2 :: 8 :: 5 :: 7).) Result Pair {Unit, State {Nat <=}}:

(tt, st (35, 1 :: 2 :: 4 :: 5 :: 7 :: 8))
Maude> (red sort (reverse (ord)) st (0, 1 :: 4 :: 2 :: 8 :: 5 :: 7).) Result Pair {Unit, State {Nat <=}}:

(tt, st (35, 8 :: 7 :: 5 :: 4 :: 2 :: 1))
```

Maude segera mengembalikan hasil yang diharapkan. Jadi, kita sekarang memiliki embedding bahasa yang dangkal (dengan instruksi push, pop, do _: = _ in _, _ ;; _, min, dan menyortir) menjadi Maude, yang berjalan secara efisien pada masukan konkret yang diberikan. Pada bagian selanjutnya kami menunjukkan bagaimana bahasa dapat diubah untuk memungkinkan eksekusi simbolik program, yaitu menjalankan program dengan input simbolik.

3. Eksekusi simbolik

Di bagian ini kami menunjukkan bagaimana seseorang dapat secara simbolis menjalankan program seperti yang ditunjukkan di bagian sebelumnya. Pendekatan ini parametrik dalam definisi bahasa yang semantik operasionalnya didefinisikan menggunakan rumus Reachability Logic.

3.1. Logika Reachability (rl)

Beberapa versi Reachability Logic telah diajukan dalam beberapa tahun terakhir [3–6] . Bahkan, rl dibangun di atas *Pencocokan Logika (* ml), yang juga ada dalam beberapa versi [38–40] . (Situasinya agak mirip dengan hubungan antara logika penulisan ulang dan logika persamaan di bawahnya.) Kami mengadopsi *semua jalur* interpretasi rl [6] , dan kembangkan seminimal mungkin ml itu cukup untuk mengekspresikan semantik bahasa dan properti program yang secara praktis relevan, dan terlebih lagi dapat menerima eksekusi simbolik dengan menulis ulang.

Rumus dari ml disebut *pola* dan dijelaskan di bawah. Asumsikan tanda tangan aljabar dengan satu set S macam, termasuk macam *Bool* dan *Cfg (konfigurasi).* Kami menulis $T, s\ (Var)$ untuk kumpulan term of sort s lebih dari satu set yang dapat dihitung *Var* dari S- variabel yang diindeks, dan T, s untuk himpunan istilah-istilah dasar s.

Contoh 1. Semacam itu *Cfg* konfigurasi pada contoh dari bagian sebelumnya adalah Sandingkan {XuUnit, State} dimana XuUnit adalah supersort terkecil dari X $ Elt dan Satuan. Sampel *Cfg* istilah adalah (n, st (t, stk)) dan urutkan (leq) st (t, n :: m :: stk) dengan variabel yang diurutkan leq: Panah {X, Panah {X, Bool}}, n: X $ Elt, m: X $ Elt, t: Nat, dan Stk: Tumpukan {X}.

Kami mengidentifikasi *Bool*- operasi yang diurutkan di dengan satu set dari predikat.

Definisi 1 (*Pola).* Pola adalah ekspresi dari bentuk $(\exists X)\ \pi \wedge \varphi$, dimana $X \subset Var$, $\pi \in T_{Cfg}(X)$ dan φ adalah Logika Orde Pertama (fol) rumus di atas tanda tangan urutan pertama (,) dengan variabel bebas di X.

Kami sering menunjukkan pola dengan ϕ dan tulis $\phi\ (\exists X)\ \pi \wedge \varphi$ untuk menekankan variabel yang dihitung X, itu *pola dasar* π, dan φ, itu *kondisi.* Kami membiarkan $FreeVars(\phi)$ menunjukkan kumpulan variabel yang muncul secara bebas dalam suatu pola ϕ, didefinisikan seperti biasa (yaitu, tidak di bawah angka kejadian). Kami mengidentifikasi pola dasar π dengan pola $(\exists\emptyset)\ \pi \wedge benar$, dan *pola dasar* dari bentuk $\pi \wedge \varphi$ dengan pola $(\exists\emptyset)\ \pi \wedge \varphi$.

Contoh 2. Penulisan pola dalam Maude berbeda dengan notasi matematisnya. Pertama, kami memperkaya struktur semacam itu Negara Bagian {X} dengan komponen ketiga Bool, ditakdirkan untuk menyandikan kondisi pola. Kedua, di Maude tidak ada parameter eksistensial (eksplisit). Kami menyandikan variabel yang dihitung dengan variabel baru, dibangun dengan fungsi

freshVar (). Contoh pola kuantifikasi dengan konvensi ini adalah ruas kanan dari salah satu persamaan
menyortir, dimana f memainkan peran kondisi dan x dari variabel yang dihitung:

```
eq sort (leq) st (t, n :: m :: stk) =
        (min (leq) ;; (do x: = pop in
                (urutkan (leq) ;;
                  dorong (x)))) st (t + 1, f, n :: m :: stk) if x: = freshVar (t)
```

Kami sekarang menjelaskan semantik pola. Kami mengasumsikan model M dari contoh tanda tangan aljabar Maude, model M adalah model awal yang didalamnya kelas kesetaraan Dalam kasus kami dan aksioma spesifikasi. Untuk macam $s \in S$
kami menulis M_s untuk interpretasi semacam itu s. Kami memanggil *penilaian* fungsinya $\rho: Var \to M$ yang menetapkan variabel
nilai dari jenis yang sesuai, dan (konkret) *konfigurasi* elemen di set pembawa M_{Cfg}. Contoh konfigurasi konkret adalah kelas ekivalensi dengan semua persamaan dan aksioma dalam spesifikasi Maude kami untuk istilah dasar
(tt, st (0, false, emptyStack)) semacam itu *Cfg*.

Definisi 2 (*Semantik pola).* Diberikan pola ϕ ($\exists$ *X)* $\pi \wedge \varphi$, $\gamma \in M_{Cfg}$ sebuah konfigurasi, dan $\rho: Var \to M$ penilaian, hubungan kepuasan (γ, ρ) $\models \phi$ memegang jika ada penilaian ρ' dengan $\rho' \restriction_{Var \setminus X} = \rho \restriction_{Var \setminus X}$ seperti yang $\gamma = \rho'$ (π) dan $\rho' \models \varphi$
(di mana yang terakhir $\models$ menunjukkan kepuasan dalam fol, dan $\rho \restriction_{Var \setminus X}$ menunjukkan batasan penilaian ρ ke set *Var* \ *X).*

Kami sekarang mengingat Reachability-Logic (rl) rumus, sistem transisi yang mereka hasilkan, dan semantik semua jalurnya [6]
yang kami gunakan dalam makalah ini.

Definisi 3 (rl *rumus).* Sebuah rl rumus adalah sepasang pola $\phi \Rightarrow \phi'$.

Contoh 3. Di bagian sebelumnya kami mengilustrasikan embedding dangkal dari bahasa dengan instruksi do _: = _ in_, _ ;; _, push, pop, min dan menyortir di Maude, yang semantiknya didefinisikan menggunakan persamaan. Kami mengubah persamaan itu menjadi Maude *menulis ulang aturan* dan menafsirkan aturan yang dihasilkan sebagai rl rumus. Misalnya persamaan urutkan (leq)
st (t, f, emptyStack) = (tt, st (t, f, emptyStack)) menjadi aturannya urutkan (leq) st (t, f, emptyStack)
=> (tt, st (t, f, emptyStack)). Baik persamaan dan aturan diterapkan dengan menulis ulang, tetapi persamaan ditakdirkan untuk menyandikan perhitungan deterministik (seperti yang konkret dalam bahasa kita), sedangkan aturan menyandikan kemungkinan perhitungan nondeterministik (seperti yang simbolis).

Membiarkan S menunjukkan satu set tetap rl rumus, misalnya, semantik bahasa tertentu. Kami mendefinisikan sistem transisi yang didefinisikan oleh S dan kemudian validitas rl rumus.

D e fin

itu {tion 4 (*Sistem Transisi didefinisikan oleh set S dari* rl *rumus).* Itu *sistem transisi* didefinisikan oleh *S* adalah ($\mathcal{H}$ *Cfg*, $\Rightarrow_S$), dimana $\Rightarrow_S = \{(\gamma, \gamma') \mid \exists\, \phi \Rightarrow \phi' \in S, (\exists\, \rho)\ (\gamma, \rho) \models \phi \wedge (\gamma', \rho) \models \phi'\}$. Kami menulis $\gamma \Rightarrow_S \gamma'$ untuk (γ, γ') $\in \Rightarrow_S$. Negara γ adalah *terminal* jika tidak ada γ' seperti yang $\gamma \Rightarrow_S \gamma'$. SEBUAH *jalan* adalah urutan $\gamma_0 \cdots \gamma_n$ seperti yang $\gamma_{saya} \Rightarrow_S \gamma_{i+1}$ untuk semua $0 \le saya \le n - 1$. Jalan seperti itu *lengkap* jika γ_n adalah terminal.

Jadi, sistem transisi diinduksi oleh satu himpunan S dari rl rumus berisi semua kemungkinan perhitungan dari semua program yang mungkin dalam semua konteks yang memungkinkan.

Definisi 5 (*Keabsahan).* Sebuah rl rumus $\phi \Rightarrow \phi'$ valid, tertulis $S \models \phi \Rightarrow \phi'$, jika untuk semua pasangan (γ_0, ρ) seperti yang (γ_0, ρ) $\models \phi$, dan semua jalur lengkap $\gamma_0 \Rightarrow_S \cdots \Rightarrow_S \gamma_n$, ada $0 \le saya \le n$ seperti yang (γ_{saya}, ρ) $\models \phi'$.

Perhatikan bahwa validitas rl rumus hanya ditentukan oleh jalur terbatas dan lengkap. Jalur tak terbatas, yang diinduksi oleh program nonterminating, tidak relevan. Jadi, terminasi diasumsikan: sebagai logika program rl adalah logika kebenaran parsial.

Asumsi 1. Selanjutnya, rl rumus memiliki bentuk $\pi_l \wedge \varphi_l \Rightarrow (\exists\, Y)\ \pi_r \wedge \varphi_r$ seperti yang $FreeVars(\pi_r) \subseteq FreeVars(\pi_l) \cup Y$, $FreeVars(\varphi_r) \subseteq FreeVars(\pi_l) \cup FreeVars(\pi_r)$, dan $FreeVars(\varphi_l) \subseteq FreeVars(\pi_l)$.

Artinya, sisi kiri adalah pola dasar, dan sisi kanan adalah pola, kemungkinan dengan pembilang; kondisi yang tersisa mencegah variabel tambahan di sisi kanan dan dalam kondisi. Rumus semacam itu biasanya cukup ekspresif untuk mengekspresikan semantik bahasa [2] dan properti program.

3.2. Eksekusi simbolik parametrik bahasa

Kami sekarang menyajikan eksekusi simbolik, teknik analisis program yang terdiri dari menjalankan program dengan input simbolik (misalnya, nilai simbolik *x)* bukannya masukan konkret (misalnya, 0) dan dalam pemeliharaan *kondisi jalur* pada input simbolik.

Kami merumuskan kembali pendekatan eksekusi simbolik tanpa bahasa yang telah disajikan di tempat lain [11] , dengan terkenal penyederhanaan (misalnya, kami tidak menggunakan koinduksi). Pendekatan ini terdiri dari transformasi semantik dari bahasa pemrograman sehingga, di bawah batasan yang wajar, ditunjukkan di bawah ini, mengeksekusi program dengan semantik yang dimodifikasi sama dengan mengeksekusi program secara simbolis. Pendekatannya juga membedakan antara *builtin (* misalnya, bilangan bulat, boolean,. . .) dan *non-builtin (* misalnya, berbagai konstruksi bahasa pemrograman) macam dan operasi. Perbedaan ini penting karena memungkinkan seseorang untuk mengganti penyatuan dan penyempitan (yang merupakan operasi alami dalam pelaksanaan simbolik dari teori penulisan ulang, cf, misalnya, [33]) dengan pencocokan dan penulisan ulang yang jauh lebih efisien, dengan cara yang tepat.

Asumsi 2. Ada *builtin subsignature builtin*. Semacam itu *Cfg* tidak ada di $_b$. Jenis dan pengoperasian di $_b$ adalah *builtin;* semua yang lainnya *non-* dalamnya.

Selain itu, simbol operasi non-builtin mungkin hanya tunduk pada satu set (mungkin kosong) *linier, reguler, dan non-builtin* persamaan.

Definisi 6. Persamaan $u = v$ is: linear jika keduanya u, v linier (istilah linier jika ada variabel yang muncul paling banyak sekali); biasa jika keduanya u, v memiliki kumpulan variabel yang sama; dan non-builtin jika keduanya u, v hanya memiliki operasi non-builtin.

Contoh 4. Dalam bahasa kami, kami memilih subsignature bawaan untuk menjadi modul TUMPUKAN {X}. Artinya, jenis bawaan adalah Nat dan Bool (diimpor dalam modul), bersama dengan X $ Elt dan Negara Bagian {X} didefinisikan dalam modul. Semua jenis lainnya adalah non-builtin. Seseorang dapat memverifikasi bahwa persamaan yang tersisa setelah transformasi persamaan-ke-aturan (lih. Contoh 3 di atas) linier, reguler dan non-builtin.

S ini pada dasarnya menulis ulang modulo kesesuaian $T(Var)$ disebabkan oleh Asumsi 2 . Membiarkan $Var_b \subseteq Var$ menjadi variabel jenis bawaan. Kita memerlukan asumsi teknis berikut, S untuk merumuskan hasil simulasi sekarang kita mendefinisikan relasi transisi yang dihasilkan oleh himpunan aturan simbolik yang berarti bahwa variabel dalam eksekusi simbolik adalah variabel bawaan, dan tidak membatasi keumuman pendekatan kita:

Asumsi 3. Untuk setiap $\pi t_l \wedge \varphi_l \Rightarrow (\exists\ Y)\ \pi t_r \wedge \varphi_r \in S,\ \pi t_l \in T\ |_{b(}\ Var),\ \pi t_l$ linier, dan $Y \subseteq Var_b$.

Asumsi selalu dapat dibuat untuk menahan dengan mengganti di πt_l semua suku non-variabel dalam variabel segar, dan dengan menyamakan dalam $_b$ dan semua variabel duplikat kondisi φ_l variabel baru ke suku yang diganti.

Contoh 5. Beberapa aturan (sebelumnya, persamaan) dalam contoh kita tidak memuaskan Asumsi 3 dan perlu diubah untuk memuaskannya. Misalnya, aturan ketiga untuk urutkan (cf. Gambar 6) berisi operasi builtin _ :: _ on builtin sort Tumpukan {X} di sisi kirinya, yang melanggar batasan $\pi t_l \in T\ |_{b(}\ Var)$. Itu dapat dibuat untuk memenuhi batasan ini dengan (secara ekuivalen) menuliskannya dalam formulir

