Teori Komposisi Komponen

I.S.W.B. Prasetya[1], S.D. Swierstra[1], dan Belawati H. Widjaja[2]

[1] Instituut van Informatiekunde & Informatica, Universiteit Utrecht.
Email : wishnu@cs.uu.nl dan doaitse@cs.uu.nl.
[2] Fakultas Ilmu Komputer, Universitas Indonesia.
Email : bela@cs.ui.ac.id.
Didukung oleh Graduate Team Research Grant Batch III, URGE Project

Abstrak - **Teori komposisi komponen adalah dasar dari modularitas dalam pengembangan dan verifikasi perangkat lunak. Teori ini terutama digunakan untuk memberikan justifikasi formal untuk konsistensi kerja komponen-komponen yang mudah dipakai ulang (*reusable*). Hambatan terbesar dalam menyusun bukti formal untuk komponen seperti itu adalah konsistensi prilaku temporal yang berhubungan dengan kemajuan (*progress*) sulit dijaga dalam konteks sistem terdistribusi atau paralel. Teori yang ada sifatnya terlalu umum dan tidak memberikan petunjuk yang jelas bagaimana cara memberi batasan (*constraint*) kepada lingkungan (*environment*) sebuah komponen untuk menjaga konsistensi prilaku komponen tersebut di dalam sistem. Teori yang dikembangkan di sini bersifat lebih terbatas. Keumuman (*generality*) sengaja dikorbankan, dan sebagai imbalannya diperoleh hasil yang lebih kuat, sehingga pada prakteknya akan lebih bermanfaat. Teori ini merupakan pengembangan lebih lanjut dari teori serupa yang didasarkan pada *mutual exclusion*. Walaupun dengan *mutual exclusion* bisa didapatkan hasil yang lebih kuat, banyak sistem di lapangan yang sikronisasinya tidak menganut prinsip ini. Contohnya sistem basis data terdistribusi. Versi yang diberikan di sini, diharapkan bermanfaat untuk domain penerapan yang lebih luas.**

Kata kunci : reusable, teori komposisi, mutual exclusion

Makalah diterima [15 Februari 2001]. Revisi akhir [25 April 2001]

1. PENDAHULUAN DAN MOTIVASI

Bayangkan skenario berikut ini. Sebuah perusahaan X ingin membangun sistem basis data terdistribusi. Sistem ini terdiri dari tiga komponen utama, yaitu P, Q, dan R yang bekerja sama secara paralel. Sistem keseluruhan biasa ditulis sebagai $P||Q||R$. Spesifikasi prilaku sistem secara keseluruhan biasanya merupakan serangkaian spesifikasi temporal, berbentuk :

$$P||Q||R \text{ sat } A$$

yang artinya sistem $P||Q||R$ memiliki sifat temporal A. Sifat temporal adalah sifat sebuah sistem selama *run-time*. Misalnya, sistem tersebut tidak pernah mengalami *dead-lock*. Prilaku temporal sangat dominan pada sistem-sistem yang operasinya diharapkan kontinu, misalnya sistem operasi, sistem pengendali lalu lintas, atau server basis data.

Dalam pendekatan tradisional, pengujian dari spesifikasi di atas membutuhkan kode lengkap (*complete code*) dari ketiga komponen yaitu P, Q, dan R. Dalam banyak konteks, hal ini tidak menguntungkan (*acceptable*). Sebagai contoh, perusahaan X tersebut memiliki divisi IT sendiri dan sudah membangun komponen P. Pembuatan dari dua komponen yang lain diserahkan pada kontraktor Y. Persoalannya, perusahaan Y menjual produk Q dan R belum tentu bersedia menyerahkan kode (*code*) dari produknya pada X, hal ini akan sangat menghambat X dalam melakukan pengujian dengan metoda pengujian tradisional. Persoalan lain akan timbul pula kalau komponen Q ingin diganti. Kalau uji tradisional yang digunakan, maka pengujian ulang harus dilakukan. Lebih buruk lagi, apabila X memutuskan untuk menjual P ke pasaran, harus dilakukan pengujian untuk

berbagai macam Q yang ada di pasar untuk menjamin kompatibilitas produknya. Hal ini akan membutuhkan biaya yang sangat mahal. Bahkan apabila ketiga komponen P, Q, dan R dikembangkan oleh X sendiri dan hanya dipakai untuk keperluan sendiri, persoalan masih masih bisa timbul. Apabila komponen P terlebih dahulu selesai dibuat, tidak bisa langsung diujinya, tetapi harus menunggu sampai kedua komponen yang lain selesai dibuat baru dapat diuji.