```
rl sort (leq) st (t, f, stk) =>
        (min (leq) ;;
          (do x: = pop in (sort (leq) ;; push (x)))) st (t + 3, f, stk)

jika n: = freshVar (t) / \ m: = freshVar (t + 1) / \
stk ': = freshStkVar (t) / \ (n :: m :: stk' = stk) / \ x: = freshVar (t + 2).
```

Setelah transformasi, operasi bawaan _ :: _ telah dipindahkan dari sisi kiri aturan ke kondisi, dengan variabel baru tambahan (agar aturan dapat dieksekusi oleh Maude) dan kondisi tambahan n :: m :: stk '= stk.
Transformasi semacam itu sistematis dan dapat diterapkan di Maude sendiri.

[2] Misalnya, bahasa yang didefinisikan dalam K kerangka: http://k-framework.org .

Untuk mendapatkan eksekusi simbolik, transformasi tambahan harus dilakukan pada aturan. Kami mendefinisikannya secara formal dan kemudian mengilustrasikannya dengan contoh.

Pertimbangkan tanda tangannya sesuai dengan definisi bahasa. Membiarkan *Fol* menjadi jenis baru yang semua istilahnya adalah fol rumus, termasuk pembilang.

Membiarkan *Indo* dan *IdSet* menjadi jenis baru yang menunjukkan pengenal dan set pengenal, dengan operasi serikat pekerja _, _. Membiarkan *Cfg* s jadilah jenis baru, dengan konstruktor (∃ _) _ ∧_: *IdSet* × *Cfg* × *Fol* → *Cfg* s. Jadi, pola (∃ *X)* $\pi \wedge \varphi$ sesuai dengan istilah (∃ *X)* $\pi \wedge \varphi$ semacam itu *Cfg* s dalam tanda tangan yang diperkaya dan secara timbal balik. Pertimbangkan juga set berikut ini rl rumus, yang disebut *versi simbolis S:*

$$S \text{ s} \quad \{(\exists X)\ \pi_l \wedge \psi \Rightarrow (\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_{r)} / \pi_l \wedge \varphi_l \Rightarrow (\exists Y)\ \pi_r \wedge \varphi_r \in S\}$$

dengan ψ variabel baru semacam *Fol,* dan X variabel baru semacam *IdSet.*

Saat menulis ulang dengan aturan seperti itu di atas sebuah pola, variabel ψ cocok dengan kondisi pola dan diperkuat dengan ketentuan aturan $\phi_{aku,}$ $\phi_{r;}$ dan variabel yang dihitung dari rumus RL ditambahkan ke pola, kemungkinan setelah mengganti nama untuk menghindari variabel yang sama dikuantifikasi dua kali.

Contoh 6. Aturan di Contoh 5 memiliki satu-satunya syarat [3] n :: m :: stk '= stk. Dalam versi simbolik aturan, ini dimasukkan ke dalam "kondisi jalur" f:

```
rl sort (leq) st (t, f, stk) =>
        (min (leq) ;;
            (do x: = pop in (sort (leq) ;; push (x)))) st (t + 3, f dan (n :: m :: stk '=== stk),
stk)
jika n: = freshVar (t) / \ m: = freshVar (t + 1) / \ stk ': = freshStkVar (t) / \ x: = freshVar (t + 2).
```

Sebagian besar aturan lain dalam bahasa tertanam Maude kami tidak berubah oleh transformasi ini karena mereka tidak terkondisi.
tional (yaitu, kedua kondisi tersebut $\phi_{aku,}$ ϕ_r adalah *benar)* dan tidak (secara eksplisit) tidak memiliki bukti eksistensial. Satu pengecualian penting adalah Maude builtin jika-maka-lain-fi konstruksi yang telah kami gunakan dalam bahasa kami, yang menghindari kebutuhan untuk mendefinisikan instruksi bersyarat bersama dengan ekspresi Boolean yang sesuai, dan memungkinkan kami untuk hanya menggunakan konstruksi yang ada dari host Maude (dalam semangat embedding dangkal). Seandainya kita mendefinisikan instruksi bersyarat, aturan simbolis yang sesuai adalah:

```
sif b lalu m1 else m2 fi st (p, f, stk) => m1 st (p, (f dan b), stk)
sif b lalu m1 else m2 fi st (p, f, stk) => m2 st (p, (f dan Not (b)), stk)
```

Efek global dari aturan tersebut adalah bahwa "kondisi jalur" saat ini f diperkaya dengan kondisi instruksi b atau dengan negasinya Tidak b). Kita gunakan sif (“Symbolic if”), bukan jika untuk menghindari kebingungan dengan bawaan Maude jika-maka-lain-fi konstruksi, dan Tidak alih-alih bawaan tidak untuk menghindari pengurangan Maude dari tidak untuk xor yang membuat kondisi yang dihasilkan tidak dapat dibaca.

Kepentingan global dari konstruksi yang dijelaskan di atas adalah menulis ulang dengan aturan dalam S s mencapai a *simulasi timbal balik* menulis ulang dengan aturan masuk S.

Definisi penulisan ulang kita perlu memperluas kesesuaian ~ = untuk istilah semacam *Cfg* s oleh (∃ *X)* $\pi_1 \wedge \varphi$ ~ = *(∃ X)* π_2 untuk beberapa π_2.

Definisi 7 (*Hubungan* $\Rightarrow_{\alpha S)}$. Untuk α s *(∃ X)* $\pi_l \wedge \psi \Rightarrow (\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_{r)} \in S$ s kami menulis (∃ *X)* $\pi \wedge \varphi \Rightarrow_{\alpha s}$ (∃ *X, Y)* $\pi' \wedge \varphi'$ *jika pola (∃ X)* $\pi \wedge \varphi$ *ditulis ulang oleh* α s *ke (∃ X, Y)* $\pi' \wedge \varphi'$, yaitu, ada substitusi σ' di *Var* ∪ { *X, ψ}* seperti yang σ' ((∃ *X)* $\pi_l \wedge \psi$) ~ = (∃ *X)* $\pi \wedge \varphi$ dan σ' ((∃ *X, Y)* $\pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_{r)} = (\exists X, Y)\ \pi' \wedge \varphi'$.

Lemma 1 ($\Rightarrow_{\alpha s}$ *mensimulasikan* $\Rightarrow_{\alpha)}$. *Untuk semua konfigurasi* $\gamma, \gamma' \in M_{Cfg,}$ *semua pola* ϕ *dengan FreeVars (* ϕ*)* ⊆ *Var b, dan semua penilaian* ρ*, jika* $(\gamma, \rho) / = \phi$ *dan* $\gamma \Rightarrow_\alpha \gamma'$ *lalu di sana ada* ϕ' *dengan FreeVars (* ϕ'*)* ⊆ *Var b seperti yang* $\phi \Rightarrow_{\alpha s} \phi'$ *dan (* γ'*,* $\rho) / = \phi'$.

Akibatnya, setiap eksekusi konkret ($\Rightarrow$ *S)* sedemikian rupa sehingga konfigurasi awal memenuhi pola awal yang diberikan ϕ disimulasikan oleh eksekusi simbolik ($\Rightarrow_{S s)}$ dimulai ϕ. Sebuah simulasi dari *balik* hubungan $\Leftarrow_S$ s oleh relasi $\Leftarrow_S$ memegang:

Lemma 2 ($\Leftarrow_\alpha$ *mensimulasikan* $\Leftarrow_{\alpha s)}$. *Untuk semua pola* ϕ*,* ϕ' *seperti yang* $\phi \Rightarrow_{\alpha s} \phi'$*, untuk semua konfigurasi* $\gamma' \in M_{Cfg}$ *dan penilaian* ρ *seperti yang (* γ'*,* $\rho) / = \phi'$*, ada konfigurasi* $\gamma \in M_{Cfg}$ *seperti yang (* γ*,* $\rho) / = \phi$ *dan* $\gamma \Rightarrow_\alpha \gamma'$.

[3] Persamaan pencocokan tidak dihitung sebagai kondisi karena hanya membantu proses pencocokan.

Jadi, eksekusi simbolik yang pola akhirnya dipenuhi oleh konfigurasi yang diberikan sesuai dengan eksekusi konkret.
diakhiri dengan konfigurasi itu. Perhatikan bahwa simulasi $\Rightarrow_S$ oleh relasi $\Rightarrow_S$ tidak berlaku secara umum: memang, beberapa eksekusi simbolis sama sekali tidak sesuai dengan eksekusi konkret. Eksekusi simbolis seperti itu disebut *tidak layak* dan
terjadi, dalam kerangka kerja kami, saat menulis ulang menghasilkan pola $(\exists X)\ \pi \wedge \varphi$ dengan kondisi yang tidak memuaskan φ.

Gagasan tentang *turunan* adalah tentang penerus simbolik pola berdasarkan aturan:

Definisi 8 (*Derivatif).* Kami mengatur $\alpha(\phi) = \phi'$, dimana ϕ' didefinisikan secara unik (hingga nama variabel) dengan transisi $\phi \Rightarrow_{\alpha_S} \phi'$, dan $S(\phi) = \{\alpha(\phi) \mid \alpha \in S\}$.

Contoh 7. Kami mengilustrasikan eksekusi simbolis pemilahan tumpukan untuk tumpukan, katakanlah, tiga elemen. Ini dicapai dengan perintah Maude yang ditunjukkan di bawah ini:

```
Maude> (search sort (ord) st (0, true, p :: q :: r) =>!
                (tt, st (t: Nat, f: Bool, stk: Stack {X}))
                sedemikian rupa sehingga f: Bool = / = false.)
Solusi 1
f: Bool -> p: X $ Elt <= q: X $ Elt dan q: X $ Elt <= r: X $ Elt dan (...); t: Nat -> 18;

stk: Tumpukan '{X'} -> p: X $ Elt :: q: X $ Elt :: r: X $ Elt
. . .
Solusi 6
f: Bool -> Not (p: X $ Elt <= r: X $ Elt) dan Not (p: X $ Elt <= q: X $ Elt) dan Not (q: X $ Elt <= r: X $ Elt) dan (...);

t: Nat -> 18;
stk: Tumpukan '{X'} -> r: X $ Elt :: q: X $ Elt :: p: X $ Elt
```

Itu Cari perintah menghitung semua istilah yang tidak dapat direduksi (=>!) yang dapat dijangkau dengan menerapkan urutkan (ord) ke keadaan yang terdiri dari pencacah variabel baru yang diinisialisasi ke 0, kondisi jalur awal benar, dan tumpukan tiga elemen p :: q :: r. Selain itu, dalam istilah-istilah tak tersederhanakan yang diperoleh hanya mereka yang memiliki kondisi jalan f yang tidak bernilai Salah disimpan. Kondisi jalur tidak ditampilkan sepenuhnya; kata sambung (...) menunjukkan persamaan yang melibatkan variabel segar (singkatan dari variabel yang dikuantifikasi secara eksistensial) yang diperkenalkan oleh penulisan ulang. Mengukur variabel tersebut secara eksistensial di seluruh kondisi jalur akan menguranginya menjadi fragmen kondisi jalur yang ditampilkan. Ada 6 (= 3!) Solusi seperti yang diharapkan, yang hanya yang pertama dan terakhir yang ditampilkan. Perintah (ord alias <=) antara p, q dan r di kondisi jalur konsisten dengan fakta bahwa tumpukan dalam solusi yang dilaporkan diurutkan.

4. Verifikasi program

Contoh 7 dapat dilihat sebagai bukti dengan eksekusi simbolik bahwa prosedur kami urutkan (ord) mengurutkan tumpukan tiga elemen
wrt hubungannya ord. Untuk tumpukan sejumlah elemen yang berubah-ubah, eksekusi simbolik, meskipun berguna, tidaklah cukup, karena itu sama saja dengan membangun pohon yang tidak terbatas. Pada bagian ini kami menunjukkan bagaimana pohon tak terbatas tersebut dapat dibuat tak terbatas, sehingga mencapai verifikasi formal penuh.

Contoh 8. Kami pertama kali menyajikan contoh untuk membantu intuisi. Kami akan memverifikasi fakta bahwa prosedur tersebut min (leq)
dari Bagian 2 mengubah tumpukan q :: stk mengandung setidaknya satu elemen, ke dalam tumpukan q ':: stk' memiliki elemen yang sama (dengan jumlah kemunculan yang sama) sebagai q :: stk, dan yang elemen pertamanya q ' adalah elemen terkecil (tulis urutannya leq) di tumpukan q ':: stk'.

```
crl min (leq) st (t, true, true, (q :: stk)) => (tt, st (t ', true,

    (leq q 'least (leq, q' :: stk ')) dan sameElements (q :: stk, q' :: stk '), q' :: stk ')))

  jika t ': = freshNatVar (t) / \ q': = freshVar (t) / \ stk ': = freshStackVar (t).
```

Seperti yang telah terlihat di atas, variabel baru dalam kondisi sebenarnya adalah variabel yang dikuantifikasi secara eksistensial, di sini, dari jenis
Nat, X $ Elt dan Tumpukan {X} masing-masing. Fungsinya paling sedikit dan elemen yang sama return, masing-masing, elemen terkecil
membuat pesanan leq dalam satu tumpukan, dan nilai kebenaran sebuah predikat yang menyatakan bahwa dua tumpukan memiliki elemen yang sama dengan jumlah kemunculan yang sama.

Rumus di atas dengan demikian menetapkan properti dari min fungsi. Ternyata itu juga dapat digunakan untuk “membuktikan” dirinya dengan secara simbolis mengeksekusi dirinya sendiri (diubah dengan benar untuk memuaskan Asumsi 3) bersama dengan aturan lain dalam definisi min fungsi. Untuk ini kami telah menambahkan ke konstruktor st semacam itu Negara Bagian {X} komponen keempat (ditempatkan di

posisi kedua, tepat setelah penghitung variabel segar t). Ini adalah Boolean yang "memaksa" penerapan aturan lain dari
min fungsi, sebelum yang terakhir dapat diterapkan. Dengan cara ini, penalaran melingkar yang kejam dapat dihindari. Pembuktiannya sendiri terdiri dari a Cari perintah yang mencoba menemukan contoh yang berlawanan dengan properti: konfigurasi simbolik yang tidak dapat direduksi yang berisi tumpukan yang tidak memiliki bentuk yang diharapkan q '::: stk', atau yang kondisinya tidak sesuai dengan yang diharapkan
kondisi (leq q 'least (leq, q' :: stk ')) dan sameElements (q :: stk, q' :: stk ').

Untuk tumpukan setidaknya dua elemen Cari perintah dan keluarannya adalah:

```
Maude> (search (min (ord) st (0, false, true, p1 :: q1 :: stk1)) =>! Pp
                    sedemikian rupa sehingga tidak ((expectedPattern (pp)) atau tidak
                    (getPathCondition (pp)
                                Menyiratkan
                            expectedCondition (ord, pp, p1 :: q1 :: stk1))).)
Tidak ada solusi.
```

Perintah serupa, dengan keluaran yang sama, digunakan untuk tumpukan satu elemen. Karena solusi sesuai dengan contoh berlawanan, dan tidak ada yang ditemukan, maka rl rumus (diklaim) valid. Tentu saja, ini adalah argumen informal; itu diformalkan di bawah ini. Itu
fungsi expectedPattern, getPathCondition dan expectedCondition didefinisikan secara sama; nama mereka
cukup jelas. Untuk mendapatkan hasil yang diharapkan di atas kita telah menggunakan beberapa persamaan pada tipe builtin sebagai penyederhanaan yang merupakan konsekuensi induktif dari definisi paling sedikit dan elemen yang sama. Hal ini bisa dibuktikan, misalnya dengan menggunakan teorema induktif Maude yang prover ITP. [4]

Di atas Cari perintah sehingga secara simbolis mengeksekusi sisi kiri rumus di bawah bukti, menggunakan aturan yang mendefinisikan min fungsi dan rumus di bawah pembuktian itu sendiri, dimulai dengan yang pertama untuk menghindari penalaran melingkar yang kejam. Dalam prosesnya, file Cari membangun perintah a *pohon* [5] yang simpulnya adalah pola; dan memeriksa kondisi tertentu pada daunnya, yaitu pola tanpa penerus. Perhatikan bahwa jalur program yang tidak layak (dibangun oleh eksekusi simbolik) berhubungan dengan daun yang tidak dapat digunakan pada pohon; karena ini menyiratkan pola lain, verifikasi tetap masuk akal. Kami sekarang menggeneralisasi dan secara resmi menetapkan kelayakan pendekatan dengan mendefinisikan prosedur konstruksi pohon yang sesuai dengan Maude. Cari perintah.

Sebelum kami menjelaskan prosedur, kami memperkenalkan komponennya. Prosedur mengasumsikan satu set rl rumus S *(* aturan semantik dari bahasa pemrograman) dan satu set rl rumus G di bawah bukti, dipanggil *lingkaran* di rl- verifikasi berbasis.

Pesanan parsial <aktif $S \cup G$ Selama eksekusi simbolik, sirkularitas dapat diterapkan secara simbolis "dalam persaingan dengan" aturan dalam semantik. Jika ini adalah kasus, prioritas diberikan pada sirkularitas, karena jika sirkularitas dan aturan dalam semantik dapat diterapkan maka yang terakhir biasanya akan menghasilkan eksekusi simbolis yang tidak terbatas, yang membahayakan penghentian prosedur. Untuk membuat ini tepat kita menggunakan relasi urutan parsial <on $S \cup G$. Kami menggunakan notasi berikut. Membiarkan *lhs (* α*)*

menunjukkan sisi kiri rumus α. Membiarkan $G <S$ menunjukkan fakta bahwa untuk semua $g \in G$ dan $\alpha \in S$, $g < \alpha$, dan *min (<)* menunjukkan <elemen -minimal dari $S \cup G$.

Asumsi 4. Kami mengasumsikan hubungan urutan parsial <on $S \cup G$ memuaskan: $G <S$, dan untuk semua $\alpha' \in S$ dan pasang (γ, ρ), jika
$(\gamma, \rho) \models lhs (\alpha')$ lalu di sana ada $\alpha \in min (<)$ seperti yang (γ, ρ) $\models lhs (\alpha)$.

Ini memberikan prioritas sirkularitas di atas aturan dalam semantik yang dapat diterapkan dalam persaingan dengannya. Dalam contoh kami, prioritas tidak memerlukan urutan parsial karena dalam satu-satunya situasi ketika sirkularitas dapat bersaing dengan aturan lain, aturan terakhir tidak diterapkan dengan menggunakan trik teknis sederhana.

Implikasi antar pola Prosedur konstruksi pohon kami menggunakan tes implikasi antar pola, yang memenuhi definisi berikut.

Definisi 9 (*Implikasi).* Tes implikasi adalah fungsi yang diberikan pola ϕ, ϕ', kembali *benar* jika untuk semua pasangan (γ, ρ), jika
$(\gamma, \rho) \models \phi$ kemudian (γ, ρ) $\models \phi'$.

Pembangunan pohon Kami sekarang siap untuk mempresentasikan prosedurnya Gambar 7 . Prosedur mengambil sebagai masukan rl rumus
$\pi \wedge \varphi \Rightarrow (\exists\ Y)\ \pi' \wedge \varphi'$ dari satu set G dari rl rumus, dan satu set S dari rl rumus dengan urutan <aktif $S \cup G$ seperti yang dibahas sebelumnya i S Di bagian ini. Itu membangun pohon (N, E) dengan N kumpulan node (awalnya terdiri dari pola awal { $\pi \wedge \varphi$} dan nya

- turunan) dan E himpunan edge (awalnya menghubungkan node awal dan node S- turunan). Ini menggunakan dua variabel untuk

[4] Saat ini tersedia di http://maude.cs.uiuc.edu/tools/itp/ .

[5] Secara umum, Cari membangun grafik, dengan mengulang kembali jika simpul yang baru dicapai sama dengan simpul yang sudah ditemukan. Namun dalam kasus kami, node baru selalu berbeda dari yang sudah ditemukan karena penghitung variabel segar yang selalu tumbuh.

0: $G = (N \quad \{\pi \wedge \varphi\} \cup s(\pi \wedge \varphi), E \quad \{\pi \wedge \varphi \overset{\alpha}{\rightarrow} \alpha(\pi \wedge \varphi) \mid \alpha \in S\})$, $Gagal \leftarrow false$, $Baru \leftarrow s(\pi \wedge \varphi)$

1: **sementara** tidak *Gagal* dan $Baru = \emptyset$

2: **memilih** $\phi_n \quad (\exists X_n)\, \pi_n \wedge \varphi_n \in Baru; Baru \leftarrow Baru \setminus \{\phi_n\}$

3: **jika** *tikar* $\vee \; ch = (\pi_n, \pi') = \emptyset$ **kemudian**

4: **jika** $\alpha \in min(<)$ $implikasi(\phi_n, lhs(\alpha)) = benar$ **kemudian**

5: **untuk semua** $\alpha \in min(<)$

6: $Baru \leftarrow Baru \cup \{\alpha(\phi_n)\}; E \leftarrow E \cup \{\phi_n \overset{\alpha}{\rightarrow} \alpha(\phi_n)\}$

7: $N \leftarrow \cup N\, Baru$

8: **lain** $Gagal \leftarrow benar$ **berakhir jika**

9: **elseif** tidak $implikasi(\phi_n, (\exists Y)\, \pi' \wedge \varphi')$ **kemudian** $Gagal \leftarrow benar$ **berakhir jika.**

Gambar 7. Konstruksi pohon. *pertandingan* ~ = () sedang mencocokkan modulo aksioma non-builtin (lih. Bagian 3.2), dan *implikasi()* adalah objek dari Definisi 9 .

mengontrol loop sementara: variabel Boolean *Gagal* (mulanya *Salah)* dan satu set node *Baru* (awalnya berisi S- turunan dari node awal).

Pada setiap iterasi w terdapat modulo matcher ~ = hile loop, sebuah node $(\exists X_n)\, \pi_n \wedge \varphi_n$ diambil dari *Baru* (baris 2) dan diperiksa apakah (lih. Bagian 3.2) dari ϕ π' ke π_n (baris 3). Jika demikian, maka π_n adalah contoh dari dasar pola π', dan prosedur menuju baris 9 untuk memeriksa apakah ϕ_n "Secara keseluruhan" termasuk dalam $(\exists Y)\, \pi' \wedge \varphi'$. Jika tidak demikian, maka ini menunjukkan konfigurasi terminal yang tidak memenuhi sisi kanan rumus di bawah pembuktian; *Gagal* dilaporkan, yang menghentikan pelaksanaan prosedur. Namun, jika pengujian di baris 3 menunjukkan hal itu π_n bukan i S nstance dari pola π', kemudian uji implikasi lain dilakukan (baris 4): apakah terdapat elemen minimal di (yaitu, dari semantik bahasa, atau di antara lingkaran) yang termasuk di sisi kiri ϕ_n. Jika tidak demikian maka prosedur diakhiri lagi dengan *Gagal = benar*.

Namun, jika uji implikasi pada baris 4 berhasil, maka semua penerus simbolis ϕ'_n dari ϕ_n dengan elemen minimal α di <are dihitung, dan tepi dari node saat ini ϕ_n ke setiap node baru, diberi label oleh artefak yang menghasilkannya, dibuat.

Prosedur konstruksi pohon tidak berhenti secara umum, sejak verifikasi rl rumus tidak dapat diputuskan. Namun, jika diakhiri dengan *Gagal = salah:*

Teorema 1 (*Kesehatan). Pertimbangkan satu set* rl *rumus* $S \cup G$. *Jika untuk semua* $g \in G$ *prosedur di* Gambar 7 *diakhiri dengan Fail = false* $S \models G$.

Teorema 1 menggunakan asumsi berikut (dan terakhir) pada rl rumus, di mana untuk pola ϕ notasi ϕ menunjukkan set $\{\gamma \mid (\exists \rho)\, (\gamma, \rho) \models \phi)\}$:

Asumsi 5. Semua aturan $\phi_l \Rightarrow \phi_r \in S$ memiliki properti berikut:

1. untuk semua (γ, ρ) seperti yang $(\gamma, \rho) \models \phi_l$ disana ada γ' seperti yang $(\gamma', \rho) \models \phi_r$.
2. $\phi_l \cap \phi_r = \emptyset$.

Asumsi pertama di atas mengatakan bahwa jika sisi kiri cocok dengan konfigurasi maka tidak ada apa pun di sisi kanan yang mencegah penerapan. Properti ini disebut *definisi baik yang lemah* di [6] dan terbukti ada kondisi yang diperlukan untuk mendapatkan sistem kedap suara rl. Kondisi kedua mengatakan bahwa sisi kiri dan kanan aturan tidak dapat berbagi contoh; ini sangat wajar karena aturan dimaksudkan untuk menggambarkan kemajuan dalam komputasi.

Keterangan 1. Asumsi 5 .1 menyiratkan ϕ_l tidak memiliki konfigurasi terminal.

Untuk menyimpulkan bagian ini kami tunjukkan bahwa fungsinya min untuk tumpukan setidaknya dua elemen (dari bentuk p1 :: q1 :: stk1) memuaskan itu rl spesifikasi (untuk tumpukan satu elemen hasilnya sepele). Ini ditulis sebelumnya di bagian:

```
crl min (leq) st (t, true, true, (q :: stk)) => (tt, st (t ', true,

(leq q 'least (leq, q' :: stk ')) dan sameElements (q :: stk, q' :: stk '), q' :: stk ')))

jika t ': = freshNatVar (t) / \ q': = freshVar (t) / \ stk ': = freshStackVar (t)
```

Itu Cari perintah untuk membuktikan aturan ini (ditafsirkan sebagai file rl rumus) adalah:

```
(cari (min (ord) st (0, false, true, p1 :: q1 :: stk1)) =>! pp sedemikian rupa sehingga bukan ((expectedPattern (pp))
atau
tidak (getPathCondition (pp)
Menyiratkan expectedCondition (ord, pp, p1 :: q1 :: stk1))).)
```

Perintah pertama kali menerapkan aturan dalam definisi min untuk tumpukan dua elemen. Ini dipastikan oleh bendera Boolean yang digunakan untuk mencegah penerapan sirkularitas sejak awal, dan sesuai dengan langkah inisialisasi prosedur konstruksi pohon kami (Gambar 7). Jadi, kumpulan node awal (seperti yang dihitung dengan menulis ulang dalam Full Maude dan sedikit diedit agar terbaca) adalah istilah tunggal:

```
(lakukan v3: = masuk
    sif p1 <= v3 lalu dorong (v3) ;; dorong (p1) lain dorong (p1) ;; dorong (v3) fi)

    snd (min (ord)
      st (4, true, (v0 :: v1 :: vstk0) === (p1 :: q1 :: stk1), q1 :: stk1))
```

Itu adalah, min masih perlu diterapkan secara rekursif ke tumpukan yang lebih kecil q1 :: stk1, dan hasilnya v3 dari memunculkan tumpukan awal p1 :: q1 :: stk1 akan didorong pada hasil panggilan rekursif. Itu snd pair-destructor adalah efek samping dari monad, dan persamaan tumpukan dalam kondisi tersebut diinduksi oleh fakta bahwa aturan diterapkan secara simbolis, yaitu, setelah transformasi aturan yang dijelaskan dalam Bagian 3.2 . The sirkularitas, yaitu di atas rl rumus di bawah bukti, sekarang dapat diterapkan, dan dua pola terminal dihasilkan, yang sesuai dengan variabel x dan y di min sedang didorong ke tumpukan dalam satu urutan atau yang lainnya. Variabel baru telah dihasilkan dalam proses, dan kondisi jalur telah diakumulasikan.

```
(tt, st (vnat4, true, p1 <= v5 dan
        v5 <= least (ord, v5 :: vstk5) dan
        (v4 :: vstk4 === q1 :: stk1 dan (v0 :: v1 :: vstk0) ===
        p1 :: q1 :: stk1 dan elemen yang sama (q1 :: stk1, v5 :: vstk5), p1 :: v5 :: vstk5))
```

```
(tt, st (vnat4, true, Not (p1 <= v5) dan
        v5 <= least (ord, v5 :: vstk5) dan
        (v4 :: vstk4) === q1 :: stk1 dan (v0 :: v1 :: vstk0) ===
          p1 :: q1 :: stk1 dan elemen yang sama (q1 :: stk1, v5 :: vstk5), v5 :: p1 :: vstk5))
```

Dapat dilihat bahwa pola-pola ini memuaskan expectedPattern () (yaitu, kedua tumpukan yang dihasilkan p1 :: v5 :: vstk5 dan v5 :: p1 :: vstk5 tidak kosong). Ini sesuai dengan lompatan pada baris 9 dalam prosedur masuk Gambar 7 . Polanya juga memuaskan getPathCondition () Menyiratkan expectedCondition () (yaitu kondisi jalur menyiratkan tumpukan diurutkan sesuai dengan ord). Jadi, *implikasi* memegang antara masing-masing pola di atas dan pola akhir yang diharapkan, oleh karena itu, prosedur diakhiri dengan *Gagal = salah*. Ini dicerminkan oleh fakta bahwa Cari perintah, yang mengimplementasikan konstruksi pohon, diakhiri dengan Tidak ada solusi.

Menggunakan hasil kesehatan (Th. 1) kita menyimpulkan bahwa fungsinya min memang memuaskannya rl spesifikasi. Alasan serupa memungkinkan untuk membuktikan rumus berikut, yang mengatakan bahwa fungsinya menyortir mengembalikan tumpukan yang diurutkan dengan elemen yang sama sebagai inputnya (predikat isSorted secara ekuivalen ditentukan dalam Maude):

```
crl sort (leq) st (t, true, f, stk) => (tt, st (t ', true, f dan isSorted (leq, stk') and sameElements (stk, stk ')), stk'))