Solusi untuk persoalan-persoalan di atas adalah *modularitas*. Konsepnya sendiri bukan konsep baru, tetapi teori tentang modularitas masih relatif baru. Teori ini juga disebut teori komposisi komponen, biasanya dibangun sebagai ekstensi dari sebuah logika pemrograman dengan menyediakan teorema-teorema yang menyatakan bagaimana prilaku dua komponen P dan Q yang berbeda mempengaruhi satu sama lain dalam komposisi $P||Q$ dari kedua komponen tersebut. Teorema tersebut biasanya ditulis sebagai :

$$\frac{P \text{ sat } A \ , \ Q \text{ sat } B}{P||Q \text{ sat } C_{(A,B)}}$$

dengan A, B, dan C masing-masing menyatakan sifat temporal dan C diperoleh dengan mengkombinasikan A dan B dengan cara tertentu. Yang menarik dari teorema seperti ini adalah, apabila digunakannya secara *goal-driven*. Apabila C adalah spesifikasi sistem yang harus dibangun, teorema tersebut mengatakan bahwa cukup dibuat komponen P yang memiliki sifat A. Komponen ini akan bekerjasama secara baik dengan *semua* komponen lain yang memenuhi sifat B. Perhatikan bahwa A adalah sifat lokal dari P, sehingga pengujiannya dapat dilakukan secara lokal. Perhatikan pula bahwa verifikasi A cukup untuk menjustifikasi sifat C yang merupakan sifat sistem secara keseluruhan.

Kebanyakan logika pemrograman memiliki teorema komposisi seperti diatas. Hanya saja mereka berbeda dalam 'kekuatannya'. Pengembangkan komponen P, biasanya menginginkan sifat B selemah mungkin. B bisa dilihat sebagai batasan bagi lingkungan dari P. Makin lemah batasan ini, makin banyak jenis komponen Q yang dapat memenuhinya, sehingga dengan kata lain P makin *reusable*. Ini tentunya sifat yang menguntungkan dalam pasaran. Sebaliknya, kalau B terlalu lemah, ini juga akan menyulitkan pengembang dari Q dalam merancang komponen yang akan memenuhi B. Jadi, pengembang Q menginginkan B yang lebih kuat (lebih spesifik), karena B yang seperti ini mengandung lebih banyak 'petunjuk'. Sayangnya tidak banyak logika pemrograman yang mampu memberikan batasan yang optimal. Logika seperti UNITY versi Chandy dan Misra [2] atau versi Udink [9] memberikan batasan yang terlalu kuat. Sebaliknya, teori komposisi umum versi Abadi dan Lamport [1] atau versi Collete dan Knapp [3] memberikan batasan yang terlalu lemah.

Teori yang dikembangkan di sini berawal dari teori komposisi yang khusus dibuat untuk komponen-komponen yang melakukan sinkronisasi lewat teknik yang disebut *mutual exclusion*. Dalam teknik ini hanya ada satu komponen yang pada suatu saat memakai sejumlah sumber daya (*resource*) bersama. Spesifikasi sebuah komponen ditulis dalam bentuk :

$$P \vdash A \text{ using } V$$

yang berarti bahwa komponen P menunjukan sifat temporal A. Sifat temporal tidak semuanya persisten. Misalnya, spesifikasi

$$x{=}0 \text{ leads-to } y{=}0$$

mengatakan bahwa apabila dalam eksekusi sebuah program harga variabel x menjadi 0, maka suatu saat harga y juga menjadi 0. Sifat ini sementara, karena pada saat y menjadi 0, maka sifat tersebut terpenuhi, jadi seolah-olah bisa dianggap sebagai 'selesai'. Klausul

using V

di atas berarti bahwa 'selama' periode berlakunya prilaku A tersebut, komponen P membutuhkan akses ekslusif ke sumber-sumber daya yang disebutkan di dalam daftar V. Konsekwensinya, untuk mempertahankan konsistensi prilaku A dalam komposisi $P||Q$, maka komponen Q harus diberi batasan supaya tidak melakukan interferensi (*update*) ke sumber daya di V. Batasan seperti ini dinyatakan dalam bentuk :

Q at a preserves V unless b

yang berarti bahwa pada saat a mulai berlaku, Q akan berhenti melakukan interferensi ke sumber-sumber daya di V. Q bisa menghentikan perioda non-interferensi ini dengan memberi nilai *true* ke b.

Teori komposisi yang dihasilkan cukup kuat dan elegan (lihat [5,6]). Kelebihan utamanya adalah bahwa batasan yang dihasilkannya, yaitu dalam bentuk klausul using dan spesifikasi preserves, jauh lebih spesifik dari yang diberikan oleh teori komposisi umum versi [1,3], tetapi tidak sesempit seperti yang diberikan oleh [2,8]. Tentunya diharapkan bahwa ini merupakan kompromi yang dapat diterima. Keuntungan lainnya adalah bahwa batasan tersebut relatif mudah diverifikasi.

Kekurangan dari teori tersebut adalah bahwa *mutual exclusion* seringkali terlalu membatasi untuk dipakai dalam jenis-jenis aplikasi tertentu. Untuk itu perlu dilakukan generalisasi. Ringkasan dari generalisasi inilah yang akan ditampilkan dalam tulisan ini. Pertama-tama akan disampaikan model sistem dan komputasi yang akan digunakan, baru kemudian diulas teori intinya. Setelah itu akan diberikan kesimpulannya.

2. PERMODELAN

Komposisi komponen perangkat lunak pada dasarnya adalah komposisi paralel. Karena itu teori komposisi paling baik diberikan apabila teori prilaku paralel digunakan sebagai basisnya. Ada banyak teori seperti ini. Di sini akan digunakan model interleaving versi [2,4] karena model ini banyak digunakan dan sederhana. Lebih spesifik, yang akan diikuti adalah model UNITY [2].

Komponen adalah program. Program pada dasarnya terdiri dari aksi-aksi atomis. Untuk menyederhanakan persoalan, setiap aksi atomis akan dianggap tidak menggantung (*terminating*). Sebuah aksi atomis dapat dispesifikasikan oleh *statement* berbentuk :

$$\begin{array}{lll} \text{case } g_1 & \rightarrow A_1 \\ \quad\; g_2 & \rightarrow A_2 \\ \quad\; \ldots & \\ \quad\; \text{else} & \rightarrow A_n \end{array}$$

dengan A_i merupakan *assignment* simultan. Penggunaan else wajib untuk membuat aksi selalu bisa dieksekusi (*always enabled*).

Sifat input/output dari sebuah komponen adalah relasi antara hasil yang diperoleh setelah komponen tersebut selesai dieksekusi dengan input yang diberikan padanya. *Sifat temporal* sebuah komponen adalah sifat dari prilakunya selama eksekusi. Sifat temporal lebih sulit untuk dipertahankan konsistensinya dalam komposisi komponen. Walaupun demikian, banyak program yang sifat temporalnya jauh lebih penting dari sifat input/outputnya. Sebagai contoh: sistem operasi, sistem pengendali lalu lintas, dan server basis data. Di sini akan lebih dikonsentrasikan pada pemodelan dari sifat temporal. Untuk itu lebih mudah apabila eksekusi sebuah komponen P dianggap tak berhingga (tak pernah berhenti). Pada setiap langkah, sebuah aksi dari P dipilih secara acak dan dieksekusi. Dengan sendirinya jumlah kombinasi eksekusi P adalah tak berhingga. Hal ini memodelkan paralelisme internal dari aksi-aksi di dalam P sendiri. Model ini sangat primitif. Struktur kontrol yang lebih canggih seperti iterasi digunakan while.

Komposisi paralel dua komponen P dan Q ditulis $P||Q$ dan didefinisikan sebagai sebuah komponen yang aksi-aksinya terdiri dari aksi dari P dan aksi dari Q. Akan diasumsikan bahwa eksekusi dari sebuah komponen selalu *weakly fair*. Artinya, sistem tidak boleh terus menerus mendiamkan sebuah aksi (sehingga aksi tersebut tidak dieksekusi). Karena itu, dalam eksekusi komponen, setiap aksi akan dieksekusi tak berhingga kali.

3. TEORI KOMPOSISI

Untuk menjaga konsistensi prilaku A dari sebuah komponen P dalam komposisi $P||Q$ dimana Q adalah komponen lain, tidak perlu diketahui seluruh prilaku Q. Seperti dikatakan sebelumnya, secara praktis ini juga tidak riil karena akan membuat P tidak *reusable*. Untuk itu, digunakan abstraksi dari Q, yaitu karakterisasi secukupnya dari prilaku Q yang diperlukan untuk menjada konsistensi A dalam $P||Q$.