jika t ': = freshNatVar (t) / \ stk': = freshStackVar (t).
```

5. Kesetaraan program

Dengan menggunakan eksekusi simbolik kami dapat memverifikasi program terkait rl rumus. Kami sekarang menunjukkan bahwa eksekusi simbolik dan verifikasi program memungkinkan kami untuk membuktikan kesetaraan antar program juga. Secara khusus, kami tertarik *kesetaraan yang lemah,* yang mengatakan bahwa dua program adalah setara jika setiap kali disajikan dengan input yang sama, jika kedua program dihentikan maka mereka menghitung output yang sama. Jenis kesetaraan ini diadaptasi untuk program deterministik — setiap penghentian komputasi pada masukan tertentu menghasilkan keluaran yang unik.

Kami memformalkan gagasan ini dalam a rl pengaturan, dan menggambarkannya dengan membuktikan kesetaraan urutkan (ord) dan urutkan (mundur (ord)) ;; putaran dimana putaran adalah program yang membalikkan tumpukan dan _ ;; _ adalah operasi pengurutan monadik.

Kami dilambangkan dengan $\Rightarrow^*_S$ penutupan relasi yang refleksif-transitif $\Rightarrow_S$. Kami menulis $\gamma_1 \Rightarrow!$ $_S\ \gamma_2$ kapan pun konfigurasi γ_2 tidak memiliki penerus dalam hubungan tersebut $\Rightarrow_S$. Kami menulis $\phi_1 \Rightarrow!$ $_S\ \phi_2$ kapanpun ϕ_2 berisi konfigurasi terminal seperti itu.

Definisi 10 (*Confluence).* S berfungsi jika untuk semua konfigurasi yang berbeda $\gamma, \gamma_1, \gamma_2$ st $\gamma \Rightarrow^*_S \gamma_1$ dan $\gamma \Rightarrow^*_S \gamma_2$ ada γ' seperti yang $\gamma_1 \Rightarrow^*_S \gamma'$ dan $\gamma_2 \Rightarrow^*_S \gamma'$.

Komentar 2.

Jika S kemudian menjadi lancar untuk semua konfigurasi γ, hubungan $\gamma \Rightarrow!$ $_S\,\gamma'$ mendefinisikan konfigurasi secara unik γ'.

Pernyataan di atas memastikan bahwa definisi berikut ini terdengar kapan pun S apakah lancar (hipotesis yang akan kita asumsikan di sisa bagian ini):

Definisi 11. *terakhir(* γ*) =* γ' kapanpun $\gamma \Rightarrow!$ $_S\,\gamma'$.

Definisi 12 (*Kesetaraan Program).* Pertimbangkan kesetaraan ~ di M_{Cfg}. Untuk konfigurasi γ_1, γ_2, kami menulis $\gamma_1 \equiv \gamma_2$ iff $\gamma_1 \sim \gamma_2$ dan keduanya *terakhir(* γ_1*), terakhir(* γ_2*)* ada menyiratkan *terakhir(* γ_1*)* ~ *terakhir(* γ_2*)*. Relasi ~ adalah kesetaraan program (lemah) jika untuk semua pasangan γ_1, γ_2, $\gamma_1 \sim \gamma_2$ menyiratkan $\gamma_1 \equiv \gamma_2$.

Persamaannya disebut "lemah" karena mengasumsikan keduanya *terakhir(* γ_1*)* dan *terakhir(* γ_2*)* ada. Versi lain dari kesetaraan, yang melonggarkan batasan ini, telah didefinisikan dalam literatur, tetapi kami tidak mempermasalahkannya di sini.

Relasi ~ dan ≡ juga dapat diangkat ke pola:

Definisi 13 (~ *dan* ≡ *pada pola).* Untuk pola ϕ_1, ϕ_2 kami menulis $\phi_1 \sim \phi_2$ (resp. $\phi_1 \equiv \phi_2$) jika untuk semua konfigurasi γ_1, γ_2 dan penilaian ρ seperti yang (γ_1, ρ) / = ϕ_1 dan (γ_2, ρ) / = ϕ_2 hubungan $\gamma_1 \sim \gamma_2$ (resp. $\gamma_1 \equiv \gamma_2$) memegang.

Lemma berikut memberikan kondisi yang cukup untuk kesetaraan pada pola (dan dengan demikian, pada konfigurasi program yang mereka tunjukkan):

Lemma 3. *Pertimbangkan dua pola* ϕ_1, ϕ_2 *seperti yang* $\phi_1 \sim \phi_2$*. Kemudian,* $\phi_1 \equiv \phi_2$ *jika untuk semua pola* ϕ' $_1, \phi'_2$ *seperti yang* $\phi_1 \Rightarrow!$ $_S\,\phi'_1$ *dan* $\phi_2 \Rightarrow!$ $_S\,\phi'_2, \phi'_1 \sim \phi'_2$ *memegang.*

Lemma 3 menyarankan bahwa itu cukup untuk menghitung semua penerus simbolik dari dua pola yang berisi beberapa konfigurasi terminal untuk memeriksa kesetaraannya. Sayangnya pada umumnya terdapat banyak penerus simbolik yang tak terbatas bahkan jika semua konfigurasi program yang ditunjukkan oleh pola diakhiri (yaitu, setiap panjang eksekusi terbatas tetapi himpunan semua panjang eksekusi tidak dibatasi).

Namun, situasi ini dapat diperbaiki jika kita membuktikannya rl properti pada program yang ingin kita buktikan ekivalennya karena kita kemudian dapat menggunakan properti sebagai aturan penulisan ulang untuk mendapatkan perkiraan berlebih dari penerus simbolik yang dapat dijangkau. Lemma berikut meresmikan intuisi ini.

Lemma 4. *Jika* S / = G *dan* $\phi \Rightarrow!$ $_S\,\phi'$ *lalu ada* $\phi'_1, \ldots, \phi'_n$ *seperti implikasi itu (* ϕ', ϕ'_1*) dan* $\phi \Rightarrow!$ $_{S'}\,\phi'_1, \ldots, \phi'_n$ *dengan* S' min (<).

Artinya, cukup mempertimbangkan aturan di min (<), yang menggantikan beberapa aturan di S dengan sirkularitas (terbukti) G .

Buktinya Lemma 3 mudah dilakukan kembali jika terlalu banyak perkiraan $\phi'_1, \ldots, \phi'_n$ $_{saya}$ dari ϕ' *saya (* i = 1, 2) digunakan sebagai pengganti ϕ' $_i$. Dikombinasikan dengan Lemma 4 ini memberi kita pendekatan praktis untuk membuktikan kesetaraan pola $\phi_1 \equiv \phi_2$:

- membuktikan S / = G_{saya} dan tentukan hubungan < *saya,* untuk i = 1, 2;
- hitung semua penerus simbolis $\phi'_1, \ldots, \phi'_n$ $_{saya}$ dari ϕ_{saya} seperti yang $\phi_{saya} \Rightarrow!$ $_{S'}\,\phi'_1, \ldots, \phi'_n$ *saya,* untuk i = 1, 2;
- untuk setiap pasang penerus tersebut $\phi'_1, \ldots, \phi'_n$ $_1$ dari ϕ_1 dan $\phi'_1, \ldots, \phi'_n$ $_2$ dari ϕ_2, menunjukkan $\phi'_1, \ldots, \phi'_n$ $_1 \sim \phi'_1, \ldots, \phi'_n$ 2.

Kembali ke contoh kami tentang membuktikan kesetaraan komputasi urutkan (ord) dan urutkan (mundur (ord)) ;; putaran sebuah tumpukan: kesetaraan ~ adalah kesetaraan tumpukan yang diberikan ke dua program, dan elemen minimal dari relasinya < terdiri dari formula tunggal yang dibuktikan di akhir Bagian 4 :

```
crl sort (leq) st (t, true, f, stk) => (tt, st (t ', true, f dan isSorted (leq, stk') and sameElements (stk, stk ')), stk'))

jika t ': = freshNatVar (t) / \ stk': = freshStackVar (t).
```

Untuk memprioritaskannya, kami menghapus semua definisi aturan lainnya menyortir; ini bagus karena jika kita menerapkan prioritas, misalnya, di metalevel, aturan yang dihapus tidak akan dijalankan. Dalam modul yang dihasilkan, kami menambahkan kondisional baru aturan (p1, p2) => (p'1, p'2) jika p1 => p'1 / \ p2 => p'2 menyatakan bahwa sepasang pola (p1, p2) ditulis ulang satu langkah ke pasangan pola lainnya (p'1, p'2) kapan pun pola penulisan ditulis ulang berdasarkan komponen, mungkin dalam beberapa langkah; dan kami gunakan Cari:

```
(search ((sort (ord) st (0, true, true, stk)),
                (sort (reverse (ord)) ;; rev) st (1, true, true, stk)) => *
```

((tt, st (t1, f'1, f1, stk1), (tt, st (t2, f'2, f2, stk2))) sedemikian rupa sehingga tidak ((f1 dan f2) Menyiratkan (stk1 === stk2)).)

Tidak ada solusi.

Artinya, dimulai dari pasangan pola awal dimana komponen berisi tumpukan yang sama stk, kami mencari satu-satunya bentuk pasangan pola yang dapat dijangkau yang mungkin berisi konfigurasi terminal *dan* yang hubungannya dengan kondisi jalan f1 dan f2 tidak *tidak* menyiratkan kesetaraan tumpukan yang dihasilkan stk1 === stk2. Sejak Cari perintah tidak mengembalikan solusi, kami mendapatkannya di semua pola (dengan tumpukan stk1, stk2) mungkin berisi konfigurasi terminal dan dapat dijangkau dari pola awal (keduanya berisi tumpukan yang sama stk) harus menjadi kasus yang kesetaraan

stk1 === stk2 memegang, yang, dengan hasil yang ditetapkan di bagian ini, membuktikan kesetaraan yang lemah dari program kami.

Keterangan 3. Kami mulai dengan 0 di salah satu program dan 1 di program lainnya untuk menghindari interferensi pada nama variabel yang dibuat selama eksekusi. Sesuai pembahasan di atas tadi Cari perintah hanya ada satu aturan yang mendefinisikan menyortir disebut — sirkularitas, direproduksi di atas — yang tidak memiliki panggilan rekursif dan hanya membuat dua variabel — a Nat dan a Tumpukan —Menggunakan bilangan asli yang disediakan dalam input. Dengan memulai dengan bilangan asli yang berbeda, empat nama variabel yang dihasilkan berbeda, dengan demikian, gangguan nama dapat dihindari.

Seperti dalam kasus verifikasi program, kami telah menggunakan di sini fungsi yang didefinisikan secara ekuivalen pada jenis bawaan Tumpukan {X}, bersama dengan persamaan untuk penyederhanaan, yang merupakan konsekuensi induktif dari definisi fungsi. Sebenarnya pembuktian mereka menggunakan, misalnya, penguji teorema ITP Maude tetap menjadi masalah pekerjaan masa depan.

6. Kesimpulan dan pekerjaan masa depan

Dalam makalah ini kami telah menunjukkan bahwa Maude adalah kerangka kerja all-in-one di mana seseorang dapat menulis, menjalankan, dan memverifikasi program-program imperatif fungsional tingkat tinggi. Kami menggunakan sistem modul Maude (Penuh) dan fitur reflektif untuk menggabungkan fungsi tingkat tinggi dan monad status di Maude, dengan demikian memperoleh embedding dangkal dari bahasa fungsional imperatif tingkat tinggi yang diinginkan. Kami kemudian mengadaptasi teknik yang ada untuk eksekusi simbolik dan verifikasi Reachability-Logic (saat ini digunakan pada dasarnya untuk definisi bahasa yang tertanam dalam) untuk program sampel dalam bahasa yang tertanam dangkal yang dihasilkan. Kami juga telah menunjukkan bahwa kesetaraan program dapat dibuktikan dengan menggunakan eksekusi simbolik dan verifikasi program.

Ada beberapa perkembangan yang kami tinggalkan sebagai pekerjaan masa depan. Pertama, properti khusus domain pada bawaan (konsekuensi induktif dari definisi) yang kita gunakan sebagai aturan penyederhanaan dapat dibuktikan menggunakan Maude's Inductive Theorem Prover ITP, sehingga menghilangkan kebutuhan untuk menegaskannya sebagai aksioma. Masalah teknis terkait interaksi antara Full Maude dan ITP perlu diselesaikan terlebih dahulu.

Kedua, pendekatan kami saat ini untuk verifikasi rumus RL (menggunakan perintah pencarian Maude di tingkat dasar) menggunakan beberapa trik teknis ad-hoc untuk mengimplementasikan prosedur suara yang terbukti secara formal yang kami sajikan di makalah. Prosedur itu sendiri membutuhkan implementasi meta-level. Karena sudah di Full Maude, implementasi meta-level berada pada level meta-meta sehubungan dengan mesin penulisan ulang Maude, yang mungkin menimbulkan pertanyaan tentang efisiensi yang perlu ditangani.

Ketiga dan terakhir, fungsi tingkat tinggi dan monad negara juga dapat digunakan dalam perkembangan Maude lainnya. Kami sedang menyelidiki perpanjangan modul berorientasi objek Maude saat ini, di mana kelas dan objek akan memiliki metode aktual (yaitu, konstanta tipe orde tinggi) dan metode akan memiliki kode penting.

Lampiran A. Bukti dari Bagian 3

Bagian ini didedikasikan untuk pembuktian Lemmas 1 dan 2 mengenai simulasi antara hubungan transisi konkret dan simbolik. Membuktikan Lemma 1 sebenarnya cukup menantang. Pembuktiannya meliputi beberapa langkah:

1. menetapkan hasil umum tentang penyatuan dengan pencocokan; membangun simulasi timbal balik antara $\Rightarrow_{S_s}$ dan hubungan lain $\Rightarrow_s$
2. $\Rightarrow_{S_s}$ dijabarkan; S, di mana detail tentang menulis ulang dengan
3. membuktikan simulasi antar relasi $\Rightarrow_S$ dan $\Rightarrow_s$ S dalam dua langkah (dan menggunakan hasil yang disebutkan di atas pada penyatuan dengan mencocokkan)
   (a) ketika relasi didefinisikan oleh rl formula tanpa pembilang; (b) saat relasi didefinisikan oleh rl rumus dengan kuantifer.

Jadi $\Rightarrow_{S_s}$ mensimulasikan $\Rightarrow_s$ S, yang mensimulasikan $\Rightarrow_S$. Dengan transitivitas, $\Rightarrow_{S_s}$ mensimulasikan $\Rightarrow_S$, yang mana Lemma 1 mengatakan (aturan-oleh-aturan). Di sisi lain, membuktikan Lemma 2 relatif mudah. Buktinya ditunjukkan di akhir bagian ini.

Uni fi kasi dengan mencocokkan Hasil ini (Lemma 6 di bawah) menunjukkan bahwa, dalam kondisi tertentu (Asumsi 2), penyatuan dapat dilakukan dengan pencocokan.

Definisi 14. Untuk penilaian $\rho: Var \to M$ dan substitusi $\sigma: X \to T(Y)$, kami menulis $\rho \prec \sigma$ jika $\rho \restriction_X = (\rho \circ \sigma) \restriction_X$.

Lemma 5. *Membiarkan σ_1: $X \to T(Y)$ dan σ_2: $Z \to T(X)$. Jika $\rho \prec \sigma_1$ dan $\rho \prec \sigma_2$ kemudian $\rho \prec \sigma_1 \circ \sigma_2$.*

Bukti. Membiarkan $z \in Z$ dipilih secara sewenang-wenang. Kami harus menunjukkan itu $\rho(z) = \rho(\sigma_1(\sigma_2(z)))$.

Pertama, t_2 $\sigma_2(z)$ adalah istilah berakhir $T(X)$, dan t_1 $\sigma_1(\sigma_2(z))$ adalah istilah berakhir $T(Y)$, diperoleh dengan mengganti di t_2 setiap $x \in FreeVars(\prec t_2)$ oleh $\sigma_1(x) \in T(Y)$.

Dari $\rho\ \sigma_1$ kami tahu itu $\rho(x) = \rho(\sigma_1(x))$ untuk semua $x \in FreeVars(t_2)$. Jadi, $\rho(t_2)$, diperoleh dengan mengganti di t_2 fungsi simbol f oleh mereka ρ- penafsiran f_ρ dan variabel dalam $FreeVars(t_2)$ oleh mereka ρ- penilaian $\rho(x)$, juga sama dengan nilai yang diperoleh dengan mengganti fun = simbol ktion menurut mereka ρ- penafsiran *dan variabel* $x \in FreeVars(t_2)$ *oleh* $\rho(\sigma_1(x))$. Tapi yang terakhir nilai persis = $\rho(\sigma_1(t_2))$ $\rho(\sigma_1(\sigma_2(z))$.

Jadi, $\rho(t_2)$ $\rho(\sigma_1(\sigma_2(z))$, dan dari $\rho \prec \sigma_2$ kami dapatkan $\rho(z) = \rho(\sigma_2(z)) = \rho(t_2)$. Kesimpulannya mengikuti transitivitas dari persamaan.

Kemtandai 4. Domain dan codomain substitusi σ_1, σ_2 di Lemma 5 dipilih sehingga komposisi $\sigma_1 \circ \sigma_2$: $Z \to$ kapal selam ′ tititusi σ_1: $X \to T(Y)$ dan σ_2: $Z \to T(X')$ dengan $X = T'(Y)$ didefinisikan dengan baik. Lemma bisa b ′ e digeneralisasikan untuk $X' X \backslash X$ sebagai berikut: pertimbangkan substitusi σ_1' yang th sebuah domain $X \cup X$, yang mana σ_1' diperpanjang sebagai identitas pada variabel di $_1 \circ \sigma_2$. Sekarang dapatkan sepele $\rho\ \sigma_1$, dan, bersama dengan hipotesis $\rho\ \sigma_2$, Lemma 5 memberi kami $\rho\ \sigma_1\ \sigma_2$, yaitu, $\rho \prec$, mondar-mandir $\prec ' \circ$ $\sigma_1 \circ \sigma_2$ sejak kita didefinisikan $\sigma_1 \circ \sigma_2$ $\sigma_1' \circ \sigma_2$. Jadi, dengan definisi umum di atas untuk komposisi substitusi, Lemma 5 memegang.

Lemma 6 (*Unification by Matching). Untuk semua suku non-variabel $t \in T(Var_b)$, istilah linier $t' \in T|_{b(}Var)$, dan semua penilaian ρ: $Var \to M$ seperti yang $\rho(t) = \rho(t')$, ada substitusi σ sedemikian rupa sehingga $t =_{SEBUAH} \sigma(t')$ dan $\rho \prec \sigma$.*

Bukti. Membiarkan D menjadi model awal dari sub teori bawaan b. Membiarkan *SEBUAH* menjadi himpunan aksioma non-builtin, yang dengan asumsi di kertas dikenal linier, teratur dan tidak runtuh. Catat dulu model awal itu M adalah isomorp $T(|_{b)(}D)/SEBUAH$, yaitu, kelas kesetaraan modulo *SEBUAH* istilah dasar di mana satu-satunya subterms dari jenis bawaan berada D hic to . Ini karena D itu sendiri adalah himpunan kelas kesetaraan istilah dasar di atas b sehubungan dengan set B dari aksioma b, jadi, $T(|_{b)(}D)/SEBUAH$ berjumlah hasil bagi pertama T oleh B dan kemudian *SEBUAH*, yaitu, membagi T oleh $SEBUAH \cup B$ sejak $SEBUAH \cap B = \emptyset$ (karena berlaku untuk istilah yang berbeda sama sekali).

Equa ˆlit ˆ y $\rho(t) = \rho(t')$ tersirat ˆ s itu untuk semua $\hat{t} \in \rho(t)$ disana ada $\hat{t}' \in \rho(t')$ seperti yang $\hat{t} =_{SEBUAH} \hat{t}'$. Kami memperbaiki persyaratan seperti itu semaunya $t, t' \in T|_{b)(}D)$. Dari $t' \in \rho(t')$ kami mengerti t' diperoleh dari $t' \in T(|_{b)(}Var)$ dengan mengganti variabel $x_1 \ldots x_n \in FreeVars(t')$ dengan perwakilan $\hat{\rho}(x_1) \ldots \hat{\rho}(x_n)$ ĉ dipilih, masing-masing, $\rho(x_1) \ldots \rho(x_n)$, yaitu, $t' = c[x_1 \ldots x_n]$ dan $\hat{t}' = [c\ \hat{\rho}(x_1) \ldots \hat{\rho}(x_n)]$ untuk beberapa konteks c. Relasi $t = '_{SEBUAH}\ t$ diperoleh dengan menggunakan sejumlah kecil aksioma dalam *SEBUAH*.

1. Pertama kita pertimbangkan kasus ketika tidak ada aksioma yang diterapkan: $\hat{t} = \hat{t}'$, yaitu, kedua istilah tersebut sama secara sintaksis. Kami membuktikan dengan induksi pada posisi (string di atas bilangan asli N) itu (♠) *posisi apapun ω di t' juga merupakan posisi di t, dan jika t'_ω adalah posisi non-variabel maka t_ω dan simbol fungsi atas t'_ω dan T_ω bertepatan.*
   (a) dalam kasus dasar ω adalah string kosong yang jelas merupakan posisi t. Sejak t, t' bukan variabel yang mereka miliki beberapa simbol teratas f, masing-masing g. Dari $\hat{t} = \hat{t}'$ kita mendapatkan $f = g$ yang membuktikan kasus dasar. (b) fo r langkah induktif: biarkan ω menjadi posisi di t' panjangnya $k+1$. Jadi, ada posisi ω' panjangnya k, dimana $t'_{\omega'}$ memiliki bentuk $f(T_1, \ldots T_q)$ dimana $q > 0$ dan f adalah simbol fungsi non-builtin. Dengan hipotesis induksi posisi ω' juga merupakan posisi t. Sekarang, $t_{\omega'}$ tidak bisa menjadi variabel; jika tidak subterm $\hat{t}_{\omega'}$ di $\hat{t}$ akan memiliki jenis bawaan, yang tidak bisa sy = kesetaraan taktis t ˆ b ˆ ê sama dengan the non-builtin $\hat{t}'_\omega$. Jadi, $t_{\omega'}$ adalah dari fôrm $g(T_1, \ldots T'_r)$ untuk beberapa $r \geq 0$. Sekarang, dimana t memiliki simbol atas g dan t memiliki simbol atas f: kami dapatkan $f = g$ dan ˆ $\hat{r} =$ ˆ $\hat{q}' > 0$. Dalam part = ic ˆ ûlar, posisinya ω panjangnya $k+1$ juga merupakan posisi t. Jadi, t_ω ada, dan begitu juga $\hat{t}_\omega$, dan t menyiratkan t_ω t'_ω. Menggunakan yang sama ′ penalaran seperti dalam kasus dasar kami memperoleh jika t'_ω bukan variabel begitu t_ω dan simbol fungsi atas dari posisi bertepatan, yang membuktikan langkah induktif juga (♠) posisi di t.

   Sejak t Kami membangun substitusi σ dengan pemetaan x_{saya} untuk $t_{\omega(x_i)}$ untuk $saya$ 1,..., n —Itu adalah substitusi, sejak itu t' linier — dan kami dapatkan $\sigma(t') = t$. Apalagi dari $\hat{t}' = c[\hat{\rho}(x_1) \ldots \hat{\rho}(x_n)]$ kami dapatkan $\hat{\rho}(x) = \hat{t}'_{\omega(x_{saya})} = \hat{t}_{\omega(x_i)} \in \rho(t_{\omega(x_i)})$ lalu $\rho(x_i) = [\hat{\rho}(x_i)]_A = [\hat{t}_{\omega(x_i)}]_A = \rho(t_{\omega(x_i)}) = \rho(\sigma((x_i)) = (\rho \circ \sigma)(x_i)$ yang membuktikan $\rho \prec \sigma$.

2. Selanjutnya, kami mempertimbangkan kasus di mana salah satu aksioma, katakanlah, $u = v$ terlibat tepat sekali dalam pembentukan $\hat{t} =_{SEBUAH} \hat{t}'$. Oleh kami asumsi aksi {oms A, u dan v adalah istilah linier yang hanya berisi operasi non-builtin, dan memiliki himpunan yang sama variabel bebas, katakanlah, $y_1, \ldots, y_m$. Kami ss ada substitusi μ $y_1, \ldots, y_m\}$ Sebuah → ume tanpa batasan keumuman itu $y_{saya} \notin vars(t, t')$ untuk $i = 1, \ldots, m$. Jadi, $T(|_{b)(}D)$ dan posisi bersama ω dari $\hat{t}$ dan $\hat{t}'$ seperti yang:

   - persyaratan $\hat{t}, t'$ sama kecuali subterms mereka pada posisi ω, yang dinyatakan sebagai $\hat{t}[w] = \hat{t}'[w]_\omega$, dimana w adalah variabel baru. ω

- axio tersebut ˆ m̂ $u = v$ dengan $\mu(u) = t'$ ω, $\mu(v) = t$ˆ penggantian persamaan ini diperoleh pada posisi tiler, yaitu $\mu(u) = \hat{t}\ \omega$ atau, secara simetris, $\mu(v) = \hat{t}\ \omega$, dan kedua istilah tersebut u, v memiliki properti yang sama, kita dapat mengasumsikan kasus pertama: $\mu(u)\ t\ \omega$ dan $\mu(v)\ t\ \omega$.

U Nyanyikan persamaan yang ditetapkan di atas hasil bukti sebagai berikut: fakta itu $\hat{t}\ \omega = \mu(u)$ dan $\hat{t}\ \omega \in T(\Sigma_{b)}(D)$, dan fakta itu u linear dan tidak memiliki subterms bawaan lainnya

- daripada variabel, ijinkan kita untuk menerapkan = alasan yang sama seperti pada Item 1, dan mendapatkan substitusi σ_u: $FreeVars(u) \to T(FreeVars(t\ \omega))$ seperti yang $\sigma_{kamu}(u)\ t\ \omega$. Selanjutnya, kami = perhatikan bahwa lemma kami ∈ memegang atau tidak indepe ≺ ndently dari nilai ρ di *Var Gratis* = *Va*ˆ *r*' *s (t, t)*. Jadi kita bisa ' menganggap $\rho(y)\ \rho(\sigma_{kamu}(y))$ untuk semua y *FreeVars (u)*, yaitu, $\rho\ \sigma_u$.
- menggunakan $\mu(v)\ t\ \omega$ dan fakta itu $t\ \omega$ linier dan $\mu(v) \in T(\Sigma_{b)}(D)$, kami menerapkan kembali alasan yang sama seperti di Item 1 dan dapatkan σ_v: lebih *Fr [eeV] ar* *s (t* | *FreeVars (t'* $\omega)) \to T(FreeVars(v))$ seperti yang $\sigma(t'$ $v\ \omega) = v$ dan $\rho \prec \sigma_v$ (dengan memperluas σ_v identitas
- dari $tw\ \omega = tw[\omega,]$ dengan menggunakan sekali lagi alasan di Item 1, kita mendapatkan bahwa e ' posisi [ch] ω' dari t' [*w], ω*, adalah juga posisi $tw\ \omega$ dan jika ω bukan posisi variabel maka simbol atas ($tw\ \omega)\ \omega'$ dan ($tw\ \omega)\ \omega'$ bertepatan. []
  Kami membangun sebagai ubs titusi σ' lebih { *w} FreeVars (t')* \ *FreeVars (t'* $\omega)$ dengan $\sigma'(w) = w$, seperti yang $\sigma'(t'[w]\omega) = t[w]$ ω dan ≺ $\rho'\ \sigma'$
- $(' \sigma[^c u]\sigma =)(t'[^c d]\omega) = \sigma_{kamu}(t[\sigma'] = \sigma_v(t$ [(lagi, σ' adalah identitas di luar domainnya) $\omega) = t[\sigma_{kamu}(u)]\ \omega = t[t\ \omega]\ \omega = t;$
- *tu* ω SEBUAH televisi $\overline{\omega}$ ' [*t* $\sigma_v(t\ \omega)\ \omega$), jadi, $((\sigma_u \circ \sigma') \circ \sigma_v)(t') =$ SEBUAH *t*.
- akhirnya, biarkan σ $((\sigma_u \circ \sigma') \circ \sigma_v)$; kami telah memperoleh $\sigma(t') =$ *SEBUAH t* di atas, dan $\rho \prec \sigma$ mengikuti dari $\rho \prec \sigma_v$, $\rho \prec \sigma_{kamu}$, $\rho \prec \sigma'$ dan Keterangan 4 .

3. Masih perlu dipertimbangkan kasus saat mo ˆ re th ˆ ân satu ax .iom̂ saya ' s terlibat dalam̂ relasi $\hat{t} =$ *SEBUAH* $\hat{t}'$. Jadi, ada aksioma *Sebuah* $_1, \ldots,$ *Sebuah* $_p$ ($p > 1$) dan istilah dalam $T(\Sigma_{b)}(D)$: t_0 $t, t_1, \ldots, t_p$ t seperti yang t tˆ $_0$ = *Sebuah* $_1$ tˆ $_1 \cdots$ = *Sebuah* $_p$ tˆ $_p$ t. Sekarang, untuk masing-masing itu *tanah* istilah t_1 ˆ, ˆ. . . , t_{p-1} dalam urutan kami membangun istilah yang sesuai *dengan variabel* $t_1, \ldots, t_{p-1}$, seperti yang t_{saya} diperoleh dari t_{saya} dengan mengganti konstanta dalam D dengan ˆ *segar* variabel dalam *Var b*. Jadi, $t_1 \ldots t_{p-1}$ linier dan milik $T(\Sigma_{b)}(Var\ b)$ (sejak istilah dasar yang sesuai t_{saya} berada di $T(\Sigma_{b)}(D))$. Ingat juga itu $t' \in T(\Sigma_{b)}(Var)$ adalah linier dengan hipotesis. Jadi, kita bisa r = appl berulang kali ≺ y reaso = 1,. . . , p ing di Item 2 dan = memperoleh substitusi σ_{saya}: $FreeVars(t_{saya}) \to T(FreeVars(t_{saya-1}))$ sukses ≺ h itu $\sigma_{saya}(t_i)$ *A* t_{saya-1} dan $\rho\ \sigma_{saya}$, untuk *saya* . Dengan σ $\sigma_p \circ \cdots \circ \sigma_1$ kami mendapatkan $\sigma(t') =$ *SEBUAH t* dan (oleh Keterangan 4) $\rho\ \sigma$. Ini menyimpulkan buktinya.

Sebuah relasi ⇒ s *S dan simulasi timbal baliknya dengan* ⇒ S_s Definisi berikut memperkenalkan relasi ⇒ *S* s pada pola. ini versi dari relasi berbasis penulisan ulang ⇒ S_s di mana detail tentang penulisan ulang (mencocokkan dengan substitusi, menerapkan substitusi ke sisi kanan aturan) diuraikan.

Definisi 15 (*hubungan* ⇒ s $\phi_l \Rightarrow \phi_r$ kacang rl formula dengan ϕ_l $\pi_l \wedge \varphi_{aku}$, ϕ_r $(\exists Y)\ \pi_r \wedge \varphi_r$, dan ($X \cup S$). Membiarkan ϕ jadilah pola dengan $\phi\ (\exists X)\ \pi \wedge \varphi$ dan α $\phi \Rightarrow_s {}_{\alpha}\ \phi$, dengan ϕ' $(\exists X, Y)\ \pi' \wedge \varphi'$, kapanpun ada korek api $\sigma \in FreeVars(\phi)) \cap (Y \cup FreeVars(\phi_{aku}, \phi_r)) = \emptyset$. Kami menulis $\sigma'(\varphi_l \wedge \varphi_r)$ dimana $\sigma' = \sigma \cup Id|_{Var \setminus FreeVars(\pi_l)}$ berbentuk $= (\pi, \pi_l)$ seperti yang $\pi' = \sigma'(\pi_r)$ dan $\varphi' = \varphi \wedge$

Lemma berikut membuat simulasi timbal balik antara ⇒ s *S* dan ⇒ S_s. Ingat pola itu (∃ *X)* $\pi \wedge \varphi$ bisa ekuivalen menjadi se ∃ en sebagai t ∧ erm ~ begitu ∃ f semacam ∧ Cfg_s, dan bahwa kami memperpanjang kongruensi = ~ dari segi jenis *Cfg* untuk istilah semacam Cfg_s, sebagai berikut: $(X)\ \pi_1\ \varphi = (X)\ \pi_2\ \varphi$ iff $\pi_1 = \pi_2$. Untuk kemudahan kita juga mengingat di sini definisi relasi ⇒ S_s (Definisi 7):

untuk α_s $(\exists X)\ \pi_l \wedge \psi \Rightarrow (\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_r) \in S_s$ *kami menulis* $(\exists X)\ \pi \wedge \varphi \Rightarrow_{\alpha_s} (\exists X, Y)\ \pi' \wedge \varphi'$ *jika* $(\exists X)\ \pi \wedge \varphi$ *ditulis ulang oleh* α_s *ke* $(X \exists Y)\ \pi'\ \varphi'$, *yaitu, ada* σ'' *seperti yang* $\sigma''((\exists X)\ \pi_l \wedge \psi) =$ $\sim (\exists X)\ \pi \wedge \varphi$ *dan* $(\exists X, Y)\ \pi' \wedge \varphi' = \sigma''((\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_r)$.

Lemma 7. ($\exists X)\ \pi \wedge \varphi \Rightarrow_s$ *S* $(\exists X, Y)\ \pi' \wedge \varphi'$ *iff* $(\exists X)\ \pi \wedge \varphi \Rightarrow S_s (\exists X, Y)\ \pi' \wedge \varphi'$.

Bukti. (⇒) Menganggap $(\exists X)\ \pi \wedge \varphi \Rightarrow_s$ *S* $_s(\exists X, Y)\ \pi' \wedge \varphi'$, dengan demikian, ada α $\pi_l \wedge \varphi_l \Rightarrow (\exists Y)\ \pi_r \wedge \varphi_r \in S$, σ: $FreeVars(\pi_l) \to T(FreeVars(\pi)) \in$ *pertandingan* $\sim = (\pi, \pi_l)$, dan σ' $\sigma \cup Id|_{Var \setminus FreeVars(\pi_l)}$ kepuasan $\pi' = \sigma'(\pi)$ dan $\varphi' = \varphi \wedge \sigma'(\varphi_l \wedge \varphi_r)$.

Mempertimbangkan α_s $(\exists X)\ \pi_l \wedge \psi \Rightarrow (\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_r) \in S_s$ dan substitusi σ'' $\sigma' \cup (X \leftarrow X) \cup (\psi \leftarrow \varphi)$. Kita h ∃ ave: $\sigma''((\exists X)\ \pi_l \wedge \psi) \cong (\exists X)$ ∧ π φ; (ingat bagaimana kita diperpanjang kongruensi = untuk suku dalam̃ Cfg_s dengan membiarkan (∃ *X)* $\pi_1 \wedge \varphi \sim=$ $(X)\ \pi_2\ \varphi$ d ly if $\pi = \sim \pi_2$); dan kita mempunyai $\sigma''((\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_r)) = (\exists X, Y)\ \sigma'(\pi_r) \wedge (\varphi \wedge \sigma'(\varphi_l \wedge \varphi_r)) = (\exists X, Y)\ \pi' \wedge \varphi'$.

Thus, $(X)\ \pi \wedge \varphi$ $\Rightarrow_{\alpha_s} (X, Y)\ \pi' \wedge \varphi'$, yang membuktikan (⇒) implikasi.

(⇐) Assume saya $(\exists X)\ \pi \wedge \varphi \Rightarrow (\exists X, Y)\ \pi' \wedge \varphi'$. Jadi, ada α_s $(\exists X)\ \pi_l \wedge \psi \Rightarrow (\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_r) \in S_s$ dan substitution σ dari *FreeVars* $((X)\ \pi_l\ \psi)$ untuk istilah berakhir *FreeVars* $((X)\ \pi\ \varphi)$, seperti yang ∃ ∧

1. $\sigma''((\exists X)\ \pi_l \wedge \psi) = \sim (\exists X)\ \pi \wedge \varphi$, dimana kesesuaian ~ = lebih Cfg_s kepuasan $(\exists X)\ \pi_1 \wedge \varphi \sim = (\exists X)\ \pi_2 \wedge \varphi$ jika dan hanya jika $\pi_1 \sim = \pi_2$. *Karena ini benar* ∈ *ty,* σ memiliki bentuk $\sigma' \cup (X \leftarrow X) \cup (\psi \leftarrow \varphi)$ dengan $\sigma' = \sigma \cup Id|_{Var \setminus FreeVars(\pi_l)}$ dan σ: $FreeVars(\pi_l) \to T(FreeVars(\pi))$ *pertandingan* $\sim = (\pi, \pi)$. Kami mendapatkan $\sigma'(\pi_l) \sim = \pi$, $\sigma''(\psi) = \varphi$, dan $\sigma''(X) = X$;

2. $\sigma''((\exists X, Y)\ \pi_r \wedge (\psi \wedge \varphi_l \wedge \varphi_r)) = (\exists X, Y)\ \pi' \wedge \varphi'$, dengan demikian, kami memperoleh $\sigma''(\pi_r) = \sigma'(\pi_r) = \pi'$ dan juga $\sigma''(\psi \wedge \varphi_l \wedge \varphi_r) = \varphi \wedge \sigma'(\varphi_l \wedge \varphi_r) = \varphi'$

Kami dengan demikian telah memperoleh $\pi' = \sigma'(\pi_r)$ dan $\varphi' = \varphi \wedge \sigma'(\varphi_l \wedge \varphi_r)$ untuk beberapa σ' st $\sigma'(\pi_l) \sim = \pi$, yang menyiratkan $(\exists X)\ \pi \wedge \varphi \Rightarrow_S^{\alpha} (\exists X, Y)\ \pi' \wedge \varphi'$. Ini membuktikan ($\Leftarrow$) implikasi dan menyimpulkan lemma.

Simulasi $\Rightarrow_S$ *oleh* $\Rightarrow_S$ Kami telah membentuk simulasi timbal balik $\Rightarrow_S$ dan $\Rightarrow_S$; there remains to prove the simulation of $\Rightarrow_S$ by $\Rightarrow_S$. We will do this in two steps. In the first step we shall consider relations $\Rightarrow_S$ and $\Rightarrow_S$ defined by rl formulas $\phi \Rightarrow \phi'$ in which existential quantifiers in the right-hand side have been replaced by fresh variables. Thus, we have formulas satisfying *FreeVars*(ϕ') *FreeVars*(ϕ), for which the definition of validity becomes:

Definition 16. An rl formula $\phi \Rightarrow \phi'$ with *FreeVars*(ϕ') ⊇ *FreeVars*(ϕ) is valid w.r.t. a set S of rl formulas, written $S \models \phi \Rightarrow \phi'$, if for all (γ_0, ρ) such that $(\gamma_0, \rho) \models \phi$, and all complete paths $\gamma_0 \Rightarrow_S \cdots \Rightarrow_S \gamma_n$, there are $0 \leq i \leq n$ and a valuation ρ' with $\rho'|_{FreeVars(\phi)} = \rho|_{FreeVars(\phi)}$ such that $(\gamma_i, \rho') \models \phi'$.

Remark 5. The new valuation ρ' is required in order to avoid the undesired capturing of additional variables (in *FreeVars*(ϕ') \ *FreeVars*(ϕ)) by the valuation ρ; however, for variables of ϕ the two valuations coincide.

We shall also use the following remarks, which follow from definitions.

Remark 6. If $S \models \alpha$ with $\alpha\ \phi \Rightarrow \phi'$ then for all pairs (γ_0, ρ) such that $(\gamma_0, \rho) \models \phi$, all complete paths $\gamma_0 \Rightarrow_S \cdots \Rightarrow_S \gamma_n$, and all $i \in \{0, \ldots, n\}$ such that $(\gamma_i, \rho') \models \phi'$ for some ρ' with $\rho'|_{FreeVars(\phi)} = \rho|_{FreeVars(\phi)}$: $\gamma_0 \Rightarrow^{\alpha} \gamma_i$.

Remark 7. From Assumptions 1 and 3 it follows that for all rules $\pi_l \wedge \varphi_l \Rightarrow (\exists Y)\ \pi_r \wedge \varphi_r$, $FreeVars(\pi_r \wedge \varphi_r) \setminus FreeVars(\pi_l \wedge \varphi_l) \subseteq Y \subseteq Var_b$.

The next remark regards the definition of the "restricted" relation $\Rightarrow_S$, i.e., generated from unquantified rl formulas with additional variables in their right-hand side as discussed above. It instantiates Definition 15 to this case.

Remark 8 (*restricted relation* $\Rightarrow_S$). Let ϕ be a pattern with $\phi\ \pi \wedge \varphi$ and $\alpha\ \phi_l \Rightarrow \phi_r$ be an rl formula with $\phi_l\ \pi_l \wedge \varphi_l$, $\phi_r\ \pi_r \wedge \varphi_r$, and $FreeVars(\phi) \cap FreeVars(\phi_l, \phi_r) = \emptyset$. We have $\phi \Rightarrow_S^{\alpha} \phi'$, with $\phi'\ \pi' \wedge \varphi'$, whenever there exists a matcher $\sigma \in match_{\sim=}(\pi, \pi_l)$ such that $\pi' = \sigma'(\pi_r)$ and $\varphi' = \varphi \wedge \sigma'(\varphi_l \wedge \varphi_r)$ where $\sigma' = \sigma \cup Id|_{Var \setminus FreeVars(\pi_l)}$.

Lemma 8 (*restricted* $\Rightarrow_S$ *simulates* $\Rightarrow_S$). *For all* $\gamma, \gamma' \in M_{Cfg}$, *pattern* $\phi\ \pi \wedge \varphi$ *with* $FreeVars(\phi) \subseteq Var_b$, *and valuation* ρ, *if* $(\gamma, \rho) \models \phi$ *and* $\gamma \Rightarrow^{\alpha} \gamma'$ *then there is* $\phi'\ \pi' \wedge \varphi'$ *with* $FreeVars(\phi') \subseteq Var_b$ *such that* $\phi \Rightarrow_S^{\alpha} \phi'$ *and* $(\gamma', \rho') \models \phi'$, *for some valuation* ρ' *such that* $\rho'|_{FreeVars(\phi)} = \rho|_{FreeVars(\phi)}$.

Proof. From $\gamma \Rightarrow^{\alpha} \gamma'$ we obtain $\alpha\ \phi_l \Rightarrow \phi_r \in S$ and $\phi_l\ \pi_l \wedge \varphi_l$, $\phi_r\ \pi_r \wedge \varphi_r$, and a valuation μ such that $(\gamma, \mu) \models \pi_l \wedge \varphi_l$ and $(\gamma', \mu) \models \pi_r \wedge \varphi_r$. Since the rules are defined up to the names of their free variables, we can assume $FreeVars(\phi) \cap FreeVars(\phi_l, \phi_r) = \emptyset$. Let then ρ'' be any valuation such that $\rho''|_{FreeVars(\phi)} = \rho|_{FreeVars(\phi)}$, $\rho''|_{FreeVars(\phi_l, \phi_r)} = \mu|_{FreeVars(\phi_l, \phi_r)}$. We thus have

1. $(\gamma, \rho'') \models \pi \wedge \varphi$, hence, (i) $\gamma = \rho''(\pi)$ and (iv) $\rho'' \models \varphi$;
2. $(\gamma, \rho'') \models \pi_l \wedge \varphi_l$, hence, (ii) $\gamma = \rho''(\pi_l)$ and (v) $\rho'' \models \varphi_l$;
3. $(\gamma', \rho'') \models \pi_r \wedge \varphi_r$, hence, (iii) $\gamma' = \rho''(\pi_r)$ and (vi) $\rho'' \models \varphi_r$.

From (i) and (ii) we obtain $\rho''(\pi) = \rho''(\pi_l)$ and, using Lemma 6 (unification by matching) we obtain $\sigma : FreeVars(\pi_l) \to T(FreeVars(\pi)) \in match_{\sim=}(\pi, \pi_l)$ such that $\rho'' \prec \sigma$, that is, $\rho''|_{FreeVars(\pi_l)} = (\rho'' \circ \sigma)|_{FreeVars(\pi_l)}$. Let $\sigma'\ \sigma \cup Id|_{Var \setminus FreeVars(\pi_l)}$. We have (vii) $\rho'' = \rho'' \circ \sigma'$.

Let $\pi'\ \sigma'(\pi_r)$, $\varphi'\ \varphi \wedge \sigma'(\varphi_l \wedge \varphi_r)$, $\phi'\ \pi' \wedge \varphi'$. Using Remark 8 we obtain $\phi \Rightarrow_S \phi'$. Moreover, $FreeVars(\phi') \subseteq Var_b$ since $FreeVars(\phi') = FreeVars(\sigma'(\pi_r)) \cup FreeVars(\varphi) \cup FreeVars(\sigma'(\varphi_l)) \cup FreeVars(\sigma'(\varphi_r))$, and σ' maps $FreeVars(\pi_l)$ to terms over $FreeVars(\pi) \subseteq Var_b$ and each of the sets $FreeVars(\pi_r) \setminus FreeVars(\pi_l)$, $FreeVars(\varphi_r) \setminus FreeVars(\pi_l)$, which are subsets of $FreeVars(\phi_r) \setminus FreeVars(\phi_l) \subseteq Var_b$, to the identity. Note that Remark 7 was used in the above reasoning.

There remains to find ρ' with $\rho'|_{FreeVars(\phi)} = \rho|_{FreeVars(\phi)}$ such that $(\gamma', \rho') \models \phi'$. We let $\rho'\ \rho''$, which satisfies $\rho''|_{FreeVars(\phi)} = \rho|_{FreeVars(\phi)}$ by construction.

1. From (iii) we have $\gamma' = \rho''(\pi_r)$, and using (vii), $\gamma' = \rho''(\sigma'(\pi_r))$;

2. from (iv) : $\rho'' \models \varphi$;

3. from (v), $\rho'(\varphi_l) = true$. Using (vii), $\rho'((\sigma'(\varphi_l)) = true$, i.e., $\rho'' \models \sigma'(\varphi_l)$;

4. from (vi), $\rho'(\varphi_r) = true$. Using (vii), $\rho'((\sigma'(\varphi_r)) = true$, i.e., $\rho'' \models \sigma'(\varphi_r)$.