Di sini akan digunakan aksi sebagai karakterisasi abstrak prilaku sebuah komponen pada perioda tertentu. Untuk sebuah aksi e didefinisikan Pset e sebagai himpunan predikat state yang bisa dijaga stabil oleh e. Dengan kata lain, aksi e tidak bisa melanggar predikat-predikat di dalam Pset e. Formalnya :

$$\text{Pset } e = \{ p \mid \{p\}\ e\ \{p\} \}$$

dimana $\{p\}\ e\ \{q\}$ adalah tripel Hoare. Dalam hal ini, artinya adalah apabila aksi e dieksekusi di sebuah state yang memenuhi predikat p, maka ia akan memindahkan sistem ke sebuah state yang memenuhi predikat q.

Definisi berikut ini menjabarkan kapan sebuah komponen dikatakan *mengimplementasi* karakterisasi abstraknya.

$$J \vdash Q \text{ at } a \text{ implements } e \text{ unless } b$$
$$=$$
$$J \text{ stable in } Q$$
$$\wedge$$
$$(\forall \alpha, p : \alpha \in Q \wedge p \in \text{Pset } e : \{J \wedge a \wedge p \wedge \neg b\}\ \alpha\ \{(a \wedge p) \vee b\})$$

Artinya adalah sebagai berikut. Katakanlah suatu saat J menjadi stabil dalam eksekusi Q. Kemudian, dalam perioda stabilitas J tersebut a berlaku. Maka, semenjak saat itu Q akan berprilaku seperti e, yaitu ia tidak akan melanggar predikat-predikat di Pset e. Disamping itu, Q tetap mempunyai pilihan untuk mengakhiri perioda dimana ia berprilaku seperti e tersebut, yaitu dengan menset predikat b.

Peranan dari parameter J akan diterangkan nanti. Sebuah predikat J dapat dipertahankan stabil oleh komponen Q (ditulis J stable in Q) apabila J tidak bisa dilanggar oleh aksi-aksi di dalam Q. Formainya :

$$J \text{ stable in } Q = (\forall \alpha : \alpha \in Q : \{J\}\ Q\ \{J\})$$

Perilaku temporal dari sebuah komponen biasanya dispesifikasikan dengan menggunakan operator seperti next, unless, dan leads-to. Sekarang akan didefinisikan varian dari operator-operator tersebut untuk mendeskripsikan prilaku yang lebih spesifik, yaitu prilaku yang tidak sensitif terhadap interferensi dari lingkungan (*environment*), selama lingkungan tersebut mengimplementasi karakterisasi abstrak tertentu.

Di sini akan digunakan unless, ensures, dan leads-to, dengan definisi axiomatis versi UNITY, karena ini lebih mudah dilakukan ketimbang menggunakan definisi operasional versi LTL [4]. Dalam interpretasi klasikal, prilaku p unless q oleh sebuah komponen P artinya: apabila dalam eksekusinya P masuk ke dalam state yang memenuhi p, maka ia akan tetap berada dalam p, atau suatu saat (tetapi tidak harus) ia akan melakukan transisi ke sebuah state yang memenuhi q. Prilaku p

ensures q artinya sama seperti p unless q, tetapi transisi ke q dijamin akan terjadi, dan transisi ini bisa dilakukan oleh satu aksi saja. Prilaku unless mendeskripsikan sifat keamanan (*safety*) sebuah komponen, sementara prilaku ensures mendeskripsikan kemajuan (*progress*). Prilaku p leads-to q adalah *progress* secara umum (diwujudkan oleh kerjasama multi-aksi), dan secara matematis adalah *closure* transitif dan disjungtif dari ensures. Lihat [2,4] untuk definisi formal mereka yang standard. Definisi formal mereka yang baru adalah sebagai berikut.

$$
\begin{array}{l}
P, J \vdash p \text{ unless } q \text{ assuming } e \\
= \\
J \text{ stable in } P \\
\wedge \\
p \in \text{Pset } e \wedge q \in \text{Pset } e \\
\wedge \\
(\ \forall \alpha : \alpha \in P : \{J \wedge p \wedge \neg q\}\ \alpha\ \{p \vee q\}\ )
\end{array}
$$

$$
\begin{array}{l}
P, J \vdash p \text{ ensures } q \text{ assuming } e \\
= \\
P, J \vdash p \text{ unless } q \text{ assuming } e \\
\wedge \\
(\ \exists \alpha : \alpha \in P : \{J \wedge p \wedge \neg q\}\ \alpha\ \{q\}\ )
\end{array}
$$

Selanjutnya, $P, J \vdash p$ leads-to q assuming e didefinisikan sedemikian rupa sehingga relasi :

$$(\lambda p\ q.\ P, J \vdash p \text{ leads-to } q \text{ assuming } e\ )$$

merupakan *closure* transitif dan disjunctif terkecil dari relasi :

$$(\lambda p\ q.\ P, J \vdash p \text{ ensures } q \text{ assuming } e\ )$$

Parameter J, yang merupakan sebuah predikat stabil, ditambahkan untuk memodelkan keberadaan sebuah komponen dalam suatu fase tertentu. Tidak seperti pendekatan versi Sanders [7], fase ini tidak harus invarian karena fase tersebut bisa jadi tidak bisa dicapai (*reachable*) kalau P dieksekusi sendiri, dan baru bisa dicapai dengan pertolongan komponen lain.

Teorema komposisi utama ini asal mula idenya berasal dari Singh, yang memberikan teoremanya dalam sebuah nota UNITY [8] di tahun 1989. Disini teorema Singh tersebut berbunyi :

$$
\frac{\begin{array}{l} P, J \vdash p \text{ REL } q \text{ assuming } e \\ J \vdash Q \quad \text{at } a \text{ implements } e \text{ unless } b \end{array}}{P, J \vdash p \wedge a \text{ REL } (q \wedge a) \vee \neg a \vee b \text{ assuming } e}
$$

dimana REL adalah operator unless, ensures, atau leads-to.

Teorema Singh tersebut dapat diartikan, kurang lebih sebagai berikut. Sebuah komponen P berprilaku, misalnya, p leads-to q, yang artinya ia dapat maju dari p ke q. Kemajuan ini, meng-assume e. Artinya, diharapkan lingkungan P selama prilaku tersebut mengimplementasikan e. Dilain pihak, komponen Q berjanji akan mengimplementasi e segera setalah a berlaku, dan sampai b berlaku. Jadi, apabila dalam komposisi $P||Q$ suatu saat p dan a berlaku, maka akan ada dua kemungkinan. Pertama, Q dapat mengimplementasi e cukup lama sehingga P dapat maju ke q. Kedua, konsistensi Q dalam mengimplementasi e terputus terlalu dini sehingga P mungkin tidak bisa maju ke q. Akan tetapi dalam hal ini diketahui dua hal, ataukah pemutusan dini ini disebabkan karena a berhenti berlaku, atau karena b berlaku.

Sebagai contoh sederhana, bayangkan dua komponen R dan W yang sama-sama menggunakan buffer x. Pekerjaan W adalah mengisi buffer x dengan 'dokumen' untuk R. R akan mengambil dokumen dari x satu persatu untuk diproses. Jadi, salah satu pekerjaan dari R adalah mengambil dokumen dari x. Spesifikasinya :

$$R, J \vdash d \in x \text{ leads-to } d \notin x \text{ assuming } e$$

J dibiarkan tidak spesifik karena peranannya tidak terlalu penting dalam contoh sederhana ini. Bagaimana dengan e? Ingat bahwa ini adalah spesifikasi abstrak dari lingkungan R (dalam hal ini W). Dalam pendekatan yang menggunakan *mutual exclusion* akan digunakan e yang menuntut W untuk memberikan R akses ekskusif ke x selama ia sibuk mengambil d dari x. Ini tentunya kurang efisien, karena W cuma bisa mengisi x dengan dokumen baru. Jadi mereka seharusnya bisa bekerja lebih paralel. Dengan teori baru ini sekarang hal tersebut bisa dinyatakan secara formal. Coba lihat spesifikasi Pset e berikut ini :

$$\begin{array}{l} \underline{\text{Pset}}\ e \\ = \\ \{\, p \mid\ p\ \underline{\text{conf}}\ (\underline{\text{loc}}\ R \cup \{x\}\,) \\ \wedge \\ (\,\forall X ::\ \{\, p\}x := [X] +\!\!+ x\ \{\, p\}\,)\ \} \end{array}$$