The boxed conclusions of items 1-4 imply $(\gamma', \rho'') \models \sigma'(\pi_r) \wedge \varphi \wedge \sigma'(\phi_l \wedge \phi_r)$, i.e., $(\gamma', \rho'') \models \phi'$, which concludes the proof of the lemma.

There remains to prove that the general relation $\Rightarrow_s$ simulates $\Rightarrow_S$.

Lemma 9 ($\Rightarrow_s$ *simulates* $\Rightarrow_S$). *For all* $\gamma, \gamma' \in M_{Cfg}$, *pattern* ϕ $(\exists X)\ \pi \wedge \varphi$ *with* $X \cup FreeVars(\phi) \subseteq Var_b$, *and valuation* ρ, *if* $(\gamma, \rho) \models \phi$ *and* $\gamma \Rightarrow_\alpha \gamma'$ *with* α $\phi_l \Rightarrow \phi_r$, ϕ_l $\pi_l \wedge \varphi_l$, ϕ_r $(\exists Y)\ \pi_r \wedge \varphi_r$, *then there is* ϕ' $(\exists X, Y)\ \pi' \wedge \varphi'$ *with* $X \cup Y \cup FreeVars(\phi') \subseteq Var_b$ *such that* $\phi \Rightarrow_\alpha \phi'$ *and* $(\gamma', \rho) \models \phi'$.

Proof. Consider the unquantified pattern $\pi \wedge \varphi$ and the unquantified rule α' $\pi_l \wedge \varphi_l \Rightarrow \pi_r \wedge \varphi_r$. We have $FreeVars(\pi \wedge \varphi) = X \cup FreeVars(\phi) \subseteq Var_b$ using our lemma's hypotheses. We also have the hypothesis $\gamma \Rightarrow_\alpha \gamma'$, thus, there exist valuations μ, μ' such that $\mu' |_{Var \setminus Y} = \mu |_{Var \setminus Y}$, $(\gamma, \mu) \models \pi_l \wedge \varphi_l$, and $(\gamma', \mu') \models \pi_r \wedge \varphi_r$. Since $FreeVars(\pi_l \wedge \varphi_l) \subseteq (Var \setminus Y)$ we also have $(\gamma, \mu') \models \pi_l \wedge \varphi_l$. Thus, using the unquantified version of α, i.e., α' $\pi_l \wedge \varphi_l \Rightarrow \pi_r \wedge \varphi_r$, we obtain $\gamma \Rightarrow_{\alpha'} \gamma'$.

We can now apply Lemma 8 and obtain a pattern $\pi' \wedge \varphi'$ with $FreeVars(\pi' \wedge \varphi') \subseteq Var_b$ such that $\pi \wedge \varphi \Rightarrow_{\alpha'} \pi' \wedge \varphi'$ and $(\gamma', \rho') \models \pi' \wedge \varphi'$, for some valuation ρ' such that $\rho' |_{FreeVars(\pi \wedge \varphi)} = \rho |_{FreeVars(\pi \wedge \varphi)}$.

Specifically, from the proof of Lemma 7 we have π' $\sigma'(\pi_r), \varphi'$ $\varphi \wedge \sigma'(\varphi_l \wedge \varphi_r)$, with σ' $\sigma \cup Id |_{Var \setminus FreeVars(\pi_l)}$ and $\sigma : FreeVars(\pi_l) \to T(FreeVars(\pi))$ $= (\pi, \pi_l)$. Let ϕ' $(\exists X, Y)(\sigma'(\pi_r) \wedge \varphi \wedge \sigma'(\varphi_l \wedge \varphi_r))$. By Definition 15, $\phi \Rightarrow_{\alpha'} \phi'$.

We first show $FreeVars(\phi') \subseteq FreeVars(\phi)$. We have ϕ $(\exists X)\ \pi \wedge \varphi$ and, since $\phi' = (\exists X, Y)(\sigma'(\pi_r) \wedge \varphi \wedge \sigma'(\varphi_l \wedge \varphi_r))$, $FreeVars(\phi') = (FreeVars(\sigma'(\pi_r)) \cup FreeVars(\varphi) \cup FreeVars(\sigma'(\varphi_l)) \cup FreeVars(\sigma'(\varphi_r))) \setminus (X \cup Y)$.

We have $FreeVars(\sigma'(\pi_r)) = FreeVars(\pi) \cup (FreeVars(\pi_r) \setminus FreeVars(\pi_l))$, and then $FreeVars(\sigma'(\pi_r)) \setminus (X \cup Y) = (FreeVars(\pi) \setminus X) \cup ((FreeVars(\pi_r) \setminus Y) \setminus FreeVars(\pi_l))$. But $(FreeVars(\pi) \setminus X) \subseteq FreeVars(\phi)$ and by Assumption 1, $(FreeVars(\pi_r) \setminus Y) \setminus FreeVars(\pi_l) = \emptyset$. Hence, $FreeVars(\sigma'(\pi_r)) \subseteq FreeVars(\phi)$.

Then, $FreeVars(\varphi) \setminus (X \cup Y) = FreeVars(\varphi) \setminus X \subseteq FreeVars(\phi)$.

Next, $FreeVars(\sigma'(\varphi_l)) \subseteq FreeVars(\sigma'(\pi_l))$ using Assumption 1, and we obtain $FreeVars(\sigma'(\varphi_l)) \subseteq FreeVars(\pi)$, and then $FreeVars(\sigma'(\varphi_l)) \setminus (X \cup Y) = FreeVars(\sigma'(\varphi_l)) \setminus X \subseteq FreeVars(\pi) \setminus X \subseteq FreeVars(\phi)$.

Finally, by Assumption 1 we have $FreeVars(\varphi_r) \subseteq FreeVars(\pi_l) \cup Y$, thus, $FreeVars(\sigma'(\varphi_r)) \subseteq FreeVars(\pi) \cup Y$. We then obtain $FreeVars(\sigma'(\varphi_r)) \setminus (X \cup Y) \subseteq FreeVars(\sigma'(\varphi_r)) \setminus Y \subseteq FreeVars(\pi) \subseteq FreeVars(\phi)$.

The proof of $FreeVars(\phi') \subseteq FreeVars(\phi)$ is now complete.

Since we already obtained $\phi \Rightarrow_{\alpha'} \phi'$ there only remains to prove $(\gamma', \rho) \models \phi'$ and $X \cup Y \cup FreeVars(\phi') \subseteq Var_b$.

We now prove $(\gamma', \rho) \models \phi'$. Using Lemma 8 we obtained above $(\gamma', \rho') \models \pi' \wedge \varphi'$, for some valuation ρ' su $\rho' |_{FreeVars(\pi \wedge \varphi)} = \rho |_{FreeVars(\pi \wedge \varphi)}$. Thus, using the definition of valuation of quantified patterns, we also obtain (γ' ch , ρ' th $|_{a=t}$ $(\exists X, Y)\ \pi' \wedge \varphi'$, i.e., $(\gamma', \rho') \models \phi'$. From $\rho' |_{FreeVars(\pi \wedge \varphi)} = \rho |_{FreeVars(\pi \wedge \varphi)}$ we also have $\rho' |_{FreeVars(\pi \wedge \varphi) \setminus X} = \rho |_{FreeVars(\pi \wedge \varphi) \setminus X}$, that is $\rho' |_{FreeVars(\phi)} = \rho |_{FreeVars(\phi)}$. Using $FreeVars(\phi') \subseteq FreeVars(\phi)$ (established at the beginning of this proof) we obtain $\rho' |_{FreeVars(\phi')} = \rho |_{FreeVars(\phi')}$, which together with $(\gamma', \rho') \models \phi'$ proves $(\gamma', \rho) \models \phi'$.

There only remains to be proved that $X \cup Y \cup FreeVars(\phi') \subseteq Var_b$. We have $X \cup Y \cup FreeVars(\phi') = FreeVars(\pi' \wedge \varphi')$ and by Lemma 8 we have that $FreeVars(\pi' \wedge \varphi') \subseteq Var_b$. This concludes the proof of our lemma.

Now, Lemma 1 is a corollary of Lemmas 9 and 7 .

Simulation of $\Leftarrow_S$ *by* $\Leftarrow_s$ Lemma 2 is the last result in Section 3 .

Lemma 2 ($\Leftarrow_\alpha$ simulates $\Leftarrow_{\alpha_s}$). *For all patterns* ϕ, ϕ' *such that* $\phi \Rightarrow_{\alpha_s} \phi'$, *for all configurations* $\gamma' \in M_{Cfg}$ *and valuations* ρ *such that* $(\gamma', \rho) \models \phi'$, *there exists a configuration* $\gamma \in M_{Cfg}$ *such that* $(\gamma, \rho) \models \phi$ *and* $\gamma \Rightarrow_\alpha \gamma'$.

Proof. Let ϕ $(\exists X)\ \pi \wedge \varphi$ and α $\pi_l \wedge \varphi_l \Rightarrow (\exists Y)\ \pi_r \wedge \varphi_r$. Thus we have $\phi' = (\exists X, Y)\ \sigma'(\pi_r) \wedge (\varphi \wedge \sigma'(\varphi_r) \wedge \sigma'(\varphi_l))$ for some substitution $\sigma' : FreeVars(\pi_l) \to T(FreeVars(\pi))$ (extended to th identity over $Var \setminus FreeVars(\pi_l)$) such that $\pi \sim \sigma'(\pi_l)$.

From $(\gamma', \rho) \models \phi'$ we obtain that there is ρ' with $\rho' |_{Var \setminus (X \cup Y)} = \rho |_{Var \setminus (X \cup Y)}$ such that $\gamma' = \rho'(\sigma'(\pi_r))$ and $\rho' \models (\varphi \wedge \sigma'(\varphi_r) \wedge \sigma'(\varphi_l))$. Let $\gamma \rho'(\pi)$. We thus have $(\gamma, \rho') \models \pi \wedge \varphi$ and then $(\gamma, \rho) \models (\exists X)\ \pi \wedge \varphi = \phi$.

There only remains to prove that $\gamma \Rightarrow_\alpha \gamma'$. From $\gamma = \rho'(\pi)$ and $\pi \sim \sigma'(\pi_l)$ we obtain $\gamma = (\rho' \circ \sigma')(\pi_l)$. From $\rho' \models (\varphi \wedge \sigma'(\varphi_r) \wedge \sigma'(\varphi_l))$ we obtain $(\rho' \circ \sigma') \models \varphi_l$ and thus $(\gamma, \rho' \circ \sigma') \models \pi_l \wedge \varphi_l$. From $\gamma' = \rho'(\sigma'(\pi_r))$ and $\rho' \models \sigma'(\varphi_r)$ we obtain $(\gamma', \rho' \circ \sigma') \models \pi_r \wedge \varphi_r$ which implies $(\gamma', \rho' \circ \sigma') \models (\exists Y)$ r.

Finally, $(\gamma, \rho' \circ \sigma') \models \pi_l \wedge \varphi_l$ and $(\gamma', \rho' \circ \sigma') \models (\exists Y)\, \pi_r \wedge \varphi_r$ and $\alpha\ \pi_l \wedge \varphi_l \Rightarrow (\exists Y)\, \pi_r \wedge \varphi_r$ together mean $\gamma \Rightarrow_{\alpha} \gamma'$, which concludes the proof.

Appendix B. Proofs from Section 4

We shall use the following notation: $S_0 \models_n G$ whenever for all pairs (γ_0, ρ) such that $(\gamma_0, \rho) \models \phi$, and all complete paths $\gamma_0 \Rightarrow_S \cdots \Rightarrow_S \gamma_n$ *of length at most n,* there exists $0 \le i \le n$ such that $(\gamma_i, \rho) \models \phi'$.

Lemma 10 *(Simulation by Graph). Consider any complete path* $\tau = \gamma_0 \Rightarrow_{\alpha_1} \cdots \Rightarrow_{\alpha_n} \gamma_n$ *with* $\alpha_1, \ldots, \alpha_n \in S$, *s.t.* $(\gamma_0, \rho) \models \pi \wedge \varphi$. *Assume* $S \models_{n-1} G$. *Then, there exist:* $k \ge 0$, *a subsequence* $(0 = i_0 < \cdots < i_k = n)$, *and a path* $\pi \wedge \varphi = \phi_{i_0} \xrightarrow{\alpha_{i_1}} \cdots \xrightarrow{\alpha_{i_k}} \phi_n$ *in the graph constructed by the procedure in* Fig. 7 *such that* $(\gamma_{i_j}, \rho) \models \phi_{i_j}$ *for* $j = 0 \ldots k$.

Proof. We show how to inductively construct the sequence of indices $(0 = i_0 < \cdots < i_k = n)$ and the corresponding path in the graph.

The first index is (by definition) $i_0 = 0$. In this case the path in the graph reduces to the sole node $\pi_{i_0} \wedge \varphi_{i_0} = \pi_0 \wedge \varphi_0 = \pi \wedge \varphi$, and the valuation ρ together with $\gamma_{i_0} = \gamma_0$ obviously satisfies $(\gamma_{i_0}, \rho) \models \pi_{i_0} \wedge \varphi_{i_0}$.

The second index is $i_1 = 1$. In this case the path in the graph reduces to the initial edges $E = \{\pi \wedge \varphi \xrightarrow{\alpha} \Delta_\alpha(\pi \wedge \varphi) \mid \alpha \in S\}$ and the path consists of one transition $\gamma_0 \Rightarrow_\alpha \gamma_1$ for some $\alpha_1 \in S$ with $(\gamma_0, \rho) \models \pi \wedge \varphi$ for some ρ. Then, Lemma 1 ensures $(\gamma_1, \rho) \models \Delta_{\alpha_1}(\pi \wedge \varphi)$, and since our graph contains the edge $\pi \wedge \varphi \xrightarrow{\alpha} \Delta_{\alpha_1}(\pi \wedge \varphi)$ the coverage of paths of length 1 by edges starting in the initial node is settled.

Assume now that we have built the subsequence up to some index $0 < i_m \le n$. Thus, we have built the sequence $(0 = i_0 < \cdots < i_m)$ and the path $(\pi \wedge \varphi =) \phi_0 \xrightarrow{\alpha_{i_1}} \cdots \xrightarrow{\alpha_{i_m}} \phi_m$ satisfying the conclusions of the lemma. If $i_m = n$ the conclusion of the lemma holds directly so we can assume $i_m < n$.

We show how to extend the sequence of indices and the path in the graph. We know that $\phi_{i_m} \triangleq (\exists Z)\, \pi_{i_m} \wedge \varphi_{i_m}$ is a node in the graph and that $(\gamma_{i_m}, \rho) \models (\exists Z)\, \pi_{i_m} \wedge \varphi_{i_m}$. Consider the configuration γ_{i_m} on the sequence τ. Since $i_m < n$ the configuration γ_{i_m} has a successor on τ i.e., there is $\alpha_{i_m} \in S$ such that $(\gamma_{i_m}, \rho) \models lhs(\alpha_{i_m})$. By Assumption 4 on the relation $<$, there exists $\alpha\ \pi_l \wedge \varphi_l \Rightarrow (\exists Y)\, \pi_r \wedge \varphi_r \in min(<)$ such that $(\gamma_{i_m}, \rho) \models \pi_l \wedge \varphi_l$. We distinguish two cases:

- $\alpha \in S$
- $\alpha \notin S$ and thus, $S \models \alpha$; $\alpha\ G$, and we obtain $S \models_n$

Next, using the definition of $\models$ and $\models$ an index, say, $i_{m+1} \le n$, on the (complete) path $\gamma_{i_m} \cdots \gamma_n$ (a nonempty, strict suffix of τ) there exists and ρ' such that $(\gamma_{i_m}, \rho) \models \pi_r \wedge \varphi_r$, that contains γ_{i_m}. Moreover, $i_{m+1} > i_m$ since $\gamma_{i_m} \in \pi_l \wedge \varphi_l$, which by Assumption 5.2 is disjunct from $\pi_r \wedge \varphi_r$.

By Remark 6 we have $\gamma_{i_m} \Rightarrow_\alpha \gamma_{i_{m+1}}$, thus, using Lemma 1 and Definition 8 of derivatives, $(\exists Z)\, \pi_{i_m} \wedge \varphi_{i_m} \Rightarrow_\alpha \Delta_\alpha((\exists Z)\, \pi_{i_m} \wedge \varphi_{i_m})$, and $(\gamma_{i_{m+1}}, \rho) \models \Delta_\alpha((\exists Z)\, \pi_{i_m} \wedge \varphi_{i_m})$. We take i_{m+1} to be the next element of the sequence $(0 = i_0 < \cdots < i_m)$, and extend the path $\pi_0 \models \Delta_\alpha((\wedge \varphi_0 \xrightarrow{\alpha_{i_1}} \cdots \xrightarrow{\alpha_{i_m}} \phi_{i_m} \xrightarrow{\alpha} \phi_{i_{m+1}} \triangleq \Delta_\alpha((\exists Z)\, \pi_{i_m} \wedge \varphi_{i_m})$. From $(\gamma_{i_{m+1}}, \rho)$ we obtain that, with the transition $(\exists Z)\, \pi_{i_m} \wedge \varphi_{i_m} \to \phi_{i_{m+1}}$, where

Thus, we have obtained the next index i_{m+1} in the sequence $(0 = i_0 < \cdots < i_k = n)$ and the next node $\phi_{i_{m+1}}$ in the path $(\pi \wedge \varphi =) \phi_0 \xrightarrow{\alpha_{i_1}} \cdots \xrightarrow{\alpha_{i_n}} \phi_n$ in the graph satisfying all the lemma's conclusions. This completes the inductive construction of the elements whose existence is stated by the lemma.

Theorem 1 (soundness). Consider a set of rl formulas $S \cup G$. If for all $g \in G$ the procedure in Fig. 7 terminates with $Fail = false$ then $S \models G$.

Proof. We prove by induction on n that $S \models_n G$. Let $g\ \pi \wedge \varphi \Rightarrow (\exists Y)\, \pi' \wedge \varphi' \in G$ be arbitrarily chosen, and a complete path $\tau \Rightarrow_{\alpha_1} \cdots \Rightarrow_{\alpha_n} \gamma_n$ with $\alpha_1, \ldots, \alpha_n \in S$, such that $(\gamma_0, \rho) \models \pi \wedge \varphi$ for some valuation ρ. In the base case $n = 0$, the theorem is trivial, since this would mean that γ_0 is terminal, which together with $(\gamma_0, \rho) \models \pi \wedge \varphi$ contradicts Remark 1. By induction hypothesis, $S \models_{n-1} G$. Thus, we can apply Lemma 10 and obtain $k \ge 0$, a subsequence $(0 = i_0 < \cdots < i_k = n)$, and a path $\pi \wedge \varphi = \phi_0 \xrightarrow{\alpha_{i_1}} \cdots \xrightarrow{\alpha_{i_k}} \phi_n = (\exists Z)\, \pi_n \wedge \varphi_n$ in the graph constructed by the procedure in Fig. 7 such that $(\gamma_{i_j}, \rho) \models \phi_{i_j}$ for $j = 0 \ldots k$. We have two cases:

- $match(\pi_n, \pi') = \emptyset$ … $implication(\phi_n, (\exists Y)\, \pi' \wedge \varphi')$ is true, which, by the definition of the implication predicate, implies $(\gamma_n, \rho) \models (\exists Y)\, \pi' \wedge \varphi'$, since $Fail = false$ we have, which proves the induction step in this case.

- *match* ~ =(π_{n}, π_{l}) = ∅: since *Fail* = *false* by construction of the procedure there exists α $\phi_{l} \Rightarrow \phi_{r} \in$ *min(<)* such that *implication(* ϕ_{n}, ϕ_{l}*)* = *true*, which implies (γ_{n}, ρ) |=ϕ_{l}. But this is impossible by Remark 1 .

The induction step and the proof are now complete.

Appendix C. Proofs from Section 5

Lemma 3 . Consider two patterns ϕ_{1}, ϕ_{2} such that ϕ_{1} ~ ϕ_{2}. Then, $\phi_{1} \equiv \phi_{2}$ if for all patterns ϕ'_{1}, ϕ'_{2} such that $\phi_{1} \Rightarrow_{!}$ Ss ϕ'_{1} and $\phi_{2} \Rightarrow_{!}$ Ss ϕ'_{2}, ϕ'_{1} ~ ϕ'_{2} holds.

Proof. Assume (γ_{1}, ρ) |= ϕ_{1} and (γ_{2}, ρ) |= ϕ_{2}. Since ϕ_{1} ~ ϕ_{2} we have γ_{1} ~ γ_{2}. In order to prove $\phi_{1} \equiv \phi_{2}$ we need to show that if both *last(* γ_{1}*)* and *last(* γ_{2}*)* exist then *last(* γ_{1}*)* ~ *last(* γ_{2}*)*. Assume then that both *last(* γ_{1}*)* and *last(* γ_{2}*)* exist. Thus, we have the concrete execution $\gamma_{1} \Rightarrow \alpha_{1} \cdots \Rightarrow \alpha_{m-1} \gamma_{m} =$ *last(* γ_{1}*)* for some $m \geq 1$. By repeatedly applying Lemma 7 we obtain the symbolic execution $\phi_{1} \Rightarrow \alpha_{1}$ s $\cdots \Rightarrow \alpha_{m-1}$ s ϕ_{m} such that (γ_{i}, ρ) |= ϕ_{i} for $i = 1, \ldots, m$. Since in particular (γ_{m}, ρ) |= ϕ_{m} and $\gamma_{m} =$ *last(* γ_{1}*)* is terminal, by setting ϕ'_{1} = ϕ_{m} we obtain $\phi_{1} \Rightarrow_{!}$ Ss ϕ'_{1}. Similarly, we obtain $\phi_{2} \Rightarrow_{!}$ Ss ϕ'_{2}. By hypothesis, ϕ'_{1} ~ ϕ'_{2}, which implies *last(* γ_{1}*)* ~ *last(* γ_{2}*)*. Using Definition 12 , $\gamma_{1} \equiv \gamma_{2}$, and $\phi_{1} \equiv \phi_{2}$ follows by Definition 13 .

Lemma 4 . If S |=G and $\phi \Rightarrow_{!}$ Ss ϕ' then there is ϕ'' such that *implication(* ϕ', ϕ''*)* and $\phi \Rightarrow_{!}$ S's ϕ'' with S' min(<).

Proof. From $\phi \Rightarrow_{!}$ Ss ϕ' we obtain γ' and ρ such that (γ', ρ) |= ϕ' and γ' is terminal. Using Lemma 2 (repeatedly) we obtain an execution $\gamma_{0} \Rightarrow \alpha_{1} \cdots \Rightarrow \alpha_{n} \gamma_{n} = \gamma'$ ($n \geq 0$) such that (γ_{i}, ρ) |=ϕ_{i} where ϕ_{0} ϕ, ϕ_{n} ϕ', and the remaining ϕ_{i} are the intermediary patterns in the symbolic execution $\phi \Rightarrow_{!}$ Ss ϕ' (if any).

We now prove (♦): there exist $k \geq 0$, a subsequence $(0 = i_{0} < \cdots < i_{k} = n)$, and a symbolic execution $\phi = \phi'_{0} \Rightarrow \alpha'_{1}$s $\cdots \Rightarrow \alpha'_{k}$s ϕ'_{k} with $\alpha'_{i} \in S$ such that (γ_{i_j}, ρ) |=ϕ'_{j} for $j = 0 \ldots k$.

The first index is (by definition) $i_{0} = 0$. In this case, the valuation ρ together with γ_{0} obviously satisfies (γ_{0}, ρ) |= ϕ'_{0} = (ϕ =ϕ_{0}). Assume now that we have b have the symbolic execution $\phi'_{0} \Rightarrow \alpha'_{1}$s $\cdots \Rightarrow \alpha'_{m}$s ϕ'_{m} the subsequence ' up to some index $(0 \leq i_{j} < n)$ |= ϕ'_{j} for $j = 0 \ldots m$. In particular, (γ_{i_m}, ρ) |= k, ϕ'_{m}, and we since $i_{m} < n$ the configuration γ_{i_m} has a successor, i.e., there is $\alpha \in S$ such that (γ_{i_m}, ρ) |= *lhs(* α*)*. By Assumption 4 on the relation <, there exists $\alpha' \in$ *min(<)* such that (γ_{i_m}, ρ) |= *lhs(* α'*)*. We distinguish two cases:

- $\alpha' \in S$, and thus, S |= α';
- $\alpha' \in G$, and thus again S |= α' .

Next, using the definition of |= on the (complete) execution $\gamma_{i_m} \cdots \gamma_{n}$ there exists an index, say, $i_{m+1} \leq n$ such that ($\gamma_{i_{m+1}}$, ρ) |= *rhs(* α'*)*. Moreover, $i_{m+1} > i_{m}$ since $\gamma_{i_m} \in$ *lhs(* α'*)* , which by Assumption 5 item 2 is disjunct from *rhs(* α'*)* that contains $\gamma_{i_{m+1}}$.

By Remark 6 we have $\gamma_{i_m} \Rightarrow \alpha' \gamma_{i_{m+1}}$, thus, using Lemma 1 , there exists ϕ'_{m+1} such that ($\gamma_{i_{m+1}}$, ρ) |=ϕ'_{m+1}, obtained the next index i_{m+1} in the sequences $(0 = i_{0} < \cdots < i_{k} = n)$ and $\phi = \phi'_{0} \Rightarrow \alpha'_{1}$s $\cdots \Rightarrow \alpha'_{m}$s ϕ'_{m} ϕ'_{m+1}. Thus, we have w t γ i h (i_{j}, ρ) |= ϕ'_{j} for $j = 0 \ldots m$. In this way the whole sequence can be built, which completes the proof of (♦).

In particular, for $\gamma_{i_k} = \gamma_{n} = \gamma'$ and $\phi'_{k} = \phi''$ we obtain (γ', ρ) |= ϕ'' . Since γ' is terminal we have $\phi \Rightarrow_{!}$ S's ϕ'' . Since we arbitrarily chose γ' and ρ such that (γ', ρ) |=ϕ' and obtained (γ', ρ) |=ϕ'' we also have *implication(* ϕ', ϕ''*)*: the lemma is proved.

References

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, All About Maude—A High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic, Springer-Verlag, 2007.

[2] The Haskell language, http://www.haskell.org . [3] G. Roşu, A. Ştefănescu, Towards a unified theory of operational and axiomatic semantics, in: Proceedings of the 39th International Colloquium on Automata, Languages and Programming, ICALP'12, in: Lecture Notes in Computer Science, vol. 7392, Springer, 2012, pp. 351–363.

[4] G. Roşu, A. Ştefănescu, Checking reachability using matching logic, in: Proceedings of the 27th Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'12, ACM, 2012, pp. 555–574.

[5] G. Roşu, A. Ştefănescu, Ş. Ciobâcă, B.M. Moore, One-path reachability logic, in: Proceedings of the 28th Symposium on Logic in Computer Science, LICS'13, IEEE, 2013, pp. 358–367.

[6] A. Ştefănescu, Ş. Ciobâcă, R. Mereuţă, B.M. Moore, T.F. Şerbănuţă, G. Roşu, All-path reachability logic, in: Proceedings of the Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications, RTA–TLCA'14, in: LNCS, vol. 8560, Springer, 2014, pp. 425–440.

[7] The Coq proof assistant, http://coq.inria.fr .

[8] W. Swierstra, A Hoare logic for the state monad, in: S. Berghofer, T. Nipkow, C. Urban, M. Wenzel (Eds.), Theorem Proving in Higher Order Logics, Proceedings of the 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17–20, 2009, in: Lecture Notes in Computer Science, vol. 5674, Springer, 2009, pp. 440–451.

[9] A. Ştefănescu, D. Park, S. Yuwen, Y. Li, G. Roşu, Semantics-based program verifiers for all languages, in: Proceedings of the 31th Conference on ObjectOriented Programming, Systems, Languages, and Applications, OOPSLA'16, ACM, 2016.

[10] The K semantic framework, http://www.kframework.org .

[11] D. Lucanu, V. Rusu, A. Arusoaie, A generic framework for symbolic execution: a coinductive approach, J. Symb. Comput., https://doi.org/10.1016/j.jsc.2016.07.012 , https://hal.inria.fr/hal-01238696 .

[12] V. Rusu, D. Lucanu, T. Serbanuta, A. Arusoaie, A. Stefanescu, G. Rosu, Language definitions as rewrite theories, J. Log. Algebraic Methods Program. 85 (1) (2016) 98–120, http://dx.doi.org/10.1016/j.jlamp.2015.09.001 .

[13] Ştefan Ciobâcă, D. Lucanu, V. Rusu, G. Rosu, A language-independent proof system for full program equivalence, Form. Asp. Comput. 28 (3) (2016) 469–497, http://dx.doi.org/10.1007/s00165-016-0361-7 .

[14] A. Arusoaie, D. Lucanu, V. Rusu, Symbolic execution based on language transformation, Comput. Lang. Syst. Struct. 44 (2015) 48–71, http://dx.doi.org/10.1016/j.cl.2015.08.004 .

[15] D. Lucanu, V. Rusu, Program equivalence by circular reasoning, Form. Asp. Comput. 27 (4) (2015) 701–726, http://dx.doi.org/10.1007/s00165-014-0319-6 .

[16] J.C. King, Symbolic execution and program testing, Commun. ACM 19 (7) (1976) 385–394.

[17] C.S. Păsăreanu, N. Rungta, Symbolic PathFinder: symbolic execution of Java bytecode, in: International Conference on Automated Software Engineering, ASE'10, ACM, 2010, pp. 179–180.

[18] P. Godefroid, N. Klarlund, K. Sen, DART: directed automated random testing, in: Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, ACM, 2005, pp. 213–223.

[19] K. Sen, D. Marinov, G. Agha, CUTE: a concolic unit testing engine for C, in: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE-13, ACM, New York, NY, USA, 2005, pp. 263–272, http://doi.acm.org/10.1145/1081706.1081750 .

[20] C. Cadar, V. Ganesh, P.M. Pawlowski, D.L. Dill, D.R. Engler, EXE: automatically generating inputs of death, in: ACM Conference on Computer and Communications Security, 2006, pp. 322–335.

[21] J. de Halleux, N. Tillmann, Parameterized unit testing with Pex, in: Tests and Proofs, Second International Conference, in: Lecture Notes in Computer Science, vol. 4966, Springer, 2008, pp. 171–181.

[22] C. Cadar, D. Dunbar, D. Engler, Klee: unassisted and automatic generation of high-coverage tests for complex systems programs, in: Proc. 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08, 2008, pp. 209–224, http://dl.acm.org/citation.cfm?id=1855741.1855756 .

[23] A. Coen-Porisini, G. Denaro, C. Ghezzi, M. Pezzé, Using symbolic execution for verifying safety-critical systems, Softw. Eng. Notes 26 (5) (2001) 142–151, http://doi.acm.org/10.1145/503271.503230 .

[24] J. Jaffar, V. Murali, J.A. Navas, A.E. Santosa, TRACER: a symbolic execution tool for verification, in: Computer Aided Verification—Proceedings of the 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7–13, 2012, in: Lecture Notes in Computer Science, vol. 7358, Springer, 2012, pp. 758–766.

[25] D.A. Ramos, D.R. Engler, Practical, low-effort equivalence verification of real code, in: Proceedings of the 23rd international conference on Computer aided verification, CAV'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 669–685, http://dl.acm.org/citation.cfm?id=2032305.2032360 .

[26] X. Leroy, Formal verification of a realistic compiler, Commun. ACM 52 (7) (2009) 107–115, http://doi.acm.org/10.1145/1538788.1538814 . [27] G.C. Necula, Translation validation for an optimizing compiler, in: M.S. Lam (Ed.), PLDI, ACM, 2000, pp. 83–94.

[28] A.M. Pitts, Operational semantics and program equivalence, in: Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9–15, 2000, in: Advanced Lectures, Springer-Verlag, London, UK, UK, 2002, pp. 378–412, http://dl.acm.org/citation.cfm?id=647424.725796 .

[29] T. Arons, E. Elster, L. Fix, S. Mador-Haim, M. Mishaeli, J. Shalev, E. Singerman, A. Tiemeyer, M.Y. Vardi, L.D. Zuck, Formal verification of backward compatibility of microcode, in: K. Etessami, S.K. Rajamani (Eds.), CAV, in: Lecture Notes in Computer Science, vol. 3576, Springer, 2005, pp. 185–198.

[30] S. Craciunescu, Proving the equivalence of CLP programs, in: P.J. Stuckey (Ed.), ICLP, in: Lecture Notes in Computer Science, vol. 2401, Springer, 2002, pp. 287–301.

[31] O. Strichman, Regression verification: proving the equivalence of similar programs, in: A. Bouajjani, O. Maler (Eds.), CAV, in: Lecture Notes in Computer Science, vol. 5643, Springer, 2009, p. 63.

[32] B. Godlin, O. Strichman, Inference rules for proving the equivalence of recursive procedures, in: Z. Manna, D. Peled (Eds.), Essays in Memory of Amir Pnueli, in: Lecture Notes in Computer Science, vol. 6200, Springer, 2010, pp. 167–184.

[33] K. Bae, S. Escobar, J. Meseguer, Abstract logical model checking of infinite-state systems using narrowing, in: F. van Raamsdonk (Ed.), 24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24–26, 2013, Eindhoven, The Netherlands, in: LIPIcs, vol. 21, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013, pp. 81–96.

[34] C. Rocha, J. Meseguer, C.A. Muñoz, Rewriting modulo SMT and open system analysis, in: Rewriting Logic and Its Applications – 10th International Workshop, WRLA 2014, Held as a Satellite Event of ETAPS, Grenoble, France, April 5–6, 2014, 2014, Revised Selected Papers, 2014, pp. 247–262.

[35] C. Rocha, J. Meseguer, C. Muñoz, Rewriting modulo {SMT} and open system analysis, J. Log. Algebraic Methods Program. 86 (2017) 269–297, http://dx.doi.org/10.1016/j.jlamp.2016.10.001 , http://www.sciencedirect.com/science/article/pii/S2352220816301195 .

[36] A. Arusoaie, D. Lucanu, V. Rusu, A generic framework for symbolic execution, in: 6th International Conference on Software Language Engineering, in: LNCS, vol. 8225, Springer Verlag, 2013, pp. 281–301, also available as a technical report: http://hal.inria.fr/hal-00853588 .

[37] V. Rusu, A. Arusoaie, Proving reachability-logic formulas incrementally, in: D. Lucanu (Ed.), Rewriting Logic and Its Applications – 11th International Workshop, WRLA 2016, Held as a Satellite Event of ETAPS, Eindhoven, The Netherlands, April 2–3, 2016, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 9942, Springer, 2016, pp. 134–151.

[38] G. Roşu, C. Ellison, W. Schulte, Matching logic: an alternative to Hoare/Floyd logic, in: M. Johnson, D. Pavlovic (Eds.), Proceedings of the 13th International Conference on Algebraic Methodology and Software Technology, AMAST '10, in: Lecture Notes in Computer Science, vol. 6486, 2010, pp. 142–162.

[39] G. Roşu, A. Ştefănescu, Matching logic: a new program verification approach (NIER track), in: ICSE'11: Proceedings of the 30th International Conference on Software Engineering, ACM, 2011, pp. 868–871.

[40] G. Roşu, Matching logic — extended abstract, in: Proceedings of the 26th International Conference on Rewriting Techniques and Applications, RTA'15, in: LIPIcs. Leibniz Int. Proc. Inform., vol. 36, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2015, pp. 5–21.