dimana p conf (loc $R \cup \{x\}$) artinya predikat p adalah predikat yang hanya membatasi harga dari x dan variabel-variabel lokal dari R. Perhatikan bahwa aksi internal W maupun aksinya mengisi dokumen baru ke x tidak bisa menghancur-kan predikat-predikat di Pset e. Jadi, e yang Pset-nya seperti ini cocok untuk dipakai sebagai karakterisasi abstrak dari W. Tepatnya, diharapkan bahwa W akan memenuhi spesifikasi :

$$J \vdash W\ \underline{\text{at}}\ \mathit{true}\ \underline{\text{implements}}\ e\ \underline{\text{unless}}\ \mathit{false}$$

dimana e adalah seperti diatas. Menggunakan teorema Singh kelihatan jelas bahwa W yang seperti itu, apabila dikomposisikan dengan R yang memenuhi spesifikasi sebelumnya, yaitu :

$$R, J \vdash d \in x\ \underline{\text{leads-to}}\ d \notin x\ \underline{\text{assuming}}\ e$$

Akan menghasilkan sistem yang berprilaku seperti yang diinginkan semula, yaitu :

$$R \,\|\, W,\ J \vdash d \in x\ \underline{\text{leads-to}}\ d \notin x\ \underline{\text{assuming}}\ e$$

Teorema Singh bukan satu-satunya teorema yang dibutuhkan untuk sebuah teori komposisi yang efektif (teorema ini memang yang terpenting). Karena keterbatasan ruangan di sini tidak bisa ditampilkan teorema-teorema lainnya. Pembaca dapat melihatnya di [5,6]. Yang ini adalah untuk versi *mutual exclusion*, tetapi variannya, untuk generalisasi yang diberikan dalam tulisan ini, bisa diharapkan untuk menunjukkan banyak kesamaan.

4. KESIMPULAN

Dengan demikian telah diberikan bagian-bagian terpenting dari teori komposisi. Teori ini merupakan teori baru dan diharapkan merupakan kompromi yang baik, dalam arti menghasilkan batasan komponen yang tidak terlalu sempit sehingga pengembang masih leluasa menghasilkan komponen yang *reusable*, tetapi di lain pihak juga cukup spesifik dalam memberikan petunjuk-petunjuk yang berguna untuk pengimplementasian sebuah komponen. Dirasakan teori ini besar potensinya untuk digunakan di lapangan, akan tetapi sejauh ini masih belum ada studi kasus yang layak untuk menunjukkan bahwa teori ini memang *scalable* untuk menangani persoalan-persaoalan realistis. Hal ini dapat dijadikan sebagai bahan penelitian lebih lanjut. Disamping itu, perlu juga diteliti cara-cara untuk memverifikasi batasan dalam bentuk assuming dan implements dengan efisien.

REFERENSI

[1] M.Abadi and L.Lamport, *"Computing specifications. ACM Transactions on Programming Languages and Systems"*, 15(1):73-132, January 1993.

[2] K.M. Chandy and J. Misra, *"Parallel Program Design a Foundation"*, Addison-Wesley, 1988.

[3] P. Collette & E. Knapp, *"Logical foundations for compositional verification*

and development of concurrent programs in UNITY. LNCS", 936:353-367, 1995.

[4] Z. Manna and A. Pnueli, *"The Temporal Logic of Reactive and Concurrent Systems Specifications"*, Springer-Verlag, 1992.

[5] I.S.W.B. Prasetya, S.D. Swierstra, and B. Widjaja, "Component-Wise Formal Approach to Design Distributed Systems". *Tech. Rep. UU-CS-2000-01*, Dept. of CS, Utrecht Univ., 2000.

[6] I.S.W.B. Prasetya, S.D. Swierstra, and B. Widjaja, "A Composition Theory for Distributed Systems, Based on Temporary Non-interference", *Draft, 2001.* Available on request: wishnu@cs.uu.nl.

[7] B.A. Sanders, Elminiating the substitution axiom from UNITY logic. *"Formal Aspects of Computing"*, 3(2):189-205, 1991.

[8] A.K. Singh, *"Leads-to and program union. Notes on UNITY"*, 06-89, 1989.

[9] R. Udink, *"Program Refinement in UNITY-like Environment"*, PhD thesis, Dept. of CS, Utrecht Univ., 1995. Downloadable from www.cs.uu.nl.